Towards Unifying Rules and Policies for Semantic Web Services

Nima Kaviani[1], Dragan Gašević[1], Marek Hatala[1], David Clement[2], Gerd Wagner[3]
[1]*Simon Fraser University Surrey, Canada*
[2]*Visiphor Corporation, Canada*
[3]*Brandenburg University of Technology at Cottbus, Germany*
{nkaviani, dgasevic, mhatala}@sfu.ca, david.clement@visiphor.com, wagnerg@tu-cottbus.de

Abstract

Simplifying the discovery of web services on one hand and protecting them from misuse on the other hand has initiated several lines of research in the area of policy-aware semantic web services. However, the diversity of approaches, ontologies and languages chosen for defining Semantic Web services and policies has made the research area cluttered. It is now ambiguous how different registries and agents with different policy languages and Semantic Web service ontologies would share their information. In this paper we try to solve the problem of exchanging information between the registries by defining an interchange framework to transform business rules and concepts from one language to another using a third intermediary language called R2ML. The expressivity of the new framework exempts any service provider or service requester from the difficulties it may encounter during the process of transformation from one business rule language to the other. It also guarantees that information loss during transformation would be minimal.

1. Introduction

Semantic Web services (SWS), as the augmentation of Web service descriptions through Semantic Web annotations, facilitate the higher automation of service discovery, composition, invocation, and monitoring on the Web [17]. Semantic Web ontologies and its ontology languages (OWL and RDF(S)) are recognized as the main means of knowledge representation for Semantic Web services [20]. Such ontology-enriched Web service descriptions are later used in the negotiation process between service clients and service provides, which is defined by a set of abstract protocols of Semantic Web Service Architecture (SWSA) [6].

However, the current proposed standards for describing Semantic Web services (i.e. OWL-S [14], WSDL-S [1], Web Service Modeling Ontology [7], and Semantic Web Service Language – SWSL [2]) demonstrate that it is important to use a rule language in addition to ontologies. This allows run-time discovery, composition, and orchestration of Semantic Web services by defining preconditions or post-conditions for all Web service messages exchanged [13]. For example, OWL-S recommends using OWL ontologies together with different types of rule languages (SWRL, KIF, or DRS), WSMO uses F-Logic, while WSDL-S is fully agnostic about the use of a vocabulary (e.g., UML, ODM, OWL) or rule language (e.g., OCL, SWRL, RuleML). Usually, Semantic Web service descriptions use only parts of rules representing logical formulas that may have a Boolean result. It is important to point out that there is no agreement upon which rule language to use for Semantic Web services or what type of reasoning (open or closed world assumption) should be supported.

Besides satisfying clients goals when using Semantic Web services, trust is another important aspect that should be established between a client and service. Addressing this problem, researchers proposed the use of policy languages. A policy is a rule that specifies under which conditions a resource (or another policy) might be disclosed to a requester [16]. To define polices on the Semantic Web, various policy languages have been proposed such as PeerTrust [16], KAoS [21], Rei [12], and PROTUNE [4]. As [16] reports, trust management policies are also defined as parts (most commonly preconditions) of Semantic Web service descriptions.

It is obvious that besides various Semantic Web services description languages, we have various policy languages and various rule languages. All these languages are based on different syntactic representations and formalisms with no explicitly

defined mapping between them. This hampers the use of Semantic Web services from two different perspectives. One perspective is automatic negotiation between service client agents and service provides and automatic matchmaking, where agents and matchmakers should be able to "understand" various rule/policy/service web service description languages. Another perspective is that of a knowledge management worker who needs to be able to express the rules and policies in a single form rather than in a broad variety of forms. To attempt to represent the same rules and policies in many forms is cumbersome, time consuming and error prone but it is the only choice currently available if a broad base of interoperability is required.

In the paper, we start from the presumption that rules encoded in policies and semantic web services are parts of business rules [23] [4], and that we should be able to share them by using the same representation. Such a rule language should enable modeling different types of rules such as reaction rules considering the event-driven nature of Web services (or so-called Event-Condition-Action Rules), derivation rules considering the importance of inferring new facts (such as RuleML), and integrity rules considering the deontic, i.e. "must", nature of policy languages. Unfortunately, current Web rule markup languages such as RuleML and SWRL are unable to express all these types of rules.

In our approach, we propose the use of REWERSE Rule Markup Language (R2ML) [22], which addresses almost all use-cases and requirements for a Rule Interchange Format (RIF) [8], along with a set of transformations between Semantic Web service description (e.g., WSDL-S), rule, and trust management policy languages. We illustrate the benefits of our approach using a Semantic Web Service Architecture example where R2ML is used to share Semantic Web service descriptions and policies in the process of matchmaking and trust negotiation. In the next section, we motivate our research by describing an example based on the present solutions to Semantic Web services.

2. Motivation

Semantic Web service registries are usually used together with matchmakers to help the requesting entities find their desired service providers based on the defined constraints and needs. Involving policies in the process of discovering relevant service providers would help both the requesting entity and the service provider to check their constraints before starting a negotiation process, protect sensitive information, and prevent from information leakage before going through the negotiation. In this case the policy constraints are placed in the Semantic Web service description for every single service available in the registry. The policies are also embedded in the SOAP messages sent by the requesters while seeking for the desired services.

In [16], the authors suggest three different architectures for service provider discovery, namely trusted matchmaking at registry side, trusted matchmaking at client side, and distributed matchmaking using a middle agent. In these suggested architectures they try to address the security concerns of both peers of negotiation, i.e. the requester and the service provider, and especially protect their sensitive policies and information. Figure 1 shows the third architecture which has been considered as the most promising one by the authors.

Regardless of the architecture used in service provider discovery, what seems to be totally missing in the proposed solutions is the communication language between the entities involved in the process of service discovery. [16] assumes that both the requester agent and the service provider have defined their policies in PeerTrust and their Semantic Web service description in WSMO. [12] gives another suggestion in which Rei, as the policy language, is used together with OWL-S as the Semantic Web service description ontology to describe the service.

However, it is not reasonable to assume that all the registries and the service requesters around the world would admit the same policy and description languages for their services. As it is shown in Figure 1 different service providers, even those placed in a single registry, may have been using different semantic web services with different policy languages. The proposed architectures and solutions does not deal with how to matchmake the requests and advertisements where the policy languages and the semantic web service descriptions do not match.

In the current proposals, the requestors, the broker agents, the registries, and the service providers must have different transformers from each policy language or Semantic Web service language to the others. This is hard to achieve due to the vast variety of these languages is really hard to achieve. Moreover, as the flexibility of a mapping from one language to another has not been considered in defining policy and web description languages, chances are that we encounter some information loss during transformation from one language to another which may result in unwanted changes in the policies of either of the peers and compromise their privacy. In order to solve the problem, the focus should be on finding an interchange

format language with the highest possible degree of interoperability to cope with different available semantic Web Service languages and policy languages. In the rest of the paper we propose a new solution to interchange the policy languages and semantic web service description languages, from one to another, with as less information loss as possible.

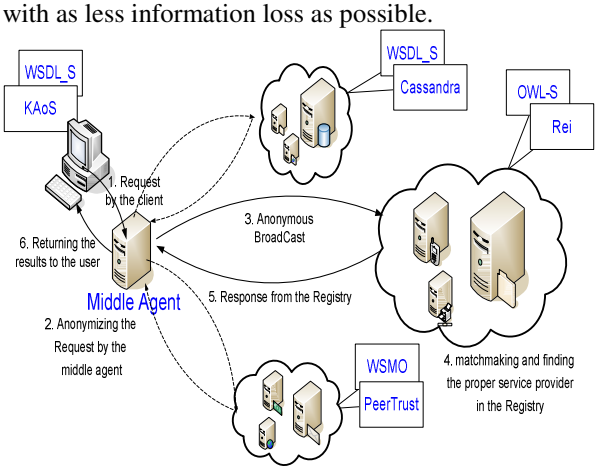

Figure 1. Distributed Architecture for Registry and MatchMaker

3. Background

3.1 Semantic Web Service Description

Considering the main feature of Semantic Web services of semantically enriching Web service descriptions, researchers have so far proposed several solutions to this problem. Here we briefly touch the submitted solutions to W3C.

OWL-S is an OWL-based ontology for describing Semantic Web services. The ontology consists of three main parts: the service *profile* for advertising and discovering services; the *process model* for detailed descriptions of services' operation; and the *grounding*, for providing details on how to interoperate with a service via messages [14]. Some of functionalities that OWL-S provides are: annotating types of input and output parameters of services with OWL-ontology concepts, defining preconditions and effects of input and output messages by logical expressions encoded as literals and XML literals, and specifying different types of composite services (e.g. sequence, split, and join-split).

Web Service Modeling Ontology (WSMO) consists of *ontologies* for defining terminology; *Web service* functional and behavioral descriptions, user *goals*, and *mediators* that automatically handles interoperability between different WSMO element. Comparing to the OWL-S, WSMO defines its own ontology language, while it also defines its own rule language for defining logic expressions. That language has backward compatibility with F-Logic. It is important to note that WSMO introduces postconditions, besides preconditions and effects that are also present in OWL-S. Postconditions cover the data output while effects specify general state changes [9].

Semantic Web Service Ontology (SWSO) is a conceptual model for describing Web services [2]. The complete axiomatization is given in first-order logic, using the Semantic Web Service Language consisting of SWSL-FOL (First Order Logic) and SWSL-Rules. In fact, SWSO presents a first-order ontology for Web services, expressed in SWSL-FOL and its partial translation expressed in SWSL-Rules. SWSO also includes material about grounding its process models with WSDL. It is important to say that SWSL-FOL and SWSL-Rules language are serialized by using RuleML.

WSDL-S is a rather evolutionary approach to Semantic Web services that only introduces a few WSDL extensions, and thus preserves the full compatibility with the standard WSDL [1]. WSDL-S is fully ontology and rule language agnostics meaning that one can annotate WSDL XML schema types with any vocabulary language (e.g., OWL or UML) and define preconditions and effects (there is not postconditions) with any rule language (e.g., OCL or R2ML).

3.2 Policy Language

As mentioned earlier, there are several policy languages proposed so far with the goal of protecting the privacy of information and authorizing requesters by providing different levels of access to the available resources and information. The syntax of these languages varies from rigid ordinary logic languages such as Cassandra [3], which is based on Constraint Language Programming, PeerTrust [16] and PROTUNE [4], which use a Prolog meta-interpreter, to more relaxed markup languages such as KAoS [21] and Rei [Kagal & Joshi, 2003] Referring to all of the available policy languages is beyond the purpose of this paper and here we just mention the most important ones.

PeerTrust is a trust negotiation engine for semantic web and P2P networks [16]. PeerTrust's language is based on first order Horn rules (definite Horn clauses), i.e. rules of the form:

$$lit_0 \leftarrow lit_1, \ldots, lit_n$$

where each lit_i is a positive literal of the form $P_j(t_1, \ldots, t_n)$, P_j is a predicate symbol, and $t_i(i=1..n)$ are the arguments of this predicate. It can be combined with WSMO to define policy-aware web services as it is proposed in [16].

Cassandra is another policy-based language based on CLP [3]. It uses a policy language based on Datalog with constraints and its expressiveness can be adjusted by changing the constraint domain. Policies are specified using the following predicates which govern access control decisions: *permits(e, a)* specifies who can perform which action; *canActivate(e, r)* defines who can activate which role (e is a member of r); *hasActivated(e, r)* defines who is active in which role; *canDeactivate(e, r)* specifies who can revoke which role; *isDeactivated(e, r)* is used to define automatically triggered role revocation. Although aggregation of Cassandra with Semantic Web service descriptions has not been proposed yet, its declarative nature makes it suitable to combine it with some of the available semantic web services, such as WSMO.

Rei is a policy framework that permits specification, analysis and reasoning about declarative policies defined as norms of behavior [Kagal & Joshi, 2003]. Rei adopts a rule-based approach to specify semantic policies. Rei policies restrict domain actions that an entity can/must perform on resources in the environment, allowing policies to be developed as contextually constrained deontic concepts, i.e., right, prohibition, obligation and dispensation. The current version of Rei (2.0) adopts OWL-Lite to specify policies and can reason over any domain knowledge expressed in either RDF or OWL.

KAoS is a framework that provides policy and domain management services for agent and other distributed computing platforms [21]. It has been deployed in a wide variety of multi-agent and distributed computing applications. KAoS policy services allow for the specification, management, conflict resolution and enforcement of policies within agent domains. KAoS adopts an ontology-based approach to semantic policy specification. In fact, policies are mainly represented in OWL as ontologies which make it possible to combine them with Semantic Web Services and then use them to define the policies of a web service provider.

3.3 Semantic Web Service Policies

As we discussed earlier in the motivation section, policies and semantic web service description ontologies are combined to provide a policy-aware specification of semantic web services. The most prominent efforts in this area have been done in [12] and [16]. In [12], OWL-S and Rei have been chosen as web service ontology and policy language respectively. The main reason in selecting OWL-S seems to be the syntactical consistency between OWL-S and Rei.

In [16] the authors have chosen WSMO as their description for semantic web services and mixed it with PeerTrust to add policies. WSMO has been selected because it allows the use of arbitrary logical expressions in the description of the service functionality which gives the authors more complete way of expressing their terms as compared to other approaches. It also uses F-Logic to describe the logical expressions used in the description of the services which in turn makes it possible to align the trust policies described in PeerTrust with functionality descriptions in WSMO.

3.4 Web rule languages

The current Semantic Web standards cover defining vocabularies and ontologies by using Resource Description Framework Schema (RDFS) and Web Ontology Language. However, there is no standard for defining and sharing rules on the Semantic Web. The most important initiative is called Rule Interchange Format (RIF) [8], which defines a set of requirements and use cases for sharing rules on the Web. It is important to point out, that the purpose of this language is to serve as an intermediary language between various rule languages, but it should not provide a formally defined semantic foundation for reasoning on the Web such as OWL for ontologies. Here we name two most prominent rule languages and briefly describe their characteristics.

RuleML is a markup language for publishing and sharing rule bases on the World Wide Web [10]. RuleML builds a hierarchy of rule sublanguages upon XML, RDF, XSLT, and OWL. The current RuleML hierarchy consists of derivation (e.g., SWRL, FOL) and production rules (e.g., Jess). RuleML is based on Datalog. RuleML rules are defined in the form of an implication between an antecedent and consequent, with the meaning whenever the logical expression in the antecedent holds, then the consequent must also hold. However, an important constraint of RuleML is that it can not fully represent all the constructs of various languages such as OCL or SWRL.

The ***Semantic Web Rule Language (SWRL)*** is a proposed rule language based on the W3C Web ontology language OWL [11]. Similar to RuleML rules, a SWRL rule is also in the form of an implication and is considered another type of an axiom on top of the other OWL axiom types. This means that SWRL rules are usually used for defining integrity constraints similar to OCL in UML. Both consequent and antecedent are collections (i.e., conjunctions) of atoms. We should say that the purpose of SWRL is not to be a universal rule syntax for interchanging rules, as it can

not represent many linguistic constructs of other rule languages (e.g., F-Logic, Rei, or OCL) utilized in Semantic Web service descriptions.

4. Our approach

We propose using one general rule representation language with the syntactical capacity to represent various rule constructs mentioned in the previous section. More specifically, we propose using the R2ML language [22]. R2ML is a general purpose rule interchange language that possesses expressivity for representing four types of rules, namely, *integrity*, *derivation*, *reaction*, and *production* rules. Besides rules, R2ML has its own set of constructs for representing vocabularies and domain ontologies similar to UML or OWL. Having in mind such an expressivity, we can use R2ML to represent the following artifacts related to Semantic Web services discussed in the previous section:

- *R2ML reaction rules* can be used to model Semantic Web service descriptions such as WSDL-S, WSMO, and OWL-S.
- *R2ML integrity and derivation rules* can be used to represent trust management policy rules such as the ones defined in Rei, PeerTrust, PROTUNE and KAoS.
- *R2ML vocabularies* that are used for annotating types of Semantic Web services can be used for representing domain vocabularies expressed as UML class diagrams or OWL ontologies.
- *R2ML itself* can be used to represent other rule languages used in Semantic Web applications such as RuleML, SWRL, or OCL.

Along with R2ML, a set of bi-directional transformations between R2ML and all the languages discussed in Section 4 need to be developed for our proposed solution. Given such as a set of transformations, we can transform SWRL rules (via R2ML) into OCL, or a UML vocabulary into OWL ontology [15]. In the rest of this section we give a brief overview of R2ML.

4.1 R2ML: A brief overview

R2ML is a general rule interchange language that tries to address all RIF requirements [8]. Its current version is 0.4 [18]. The abstract syntax of the R2ML language is defined with a metamodel by using OMG's Meta-Object Facility (MOF). This means that the whole language definition can be represented by using UML diagrams, as MOF uses UML's graphical notation. The full description of R2ML in the form of UML class diagrams is given in [18], while more details about the language can be found in [22]. In Figure 2, we give an excerpt of the metamodel that defines derivation (a) and reaction rules (b) which we will use in Sections 5 and 6.

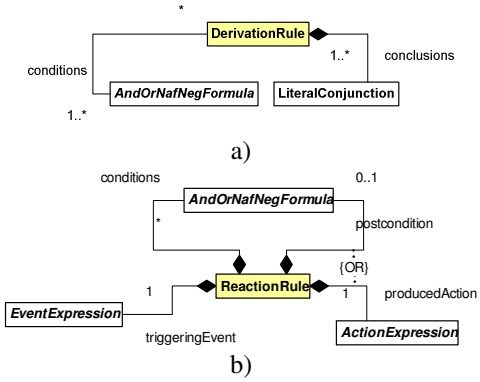

Figure 2. The R2ML definition of derivation and reaction rules

A derivation rule has *conditions* and a *conclusion* (see Figure 2a) with the ordinary meaning that the conclusion can be derived whenever the conditions hold. While the conditions of a derivation rule are instances of the *AndOrNafNegFormula* formula class, representing quantifier-free logical formulas with conjunction, disjunction and negation; conclusions are restricted to quantifier-free disjunctive normal forms without NAF (Negation as Failure, i.e. weak negation). An example of an integrity rule is:

Example 1. *If reservation date of a rental is five days in advance of the rental start date then rental discount is 10.*

A reaction rule, also called an Event-Condition-Action (ECA) rule, consists of a triggering event, a list of conditions, a triggered action and an optional post-condition, which formalizes the state change after the execution of the action. Here we give an example of a reaction rule:

Example 2. *On customer book request, if the book is available, then approve order and decrease amount of books in stock by order quantity.*

The next component of R2ML is a textual concrete syntax defined by an XML schema (so called R2ML XML). The purpose of this schema is to enable sharing R2ML rules among different applications by using XML. Another (but graphical) concrete syntax is a UML-based Rule Language (URML) that extends the UML metamodel with rule concepts (e.g., derivation rules) from R2ML, while URML represents vocabularies by using standard UML classes and their relationships. In Figure 3, we give a URML representation for the reaction rule from Example 2. We should point out that there is a plug-in for Fujaba (a well-known UML tool), called *Strelka*, for modeling

rules by using URML. Strelka serializes URML models in the R2ML XML format.

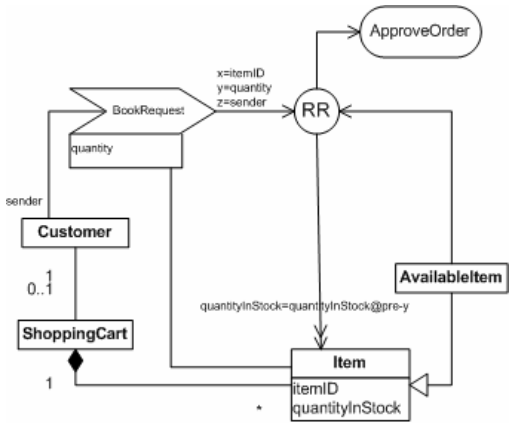

Figure 3. An example of reaction rule

Several transformations between R2ML and other languages (e.g., RuleML, F-Logic, OWL/SWRL, Jess, and UML/OCL) have been implemented. These transformations have been implemented with XSLT or the ATLAS Transformation Language (ATL) [19]. In the rest of the paper, we show how this set of transformations can be extended by providing transformations between R2ML and Semantic Web service description languages and R2ML and trust policy management languages.

5. Mapping R2ML and Semantic Web Service Descriptions

In this section, we describe how R2ML reaction rules are mapped to Semantic Web service descriptions, logical formulas used in their descriptions, and ontology vocabularies referred to from such descriptions. The approach is demonstrated on WSDL-S, while differences with OWL-S and WSMO are discussed later in the section. In a nutshell, WSDL-S [1] has the following features:

- It is defined as an extension of the standard WSDL with the goal to be fully compatible with standard Web service specifications and tools;
- It enables annotating datatypes using domain ontologies through the *modelReference* and *schemaMapping* attributes, but still data types are defined by using XML Schema.
- *Precondition* XML element defines a set of assertions that must be met before a Web service operation can be invoked;
- *Effect* XML element can make statements about what changes in the state are expected to occur upon invocation of the service;
- It is fully agnostic about the language used for defining domain ontologies, referred to in *modelReference* (e.g., UML, ODM, and OWL) and rules used in *precondition* and *effect* elements (e.g., OCL, SWRL, RuleML, R2ML).

As can be seen from the listed features of WSDL-S, this language has many concepts similar to the concepts of reaction rules described above. A WSDL interface operation corresponds to an R2ML reaction rule in the following way:

- A WSDL *input* element is mapped to an incoming message that represents the triggering event of an R2ML reaction rule;
- A WSDL *output* element is modeled as an outgoing message representing the triggered action of an R2ML reaction rule;
- R2ML reaction rule conditions and post-conditions are mapped to the WSDL-S *preconditions* and WSDL-S *effects*, respectively;
- A WSDL *outfault* element is mapped to an alternative action (in the form of an outgoing message) bound to a corresponding error condition in a reaction rule of the form *ON-IF-THEN-ELSE*.

Since WSDL-S does not have any presumption about rule language that will be used in its preconditions and effects, we can transform R2ML conditions and postconditions onto WSDL-S preconditions and effects, respectively, to be expressed in any rule language for which there are transformations implemented with R2ML (e.g., RuleML, SWRL, OCL, and F-Logic). The same is applied to the opposite direction of transformations (i.e., from WSDL-S to R2ML). Our recommendation is that WSDL-S preconditions and effects should be represented in R2ML and then, based on the need in a specific situation, translated to a target rule language. However, there is one constraint in WSDL-S caused by the XML schema definition of WSDL-S about precondition and effect XML elements. Both these elements have the *expression* attribute for defining logical conditions, which can only be set in the form of a text-based syntax (e.g., OCL or F-Logic). This means that we can not used an XML format (e.g., SWRL and R2ML) that will be wrapped by, for example, the *effect* XML element of WSDL-S. So, the R2ML conditions can only be addressed by using XPath referring to specific parts of rules defined in R2ML XML documents (arrow 1 in Figure 4). Another approach is to change the definition of the WSDL-S precondition and effect XML elements (having in mind that it is still only a W3C member submission), so that they can also wrap in an XML literal like it is shown with arrow 2 in Figure 4.

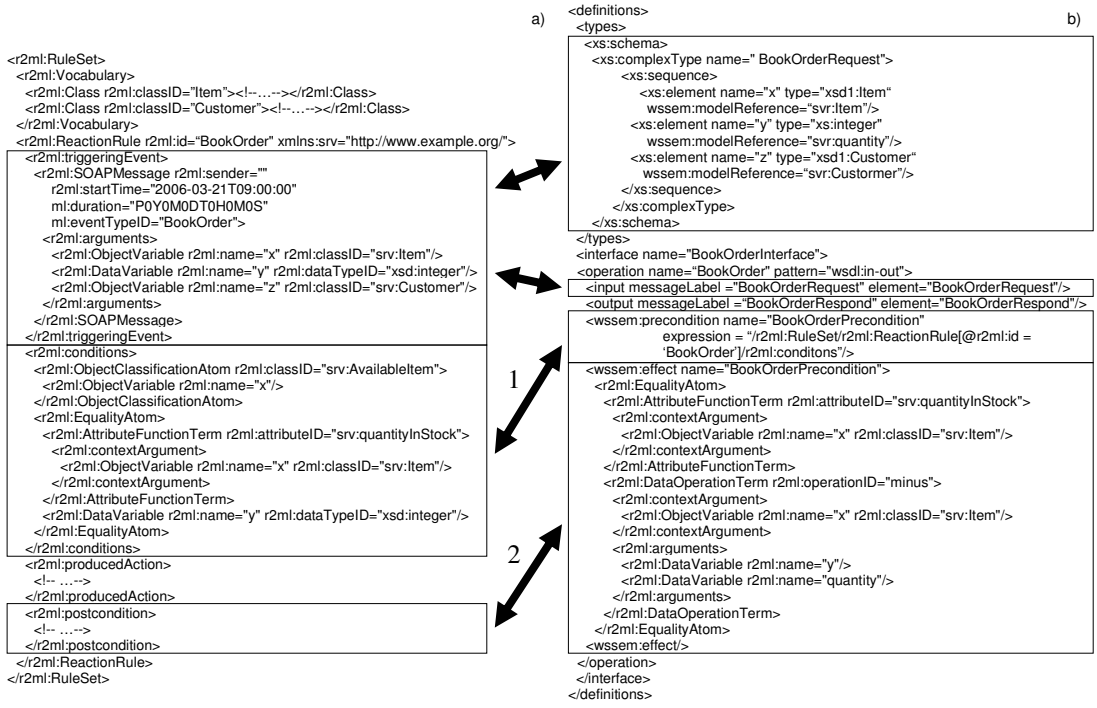

Figure 4. Mappings between reaction rules represented in the R2ML XML format (a) and WSDL-S (b)

We can also apply similar mappings between R2ML and OWL-S. Definition of OWL-S preconditions and effects can be both literal and XML literals, which is fully compatible with two alternatives discussed for WSDL-S (arrows 1 and 2 in Figure 4). Furthermore, vocabularies can be defined in different ontology languages, thus R2MlL transformation can increase the interoperability level. However, OWL-S contains the Process ontology that is used for modeling atomic and composite (e.g., sequence, split, split+join, and any-order) processes. These different types of process execution are represented in R2ML reaction rules by using EventExpressions – see Figure 2 (e.g., SequenceEventExpression or ChoiceEventExpression). A similar situation is with mappings between WSMO and R2ML. R2ML vocabulary elements fully capture WSMO ontologies, while WSMO rules that are based on F-Logic are also fully captured by R2ML rules. Actually, there is already a transformation between R2ML and F-Logic [19]. However, when mapping WSMO service descriptions to R2ML, there is one difference compared to mappings between R2ML and WSDL-S and between R2ML and OWL-S. It is related to output messages where WSMO has two properties, effect and post-condition. Post-conditions are related to the data output, whereas effects are used to define general state changes caused by the execution of the output. We can say that R2ML, OWL-S, and WSDL-S do not distinguish between these two different types of output conditions and both of them are captured with OWL-S and WSDL-S effect and R2ML post condition. However, to provide automatic transformations between R2ML and WSMO, R2ML should be slightly be extended to distinguish between data only and general state change conditions.

6. Policy languages and R2ML

In the previous section we showed that SWS description languages can be modeled using reaction rules. The last step in providing a powerful interchange format between different policy-aware SW services is to map policy rules from a source policy language to R2ML and from R2ML to the target policy language.

Policies for semantic web services are generally divided into privacy policies, in which the global constraints over accessing a service are defined, and authentication policies in which certain privileges are given to each user in accordance to the credentials s/he provides. Privacy policies can be perfectly modeled as integrity rules in R2ML. Each integrity rule is a sequence of logical predicates which through resolution process can eventually hold a value of either *true* or *false*. This logical value can be neatly evaluated as a privacy policy to determine whether or not the process of information exchange can happen after resolving the constraints of the integrity rule. Authentication policies on the other hand can be efficiently modeled as derivation rules. Providing a set of credentials by a peer may satisfy the sequence of constraints in the

condition part of a derivation rule and consequently a series of privileges are given to the peer.

To give the reader a clear understanding of how the policy rules can be exchanged using R2ML, we provide some mappings from policy languages to R2ML and vice versa. However, due to the length limitations in the paper, we limit our samples to authentication policies which are more general than privacy policies in the sense of dealing with conditions and consequences. We choose Rei and PeerTrust to define policies, we consider the same scenarios mentioned in [12] and [16], then we convert them to identical R2ML definitions and discuss the similarities of the results.

The first scenario based on [16] defines an authentication policy in which a discount is given to a buyer only if s/he proves that s/he is a student in a valid university (Figure 5). As it is obvious in Figure 5 the user should prove that s/he belongs to a valid university and the university must certify the validity of the student ID the user provides with regards to her/his name and information. It is worth mentioning that the object following "$" in PeerTrust illustrates the requester to whom the results will be returned and the object following "@" is the entity responsible for certifying the expression that appears before "@". For example the last constraint in Figure 5 considers *University* as the certifying authority for the expression *studentId(Buyer).* The expressivity of R2ML enables us to define the above rule in several different ways; however, we finally chose the transformation in Figure 6 as the most suitable one.

```
discount(BookTitle) $ Buyer ←
    studentId(Buyer) @ University @ Buyer,
    validUniversity(University),
    studentId(Buyer) @ University.
```

Figure 5. An Authentication Policy in PeerTrust

In Figure 6, we have mapped each predicate in PeerTrust to a *GenericAtom* in R2ML with the variables of each predicate defined as an *ObjectVariable* in R2ML. Certification authorities of a predicate in PeerTrust are also modeled as *GenericFunctionTerms* in which the *FunctionID* refers to the role of the authority and an *ObjectVariable* refers to the certifying authority itself. To increase the expressivity and flexibility of the mapping we have explicitly defined a certification authority for the second constraint of Figure 5 as "*self*" which is implicit in the original model defined in PeerTrust. "*self*" in PeerTrust, and correspondingly in our mapping, refers to the entity that is defining the policy. Digging into the sample provided above and our conventions in transforming the concepts, the reader may find it easy to transfer a similar concept from PeerTrust to R2ML and back.

Now that a transformation from a rule in PeerTrust to R2ML has been provided, we follow a completely different scenario mentioned in [12] for a policy defined in Rei. Figure 7a shows the sample scenario in Rei.

```
<r2ml:DerivationRuleSet>
<r2ml:DerivationRule xmlns:plcy="http://www.services.org/Policy/">

<r2ml:conditions>
  <r2ml:ObjectClassificationAtom r2ml:classID="plcy:Buyer">
     <r2ml:ObjectVariable r2ml:name="buyer"/>
  </r2ml:ObjectClassificationAtom>
<!-- Similarly we define university and book as variables from classes University
and Book respectively -->

  <r2ml:GenericAtom r2ml:predicateID="plcy:Buyer:studentID">
     <r2ml:arguments>
        <r2ml:ObjectVariable r2ml:name="buyer"/>
        <r2ml:GenericFunctionTerm r2ml:genericFunctionID="plcy:isCertifiedBy"
           r2ml:typeCategory="order">
           <r2ml:arguments>
              <r2ml:ObjectVariable r2ml:name="buyer"/>
              <r2ml:ObjectVariable r2ml:name="university"/>
           </r2ml:arguments>
        </r2ml:GenericFunctionTerm>
     </r2ml:arguments>
  </r2ml:GenericAtom>

  <r2ml:GenericAtom r2ml:predicateID="plcy:validUniversity">
     <r2ml:arguments>
        <r2ml:ObjectVariable r2ml:name="university"/>
        <r2ml:GenericFunctionTerm r2ml:genericFunctionID="plcy:isCertifiedBy">
           <r2ml:arguments>
              <r2ml:ObjectName r2ml:objectID="plcy:self" />
           </r2ml:arguments>
        </r2ml:GenericFunctionTerm>
     </r2ml:arguments>
  </r2ml:GenericAtom>

<!—The last predicate is similar to the others -->

<r2ml:conclusion>
   <r2ml:GenericAtom r2ml:predicateID="plcy:discount">
     <r2ml:arguments>
        <r2ml:ObjectVariable r2ml:name="book"/>
        <r2ml:GenericFunctionTerm r2ml:genericFunctionID="plcy:requester">
           <r2ml:arguments>
              <r2ml:ObjectVariable r2ml:name="buyer"/>
           </r2ml:arguments>
        </r2ml:GenericFunctionTerm>
     </r2ml:arguments>
   </r2ml:GenericAtom>
</r2ml:conclusion>

</r2ml:DerivationRule>
</r2ml:DerivationRuleSet>\
```

Figure 6. Transformation from PeerTrust to R2ML

In this example permission in using a service will be granted to the requesting entity in case the requester is working on the same project as the provider. Rei is built upon the concepts of OWL and RDF so the relationships are binary as opposed to PeerTrust which can have predicates of the form ternary or higher. Moreover, Rei does not address the certifying authorities as explicitly as PeerTrust and the authorities are dealt with again through defining the binary relationships between the constraints and the entities involved. Figure 7b is showing the transformation from the sample in Figure 7a to R2ML.

`<!— Rei variables used —>

<r2ml:DerivationRuleSet>

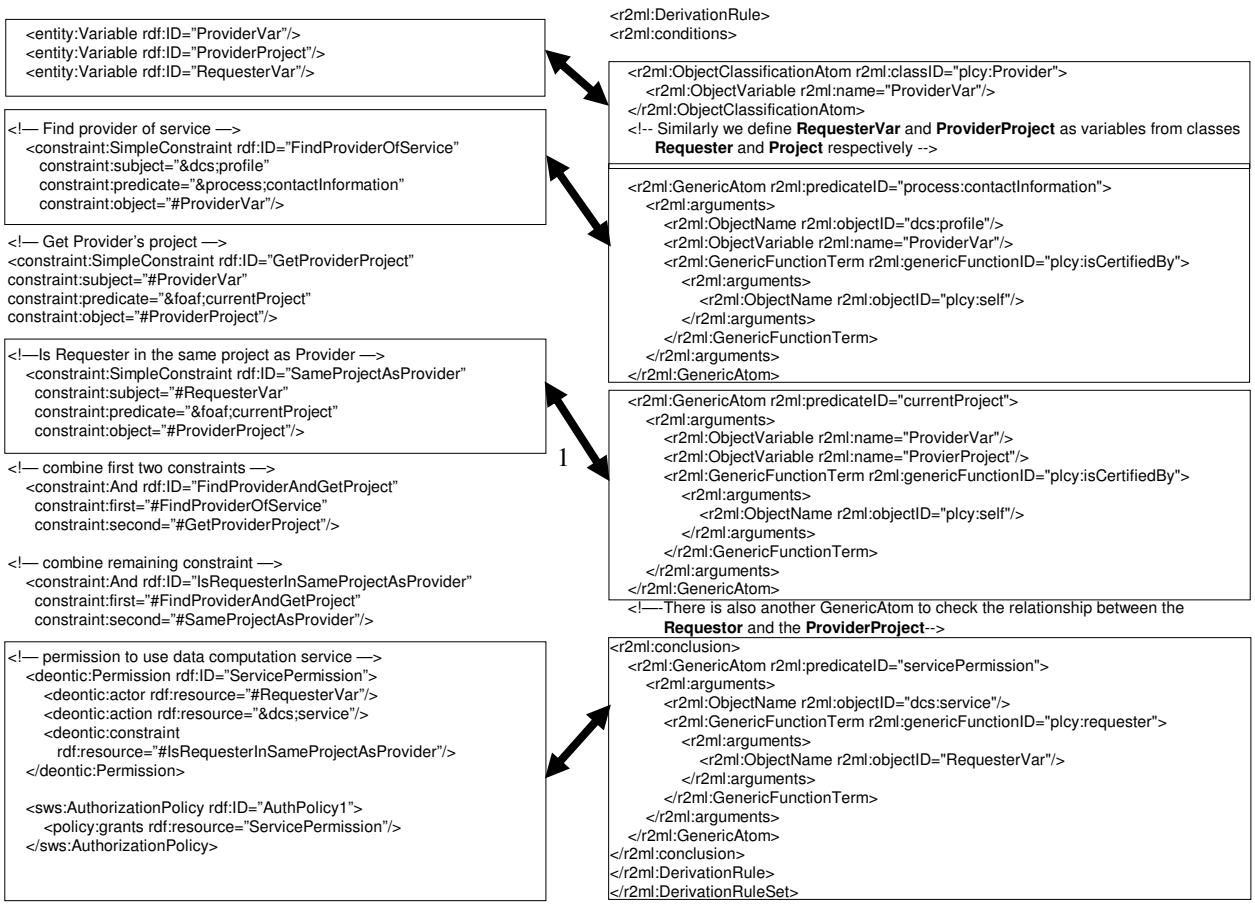

Figure 7. Mapping between Rei and R2ML: a) A Rei policy and b) its counterpart in R2ML

Finally it should be noticed that there are a variety of possible mappings from each policy language to R2ML, for example a *SimpleConstraint* in Rei (as it has been marked with 1 in Figure 7) can be easily and expressively mapped to a *ReferencePropertyAtom* in R2ML which is likely even more expressive than the format we have already come up with (Figure 8). However, transforming the derived R2ML definition to another language, say PeerTrust, is not easy as there is no room for the concept of certification authority in a ReferencePropertyAtom.

```
<r2ml:ReferencePropertyAtom referencePropertyID="currentProject">
  <subject>
    <r2ml:ObjectVariable r2ml:name="RequesterVar"/>
  </subject>
  <object>
    <r2ml:ObjectVariable r2ml:name="ProviderProject"/>
  </object>
</r2ml:ReferencePropertyAtom>
```

Figure 8. A Simple Constraint in Rei expressed as a *ReferencePropertAton* in R2ML

7. Discussion and Conclusion

In this paper we discussed how a comprehensive and general purpose rule markup language such as R2ML can be used as an intermediary language to transform different business rules, descriptions, and policies from one language to another. It was shown how the descriptions for a semantic web service written in WSDL-S or OWL-S can be mapped to and from R2ML. It was also illustrated how the policy languages defined in Rei or PeerTrust look similar after being converted to R2ML and how this similarity could ease the conversion from one language to the other. Our solution in designing an interchange format for Semantic Web services is completely independent of the architecture adopted for service discovery. Availability of transformations from each policy-aware Semantic Web service description to R2ML and back enables the brokers to surf the Web for the desired services regardless of the language they have been originally described in.

Since there is a transformation engine from R2ML to F-Logic already developed, the problem of defining PeerTrust in F-Logic and integrating it with WSMO in order to enforce the policy rules to web services, as it has been discussed in [16], can be neatly covered and solved in our interchange format.

In our framework we can also exploit the graphical notation of URML due to coverage of all the concepts of URML by R2ML and availability of a serialization from URML to R2ML. The graphical notation of URML gives the users the opportunity of designing their Semantic Web service descriptions in a graphical UML-like environment without being familiar with the syntax of any of the policy languages or Semantic Web service ontologies. The designed URML model can then be transformed into R2ML and then into any of the previously mentioned languages for policy and Web service description. Ease of implementation and design is one of the main goals pursued in the community of software engineering and Semantic Web which is addressed in our proposal too.

Despite all the positive points listed above, this paper should be regarded as the starting point for the whole framework we have in mind. This paper is mostly a conceptualization of the possibilities and most of the proposed ideas are targets of the future work. First and foremost, the amount of information loss during transformation from one language into R2ML and then into another language should be investigated more closely. Although we believe R2ML is capable of defining a wide variety of the concepts and although it is open to expansion, we still need to investigate which parts of the language need to expand. Our ultimate goal is to design a framework capable of conforming to the concepts in all business languages. This framework, when fully implemented, will provide an easy way to convert all rule languages to R2ML and then reason over their concepts.

Acknowledgement

We would like to thank to Adrian Giurca and Sergey Lukichev for their valuable comments on the ideas presented in the paper.

9. References

[1] Akkiraju, R., et al., "WSDL-S Web Services Semantics—WSDL-S," *W3C Member Submission*, www.w3.org/Submission/WSDL-S/, 2005.

[2] Battle, S., et al., "SWSL, Semantic Web Service Language," *W3C Member Submission*, www.w3.org/Submission/SWSF-SWSL/

[3] Becker, M. Y., and Sewell, P., "Cassandra: flexible trust management, applied to electronic health records," *In Proc. of 17th IEEE WSh on Computer Security Foundations,* pp. 139-154, 2004.

[4] Bonatti, P. & Olmedilla, D.. "Driving and Monitoring Provisional Trust Negotiation with Metapolicies," *In Proc. of the 6th IEEE Int'l WSh. on Policies For Dist. Sys. and Networks*, Washington, DC, 2005, pp. 14-23.

[5] Bradshaw, J.M. et al., "KAoS: Toward An Industrial-Strength Open Agent Architecture," *Software Agents,* pp. 375-418, 1997.

[6] Burstein, M., Bussler, C., Zaremba, M., Finin, T., Huhns, M.N., Paolucci, M., Sheth, A.P., Williams, S., "A Semantic Web Services Architecture," *IEEE Internet Computing*, 9(5), 2005, pp. 72-81.

[7] de Bruijn, J., et al., "WSMO Web Service Modeling Ontology (WSMO)," *W3C Member Submission*, www.w3.org/Submission/WSMO/, 2005.

[8] Ginsberg, A., "RIF Use Cases and Requirements," *W3C Working Draft*, http://www.w3.org/TR/rif-ucr/, 2006.

[9] Grønmo, R., Jaeger, M.C., Hoff, H., "Transformations between UML and OWL-S," *In Proc. of the 1st European Conf. on Model Driven Architecture: Foundations and Applications*, Nuremberg, Germany, 2005, pp. 269-283.

[10] Hirtle, D., et al., "Schema Specification of RuleML 0.91," http://www.ruleml.org/spec/, 2006

[11] Horrocks, I., et al., "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," *W3C Member Sub.*, http://www.w3.org/Submission/SWRL/, 2004.

[12] Kagal, L. et al., "Authorization and privacy for semantic web services," *In AAAI 2004 Spring Symposium on Semantic Web Services*, Stanford University, 2004.

[13] Lausen, H. et al., "Semantic web portals: state-of-the-art survey," *J. of Know. Man.*, 9(4), 2005 pp. 40-49.

[14] Martin, D. et al., "OWL-S: Semantic Markup for Web Services," *W3C Member Submission*, http://www.w3.org/Submission/OWL-S/

[15] Milanović, M. et al., "On Interchanging between OWL/SWRL and UML/OCL," *In Proc. of the 6th WSh. on OCL for (Meta-)Models in Multiple Application Domains (OCLApps)*, Genoa, Italy, 2006.

[16] Olmedilla, D. et al., "Trust negotiation for semantic web services," *In Proc. of the 1st Int'l WSh. on Semantic Web Services and Web Process Composition*, San Diego, CA, USA, 2004, pp. 81-95.

[17] Payne, T. & Lassila, O., "Guest Editors' Introduction: Semantic Web Services," *IEEE Intelligent Systems*, 19(4), 2004, pp. 14-15.

[18] R2ML specification, http://oxygen.informatik.tu-cottbus.de/R2ML/, 2006

[19] R2ML Translators, http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/15

[20] Sheth, A. et al., "Semantics to energize the full services spectrum," *Comm. of the ACM*, 49(7), 2006, pp. 55-61.

[21] Uszok, A. et al., "KAoS policy and domain services: toward a description-logic approach to policy representation, deconfliction, and enforcement," *In Proc. of the 4th IEEE I'l WSh. on Policies for Distributed Systems and Networks*, 2003, pp. 93-96.

[22] Wagner, G. et al., "A Usable Interchange Format for Rich Syntax Rules Integrating OCL, RuleML and SWRL," *In Proc. of WSh. Reasoning on the Web* (RoW2006), Edinburgh, UK, 2006.

[23] Wagner, G., "How to design a general rule markup language?," *In Proc. of the WSh. on XML Tech. fur das Semantic Web, 2002*, Berlin, Germany, 2002.