Web Rule Languages to Carry Policies

Nima Kaviani[1], Dragan Gašević[1], Marek Hatala[1], Gerd Wagner[2]
[1]Simon Fraser University Surrey, Canada
[2]Brandenburg University of Technology at Cottbus, Germany
{nkaviani, dgasevic, mhatala}@sfu.ca, wagnerg@tu-cottbus.de

Abstract

Recent efforts in the area of Web policy languages show concerns on how to better represent both context and rules of a domain to deal with large number of resources and users. Interaction between domains with different business rules is also another questionable issue in this same area. Web rule languages have been recently introduced as a means to facilitate interaction between parties with dissimilar policies and business rules. Efforts have been placed to further review the possibility of the proposed solutions and extend them to work with other Web technologies. In this paper, we introduce REWERSE Rule Markup Language (R2ML) as a Web rule language that can be employed to make concepts, policies, and elements of a domain digestible by another domain through the use of vocabularies, rules, and annotations. We also show how R2ML elements can model the concepts and elements of different policy languages and assist systems with diverse policies with their interactions.

1. Introduction

Integrating various technologies and services (e.g., mobile devices and Web) offers many opportunities for sharing resources in different contexts. For example, creating a new Web service does not mean that it will always be used in the same way and by the same users, as it has originally been designed for. On the contrary, a highly-dynamic nature of today's environments [15] requires the behavior of systems to dynamically change. In such systems one can discover and negotiate the use of new services. However, we need to define ways how one can understand what services are offered. It can be solved by using Semantic Web ontologies [1, 5]. Moreover, we also need to specify how, where, and by whom these services can be used. In fact, policies are used to address this problem, as a means to dynamically regulate the behavior of system components without changing the system's code and without requiring the consent or cooperation of the components being governed.

Currently, there are many different policy languages including Rei, KAoS, and PeerTrust; but there is no common agreement upon one universal policy language [26]. Each policy language has its own syntax and semantics that is usually grounded in a particular type of logic such as first order logic or description logic. This actually introduces a problem of mutual understanding and sharing of policies between different parties that use different policy languages. Nevertheless, it is very important to inspect policies in order to check whether there are some conflicting policies or understand what they are referring to (e.g., with the help of domain ontologies [27]). This can be very challenging having it in mind that different policy languages are based on different logics (e.g., description logic (DL) and declarative logic). On the other hand, Toninelli et al. [24] recognized that next-generation of policy languages should combine features of ontologies (i.e., description logic) and rule-based systems, which actually affects the development of languages such as Rei and KAoS. Finally, policies should also be in compliance and/or combined with business rules that can be defined by using various rule languages (e.g., F-Logic, Jess, Semantic Web Rule Language–SWRL, or Prolog) [2].

In this paper, we propose an approach to sharing policies by using Web rule languages [12]. This actually follows up two initiatives: the Rule Interchange Format (RIF) [6], an initiative for the standard for sharing rules on the Web, and Policy RuleML [20], an initiative for sharing policies by using various types of rules (e.g., derivation and production) of the RuleML language [8]. However, to the best of our knowledge, there has not been any practical attempt responding to either of these initiatives. Here, we propose using REWERSE Rule Markup Language (R2ML), a general purpose rule language and a future proposal for the RIF standard, to carry policies. To do so, we first motivate our research by analyzing two examples where there is a need to share policies that have originally been encoded in different languages. In Section 3, we describe three policy languages (KAoS, Rei, and PeerTrust), while in section 4 we briefly describe the most-known Web rule language efforts and their potentials to carry policies. Section 5, is the core section of the paper and it describes our approach to using R2ML for sharing policies in detail.

2. Motivation

As it has been mentioned in the previous section and based on the arguments in [24], there is a need to combine the features of description logic and the properties of the rule languages to define context-based policies with supports for conflict resolution, expansion, and classification on one hand, and rule enforcement on the other hand.

However, due to the variety of the policies available for protecting the resources and the approaches they take to codify these policies, i.e., either rules with ontologies as the backend knowledge bases [10, 18], or DL and corresponding reasoners on top [27], the process of information exchange becomes challenging and sometimes hard to achieve. This diversity of methods to describe rules and policies even sometimes forces the requestors to accept the language of the source entities for the sake of consistency and global compliance. To make it more precise let us review two of the scenarios addressed in the relevant literature.

In [24], the authors considered a scenario in which a certain traveling company has provided its travelers with wireless connectivity for their portable devices, e.g., PDAs and laptops, as well as some other services such as using public printers located in the airport. Alice as a traveler may wish to access the printer and print some of the available documents on her laptop. So, she sends the request to the service provider at the airport. The service provider checks Alice's credentials such as the boarding number and the name of the traveling agency. In case they match, Alice is given the right to access the printer to print the documents out. This scenario works fine as long as the service provider and the software agent on Alice's laptop speak in the same policy language.

The question that arises in this case is how Alice's web agent will communicate with the service provider in case the policy language of Alice's agent is different from that of the service provider. Recall that there is still no generally adopted agreement on defining and using Semantic Web services and Web policy languages. Thus, different Web entities may use dissimilar policies to protect their services. Let us assume that Alice's web agent is using the Rei policy language [11] and the printing service is defined based on WSMO [5]. Is Alice going to convert her policy language to F-Logic, which is the supported rule language in WSMO, and use the WSMO rule engine, or, is she going to expect the service provider to understand her policies?

Another example is based on [16] where the authors try to integrate their policy language called PeerTrust [18] with the description of a Semantic Web service. Among all the available semantic web service description languages, including OWL-S [13], WSDL-S [1], and WSMO, the authors choose WSMO as it allows arbitrary use of logical expressions in the description of the services and also uses F-Logic to describe the logical expressions used in the description of the services [18]. In contrast, WSDL-S and OWL-S are agnostic to employing rule and ontology languages. However, using F-Logic to define the concepts of PeerTrust in WSMO means that the whole concepts of PeerTrust need to be converted to a format suitable for an F-Logic-based inference engine. That is, one should totally forget about the engine that already exists for PeerTrust and develop a new F-Logic engine that can reason over the policies defined in PeerTrust.

Problems as such necessitate the development of a unified method of policy exchange that supports the conversion of different policy languages from one to another. Additionally, it seems that the viability of the future policy languages is tied to their capability in combining rule- and ontology-based languages (i.e., declarative and descriptive logic) [24]. Thus, the intermediary exchange language should have the required constructs and elements to support both rules and ontology concepts. Here we propose using a web rule (markup) language to carry the policies from one party to another one. Besides web rule languages, we also need two way transformations between the web rule language and policy languages, so that we can fully address the problem of diversity of policy languages.

3. Policy Languages

Policies in the domain of autonomous computing are guiding plans that restrict the behavior of autonomous agents in accessing the resources [24]. They also legitimatize the behavior of an agent by identifying its liberties and suppressions during the process of authorization. The main advantage in using policies is the possibility to dynamically change the behavior of the system by adjusting the policies without interfering with the internal code of the system [3]. A policy-aware system can be simply conducted to act based on the role of the requesting entities and or the context of its performance.

Policy-aware systems have been constantly evolved starting from group-based policy systems (e.g. operating systems) to role-based systems (e.g. Cassandra and RT), and recently to context-based systems. Although the earlier versions of policy-aware systems are still applicable to the static contexts with precise number of users, groups, and resources, they are not applicable to the Web with its enormous number of resources and users. Context-aware policy systems are targeting the problem of extending the number of to-be-protected resources and contexts as well as treating unknown or partly-known requesting entities[25].

KAoS [27] and Rei [11] are two of the most known policy systems that go beyond the traditional policy systems by giving special care to the context to which the policies are applied. They are both enriched semantically by using ontologies to define and describe the entities involved in the process of authorization and access control. PeerTrust [18] is another policy based system that operates in a lower level of abstraction, as compared to KAoS and Rei, and addresses access control problems through the use of trust negotiation. However, the similarities between PeerTrust, KAoS, and Rei in using ontologies to describe the policies and entities, as well as the semantic and logic that PeerTrust chooses to address rule enforcement and conflict resolution makes it an interesting language to be compared to KAoS and Rei. Aside from the conceptual differences in the level of access control and authorization, KAoS, Rei, and PeerTrust also differ in their syntax and also the types of the logic they are based on.

All the above languages have similar constructs to address *Permission, Prohibition, Obligation,* and *Dispensation,* but they add additional elements and building blocks to make the process of policy definition, harmonization, and enforcement more precise. [26] has already provided a detailed comparison of KAoS, Rei, and a traditional policy language called Ponder [4]. The comparison, however, is based on the older version of Rei, namely Rei 1.0, which was not as advanced as Rei 2.0 in defining and annotating the resources semantically. The comparison was also on the level of features and properties and not on the level of syntax and logical foundations that policy languages are built on. Here we give a quick comparison of Rei and KAoS, pointing out some of their properties that conform to the properties of PeerTrust, and briefly review the syntactical and logical differences they entail in their definitions. KAoS and Rei are important for our purpose because they are widely known in the level of context-aware policy languages with markup syntax. PeerTrust helps to show the possibility of applying the transformations to a language with a more traditional EBNF structure. The syntactical differences will be later argued when deliberating the transformation problems between policy and web rule languages.

KAoS is a policy language with the possibility of specification, management, conflict resolution, and enforcement of policies [27, 28]. The policies and domain objects have been represented as OWL ontologies which make the systems easily expandable and adaptable to different domains. Each KAoS policy rule is an instance of the Policy class (i.e., its *PosAuthorizationPolicy*, *NegAuthorizationPolicy*, *PosObligationPolicy*, and *NegObligationPolicy* subclasses), with properties for resources to be controlled, conditions, actors, triggering events and actions, site of enforcement, etc. Thanks to the features of OWL, policies can be defined to cover concepts like minimum and maximum cardinality for the entities as well as universal and existential quantifiers over the objects instantiated from the concepts and classes. However, the lack of mechanisms to define variables in OWL has made the developers use role-value-map technique to implement dynamic and runtime role/entity assignment. The arity of the predicates represented in KAoS is restricted to one or two corresponding to their definitions in OWL. By using and extending Stanford's Java Theorem Prover (JTP), KAoS enables static conflict resolution, intelligent lookup and dynamic policy refinement. KAoS has its enforcement engine, but it needs to be customized with regards to the domain it is going to be deployed in.

Rei is a rule-based approach to specify, analyze and reason over the policies in pervasive environments [10]. Although the first version of Rei was following a Prolog-like syntax with a Prolog engine as the reasoner, Rei 2.0 migrated to a new representation format, exploiting the RDF notations to define policies [22]. Thus, in the new version, Rei expressions are defined as triples compliant to the RDF format. Unlike KAoS, in which the knowledge about the domain and the policies are all defined in OWL, Rei only uses ontologies as knowledge bases to keep the information of the domain and although its syntax is in the form of RDF, the semantics follow the rule-based language conventions. Rei relies on a rich set of speech acts for the purpose of message passing and dynamic exchange of the rights between the entities. A main drawback in Rei is that there is no enforcement engine designed for it and the process of rule enforcement should be addressed outside the Rei engine. Moreover, because the Rei policy engine treats the inferences from OWL axioms as virtual fact base, there are no capabilities for ontological reasoning and consequently no chance for policy disclosure and conflict resolution as opposed to KAoS. However, the syntax used by Rei is much easier to grasp for the users with a basic understanding of rule languages. Rei is suitable for a lower level of security than KAoS, dealing with identification of entities and concepts.

PeerTrust is a trust negotiation engine with the possibility of dynamic exchange of certificates and establishment of trust without any third party being involved in the process of trust act [18]. Similar to Rei, PeerTrust also uses a Prolog-based engine to reason over the defined policies for the exchange of trust information but instead of dealing with contexts (as in KAoS) or the identities of the entities (as in Rei), it goes further down in the level of security and defines policies over attributes of the resources and the entities. PeerTrust uses its own EBNF syntax for the representation of policies with possibility of defining n-ary predicates in the rules, but the policies can be imported to the PeerTrust engine in the form of RDF metadata as well. PeerTrust has been deployed and used in the ELENA distributed e-learning environment.

While KAoS is based on description logic, Rei and PeerTrust follow the conventions of declarative logic programs. This makes a lot of difference in the way they refer to existence or nonexistence of objects and the relations between classes and elements of the classes. So, a mapping between the languages goes beyond a conceptual matching in the level of policies and has to delineate the mappings in the level of logic as well (see Section 5.2 for details).

Figure 1 summarizes features of the three languages mentioned above. It compares KAoS, Rei, and PeerTrust in terms of the format they use to describe their rules, the possibility of expanding the concepts of the languages, and also the way they store the domain information and reason over it.

4. Web Rule Languages for Policies

In Section 2, we have shown the existence of a need to define a medium for transforming the rules, and specifically policies, among different resources. We have also discussed the necessity of a support for both ontological definition and rule based representation of the policies for the purpose of achieving conflict resolution, harmonization, and enforcement.

Decreased level of Abstraction $\longrightarrow$

Properties		**KAoS**	**Rei**	**PeerTrust**
	Policy Representation	OWL	Rei (RDF format)	PeerTrust
	Expandability	High	High	Low
	Knowledge Base	OWL	RDF/OWL	Prolog
	Reasoning Support	JTP	Prolog Engine	Prolog Engine
	Enforcement Engine	Extending KAoS engine	No engine	Domain specific (for ELENA)

Figure 1. Comparison of the features in KAoS, Rei, and PeerTrust

We believe that Semantic Web Rule languages are the solution to the problem. To be precise, let us start with reviewing some of the proposed Semantic Web Rule Languages and then we will review the properties they offer to facilitate the exchange of policies.

Rule Interchange Format (RIF) [6] is one of the most important initiatives in this area. It defines a set of use cases and requirements for sharing rules on the Web. However, as it is desired in our case, the purpose of RIF is to serve as an intermediary language between various languages and not as a formally defined semantic foundation for the purpose of reasoning on the Web. Among all the use cases defined for RIF, special care has been given to policies by relating three of the ten introduced use cases to the issues specific to policies. These use cases are: *Collaborative Policy Development for Dynamic Spectrum Access*, *Access to Business Rules of Supply Chain Partners*, and *Managing Inter-Organizational Business Policies and Practices*. Still there has been no concrete example on how to use RIF for this purpose.

Here we point out some of the main efforts in the area of Semantic Web Rules.

Semantic Web Rule Language (SWRL) is a rule language based on the W3C Web ontology language OWL [9]. A SWRL rule is also in the form of an implication and is considered to be another type of an axiom on top of the other OWL axiom types. This means that SWRL rules are usually used for defining derivation and integrity rules. 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 interchange language, but its purpose is to define an additional logic layer over the present ontology languages (i.e., OWL). As such it can not represent many linguistic constructs of other rule languages (e.g., F-Logic, Rei, or OCL).

RuleML is a markup language for publishing and sharing rule bases on the World Wide Web [8].

RuleML builds a hierarchy of rule sublanguages upon XML, RDF, XSLT, and OWL and currently supports rules in the form of derivation (e.g., SWRL, FOL) and production (e.g., Jess). It is based on Datalog and defined as an implication between antecedent and consequent. The antecedent part of the rule is evaluated as true whenever the consequent of the rule holds. However, an important constraint of RuleML is that it can not fully represent all the constructs of various languages such as OCL or SWRL.

Built upon the concepts of RuleML, the Policy RuleML Technical Group [20] has been formed as a committee to outline the use of RuleML as a semantic interoperation vehicle for heterogeneous policy languages, standards, protocols, and mechanisms; both currently existing and those developed in the near future. The goal was stated as encoding, translating and integrating rules between disparate policy systems. They have addressed their long-term plan as incorporation of deontic expressive features such as logics to capture rights, obligations, and empowerment as aspects of policy rules. The deontic rules they consider to model policy languages are enlisted as *Permission and Prohibition*, *Duty Assignment*, and *Empowerment*. Nevertheless, to the best of our knowledge, there is still no real work done by the Policy RuleML technical committee in this area. In the next section, we explain our efforts to address these goals by using R2ML [12].

5. R2ML for Representing Policies

This section presents our approach to sharing policies by using Web Rule Languages, that is based on the REWERSE Rule Markup Language (R2ML). We first describe R2ML features and how they can be used to represent various types of policies. We then further clarify our idea by demonstrating how we mapped some of the policy languages such as KAoS and Rei to R2ML, and thus address the problem of sharing policies between diverse policy languages. As it was already discussed the problem of mapping KAoS to Rei and back is not only a matter of policy term matching, but it is also a transformation from description logic to declarative logic, which KAoS and Rei are respectively based on. The main reason we chose KAoS and Rei in our first attempt for providing a mapping, beside their reputation in the area, was their XML-like syntax with easier practical implementation of transformation to triples of type subject-predicate-object.

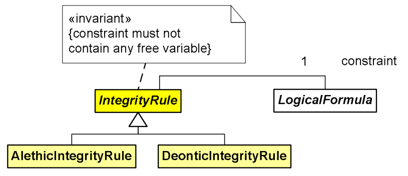
5.1. R2ML: An overview

R2ML is a general rule interchange language that tries to address all RIF requirements. The abstract syntax of R2ML language is defined with a metamodel by using the 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 current version of R2ML is 0.4 [21]. The full description of R2ML in the form of UML class diagrams is given in [21], while more details about the language can be found in [29]. The language also has an XML concrete syntax defined by an XML schema, while there are a number of transformations implemented between R2ML and rule based languages (e.g., OCL, SWRL, Jess, and F-Logic).

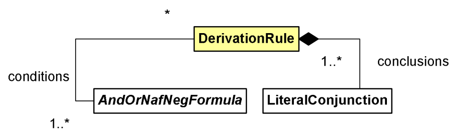
Vocabulary. R2ML provides a vocabulary that enables users to define their own world in the form of objects and elements available in the domain of discourse. The vocabulary can be defined as a combination of *Basic Content Vocabulary*, *Relational Content Vocabulary*, and *Functional Content Vocabulary*. Basic Content Vocabulary allows the user to specify the basic elements of the domain such as individual objects and data values, classes and data types, and object and data variables. Relational Content Vocabulary helps to associate different objects from different classes through defining n-ary association and association classes. Finally, Functional Content Vocabulary assists with defining functors that correspond to the standard logic of functions. The functions can be data operations to manipulate data values, they can be object operation functions that define object-value operations, or they can be role functions which correspond to functional association (binary association) of the class elements. In [14], authors showed how the basic constructs and elements of the OWL language can be transferred and modeled by R2ML atoms and elements. For example *sameAs* in OWL is equivalent to an *EqualityAtom* in R2ML and *oneOf* in OWL carries the same meaning as *Disjunction* of a set of atoms in R2ML. This means any language with its concepts defined based on OWL (including KAoS and Rei) can be modeled with R2ML constructs elaborately.

Rules. Having the objects and concepts of a domain defined, R2ML makes the definition and harmonization of rules over these concepts possible through the use of four different types of rules: *Integrity Rules*, *Derivation Rules*, *Reaction Rules*, and *Production Rules*. Since in this paper we are limited in space, we only review the first two rules and more information about the other rules and constructs of the language can be found in [29].

R2ML integrity rules, 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 (see Figure 2a). R2ML supports two kinds of integrity rules: the *alethic* and the *deontic* ones. The alethic integrity rule can be expressed by a phrase, such as *"it is necessarily the case that"* and the deontic one can be expressed by phrases, such as *"it is obligatory that"* or *"it should be the case that"*. A LogicalStatement is a LogicalFormula that has no free variables, i.e., all the variables from this formula are quantified. In terms of policy languages, integrity rules can be considered as constraints that must hold consistently especially in the level of rule enforcement, e.g. *"it is necessary to give a higher priority to the commands of the administrator than to the commands of the regular users on a system."*



a)



b)

Figure 2. The R2ML definition of Integrity (a) and Derivation Rules (b)

A R2ML derivation rule has conditions and a conclusion (see Figure 2b) 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* 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). In the context of policies, we consider each deontic policy rule as a single derivation rule with the constraints making the conditions of the derivation rule and the policy decision forming the conclusion of the rule, e.g. *"If the user is from Simon Fraser University with a valid student ID then give her the permission to enter the area of the university."* It may sound more expressive to define deontic policy rules with deontic integrity rules in R2ML. However, our attempts in doing so showed that deontic rules in the context of policies carry a different meaning from their interpretation in R2ML. In R2ML, a deontic integrity rule represents a constraint that should be satisfied or must hold with a concrete proof for its truthfulness, though a doentic policy demonstrates concerns over performing a duty or obligation as a result of satisfying a series of related conditions [19, 23].

Atoms are the basic logical constituents of a rule which are compatible with the concepts of OWL, RuleML, and SWRL. Atoms connect objects to values, classes to instances, and objects to objects, put restrictions on the objects and data values, and so on. Here we briefly represent some of the atoms that are relevant to our purpose of representing policy languages. *ReferencePropertyAtoms* associate object terms as subjects with other terms (objects or data values) as objects. A *ReferencePropertyAtom* in R2ML corresponds to an OWL (and similarly a KAoS) object property, or to the OWL concept of value for an individual-valued property. *ObjectDescriptionAtoms* are another class of useful atoms for our purpose. They refer to a class as a base type and to zero or more classes as categories, and consist of a number of property/term pairs (i.e., attribute data term pairs and reference property object term pairs). Any instance of such atom refers to one particular object that is referenced by an *objectID*, if it is not anonymous. This atom corresponds to the instantiation of an object from a class in OWL, which matches a deontic object, with all its properties instantiated, in either Rei or KAoS.

5.2 The Logic of Transformation

Providing transformations from Rei and KAoS to R2ML and then from R2ML to KAoS or Rei, as we mentioned before, is not just a straightforward keyword matching using lookup tables. KAoS models the world by specifying the elements and the objects in description logic while Rei assembles its world with declarative logic building blocks. So, the problem of transformation expands to the problem of bridging the declarative logic world to its descriptive logic counterpart. It is important for R2ML because we need R2ML to serve as a conductor between the two worlds. R2ML has been designed having the properties of both open world (i.e. descriptive logic) and close world (i.e. declarative logic) in mind. Knowing the logic of transformation from declarative logic to descriptive logic and back would help in providing more meaningful transformations with less information loss.

[7] gives an elaborate method of mapping the basic elements of description logic to declarative logic. OWL as a subset of RDFS corresponds to a fragment of classical FOL. It is shown in [7] that OWL elements are convertible to definite Horn FOL elements which in turn are convertible to definite Datalog Logic Programs as a restricted model of Logic Programs (LPs). For example, classes and class expressions are equivalent

to FOL formulae with one free variable, and properties (and property expressions when supported by description logic) are equivalent to FOL formulae with two free variables. Classes and property inclusion axioms are also considered as FOL sentences consisting of an implication between two formulae with the free variables universally quantified at the outer level.

Figure 3 shows a selection of OWL constructs with their corresponding description logic syntax and FOL expressions. Details of the definitions can be found in [7].

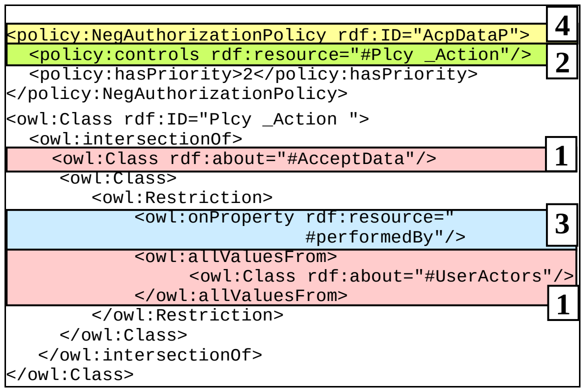
OWL Constructor	DL Syntax	FOL Expressions
subClassOf	$C \subseteq D$	$D \leftarrow C$
transitiveProperty	$P^{+} \subseteq P$	$\forall x,y,z(P(x,y) \wedge (P(y,z)) \rightarrow P(x,z)$
inverseOf	$P \equiv Q^{-}$	$\forall x, y P(x,y) \Leftrightarrow Q(y,x)$
intersectionOf	$C_1 \cap ... \cap C_n$	$C_1(x) \wedge ... \wedge C_n(x)$
unionOf	$C_1 \cup ... \cup C_n$	$C_1(x) \vee ... \vee C_2(x)$
complementOf	$\neg C$	$\neg C(x)$
oneOf	$\{a_1,...,a_n\}$	$x = a_1 \vee ... \vee x = a_n$
hasClass	$\exists P.C$	$\exists y(P(x,y) \wedge C(y))$
toClass	$\forall P.C$	$\forall y(P(x,y) \rightarrow C(y))$

Figure 3. Some of the OWL constructors and the equivalent description logic and FOL expressions

For the purpose of transforming policies from KAoS to Rei, we need R2ML rules that can precisely transfer the deontic meaning of the policy rules in the same way we explained in the previous section. These R2ML rules also have to either implicitly or explicitly demonstrate the possibility of reasoning over the content which is transferred. Furthermore, as it has been also argued in [7], class and property inclusions are better to be declared in the form of implications. Considering all the points above, we finally chose derivation rules as the most suitable rules for this purpose. Derivation rules precisely support implication and show derivation of new facts upon reasoning on a priori facts.

Let us further clarify the idea by reviewing an intuitive example of the policy languages. Consider that we need to define a policy to *"prohibit our system from using data that is accepted by the members of a group called UserActors"*. Figure 4 shows an excerpt of the policy rule in KAoS to define this policy.

The highlighted parts in the policy of Figure 4 show the main elements that carry the intended meaning of the policy. During the process of transformation these elements should be captured and converted to the appropriate R2ML elements



```
<policy:NegAuthorizationPolicy rdf:ID="AcpDataP">
  <policy:controls rdf:resource="#Plcy _Action"/>
  <policy:hasPriority>2</policy:hasPriority>
</policy:NegAuthorizationPolicy>
<owl:Class rdf:ID="Plcy _Action ">
  <owl:intersectionOf>
    <owl:Class rdf:about="#AcceptData"/>
     <owl:Class>
        <owl:Restriction>
            <owl:onProperty rdf:resource="
                            #performedBy"/>
            <owl:allValuesFrom>
                 <owl:Class rdf:about="#UserActors"/>
            </owl:allValuesFrom>
        </owl:Restriction>
     </owl:Class>
   </owl:intersectionOf>
</owl:Class>
```

Figure 4. An excerpt of a KAoS policy rule

R2ML in our transformation plays the same role as definite Horn FOL plays in description logic programs in [7], i.e. an intermediary world to capture similarities between logic programs and description logic. In order to make the idea easier to grasp, we start with a conversion of the policy in Figure 4 to definite Horn FOL according to the mappings from Figure 3 and then further develop it to the identical R2ML representation. Figure 5 shows the result of applying the conversions to the code above.

$$
\begin{aligned}
&NegAuthorizationPolicy(x) \leftarrow \\
&\qquad \exists controls(x,y) \wedge Plcy\_Action(y) \wedge \\
&\qquad hasPriority(x,2) \\
&AcceptData(x) \leftarrow Plcy\_Action(x) \\
&UserActors(y) \leftarrow Plcy\_Action(x) \wedge \\
&\qquad performedBy(x,y)
\end{aligned}
$$

Figure 5. The KAoS policy from Figure 4 described in the definite Horn FOL

Similar to the conversion in Figure 5, our R2ML transformation should syntactically demonstrate the possibilities to evaluate *AcceptData(x)*, *UserActors(y)*, and eventually *NegAuthorizationPolicy(x)* as *TRUE* knowing that *Plcy_Action(x)* and *performedBy(y, x)* are true for the instantiated variables *x* and *y*. It also has to show that *NegAuthorizationPolicy(x)* is a logical consequence of *AcceptData(x)* and *UserActors(y)*. To model the desired rule we place the values for *Plcy_Action(x)* and *performedBy(y, x)* in the head, and *AcceptData(x)* and *UserActors(y)* in the tail of the R2ML transformation rule. Note that the policy rule is fired when action *x* as a *PolicyAction* is performed by actor *y* who belongs to the group of *UserActors*, so the values for *x* and *y* become known. *NegAuthorizationPolicy(x)* is the ultimate result of the derivation rule, which we keep as a conclusion in the head of the derivation rule, converted to *Prohibition*, to show the final result of authorization. Finally, the result of a mapping from KAoS to R2ML based on this argument can be formulated similar to Figure 6.

```
<r2ml:DerivationRule>
  <r2ml:conditions>                                                    1
    <r2ml:ReferencePropertyAtom
      r2ml:propertyID="#instanceOf"
      <r2ml:subject>
        <r2ml:ObjectVariable r2ml:name="x"/>
      </r2ml:subject>
      <r2ml:object>
        <r2ml:ObjectName r2ml:objectID="#AcceptData"/>
      </r2ml:object>
    </r2ml:ReferencePropertyAtom>
    <r2ml:ReferencePropertyAtom>
          r2ml:propertyID="#instanceOf">
      <r2ml:subject>
        <r2ml:ObjectVariable r2ml:name="y"/>
      </r2ml:subject>
      <r2ml:object>
        <r2ml:ObjectName r2ml:objectID="#UserActors"/>
      </r2ml:object>
    </r2ml:ReferencePropertyAtom>
  </r2ml:conditions>
  <r2ml:conclusion>
    <r2ml:ObjectDescriptionAtom                                        4
          r2ml:classID="Prohibition">
      <r2ml:subject>
           <r2ml:ObjectVariable r2ml:name="AcpDataP"/>
      </r2ml:subject>

      <r2ml:ObjectSlot                                                 2
            r2ml:referencePropertyID="controls"/>
       <r2ml:ObjectVariable r2ml:name=”x”
                      r2ml:classID=”#Plcy_Action”>
      </r2ml:ObjectSlot>

      <r2ml:ObjectSlot                                                 3
            r2ml:referencePropertyID="performedBy">
        <r2ml:ObjectVariable r2ml:name="y"/>
      </r2ml:ObjectSlot>
    </r2ml:ObjectDescriptionAtom>
  </r2ml:conclusion>
</r2ml:DerivationRule>
```

Figure 6. R2ML representation of the KAoS policy rule from Figure 4

The parts of Figure 4 and 6 numbered similarly are the conceptually equivalent pieces in the referred languages, namely KAoS and R2ML. As we have already mentioned and it can also be seen in the transformations, R2ML *ReferencePropertyAtom* is used to model property values in OWL and *ObjectDescriptionAtom* is used to model the instantiation of the objects in OWL.

A *ReferencePropertyAtom* connects the objects of a class to the objects from other classes with properties as connectors. In this sense a *ReferencePropertyAtom* neatly models a triple similar to their representation in RDF. Now that we have the KAoS policy simulated in the form of triples in R2ML, mapping the result to the RDF format of Rei is easy to achieve. Figure 7 shows the corresponding Rei model for the above R2ML code snippet (Figure 6).

Although the policy that we have reviewed here has only few constraints, the other constraints for the other properties of a policy rule in KAoS (such as triggering constraints, preconditions, and effects) follow the same procedure and are eventually converted to constraints which are placed in the *condition* part of an R2ML rule.

The transformation of the policies from Rei to R2ML is much simpler because the R2ML rule representation that we have chosen is closer to the model of expressing the rules in Rei (the same way that def Horn FOL is closer to LP). Yet grouping of the R2ML atoms as classes in order to keep the derivation rules consistent with the format of OWL and KAoS is important. To achieve that, we group *ReferencePropertyAtoms* with a common *subject* element in the form of an instantiated object from a class. The restrictions placed over the subject part of a *ReferencePropertyAtom* can then be shown as restrictions in the OWL class.

We shorten the discussion at this point, but note that we have implemented transformations from Rei and KAoS to R2ML and also from R2ML to Rei and KAoS using XSLT which fully follow mapping rules discussed in this section. All these transformations can be found in [30].

```
<entity:Variable rdf:ID=”x”/>
<entity:Variable rdf:ID=”y”/>
<entity:Variable rdf:ID=”negAuth”/>
<constraint:SimpleConstraint rdf:ID="constraint1">                     1
   <constraint:subject rdf:resource="#x"/>
   <constraint:predicate rdf:resource="&rdfs;type"/>
   <constraint:object rdf:resource="#AcceptData"/>
</constraint:SimpleConstraint>

<constraint:SimpleConstraint rdf:ID="constraint2">
   <constraint:subject rdf:resource="#y"/>
   <constraint:predicate rdf:resource="&rdfs;type"/>
   <constraint:object rdf:resource="#UserActors"/>
</constraint:SimpleConstraint>

<constraint:And rdf:ID="conditions">
   <constraint:first rdf:resource="#constraint1"/>
   <constraint:second rdf:resource="#constraint2"/>
</constraint:And>

<constraint:SimpleConstraint rdf:ID="actor_value">                     3
   <constraint:subject rdf:resource="#y"/>
   <constraint:predicate rdf:resourc="#performedBy"/>
   <constraint:object rdf:resource="#x"/>
</constraint:SimpleConstraint>

<constraint:SimpleConstraint rdf:ID="actio_value">                     2
   <constraint:subject rdf:resource="#x"/>
   <constraint:predicate rdf:resource="controls"/>
   <constraint:object rdf:resource="#Plcy_Action"/>
</constraint:SimpleConstraint>

<deontic:Prohibition rdf:ID=”AcpDataP”>                                4
    <deontic:actor rdf:resource=”#actor_value”/>
    <deontic:action rdf:resource=”#action_value”/>
    <deontic:constraint rdf:resource=”#conditions”/>
</deontic:Prohibition>
```

Figure 7. A Rei equivalent of the R2ML rule from Figure 6

6. Discussion and Conclusion

Despite a wide recognition of necessity to integrate business rules and specially policy rules [2, 6, 20], our approach to representing the policy languages seems to be the first practical attempt in this area. We have shown the possibility of providing interoperability between two well-known policy languages, namely, KAoS and Rei, through the use of R2ML as a mediator language. We have also discussed how the concepts of description logic can be converted to declarative logic and back. PeerTrust, due to its similarities to Rei in following the conventions of declarative logic, can equally be transformed to R2ML and then from R2ML to any other language that has a transformation from R2ML, such as KAoS or Rei.

A question that arises in using Web rule languages to share policies is whether the currently-defined rules (especially derivation rules) are sufficient for the purpose of transformation, or whether we need deontic rules (i.e., a sub-type of integrity rules in terms of R2ML) such as *Permission* and *Prohibition* to be supported and defined by Web rules. Based on our experiments with both policy rules (e.g., including permission, prohibition, obligation, and dispensation) and web rules (e.g., derivation and integrity rules), the current Web rules (i.e., R2ML) are expressive enough to carry the intended meaning of policy rules. Having in mind that Web rules are currently considered only as mediums and not as rules for the purpose of reasoning, the explicit discrimination between policy rules of different types does not seem to be a must. Additionally, Web rules can be annotated with respect to the ontologies that hold the conceptual information about policy languages, policy domains, and policy rule elements. This in turn highly improves the readability and machine understandability of the derived rules and exempts us of defining further components for Web rules.

Given the appropriate transformation between several policy languages and R2ML, the problems mentioned in Section 2 can be proven to be solvable. Alice, as a traveler, should no more worry about the policy language used by the printing service provider. The broker agents available either on Alice's system or the service provider's side can determine the policy language used by the other party, once the request has been sent or received. Through conversion of the policies using the available transformations, both of the parties (i.e. the service provider and the requester) can decide whether the constraints placed on the transaction by each of the two sides are acceptable with regards to the policies of the other side, and if so, they can start a healthy and authorized communication.

R2ML as a Web rule language provides transformations from/to many other business rule and Semantic Web languages. It has transformations from/to F-Logic, OCL, SWRL, UML, etc. In the case of our second example in Section 2, existence of a conversion from F-Loigc to PeerTrust through R2ML omits the need for developing a separate reasoning engine to work over PeerTrust rules defined in F-Logic. One can simply transform the F-Logic rules to PeerTrust rules and then take advantage of the existing rule engine for PeerTrust. Any other user with any other policy language can still communicate to the service. Moreover, existence of vocabularies for R2ML facilitates defining the concepts that one user or system may want to share with the other parties with which it interacts.

Beside all the abovementioned benefits, the presence of some flaws and problems in the given approach is inevitable. The level of abstraction in protecting the resources of a domain varies in different policy languages, as we have also shown in Figure 1, which is a big problem in providing accurate mappings. This means, although the policy for a source language can be transformed to R2ML, the transformed concepts might not be available in the desired target language. For example, in KAoS there is support for keeping the history of actions and actors of a domain, while Rei does not explicitly keep the track of preformed actions and their corresponding actors. Therefore, during the process of transformation these concepts should either be addressed by expanding Rei or be ignored. In our solution, we have taken the first approach, trying to add these concepts to Rei by defining ontologies that cover those concepts. However, for languages with low expandability support it might be impossible.

Furthermore, in Rei there is a rich set of Speech Acts to manage remote policy control which are missed in KAoS. In our transformations from Rei to KAoS, although the speech act elements are converted from Rei to the R2ML elements, they (and also their related constraints) are ignored during the conversion from R2ML to KAoS.

Another problem is caused by the transformation from declarative logic (e.g. Rei policy rules) to descriptive logic (e.g. KAoS policy rules) or vice versa. Ongoing efforts are being undertaken to make the transformation from descriptive logic to declarative logic promising, however, it seems that more work in this area has to be done to solve all the problems in representing the concepts in these two domains.

Considering all the drawbacks and advantages, our long term-goal is to make use of these transformations in a practical test bed (e.g., in the context of semantic Web services) and among several parties with different policy languages, showing the real opportunities to exploit the transformations in real scenarios. By expanding the transformations, we hope to target eventually the goal of globalizing system interaction on the Web.

7. Acknowledgment

The authors would like to thank Andrzej Uszok and Lalana Kagal for their valuable comments and helps on the concepts mentioned in this paper. The research of Simon Fraser University is supported by Canada's NSERC-funded LORNET Research Network, while the research of Brandenburg University of Technology at Cottbus is supported by the EU IST-funded REWERSE Network of Excellence.

8. References

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

[2] 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 Nets*, Washington, DC, 2005, pp. 14-23.

[3] Bradshaw, J. M, Jung, H., Kulkarni, S., Taysom, W. "Dimension of adjustable autonomy and mixed-initiative interaction". In Klusch, G. Weiss, and M. Rovatsos (Ed.), Computational Autonomy, Springer, Germany, 2004.

[4] Damianou, N., Dulay, N., Lupu, E., and Sloman, M. "The ponder policy specification language," *In Proc. of the Workshop of Policies for Dist. Sys. and Nets.*, Bristol UK, 2001, pp. 18-38 .

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

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

[7] Grosof, B. N., Horrocks, I., Volz, R., Decker, S. "Description Logic Programs: Combining Logic Programs with Description Logic". *In Proc. of the 12th Int'l. Conf. on the World Wide Web*, Budapest, Hungary, 2003, pp. 48–57.

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

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

[10] Kagal, L., Finin, T., and Joshi, A. "A policy language for a pervasive computing environment," *In IEEE 4th Int'l. Workshop of Policies for Dist. Sys. & Nets*, 2003, pp. 6-74.

[11] Kagal, L., "A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments", PhD Thesis, University of Maryland, 2004.

[12] Kaviani, N., Gašević, D., Hatala, M., Clement, D., Wagner, G., "Towards Unifying Rules and Policies for Semantic Web Services," *In Proc. of the 3rd Annual LORNET Conf. on Intelligent, Interactive, Learning Object Repositories Network*, Montreal, QC, Canada, 2006

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

[14] Milanović, M., Gašević, D., Guirca, A., Wagner, G., Devedžić, V., "On Interchanging between OWL/SWRL and UML/OCL," *6th Workshop on OCL for (Meta-)Models in Multiple Application Domains (OCLApps)*, Genoa, Italy, 2006, pp. 81-95.

[15] Müller, G. "Guest Editor's Introduction: Privacy and security in highly dynamic systems," *Communications of the ACM*, vol. 49, no. 9, 2006, 28-31.

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

[17] Nejdl, W., Olmedilla, D., and Winslett, M., Zhang, C. C. "Ontology-based policy specification and management" *In 2nd European Semantic Web Conference (ESWC), LNCS 3532*, Heraklion, Crete, Greece, 2005, pp. 290-302

[18] Nejdl, W., Olmedilla, D., and Winslett, M. "PeerTrust: automated trust negotiation for peers on the semantic web". *Technical Report*, 2003.

[19] Ortalo, R. "Using Deontic Logic for Security Policy Specification", 1996 http://citeseer.ist.psu.edu/ortalo96using.html

[20] The Policy RuleML Technical Group. "The RuleML Initiative", March04 http://policy.ruleml.org

[21] R2ML Specification, http://oxygen.informatik.tu-cottbus.de/R2ML/, 2006

[22] Rei Ontology Specification, Version 2.0, http://www.cs.umbc.edu/~lkagal1/rei/

[23] Sergot, M., "Deontic Logic in Policy Specification", *In Proc. of the Policy Workshop*, Bristol, UK, 1999.

[24] Toninelli A., Bradshaw J., Kagal L., Montanari, R. "Rule-based and Ontology-based Policies: Toward a Hybrid Approach to Control Agents in Pervasive Environments" *In Proc. of the Semantic Web and Policy Workshop*, Galway, Ireland, 2005.

[25] Toninelli A., Montanari, R, Kagal L., Lassila, O. "A Semantic Context-Aware Access Control Framework for Secure Collaborations in Pervasive Computing Environments" *In Proc. of the 5th Int'l Semantic Web Conf.*, Athens, Georgia, US, 2006.

[26] Tonti, G., Bradshaw, J., Jeffers, R., Montanari, R., Suri, N. and Uszok, A. "Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder". *In Proc. of the 2nd Int'l Semantic Web Conf., LNCS 2870*, 2003, pp. 419-437

[27] 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 Int'l Workshop on Policies for Distributed Systems and Networks*, 2003, pp. 93-96.

[28] Uszok, A., Bradshaw, J., Jeffers, R. (2004). "KAoS: A Policy and Domain Services Framework for Grid Computing and Semantic Web Services". *In Proc. of the Second In'l Conf. on Trust Management (iTrust 2004)*, Springer.

[29] 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.

[30] XSLT transformations: http://cgi.sfu.ca/~nkaviani/cgi-bin/index.php?linkLocation=6projects