I1-D12

Consolidation of Language Extensions – Preliminary Draft Version

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Cottbus/I1-D12/D/PU/b1
Responsible editors:	Sergey Lukichev
Reviewers:	Dragan Gasevic
Contributing participants:	Cottbus
Contributing workpackages:	I1
Contractual date of deliverable:	April, 2008
Actual submission date:	18 January 2008

Abstract

The report consists of three parts: i). Using R2ML as an interchange format and XML syntax for the policy languages like KAoS, Rei, PeerTrust, including the Protune policy language, developed by the REWERSE Working Group I2; ii). R2ML Compatibility with the Rule Interchange Format by W3C, where we discuss the current status on I1 activities on standardization; iii). R2ML Extensions for Accommodating Production Rules, where we explain, which features of some production rule languages are missing in R2ML and discuss whether they have to be added to R2ML and how.

Keyword List

Rule Interchange, Policies, Policies Interchange, R2ML, Rule Markup Language, RIF, PRR, Production Rules, Semantic Web

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

Consolidation of Language Extensions – Preliminary Draft Version

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

[1] Institute of Informatics, Brandenburg University of Technology
Email: {G.Wagner, Giurca, Lukichev }@tu-cottbus.de

18 January 2008

Abstract
The report consists of three parts: i). Using R2ML as an interchange format and XML syntax for the policy languages like KAoS, Rei, PeerTrust, including the Protune policy language, developed by the REWERSE Working Group I2; ii). R2ML Compatibility with the Rule Interchange Format by W3C, where we discuss the current status on I1 activities on standardization; iii). R2ML Extensions for Accommodating Production Rules, where we explain, which features of some production rule languages are missing in R2ML and discuss whether they have to be added to R2ML and how.

Keyword List
Rule Interchange, Policies, Policies Interchange, R2ML, Rule Markup Language, RIF, PRR, Production Rules, Semantic Web

Contents

Chapter 1

Integration of R2ML with Policy Languages

Web rule languages with the ability to cover various types of rules have been recently emerged to make interactions between web resources and broker agents possible. The chance of describing resources and users of a domain through the use of vocabularies is another feature of Web rule languages. Combination of these two properties makes Web rule languages an appropriate medium to make a hybrid model of representing both contexts and rules of a policy-aware system, such as a web service. In this report, we describe how REWERSE Rule Markup Language (R2ML) ([WGL^{+}06c], [WGL05]), developed by the Working Group I1[1] can be employed to bridge between different policy languages using its rich set of rules, vocabulary, and built-in constructs. We show how the concepts of the KAoS, Rei and Protune policy languages can be transformed to R2ML and then from R2ML to the other policy languages. Following these mappings, we have implemented transformers, which enable us not only to share policies between KAoS and Rei, but also to transform policies onto other rule languages (e.g., F-Logic) for which transformations from/to R2ML are already developed.

The research on policies interchange using R2ML, described in this report, actually follows up two initiatives: the Rule Interchange Format (RIF) [rif], an initiative for the standard for sharing rules on the Web, and Policy RuleML [Pol], an initiative for sharing policies by using various types of rules (e.g., derivation and production) of the RuleML language [BTW01]. However, to the best of our knowledge, there has not been any practical attempt responding to either of these initiatives. In Section 1.2, we describe four policy languages (KAoS, Rei, PeerTrust and Protune), while in section 1.3 we briefly describe the most-known Web rule language efforts and their potentials to carry policies. Section 1.5, is the core section of the report and it describes our approach to using R2ML for sharing policies in detail.

1.1 Motivation

As it has been mentioned in the previous section and based on the arguments in [TBKM05], there is a need to combine the features of description logic and the properties of the rule

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

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 [KFJ03], [NOW04], or DL and corresponding reasoners on top, 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 [TBKM05], 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 [Kag04] and the printing service is defined based on WSMO [wsm]. 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 [OLPL05] where the authors try to integrate their policy language called PeerTrust [NOW04] with the description of a Semantic Web service. Among all the available semantic web service description languages, including OWL-S [owl], WSDLS [wsd], 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 [NOW04]. 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) [KMLT06]. 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.

1.2 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 [BFJ^{+}04]. 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.

Protune (PROvisional TrUst Negotiation) is the policy language and metalanguage, developed in REWERSE. Trust negotiation in Protune is directly inspired by PAPL [BS00] and PeerTrust [GNO^{+}04], that build on ideas introduced in [WCJS97].

1.2.1 KAoS

KAoS is a policy language with the possibility of specification, management, conflict resolution, and enforcement of policies [UBJ+03]. 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.

1.2.2 Rei

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 [Kag06]. 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.

1.2.3 Protune

The **Protune** language can specify access control policies, privacy policies, reputation-based policies, provisional policies, and a class of business rules. The language is declarative, still it can describe actions which modify the current state. In summary, each party (client and server) makes decisions based on a set of rules that entail decision atoms such as `allow(X)`, based on conditions over currently available credentials and declarations (sent by the other party) and a time-dependent state, covering the negotiation state, user profiles, etc. (see [BS00] for further details). On the server, `X` is typically a service; on the client `X` may denote credential release, declaration release and actions execution. The rule language is based on normal logic program

Table 1.1: Comparison of the features in KAoS, Rei, PeerTrust and Protune

	KAoS	**Rei**	**PeerTrust**	**Protune**
Policy Representation	OWL	Rei (RDF format)	PeerTrust	Protune
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	

rules

$$A \longleftarrow L_1, ..., L_n$$

where A is a standard logical atom (called the head of the rule) and $L_1, ..., L_n$ (the body of the rule) are literals, that is, L_i equals either B_i or $\neg B_i$, for some logical atom B_i.

A policy is a set of rules shaped like above, such that negation is applied neither to provisional predicates (defined below), nor to any predicate occurring in a rule head. This restriction ensures that policies are monotonic, that is, as more credentials are released and more actions executed, the set of permissions does not decrease. Moreover, the restriction on negation makes policies stratified programs; therefore negation as failure has a clear, PTIME computable semantics that can be equivalently formulated as the perfect model semantics, the well-founded semantics or the stable model semantics.

1.2.4 PeerTrust

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 [NOW04]. 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 nary 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.

Table 1.1 summarizes features of the three languages mentioned above. It compares KAoS, Rei, PeerTrust and Protune 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.

1.3 Web Rule Languages for Policies

The new generation of policy languages goes beyond defining rules that control only the users. They also put constraints on the resources of a domain. The resources may evolve and expand during time, so the policy languages should be expandable as well. Since ontologies are easy to extend, Semantic Web has gained a lot of reputation in this area. Consequently, the intermediary language that is going to be used to transform the information should not only support the definition of the rules, but also be able to transfer the domain properties and features. In this paper, we argue that Semantic Web rules are appropriate solutions to this problem, as they can cover the policies through defining rules and ontologies. Further on this can be found in [KGHW07], where we have discussed and explained the logic of transforming between policy languages.

Most of the proposals on Web rule languages are trying to address the use cases and requirements defined by Rule Interchange Format Working Group [rif].

Rule Interchange Format (RIF) [rif] is an initiative to address the problem of interoperability between existing rule-based technologies. RIF is desired to play as an intermediary language between various rule languages and not as a semantic foundation for the purpose of reasoning on the Web. RIF Working Group has defined ten use cases which have to be covered by a language compliant to the RIF properties, three of which are dealing with policies and business rules, namely: Collaborative Policy Development for Dynamic Spectrum Access, Access to Business Rules of Supply Chain Partners, and Managing Inter-Organizational Business Policies and Practices. SWRL [swr] and RuleML [BDG$^+$05] are two of the ongoing efforts in this area trying to serve as rule languages to publish and share rule bases on the Web.

In this report, we use REWERSE Rule Markup language (R2ML) as an attempt to address the use cases and requirements of RIF. In the next subsection, we briefly describe some concepts of R2ML before going through the description of the mappings between R2ML and KAoS and between R2ML and Rei.

1.4 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.

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 [2]. The full description of R2ML in the form of UML class diagrams is given on the R2ML homepage (see section on Language Metamodel), while more details about the language can be found in [WGL06a]. The language also has an XML concrete syntax defined by an XML

[2]R2ML homepage: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=R2ML

schema, while there are a number of transformations implemented between R2ML and rule based languages (e.g., OCL, SWRL, Jess, and FLogic).

1.4.1 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 [MGG$^+$06], 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.

1.4.2 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 [WGL06a].

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 **??**a). 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 R2ML derivation rule has conditions and a conclusion (see Figure **??**b) 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].

1.4.3 Atoms

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.

1.5 Mapping between R2ML and Policy Languages

In this subsection, we discuss the mappings between different policy languages using R2ML derivation rules. In R2ML, we have both integrity rules and derivation rules defined; with the integrity rules divided into deontic rules and alethic rules. An integrity rule, also known as (integrity) constraint, consists of a constraint assertion, which is a sentence in a logical language such as first-order predicate logic. A derivation rule in R2ML is different from an integrity rule in the sense that there is no concrete proof for the correctness of a derivation rule. A derivation rule is a better construct to show the inference capabilities over existing facts to obtain new facts. This is exactly the same with policies as in most of the cases approval or denial of performing an action is based on an inference over the credentials provided by the user.

1.5.1 Transforming policies from KAoS to R2ML

A KAoS policy is an object of the Policy class in KPO with its attributes instantiated to a set of users, events, and resources that make the policy fire. Considering the KAoS policy element (e.g. Example 1.1) as a rule, the *controls* element is executed upon the occurrence of the events described in the *requiresConditions* element.

Example 1.1 (A sample of a KAoS Policy)
```
<owl:Class rdf:ID="Policy_CommunicationCertificate_Action">
  <owl:intersectionOf>
```

```
    <owl:Class rdf:about="&KAoSAction;CommunicationAction"/>
      <owl:Class>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&KAoSAction;performedBy"/>
            <owl:someValuesFrom>
              <owl:Class>
                <owl:oneOf rdf:parseType="Collection">
                  <owl:Thing rdf:about="&InstanceElem;ClientA"/>
                </owl:oneOf>
              </owl:Class>
            </owl:someValuesFrom>
        </owl:Restriction>
      </owl:Class>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&KAoSPolicy;hasPartner"/>
             <owl:allValuesFrom rdf:resource="&KAoSActor;
                              #ServiceProviderSupportingX509Certificates"/>
      </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<!--Checks on whether the entity that is receiving
    the request by Client A is a trusted entity-->
<owl:Class rdf:about=" Policy_TrustedEntity ">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="&KAoSAction;ApproveAction"/>
      <owl:Restriction rdf:about="#checkOnCarryingMessages">
         <!-Similar to the above restriction;
                 checks if the action is carrying a message -->
      </owl:Restriction>
      <owl:Restriction rdf:about="#TrustedEntity">
        <!-Similar to the above restriction; checks on whether the
                 destination is a Trusted Entity -->
        </owl:Restriction>
      <owl:Restriction rdf:about="#actorRestriction">
        <!-Similar restriction; checks on whether the actor ClientA -->
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

<policy:PosAuthorizationPolicy rdf:ID="Policy_CommunicationCertificate">
   <policy:requiresConditions rdf:resource="#Policy_ TrustedEntity "/>
   <policy:controls rdf:resource="#Policy_CommunicationCertificate_Action"/>
   <policy:hasPriority>2</policy:hasPriority>
</ policy:PosAuthorizationPolicy>
```

Example 1.2 (An equivalent Rei policy of Example 1.1)

```
<entity:Variable rdf:ID="ActorVar"/>

<policy:Policy rdf:ID="policy_N100CE">
    <deontic:actor rdf:resource="#ActorVar" />
    <policy:grants rdf:resource="#Policy_CommunicationCertificate" />
    <policy:context rdf:resource="#CommunicationActionConstraint" />
</policy:Policy>

<deontic:Permission rdf:ID="Policy_CommunicationCertificate">
      <deontic:action rdf:resource="&KAoSAction;CommunicationAction"/>
      <deontic:actor rdf:resource="#ActorVar"/>
      <deontic:constriant rdf:resource="#ActionApprovalConstraint"/>
</deontic:Permission>

<constraint:And rdf:ID="ActionApptovalConstraint">
        <!--Conjunction of Constraints for the precondition
            plus the CommunicationActor constraint-->
</constraint:And>

<!-- Constraints for the TrustedEntity and CarryMessage
     to control the behavior of the system -->

<constraint:SimpleConstriant rdf:ID="CommunicationActor">
    <constriant:subject rdf:resource="#ActorVar"/>
    <constriant:object rdf:resource="&rdfs;type"/>
    <constraint:predicate rdf:resource="&KAoSActor;ClientA"/>
</constraint:SimpleConstriant>

<constraint:SimpleConstriant rdf:ID=" CommunicationActionConstraint">
    <constriant:subject rdf:resource="#ActionVar"/>
    <constriant:object rdf:resource="&KAoSActor;
                        ServiceProviderSupportingX509Certificates"/>
    <constraint:predicate rdf:resource="&KAoSPolicy;hasPartner"/>
</constraint:SimpleConstriant>
```

Thus, to model the KAoS policy with a derivation rule, we place the content of the controls element in the conclusion part and the content of the *requiresConditions* element in the condition part of the rule. A *controls* element consists of the action to be performed, the actor of the action, and the context of performing the action. To model the actor, the action and the restrictions defined over the context of the to-beexecuted action, we chose R2ML *ObjectDescriptionAtom*. This atom can neatly embed all of the mentioned concepts as arguments in its definition.

To model the condition part of a KAoS policy, we employed R2ML *ReferencePropertyAtoms*. Similar to the control part, conditions in KAoS are usually represented as a class defined over an occurred action or state with a set of properties that restrict the action. Although the conditions of a policy rule could also be modeled with *ObjectDescriptionAtom*, the main reason in choosing *ReferencePropertyAtom* was to be compliant with the definitions of Rei (defined in the form of triples) and also other R2ML transformations (e.g transformations between F-Logic and R2ML also have *ReferencePropertyAtom* in the condition part). It simplifies the later conversions of the policies to other rule languages for which we have R2ML transformations already defined (e.g F-Logic). Moreover, a ReferencePropertyAtom triple models a binary predicate. A set of ReferencePropertyAtoms with the same subject element can always be combined and converted to any element of higher arity (e.g. ObjectDescriptionAtom), and thus using ReferencePropertyAtom does not contradict with the use of ObjectDescriptionAtom. Furthermore, in our case, ReferencePropertyAtoms carry even a better semantic meaning for the transformations. Semantically they are equivalent to an OWL object property, and as KAoS is nothing but pure OWL, they model object properties of KAoS too.

A KAoS policy might also have a *trigger* element. This element is only used with NegObligation- and PosObligation-Policies showing a set of events that trigger the occurrence of an action. In our transformations, we deal with those elements as ReferencePropertyAtoms in the condition part as well. However, to discriminate them from the preconditions, we annotate them as triggering elements.

We show an excerpt of transformation rules between KAoS and R2ML in Figure **??**. The XSLT implementations of our transformations are available in [Kav07], where further details can be found.

Example 1.3 (Mappings between KAoS and R2ML elements) ?? *A KAoS Policy Element:*

```
<policy:PosAuthorizationPolicy
         rdf:ID="Policy_CommunicationCertificate">
 <policy:requiresConditions rdf:resource="#Policy_ TrustedEntity "/>
 <policy:controls
         rdf:resource="#Policy_CommunicationCertificate_Action"/>
</policy:PosAuthorizationPolicy>
```

corresponds to the R2ML derivation rule:

```
<r2ml:DerivationRule>
     <r2ml:conditions>
      <!-- The mappings for the condition part goes here -->
    </r2ml:conditions>
    <r2ml:conclusion>
    <r2ml:ObjectDescriptionAtom r2ml:classID="Permission">
       <!-- The mappings for the conclusion part goes here -->
    </r2ml:ObjectDescriptionAtom>
  </r2ml:conclusion>
</r2ml:DerivationRule>
```

KAoS control element:

```
<owl:Class rdf:ID="Policy_CommunicationCertificate_Action">
    <owl:intersectionOf>
```

```
<!-- the set of constraints on the action in KAoS are defined as
intersection of a series of restrictions in OWL  -->
</owl:intersectionOf>
</owl:Class>
```

corresponds to the R2ML object description atom:

```
<r2ml:ObjectDescriptionAtom r2ml:classID="Permission">
  <!-- The mappings for the conlcusion part goes here -->
</r2ml:ObjectDescriptionAtom>
```

KAoS control element:

```
      <owl:Class>
        <owl:Restriction>
          <owl:onProperty rdf:resource="http://ontology.ihmc.us/
                                        Action.owl#performedBy"/>
            <owl:allValuesFrom>
              <!--Defines the set of actors that are responsible for the action-->
            </owl:allValuesFrom>
        </owl:Restriction>
      </owl:Class>
```

corresponds to the R2ML object slot:

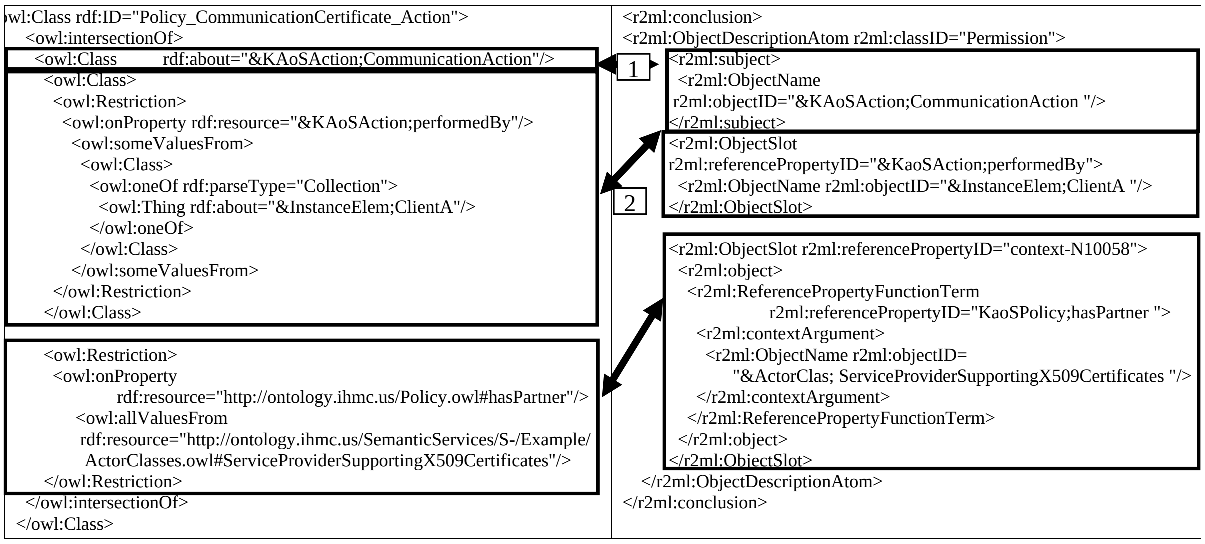
```
<r2ml:ObjectSlot r2ml:referencePropertyID="
        http://ontology.ihmc.us/Action.owl#performedBy">
      <!--We can use either r2ml:ObjectName or
             r2ml:ObjectVariable define the actors -->
</r2ml:ObjectSlot>
```

KAoS control element:

```
      <owl:Restriction>
        <owl:onProperty
               rdf:resource="http://ontology.ihmc.us/Policy.owl#hasPartner"/>
             <owl:allValuesFrom
                    rdf:resource="#TrustedServiceProvider"/>
      </owl:Restriction>
```

corresponds to the R2ML object slot:

```
<r2ml:ObjectSlot r2ml:referencePropertyID="context-N10058">
   <r2ml:object>
      <r2ml:ReferencePropertyFunctionTerm
         r2ml:referencePropertyID="
         http://ontology.ihmc.us/Policy.owl#hasPartner">
         <r2ml:contextArgument>
           <r2ml:ObjectName r2ml:name="#TrustedServiceProvider"/>
         </r2ml:contextArgument>
       </r2ml:ReferencePropertyFunctionTerm>
     </r2ml:object>
</r2ml:ObjectSlot>
```



```xml
wl:Class rdf:ID="Policy_CommunicationCertificate_Action">
  <owl:intersectionOf>
   <owl:Class         rdf:about="&KAoSAction;CommunicationAction"/>
    <owl:Class>
     <owl:Restriction>
      <owl:onProperty rdf:resource="&KAoSAction;performedBy"/>
       <owl:someValuesFrom>
        <owl:Class>
         <owl:oneOf rdf:parseType="Collection">
          <owl:Thing rdf:about="&InstanceElem;ClientA"/>
         </owl:oneOf>
        </owl:Class>
       </owl:someValuesFrom>
     </owl:Restriction>
    </owl:Class>

    <owl:Restriction>
     <owl:onProperty
             rdf:resource="http://ontology.ihmc.us/Policy.owl#hasPartner"/>
       <owl:allValuesFrom
        rdf:resource="http://ontology.ihmc.us/SemanticServices/S-/Example/
         ActorClasses.owl#ServiceProviderSupportingX509Certificates"/>
    </owl:Restriction>
  </owl:intersectionOf>
 </owl:Class>
```

```xml
<r2ml:conclusion>
<r2ml:ObjectDescriptionAtom r2ml:classID="Permission">
<r2ml:subject>
  <r2ml:ObjectName
 r2ml:objectID="&KAoSAction;CommunicationAction "/>
</r2ml:subject>
<r2ml:ObjectSlot
r2ml:referencePropertyID="&KaoSAction;performedBy">
  <r2ml:ObjectName r2ml:objectID="&InstanceElem;ClientA "/>
</r2ml:ObjectSlot>

<r2ml:ObjectSlot r2ml:referencePropertyID="context-N10058">
  <r2ml:object>
   <r2ml:ReferencePropertyFunctionTerm
                r2ml:referencePropertyID="KaoSPolicy;hasPartner ">
    <r2ml:contextArgument>
     <r2ml:ObjectName r2ml:objectID=
        "&ActorClas; ServiceProviderSupportingX509Certificates "/>
    </r2ml:contextArgument>
   </r2ml:ReferencePropertyFunctionTerm>
  </r2ml:object>
</r2ml:ObjectSlot>
 </r2ml:ObjectDescriptionAtom>
</r2ml:conclusion>
```

Figure 1.1: Mapping of a KAoS controls element (left) to R2ML (right)

KAoS uses role-value-map technique to deal with dynamic allocation of values to variables. However, in our implementation, we use a simpler model of defining variables (similar to what Rei does) and convert role-value-mapped elements of KAoS to a variable-like definition using R2ML's *ObjectVariable*. This makes the process of converting R2ML variables to the variables of other languages easier. Figure 1.1 shows an excerpt of the policy that we described in Example 1.1, converted to its R2ML equivalent based on the transformation rules explained in Example **??**. The conversion shows how a controls element of KAoS is represented in the conclusion part of an R2ML derivation rule.

1.5.2 Transforming policies from Rei to R2ML

A Rei policy, similar to KAoS, is an instantiation of the Policy class, defined in the Rei ontology [Kag06]. However, a policy element in Rei represents a list of policy rules (each defined as a deontic child element), while a policy construct in KAoS represents only one rule. Each R2ML derivation rule is also equivalent to one policy rule. Therefore, converting a policy from Rei to KAoS or R2ML may result in having more than one KAoS policy or R2ML rule. Furthermore, as Rei deals with variables similar to Prolog, and because variables can have different values during run-time, a single policy in Rei might be converted to multiple KAoS policies based on different combinations of values that the variables can take. Fortunately R2ML can accept a set of derivation rules by defining them as derivation rule set, in case more than one policy rule can be derived from a Rei policy.

Our R2ML rule structure is more similar to the structure of Rei than to KAoS and hence it is easier to convert a Rei policy rule to R2ML. Rei uses *SimpleConstraints* and *BinaryConstraints* to define the conditions of a deontic rule. All constraints for a deontic element are considered as preconditions of that deontic rule and treated the same way as requiresConditions element in KAoS. A deontic element in Rei has an action and an actor as its child elements as well.

The actor is mapped as an object argument with a *performedBy* connector under the R2ML ObjectDescriptionAtom in the conclusion part of a rule, similar to what we did for KAoS. The actions in Rei are defined either by using Rei elements or OWL classes. For mapping the actions to R2ML, we can just refer to the already defined action or use R2ML vocabulary to redefine it. The action can then be placed in the subject element of our R2ML ObjectDescriptionAtom which is again similar to what we did for KAoS.

A policy element in Rei has also a child element, named context. The deontic set of rules operate on this context. So, in our R2ML transformation, we copy this same context for all derivation rules and store it as a ReferencePropertyFunction- Term in the conclusion part of our derivation rule under the ObjectDescriptionAtom for the policy element. Thanks to the use of ReferencePropertyAtom in R2ML, we can easily convert each triple constraint of Rei to R2ML through a one to one mapping of the subject, object, and predicate elements. Example **??** shows some of the mapping rules used to convert a Rei rule to an equivalent R2ML construct.

Example 1.4 (Mappings between Rei and R2ML elements) ?? *A Rei deontic element:*

```
<deontic:Permission rdf:ID="PolicyName">
   <deontic:action rdf:resource="#actionToPerform"/>
   <deontic:actor rdf:resource="#ActorSet"/>
   <deontic:constriant rdf:resource="#ConstraintsSet"/>
</deontic:Permission>
```

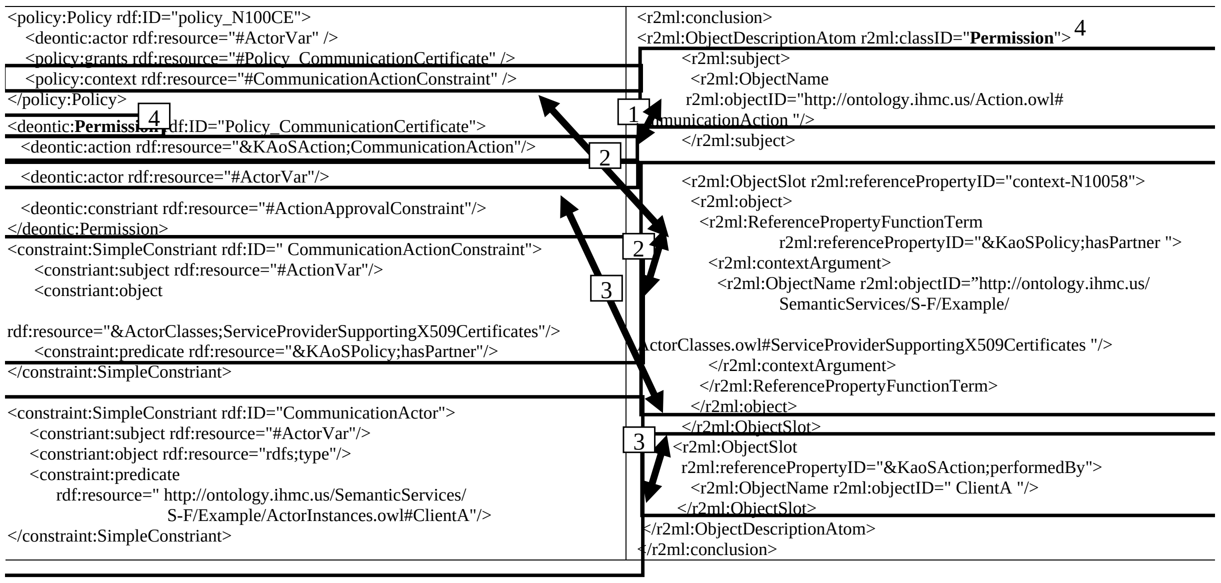
A corresponding R2ML derivation rule:

```
<r2ml:DerivationRule>
  <r2ml:conditions>
      <!-- The constraints are placed here -->
  </r2ml:conditions>
  <r2ml:conclusion>
      <r2ml:ObjectDescriptionAtom r2ml:classID="Permission">
            <!-- The mappings for the conclusion part goes here -->
            <!-- action and actor elements go here -->
      </r2ml:ObjectDescriptionAtom>
  </r2ml:conclusion>
</r2ml:DerivationRule>
```

A constraint Rei element:

```
<constraint:SimpleConstriant rdf:ID="ActorElem">
    <constriant:subject rdf:resource="#Actor"/>
    <constriant:object rdf:resource="#Student "/>
    <constriant:predicate rdf:resource="&rdfs;type "/>
</constraint:SimpleConstriant>
```

A corresponding R2ML reference property atom:



```
<policy:Policy rdf:ID="policy_N100CE">
   <deontic:actor rdf:resource="#ActorVar" />
   <policy:grants rdf:resource="#Policy_CommunicationCertificate" />
   <policy:context rdf:resource="#CommunicationActionConstraint" />
</policy:Policy>
<deontic:Permission rdf:ID="Policy_CommunicationCertificate">
  <deontic:action rdf:resource="&KAoSAction;CommunicationAction"/>
  <deontic:actor rdf:resource="#ActorVar"/>
  <deontic:constriant rdf:resource="#ActionApprovalConstraint"/>
</deontic:Permission>
<constraint:SimpleConstriant rdf:ID=" CommunicationActionConstraint">
     <constriant:subject rdf:resource="#ActionVar"/>
     <constriant:object

rdf:resource="&ActorClasses;ServiceProviderSupportingX509Certificates"/>
     <constraint:predicate rdf:resource="&KAoSPolicy;hasPartner"/>
</constraint:SimpleConstriant>

<constraint:SimpleConstriant rdf:ID="CommunicationActor">
    <constriant:subject rdf:resource="#ActorVar"/>
    <constriant:object rdf:resource="rdfs;type"/>
    <constraint:predicate
        rdf:resource=" http://ontology.ihmc.us/SemanticServices/
                        S-F/Example/ActorInstances.owl#ClientA"/>
</constraint:SimpleConstriant>
```

```
<r2ml:conclusion>
<r2ml:ObjectDescriptionAtom r2ml:classID="Permission">
      <r2ml:subject>
        <r2ml:ObjectName
       r2ml:objectID="http://ontology.ihmc.us/Action.owl#
municationAction "/>
      </r2ml:subject>

      <r2ml:ObjectSlot r2ml:referencePropertyID="context-N10058">
        <r2ml:object>
          <r2ml:ReferencePropertyFunctionTerm
                    r2ml:referencePropertyID="&KaoSPolicy;hasPartner ">
            <r2ml:contextArgument>
              <r2ml:ObjectName r2ml:objectID="http://ontology.ihmc.us/
                    SemanticServices/S-F/Example/

ActorClasses.owl#ServiceProviderSupportingX509Certificates "/>
            </r2ml:contextArgument>
          </r2ml:ReferencePropertyFunctionTerm>
        </r2ml:object>
      </r2ml:ObjectSlot>
     <r2ml:ObjectSlot
      r2ml:referencePropertyID="&KaoSAction;performedBy">
        <r2ml:ObjectName r2ml:objectID=" ClientA "/>
     </r2ml:ObjectSlot>
</r2ml:ObjectDescriptionAtom>
</r2ml:conclusion>
```

Figure 1.2: Mapping of a KAoS controls element (left) to R2ML (right)

```
<r2ml:ReferencePropertyAtom r2ml:propertyID="&rdfs;type">
   <r2ml:subject>
       <r2ml:ObjectVariable r2ml:name="#Actor"/>
   </r2ml:subject>
   <r2ml:object>
           <r2ml:ObjectName r2ml:objectID="#Student"/>
   </r2ml:object>
</r2ml:PropertyAtom>
```

Now that we have the transformations defined, let us apply them to the Rei policy in Example 1.2.

Figure 1.2 shows the result of applying the transformation rules. Comparing the R2ML snippet generated from the defined Rei policy in Figure 1.2 with the R2ML snippet of Figure 1.1 one would immediately realize that these two rules are identical although they have been generated from two completely different policy rules. It should be mentioned that the transformations are not always as straightforward as in this example. This might be a result of difference in the level of abstraction in the policy languages, the way they address variables, concepts, and entities, etc. The arrows in Figures 1.1 and 1.2 show the equivalent information pieces in the source and target languages.

1.5.3 Transforming Protune to R2ML

As we have already mentioned in the section 1.2.3, the rule language of Protune is based on normal logic program rules, therefore the most suitable corresponding rule type in R2ML for the representation of Protune policies are derivation rules.

According to the Protune language description [Bon05], the vocabulary of predicates occurring in the rules is partitioned into a number of categories. We describe the mapping solutions for each predicate type below and give appropriate examples of Protune expressions and corresponding R2ML XML.

1.5.3.1 Decision predicates

Currently this class comprises predicates `allow` and `sign`. These predicates are defined in the policy, that is, they occur in the head of some policy rules. The unary predicate `allow` is queried by the negotiator for access control decisions. The argument of `allow` can denote a service call (for access control decisions) or it can be `release`(*credential*) or `execute`(*action*) (for privacy protection). In response to a service request s, the negotiator looks for a (partial) proof of `allow`(s), and handles it as sketched in the previous section. Similarly, in response to a credential request or an action request r, the negotiator looks for a proof of `allow`(r) and processes it appropriately.

Predicate `sign` is used to issue statements signed by the principal owning the policy. The argument of `sign` can be any term, possibly consisting of attribute-value pairs. This feature is useful to issue new credentials stating domain-dependent properties. In response to a statement request r, a (partial) proof of `sign`(r) is searched for.

Policy heads, which contain these predicates, correspond to R2ML generic atoms. R2ML rule base, with resulting policies must have a policy vocabulary, where predicates `allow` and `sign` are defined in the separate namespace, which is used for denoting Protune built-ins.

Example 1.5 *Let us consider the head of a policy:*

```
allow(enter site())
```

Here `enter site()` *is a user-defined operation. This head corresponds to the following R2ML conclusion with a generic atom:*

```
<r2ml:GenericAtom r2ml:predicate="protune:allow">
  <r2ml:arguments>
   <r2ml:DataOperationTerm r2ml:operation="enter_site">
    <r2ml:arguments/>
   </r2ml:DataOperationTerm>
  </r2ml:arguments>
</r2ml:GenericAtom>
```

1.5.3.2 Abbreviation/abstraction predicates

These are predicates defined in the policy. They have many purposes ranging from the definition of high-level client properties (e.g. by combining low-level data and/or different credentials) to the specification of new credential semantics.

Policy atoms with these predicates are mapped into corresponding R2ML atoms, depending on the predicate structure.

1.5.3.3 Constraint predicates comprise the usual equality and inequality predicates

Policy atoms with these predicates are mapped into R2ML equality and inequality atoms.

1.5.3.4 State predicates

Policy decisions have to be taken with respect to a time-dependent system state, encoding the current negotiation state, legacy data, user profiles, and so on. State predicates are further partitioned into the state query predicates and provisional predicates. State query predicates read the current state without modifying it. They comprise both built-in and application dependent predicates and, therefore, are mapped into R2ML generic predicates, defined in the Protune built-in vocabulary and into R2ML user defined predicates.

Built-in state predicates model the state of the negotiation, and provide a uniform interface to external packages in the style of Hermes [Subrahmanian et al., 1995]. An example of negotiation state atom is `request`(n,R); it holds if R is the n-th request in the negotiation. It corresponds to the R2ML generic atom:

```
<r2ml:GenericAtom r2ml:predicate="protune:request">
  <r2ml:arguments>
    <r2ml:DataVariable r2ml:name="n" r2ml:datatype="xs:integer"/>
    <r2ml:DataVariable r2ml:name="R" r2ml:datatype="xs:string"/>
  </r2ml:arguments>
</r2ml:GenericAtom>
```

Provisional predicates may be made true by means of appropriate actions that may modify the current state. Such actions may be carried out by the server, by the client, or both. An important example is `credential`. An atom `credential`(C,K) is true when the current negotiation state contains a verified credential matching C and signed by the principal whose public key is K. These predicates are mapped into R2ML generic predicates.

```
<r2ml:GenericAtom r2ml:predicate="protune:credential">
  <r2ml:arguments>
    <r2ml:ObjectVariable r2ml:name="C" r2ml:class="protune:Credential"/>
    <r2ml:DataVariable r2ml:name="K" r2ml:datatype="xs:string"/>
  </r2ml:arguments>
</r2ml:GenericAtom>
```

1.5.3.5 Protune metapolicies representation in R2ML

Metapolicies in Protune are used for fine-grained tuning of (state) predicate evaluation. For example, a strategy that selects negative (state) literals for immediate evaluation only if they are ground can be expressed simply with:

$(\neg A)$`:evaluation` $\longleftarrow$ `immediate ground`(A)

They can be considered as instructions to the Protune engine, which tells how to evaluate predicates. Metapolicies have a technical nature, not a business, since a developer, who writes policies from the business perspective, does not know a lot about such concepts as "groundness" and "immediate evaluation". R2ML being a general rule markup language does not support the markup of platform-specific expressions, which are the Protune metapolicies. However, while translating from Protune to R2ML, metapolicies have to be marked up. A translator of Protune policies in R2ML to some other policy language should preserve the semantics of the metapolicies and transfer them from the R2ML annotations into their counterpart in the target policy language. This is a difficult task, which requires an additional research and evaluation.

1.5.3.6 Conclusion on Protune to R2ML transformation

Even if the task of policies interchange from the Protune language to some other policy language via R2ML seems rather complex, the mechanism of Protune policies serialization into R2ML XML gives a number of advantages to policy modelers.

- R2ML can be used as an XML syntax for Protune, which can easily be processed using XML technologies like XSLT and XQuery.
- The Working Group I1 has developed verbalization component for R2ML [LW06b], which can be used for policies verbalization.
- The R2ML rules visualization components [3] can be used to represent policies in a visual form for easier understanding.
- With the help of the UML-based Rule Modeling Language of I1[4] [WGL+06c], which is supported by the Strelka tool [WGL+06b], [LW06a], the policies can be modeled visually. The visual policy models can be translated into Protune via R2ML.

1.5.4 Transforming from R2ML to Rei or KAoS

Transforming back to either of these policy languages is a much simpler task now that we have the transformation rules defined. Of course, the mappings are bidirectional. That is, the same way that, for example, a SimpleConstraint element of the Rei language would be converted to a ReferencePropertyAtom in R2ML, a ReferencePropertyAtom in the condition part of a rule can be converted to a SimpleConstraint in Rei. However, there are some concepts that are not simple to map. For example one may think how an OWL class will be created out of a set of ReferencePropertyAtoms. As we have already mentioned, the set of ReferencePropertyAtoms with the similar subject will create an OWL class with the properties modeled as restrictions on values in the object part of the ReferencePropertyAtom. In situations where the ReferencePropertyAtom carries a *trigger* tag, the obtained OWL class will be considered as a triggering class and otherwise it will be a precondition.

Another similar issue in our transformations happens when dealing with priorities. Priorities in KAoS are defined with numbered values, but in Rei we have meta-rules to give priority to one rule over the other. We have tried to solve this problem by converting the meta-rules of Rei to a numbered model in our R2ML transformation. During conversion from R2ML to Rei we convert the numbers to a form of meta-rules compliant with Rei syntax.

[3] Visualizing R2ML rules: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/16
[4] URML homepage: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=URML

1.6 Discussion and Conclusion

We have so far described the transformations between policy languages and R2ML, thus proved that the transformations are possible. But it does not mean that we can fully transform between the two languages without any information loss. The characteristics of policy languages, especially KAoS and Rei (due to the differences in their underlying logic), makes an exact mapping between different elements difficult. KAoS follows description logic in which we have constructs for different quantifiers (both existential and universal), but Rei is similar to Prolog in which all the variables are universally quantified. This means while converting a KAoS rule to an equivalent Rei policy, we can not explicitly express that the defined variable in Rei should be existentially quantified.

In KAoS, we can define maxCardinality and minCardinality as restrictions on the number of events, but to the best of our knowledge there is no way to define such concepts in Rei. So, either these restrictions should be ignored or Rei should be expanded to cover them. It is worth mentioning that generality of R2ML helps in defining all the concepts above, because it has the required elements to cover the concepts in both domains (i.e. descriptive logic and declarative logic). We are now conducting research on the amount of harms and threats that may happen to the system due to the information loss during transformations.

As we have already mentioned, Rei has SpeechAct elements to describe the inter-process communications for remote policy control. In KAoS, there is no way to model these concepts and handling the communication between remote systems is supposed to be done by the underlying system. Therefore, although we can transfer SpeechActs from Rei to R2ML, they will be left out during the transformation to KAoS. Unfortunately, most of the time the constructs used in SpeechActs are inter-woven to the other policy elements of the language, such as Permission or Prohibition. In this case, a transformation to KAoS would not be helpful at all as the intended meaning of the policy will be lost.

We have also explained why derivation rules work better to define the policies. Now that we have the policies defined with derivation rules, we can use some other transformations that map a R2ML derivation rule with a similar structure to another rule language. One possible transformation is R2ML to F-Logic [17]. The existence of the transformation to F-Logic enables us to convert our policies into F-Logic and use them in systems compatible with F-Logic rules. According to the transformation rules provided in [GW06], the ObjectDescriptionAtom in our R2ML excerpt (Figure 1.1-right and Figure 1.2-right) for the policy language of Section **??** can be expressed as Figure 1.6. The given excerpt for the F-Logic transformation shows only the atom in the conclusion part of a rule, which illustrates the appropriate permission to be given. The ongoing efforts to provide transformations from R2ML to the other rule languages would add to the generality of R2ML and make it a suitable asset for the purpose of information exchange. For example, our transformations from Rei and KAoS to R2ML showed the need for having both existential and universal quantifiers defined for derivation rules, which currently only entail the universal quantifiers. The modifications to the language will appear in the next version of R2ML.

Example 1.6 (An R2ML element represented in F-Logic)

```
"Policy_CommunicationCertificatePermission" [
hasAction ? "CommunicationAction";
hasContext\_N10058 ? hasPartner(
"ServiceProviderSupportingX509Certificates");
performedBy ? "ClientA"]
```

Policy-RuleML [Pol] is a similar attempt in the area of policy transformation. It aims at making RuleML a semantic interoperation vehicle for heterogeneous policies and protocols on the Web. However, to the best of our knowledge there is no proposed work done by this group, and thus our solution seems to be the first practical attempt in the area. The future goal of the research will be combining the semantic web service description languages with policy languages and trying to get different web services with different descriptions and policy languages to work together. The current proposals on combining semantic web services with policy languages have been proposed in [OLPL05] that combines WSMO and PeertTrust, [UBJ$^+$03] that uses KAoS to protect web services, and [KPS$^+$04] that combines Rei and OWL-S. Our goal will be to let all of these services to communicate regardless of the languages they use for defining and describing their services and policies. Furthermore, we are working on developing transformations from OCL to R2ML [WGL06a]. Another goal will be to make all the transformations between policy languages and OCL consistent, so that we can eventually integrate Semantic Web policies for services with Model-Driven software development approaches. Developing a general policy language based on R2ML to cover the concepts of all above mentioned policy languages is also another future activity.

Chapter 2

R2ML Compatibility with the Rule Interchange Format by W3C

2.1 Rule Interchange Format

The W3C Rule Interchange Format Working Group was started in 2005 to produce a core rule language plus extensions which together *allow rules to be translated between rule languages and thus transferred between rule systems.* The Working Group have to balance the needs of a diverse communities - including Business Rules and Semantic Web users - specifying extensions for which it can articulate a consensus design and which are sufficiently motivated by use cases.

The actual deliverables concerning the RIF activity are:

1. Second Public Working Draft of RIF Use Cases and Requirements (RIF-UCR) A. Ginsberg, D. Hirtle, F. McCabe, P. Patranjan, Eds.

2. First Public Working Draft of RIF Basic Logic Dialect (RIF-BLD) H. Boley and M. Kifer, Eds.

3. First Public Working Draft of RIF RDF and OWL Compatibility (RIF-RDF-OWL) J. de Bruijn, Ed.

2.2 R2ML compatibility with RIF goals

The RIF-UCR document (W3C Working Draft 10 July 2006) states a number of goals of the future Rule Interchange Format. The sections below presents the compatibility of the REWERSE I1 Rule Markup Language (R2ML) with these goals.

2.2.1 G1: Consistency with W3C Specifications

RIF is intended to be a W3C specification that builds on and develops the existing range of specifications that have been developed by the W3C. This implies that existing W3C technologies

Criteria	**R2ML Compatibility**
XML Syntax	Yes
Support XML Data	No
OWL Data	Yes
RDF Data	Yes

Table 2.1: R2ML compatibility with RIF G1

should fit well with the RIF.

2.2.2 G2-3: Exchange of Rules, Widescale Adoption

The RIF-UCR stands that *"...it is important that the RIF is able to incorporate rule languages not currently envisaged."*
R2ML experimentally covers translators for: F-Logic, Jena Rules, Jess, RuleML 0.9, SWRL, OCL, JBoss Rules.

Also the same document claim that *"... it must be predictable what is exchanged when a ruleset is exchanged between partners and/or tools."*
This is covered by R2ML by offering the main ontological concepts existent in the intended target languages such as RuleBase, RuleSet, Rules, conditions, conclusion, action etc.

Low Cost of Implementation.
R2ML interchange is based on simple translators (equivalent transformations are carried out by special R2ML tools)

"RIF must be implementable using well understood techniques."
Actual R2ML translators use XSLT and Java.

"RIF implementations must be able to use standard support technologies such as XML parsers and other parser generators." R2ML is XML Schema-based therefore complains with any XML parser (we use now Saxon 8)

Finally we have to say that R2ML tries to accommodate all RIF goals:

- It is consistent with W3C specs,
- Consider exchange of rules as its main goal,
- Consider as target languages the main existent rule languages on the market
- Has low cost implementation
- Is extensible inside of XML Schema specification

To achieve these goals R2ML adopted a number of principles:

- Clear distinction between *objects* and *data.* All targeted languages make this distinction.
- Distinction between *different kinds of rules.* This is necessary since different rule platforms use different types of rules (see for example Jena, Jess, JBoss Rules).
- Accommodates UML and OCL towards quickly implementation in software engineering.

2.3 R2ML compatibility with RIF Requirements and Use Cases

Compliance model. *"RIF must define a compliance model that will identify required/optional features."* Actually R2ML does not identify optional and required concepts. Three is no systematical research in this direction. However, the interchange experience proved that all important concepts from R2ML are necessary and some other new optional features has to ba added (such as providing markup for production rule priorities).

Default behavior. *"RIF must specify at the appropriate level of detail the default behavior that is expected from a RIF compliant application that does not have the capability to process all or part of the rules described in a RIF document, or it must provide a way to specify such default behavior."* Actually R2ML provides generic concepts like GenericAtom, GenericEntityName, GenericFunction such that any translation may use them as defaults if other solution is not available.

Different intended semantics. *"RIF must cover rule languages having different intended semantics."* R2ML does not have a fixed semantics

Embedded comments. *"RIF must be able to pass comments."* R2ML provides comments in the form of XML comments.

Embedded metadata. *"RIF must support metadata such as author and rule name."* R2ML provides `ruleID` attribute for rule names include Dublin Core as well as elements such as RuleText and SourceCode for capturing other rule annotations.

Implementability. *"RIF must be implementable using well understood techniques."* R2ML is XML Schema based. Translators are built with XSLT and Java.

Limited number of dialects. *"RIF must have a limited number of standard dialects and/or a common core."* R2ML does not define yet dialects. The block Schema offers the possibility to interchange with good results.

OWL data. *"RIF must cover OWL knowledge bases as data where compatible with Phase 1 semantics."* R2ML covers OWL data in the form of ObjectName for individuals, and rules expressing OWL class descriptions.

RDF data. *"RIF must cover RDF triples as data where compatible with Phase 1 semantics."* R2ML directly supports RDF triples by meaning of ReferencePropertyAtom and AttributionAtom.

Rule language coverage. *"RIF must cover the set of languages identified in the Rulesystem Arrangement Framework."* R2ML targets a number of rule languages which are present if RIF-RAF document too.

Semantic precision. *"RIF must have a clear and precise (unambiguous) semantics to reduce the potential for error in the exchange of rules."* R2ML does not provide a semantics yet. Its goal is to be compatible with the RIF-BLD since that one is going to became W3C standard.

Semantic tagging. *"RIF must have a standard way to specify the intended semantics (or semantics style) of the interchanged rule set in a RIF document. "* Actually R2ML provides `PredicateCategory` type with the following possible values: `Closed` (default), `Open`, `Partial`.

Standard components. *"RIF implementations must be able to use standard support technologies such as XML parsers and other parser generators."* R2ML is XML Schema Based. Therefore the language is compatible with any XML parser and authoring and validation tools can be used.

Translators. *"RIF must not require rule systems to be changed; it must be implementable via translators."* R2ML already provide a large number of translators.

XML syntax. *"RIF must have an XML syntax as its primary normative syntax."* R2ML has a primary normative XML syntax.

XML types. *"RIF must permit XML information types (where appropriate) to be expressed using XML Schema."* R2ML completely accommodates XML Schema datatypes, also user-defined types.

As a conclusion we can say that R2ML follows in a large scale the RIF requirements. A number of improvements can be done: Research towards a R2ML compliant model, Research of alignment with RAF, More research on extensibility.

The RIF-UCR describes a set of ten use-cases representing a commitment for the future RIF specification, since it is expected that the relevant functionality specified in the use cases will ultimately be enabled by applications or systems that incorporate RIF technical specifications. However, these use cases are provided in a very basic description. Therefore we choose to test R2ML on a more complicated use cases. One important use case is UServ Product Derby 2005 The below sections present the most well described use cases. They are completely supported by R2ML.

2.3.1 Negotiating eBusiness Contracts Across Rule Platforms

This use case illustrates a fundamental use of the RIF: *to supply a vendor-neutral representation of rules, so that rule-system developers and stakeholders can do their work and make product investments without concern about vendor lock-in, and in particular without concern that a business partner does not have the same vendor technology.* Two rules are exemplified here:

`If an item is perishable and it is delivered more than 10 days after the scheduled delivery date then the item will be rejected.`

`If an item is perishable and it is delivered more than 7 days after the scheduled delivery date but less than 14 days after the scheduled delivery date then a discount of 18.7% will be applied to this delivery.` These rules are based on a business vocabulary involving classes such as `Item` or `Delivery` and properties such as `delivery date`. They are very similar with Userv Use Case already implemented by R2ML. For example, the Userv rule `If young driver and married and not located in CA, NY or VA, then increase premium by $300` is represented in R2ML as:

```
<r2ml:ProductionRule r2ml:ruleID="DP_03">
    <r2ml:conditions>
      <r2ml:qf.Conjunction>
        <r2ml:AttributionAtom r2ml:attributeID="userv:Driver.driverAgeCategory">
          <r2ml:subject>
            <r2ml:ObjectVariable r2ml:name="driver" r2ml:classID="userv:Driver"/>
          </r2ml:subject>
          <r2ml:dataValue>
            <r2ml:TypedLiteral r2ml:lexicalValue="young driver" r2ml:datatypeID="xs:string"/>
          </r2ml:dataValue>
        </r2ml:AttributionAtom>
        <r2ml:AttributionAtom r2ml:attributeID="userv:Driver.maritalStatus">
          <r2ml:subject>
            <r2ml:ObjectVariable r2ml:name="driver" r2ml:classID="userv:Driver"/>
          </r2ml:subject>
          <r2ml:dataValue>
            <r2ml:TypedLiteral r2ml:lexicalValue="married" r2ml:datatypeID="xs:string"/>
          </r2ml:dataValue>
        </r2ml:AttributionAtom>
        <r2ml:qf.Disjunction>
          <r2ml:AttributionAtom r2ml:attributeID="userv:Driver.usState" r2ml:isNegated="true">
            <r2ml:subject>
              <r2ml:ObjectVariable r2ml:name="driver" r2ml:classID="userv:Driver"/>
            </r2ml:subject>
            <r2ml:dataValue>
              <r2ml:TypedLiteral r2ml:lexicalValue="CA" r2ml:datatypeID="xs:string"/>
            </r2ml:dataValue>
          </r2ml:AttributionAtom>
          <r2ml:AttributionAtom r2ml:attributeID="userv:Driver.usState" r2ml:isNegated="true">
            <r2ml:subject>
              <r2ml:ObjectVariable r2ml:name="driver" r2ml:classID="userv:Driver"/>
            </r2ml:subject>
            <r2ml:dataValue>
              <r2ml:TypedLiteral r2ml:lexicalValue="NY" r2ml:datatypeID="xs:string"/>
            </r2ml:dataValue>
          </r2ml:AttributionAtom>
          <r2ml:AttributionAtom r2ml:attributeID="userv:Driver.usState" r2ml:isNegated="true">
            <r2ml:subject>
              <r2ml:ObjectVariable r2ml:name="driver" r2ml:classID="userv:Driver"/>
            </r2ml:subject>
            <r2ml:dataValue>
              <r2ml:TypedLiteral r2ml:lexicalValue="VA" r2ml:datatypeID="xs:string"/>
            </r2ml:dataValue>
          </r2ml:AttributionAtom>
        </r2ml:qf.Disjunction>
      </r2ml:qf.Conjunction>
    </r2ml:conditions>
    <r2ml:producedAction>
      <r2ml:AssignActionExpression r2ml:propertyID="userv:VehicleInsurancePolicy.premium">
        <r2ml:contextArgument>
          <r2ml:ObjectVariable r2ml:name="vehicleInsurancePolicy" r2ml:classID="userv:VehicleInsurancePolicy"/>
        </r2ml:contextArgument>
        <r2ml:DatatypeFunctionTerm r2ml:datatypeFunctionID="op:numeric-add">
          <r2ml:dataArguments>
            <r2ml:AttributeFunctionTerm r2ml:attributeID="userv:VehicleInsurancePolicy.premium">
              <r2ml:contextArgument>
                <r2ml:ObjectVariable r2ml:name="vehicleInsurancePolicy" r2ml:classID="userv:VehicleInsurancePolicy"/>
              </r2ml:contextArgument>
            </r2ml:AttributeFunctionTerm>
            <r2ml:TypedLiteral r2ml:lexicalValue="300" r2ml:datatypeID="xs:integer"/>
          </r2ml:dataArguments>
        </r2ml:DatatypeFunctionTerm>
      </r2ml:AssignActionExpression>
    </r2ml:producedAction>
  </r2ml:ProductionRule>
```

The reader may check our rules browser for more examples.

2.3.2 Negotiating eCommerce Transactions Through Disclosure of Buyer and Seller Policies and Preferences

This use case concerns the ability of parties involved in formal transactions or procedures, e.g., *credit card authorization of a purchase*, *access of private medical records*, etc., to express and protect their interests within a policy-governed framework. The goal is to formally encode the *preferences*, *priorities*, *responses*, etc., of the parties in such a way that the overall policy can work as intended while providing opportunity for automatic negotiation of terms when allowed by the policy.

While RIF does not respond yet to this use case, R2ML provides Integrity Rules as a representation for policy constraints.

An integrity deontic rule extracted from UServ Product Derby 2005 Use Case: `A preferred client must have a portfolio that includes at least three products (for example, a preferred client may have a portfolio that includes vehicle and life insurance policies and an individual retirement account).` is represented in R2ML as:

```
<r2ml:DeonticIntegrityRule r2ml:ruleID="CC_01">
```

```
  <r2ml:Documentation>
   <r2ml:RuleText r2ml:textFormat="plain">
    <![CDATA[
      A preferred client must have a portfolio that includes
      at least three products (for example, a preferred client
      may have a portfolio that includes vehicle and life
      insurance policies and an individual retirement account).
    ]]>
   </r2ml:RuleText>
  </r2ml:Documentation>
  <r2ml:constraint>
   <r2ml:UniversallyQuantifiedFormula>
    <r2ml:ObjectVariable r2ml:name="client" r2ml:class="userv:Client"/>
    <r2ml:Implication>
     <r2ml:antecedent>
      <r2ml:Conjunction>
       <r2ml:ReferencePropertyAtom r2ml:referenceProperty="userv:hasPortfolio">
        <r2ml:subject>
         <r2ml:ObjectVariable r2ml:name="client" r2ml:class="userv:Client"/>
        </r2ml:subject>
        <r2ml:object>
         <r2ml:ObjectVariable r2ml:name="portfolio" r2ml:class="userv:Portfolio"/>
        </r2ml:object>
       </r2ml:ReferencePropertyAtom>
       <r2ml:AtLeastQuantifiedFormula r2ml:minCardinality="3">
        <r2ml:ObjectVariable r2ml:name="product" r2ml:class="userv:Product"/>
        <r2ml:ReferencePropertyAtom r2ml:referenceProperty="userv:hasProduct">
         <r2ml:subject>
          <r2ml:ObjectVariable r2ml:name="portfolio" r2ml:class="userv:Portfolio"/>
         </r2ml:subject>
         <r2ml:object>
          <r2ml:ObjectVariable r2ml:name="product"/>
         </r2ml:object>
        </r2ml:ReferencePropertyAtom>
       </r2ml:AtLeastQuantifiedFormula>
      </r2ml:Conjunction>
     </r2ml:antecedent>
     <r2ml:consequent>
      <r2ml:ObjectClassificationAtom r2ml:class="userv:PreferredClient">
       <r2ml:ObjectVariable r2ml:name="client"/>
      </r2ml:ObjectClassificationAtom>
     </r2ml:consequent>
    </r2ml:Implication>
   </r2ml:UniversallyQuantifiedFormula>
  </r2ml:constraint>
</r2ml:DeonticIntegrityRule>
```

Other similar RIF use cases supported by R2ML are: **Collaborative Policy Development for Dynamic Spectrum Access**, **Access to Business Rules of Supply Chain Partners** and **Managing Inter-Organizational Business Policies and Practices**. The use case **Ruleset Integration for Medical Decision Support** is not supported but this one seems to don't be supported by actual RIF specification too.

2.3.3 Interchanging Rule Extensions to OWL

2.3.4 Vocabulary Mapping for Data Integration

R2ML provides an internal vocabulary markup but it is not its main goal to interoperate between vocabularies.

2.3.5 BPEL Orchestration of Rule-Based Web Services

Rule-based Web services depend on the use of XML data for their request and response format. The involved rules must be able to access and compare XML data in their conditions and modify and generate XML data in their actions.

R2ML experiences for modeling of Web Services were published in a number of papers. The main achievements were modeling of Web Services with the help of R2ML Reaction Rules. For example the rule `ON CheckAvailability event (i.e. checkInDate < checkOutDate) IF a room is available then DO send a CheckAvailabilityResponse message.` has the following R2ML representation:

```
<r2ml:ReactionRule r2ml:ruleID="id52461">
  <r2ml:triggeringEventExpr>
   <r2ml:MessageEventExpr r2ml:eventTypeID="ex:CheckAvailability">
   <r2ml:contextArgument>
    <r2ml:ObjectVariable r2ml:name="chkAv" r2ml:classID="ex:CheckAvailability"/>
    </r2ml:contextArgument>
   </r2ml:MessageEventExpr>
  </r2ml:triggeringEventExpr>
  <r2ml:conditions>
   <r2ml:DatatypePredicateAtom r2ml:datatypePredicateID="swrlb:lessThan" r2ml:isNegated="true">
    <r2ml:dataArguments>
     <r2ml:AttributeFunctionTerm r2ml:attributeID="ex:CheckAvailability.checkInDate">
      <r2ml:contextArgument>
       <r2ml:ObjectVariable r2ml:name="chkAv" r2ml:classID="ex:CheckAvailability"/>
      </r2ml:contextArgument>
     </r2ml:AttributeFunctionTerm>
     <r2ml:AttributeFunctionTerm r2ml:attributeID="ex:CheckAvailability.checkOutDate">
      <r2ml:contextArgument>
       <r2ml:ObjectVariable r2ml:name="chkAv" r2ml:classID="ex:CheckAvailability"/>
      </r2ml:contextArgument>
     </r2ml:AttributeFunctionTerm>
    </r2ml:dataArguments>
   </r2ml:DatatypePredicateAtom>
  </r2ml:conditions>
  <r2ml:producedActionExpr>
   <r2ml:MessageEventExpr  r2ml:eventTypeID="ex:InfaultDataError">
    <r2ml:DataSlot r2ml:attributeID="ex:InfaultDataError.message">
     <r2ml:value>
      <r2ml:TypedLiteral
          r2ml:lexicalValue="&lt;ex:InfaultDataError.message
             &gt;wrong input data!&lt;ex:InfaultDataError.message&gt;"
          r2ml:datatypeID="xs:string"/>
     </r2ml:value>
    </r2ml:DataSlot>
   </r2ml:MessageEventExpr>
  </r2ml:producedActionExpr>
  </r2ml:ReactionRule>
```

Then the R2ML markup can be used to generate the WSDL representation of the service. The reader may consult our reaction rules examples for more information.

2.3.6 Publishing Rules for Interlinked Metadata

The Semantic Web includes technologies (e.g., RDF) that allow metadata to be published in machine-readable form. Currently, this information is mostly enumerated as a set of facts. It is often desirable, however, to supplement such facts with rules that capture implicit knowledge. To maximize the usefulness of such published rules, a standard rule format such as RIF is necessary.

A similar use case is already implemented (and reported in our I1-D14 deliverable) by using ERDF rules. It can be tested online. The ERDF XML syntax is going to be embedded in the first R2ML dialect. As a conclusion we must say that R2ML supports a large number from official RIF use cases. These use cases covers the most widely usage of rule nowadays.

2.4 R2ML compatibility with RIF-BLD

The RIF-BLD document (W3C Working Draft 30 October 2007) is the core document intended to specify both the syntax and the semantics of the W3C Rule Interchange Format. Below we will analyze the R2ML compatibility with this document.

2.4.1 Semantics

As we already discuss in this report, R2ML does not provide a specific semantics. Its goal is to be compatible with the RIF semantics. There is no need for a fixed R2ML semantics. It is sufficient to be compatible with the default RIF semantics. R2ML (as well as RIF), as an interchange language must allow interchanging rules from different platforms therefore should accommodate the semantics of the target languages. However, a mapping from the concrete languages semantics to the RIF/R2ML default semantics is desirable to assure the semantic precision of the translations. Unfortunately not all target languages provide a semantics and this makes this mission difficult.

2.4.2 Syntax

While RIF provides an uniform and common markup for both atoms and terms, R2ML keep the distinction and moreover provides a large set of specialized atoms which helps identifying via the markup the main constructs from intended target languages. The actual EBNF grammar of RIF-BLD is:

```
CONDITION      ::= 'And' ' ( ' CONDITION* ' ) ' |
                     'Or' ' ( ' CONDITION* ' ) ' |
                     'Exists' Var+ ' ( ' CONDITION ' ) ' |
                     ATOMIC
  ATOMIC         ::= Uniterm | Equal | CLASSIFICATION | Frame
  Uniterm        ::= Const ' ( ' (TERM* | (Const ' -> ' TERM)*) ' ) '
  Equal          ::= TERM '=' TERM
  CLASSIFICATION ::= TERM ' # ' TERM | TERM ' ## ' TERM
```

RIF Conditions	**R2ML Conditions**
Conjunction	Conjunction
Disjunction	Disjunction
Existentially quantified condition	implicit
Atomic	Atom
n/a	NAF
n/a	NEG

Table 2.2: RIF Condition and R2ML Conditions

```
Frame          ::= (TERM | CLASSIFICATION) ' [ ' (TERM ' -> ' (TERM | Frame))* ' ] '
TERM           ::= Const | Var | Uniterm
Const          ::= LITERAL '^^' SYMSPACE
Var            ::= '?' VARNAME

Ruleset  ::= RULE*
RULE     ::= ' Forall ' Var+ ' ( ' RULE ' ) ' | Implies | ATOMIC
Implies  ::= ATOMIC ' :- ' CONDITION
```

The R2ML model is described using MOF/UML. The reader can consult it at http://oxygen.informatik.tu-cottbus.de/R2ML/0.5/metamodel/R2MLv0.5.htm.

First we have to say that while RIF-BLD seems to describe now just derivation rules, R2ML provides support for integrity constraints, derivation rules, production rules and reaction rules. Therefore a complete compatibility between R2ML and RIF cannot be discussed before specific RIF dialects (such as one for production rules) will be provided. However, just considering derivation rules a comparison can be done as in the below table:

The R2ML conclusion is more general than RIF conclusion. While R2ML supports Literal-Conjunction (i.e a conjunction of atoms and/or negation of atoms) as conclusion, RIF supports just atoms in the rule conclusion.

2.4.2.1 Atoms

The main syntactical difference between R2ML and RIF-BLD yields in the markup of atoms. RIF-BLD provides the following categories of atoms in the rule conditions: `Uniterm`, `Equal`, `CLASSIFICATION` (i.e. *instanceOf* and *subclassOf*) and `Frame` while R2ML provides GenericAtom, ObjectClassificationAtom, ReferencePropertyAtom, AttributionAtom, EqualityAtom, InequalityAtom, ObjectDescriptionAtom, DatatypePredicateAtom, DataClassificationAtom

A sketch of mapping is provided in the below table:

R2ML atoms are much more specialized than RIF atoms and they can be represented either by RIF Uniterm, RIF Frame or by similar concepts. Therefore a translation from R2ML to RIF is straightforward. However, translation *from R2ML to RIF* is performed by loosing information which was previously encoded in R2ML. Therefore an inverse translation *from RIF to R2ML* will not preserve the initial R2ML markup.

From this point of view we can say that RIF concepts are very general and this does not make them suitable to encode all kind of information existent in the intended target languages. However, there is no attempt yet for translators from RIF to specific platform rule languages therefore we cannot say nothing about its capability for interchange. By contrary, R2ML

RIF Atoms	**R2ML Atoms**
Uniterm	GenericAtom
#	ObjectClassificationAtom
##	n/a
Equal	EqualityAtom
n/a	InequalityAtom
Frame	ObjectDescriptionAtom
n/a	ReferencePropertyAtom
n/a	AttributionAtom
builtins?	DatatypePredicateAtom
## ?	DataClassificationAtom

Table 2.3: RIF and R2ML Atoms

provides experimental translators for F-Logic, Jena Rules, JBoss Rules, Jess Rules, RuleML, SWRL and OCL which proves its capability for interchange.

2.5 Conclusion

According with the concept of dialect and extension of Rule Interchange Format we consider R2ML and both dialect and extension of the future RIF. Is a dialect since semantically all R2ML positive atoms can be encoded semantically into RIF atoms and it is an extension since it adds support for two negations, explicit support for ERDF rules, and support for production rules, integrity rules and reaction rules.

Chapter 3

R2ML Extensions for Accommodating Production Rules

In this chapter we discuss two features of production rule systems, which are not yet supported in R2ML. The first issue is priorities, which are supported by many production rules systems and are used to control the production rules execution order. Another issue is opaque expressions, which are allowed in many production rule systems, for instance JBoss Rules. We discuss how R2ML can be extended in order to support priorities and opaque expressions.

3.1 Production Rules in R2ML

The support of production rules in R2ML is implemented following the principles, stated in the Production Rules Representation (PRR) proposal by OMG [OMG07]. The goal of the PRR is to specify set of requirements to the rule language for production rules. However, there are a lot of production rules systems already available and now all of them strictly follow the OMG proposal. The main incompatibility issue between PRR (and consequently R2ML) and existing rule languages for production rules is the support of opaque expressions by the latter. Following PRR proposal, R2ML action part of a rule may contain a sequence of *actions*. An R2ML action is a sequence of *UpdateStateActionExpression*s. Each *UpdateStateActionExpression* is either an *UpdateActionExpression*, *AssertActionExpression* and *RetractActionExpression*.

3.2 Priorities in R2ML

Priorities in production rules are used to control the execution order of rules. This is one of the possible conflict resolution strategies, implemented for helping the production rule system to decide which rule should be taken from the rules agenda and executed. Priority is a number, assigned to a rule. The rule with a highest priority is executed first. One of the advantage of rule based systems is its declarative nature: the business rules are declared and then the system executes them in order to achieve the goal; the rule modeler does not have to care about algorithmical aspect, he just must know the rules of the business. However, adding priorities to rules forces the rule modeler to change his mind from the declarative way of thinking into procedural, where the order of rules makes sense. The "ideal" rule base should perform properly without forcing the modeler to think about rules priority. Therefore, the good practice guideline is to avoid the use of priorities. R2ML is a general rule language and does not count the specific semantics of a production rule system (even if the formal semantics can be defined for some purposes, for instance, for the purpose of rule interchange). The initial R2ML design goal was not to include platform-specific mechanisms into the language. The practical evaluations have shown that the real rules development can hardly be done without priorities, mainly because some rules become too large and verbose and priorities may help to avoid it. Another reason why it is difficult to avoid priorities is syntactic restrictions of some rule languages, for instance, absence of negation or inability to define a constraint (example?).

Therefore, the support for priorities may simplify some rule modeling and interchange tasks. One of the possible implementation solutions for priorities is by means of annotation properties. Since the main R2ML syntax is XML syntax, for which the XML Schema is defined, the schema can be extended in order to allow special kind of annotation property *priority*.

Examples:

3.3 Opaque expressions in R2ML

As we have already mentioned, R2ML design principles follow the PRR submission of OMG. In particular, the action part of an R2ML production rule consists of a sequence of atomic actions. However, existing rule systems like JBoss allow opaque expressions in rule action parts, for instance, action parts may contain Java code or other code in a programming language. Moreover, opaque expressions can be mixed with standard action expressions. The presence of opaque expressions in rules complicates the rule interchange since it is difficult to translate, for instance, arbitrary Java code into an interchange format and then to the target rule language. Since the work on Rule Interchange Format (RIF) [rif] is still in progress as well as the work on PRR, it is not yet finally clear how the problem of opaque expressions interchange will be finally solved on the standardization level. At the same time, there is a trend in the development of production rule systems to move towards the PRR and RIF. It means that most likely the use of opaque expressions in rules will be minimized or prohibited at all. Our work on rule-based case study implementation [WGL$^+$07] has shown that there is no real need for opaque expressions and standard actions, i.e. which consists of predefined action expressions, are enough for successful implementation of rule-based applications. The JBoss documentation also recommends to keep action parts of rules declarative and do not use imperative or conditional code in the action part. Usually opaque expressions in JBoss are used to do logging or printing out debugging information. Technically, representation of opaque expressions in R2ML can be implemented by means of annotation properties, defined for each action expression in the rule head. Let us consider an example of a JBoss rule with the head, which contains Java code.

Example 3.1 (JBoss rule with opaque expression in the head)
```
rule "Platinum Priority"
 when
  customer : Customer( subscription == "Platinum" )
  ticket : Ticket( customer == customer, status == "New" )
 then;
```

```
  System.out.println("Current status: "+ticket.getStatus());
  ticket.setStatus( "Escalate" );
  update( ticket );
end
```

In this example, the opaque expression is `System.out.println()`*, which is used to print out a text string to the standard output. The action part of this rule in R2ML is as following:*

```
<r2ml:UpdateActionExpression r2ml:property="userv:status">
  <r2ml:opaque>
    System.out.println("Current status: "+ticket.getStatus());
  <r2ml:opaque>
  <r2ml:contextArgument>
   <r2ml:ObjectVariable r2ml:name="ticket" r2ml:class="userv:Ticket"/>
  </r2ml:contextArgument>
  <r2ml:TypedLiteral r2ml:lexicalValue="Escalate" r2ml:datatype="xs:string"/>
</r2ml:UpdateActionExpression>
```

In this example, the setter `ticket.setStatus(` Escalate");", preceding the update action `update( ticket );` is merged into one UpdateActionExpression in R2ML. The line, which prints out the message about the current status of the ticket is considered as an opaque expression and put as annotation property of the first standard action next to it (`update(ticket)` action).

Bibliography

[BDG$^+$05] Harold Boley, Mike Dean, Benjamin Grosof, Michael Kifer, Said Tabet, and Gerd Wagner. Ruleml position statement. In *Proceedings of W3C Workshop on Rule Languages for Interoperability*, Washington, DC, USA, 27-28 April 2005. W3C.

[BFJ$^+$04] Jeffrey M. Bradshaw, Paul J. Feltovich, Hyuckchul Jung, Shriniwas Kulkarni, William Taysom, and Andrzej Uszok. Dimensions of adjustable autonomy and mixed-initiative interaction. pages 17–39. 2004.

[Bon05] P. A. Bonatti. Policy language specification. Technical report, Munchen, Germany, February 2005.

[BS00] Piero Bonatti and Pierangela Samarati. Regulating service access and information release on the web. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 134–143, New York, NY, USA, 2000. ACM.

[BTW01] Harold Boley, Said Tabet, and Gerd Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Proc. Semantic Web Working Symposium (SWWS'01)*. Stanford University, July/August 2001.

[GNO$^+$04] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *Proceedings of the 1st European Semantic Web Symposium*, Heraklion, Greece, 2004.

[GW06] Adrian Giurca and Gerd Wagner. Translating r2ml into f-logic, white paper, 2006.

[Kag04] Lalana Kagal. *A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments.* PhD thesis, University of Maryland Baltimore County, Baltimore MD 21250, September 2004.

[Kag06] Lalana Kagal. Rei ontology specification, version 2.0, 2006.

[Kav07] N. Kaviani. Xslt transformations for policies interchange, 2007.

[KFJ03] Lalana Kagal, Tim Finin, and Anupam Joshi. A policy language for a pervasive computing environment. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, page 63, Washington, DC, USA, 2003. IEEE Computer Society.

[KGHW07] Nima Kaviani, Dragan Gasevic, Marek Hatala, and Gerd Wagner. Web rule languages to carry policies. In *Proceedings of the 8th IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 188–192, Bologna, Italy, 2007.

[KMLT06] Lalana Kagal, Rebecca Montanari, Ora Lassila, and Alessandra Toninelli. A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In *Proc. of 5th International Semantic Web Conference, Athens, GA, USA, November 5-9, 2006, LNCS 4273*, 2006.

[KPS$^+$04] Lalana Kagal, Massimo Paoucci, Naveen Srinivasan, Grit Denker, Tim Finin, and Katia Sycara. Authorization and privacy for semantic web services. *IEEE Intelligent Systems (Special Issue on Semantic Web Services)*, 19(4):50–56, July 2004.

[LW06a] Sergey Lukichev and Gerd Wagner. Uml-based rule modeling with fujaba. In Holger Giese and Bernhard Westfechtel, editors, *Proceedings of the 4th International Fujaba Days 2006, Universityof Bayreuth, Germany*, pages 31–35, 28-30 September 2006.

[LW06b] Sergey Lukichev and Gerd Wagner. Verbalization of the rewerse i1 rule markup language. Technical report, September 2006.

[MGG$^+$06] Milan Milanovic, Dragan Gasevic, Adrian Giurca, Gerd Wagner, and Vladan Devedzic. On interchanging between owl/swrl and uml/ocl. In *Proceedings of the OCLApps Workshop*, pages 81–95, Genova, Italy, 2 October 2006.

[NOW04] W. Nejdl, D. Olmedilla, and M. Winslett. Peertrust: automated trust negotiation for peers on the semantic web. pages 118–132, 2004.

[OLPL05] Daniel Olmedilla, Ruben Lara, Axel Polleres, and Holger Lausen. Trust negotiation for semantic web services. In *Semantic Web Services and Web Process Composition*, pages 81–95. Springer Berlin / Heidelberg, 2005.

[OMG07] OMG. Production rule representation (prr), beta 1. Technical report, November 2007.

[owl] OWL-S: Semantic Markup for Web Services. W3C Member Submission 22 November 2004. http://www.w3.org/Submission/OWL-S.

[Pol] The policy ruleml technical group.

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

[swr] SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004. http://www.w3.org/Submission/SWRL/.

[TBKM05] Alessandra Toninelli, Jeffrey Bradshaw, Lalana Kagal, and Rebecca Montanari. Rule-based and ontology-based policies: Toward a hybrid approach to control agents in pervasive environments. In *Proceedings of the Semantic Web and Policy Workshop*, November 2005.

[UBJ+03] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. *policy*, 00:93, 2003.

[WCJS97] Marianne Winslett, Neil Ching, Vicki Jones, and Igor Slepchin. Assuring security and privacy for digital library transactions on the web: client and server security policies. In *IEEE ADL '97: Proceedings of the IEEE international forum on Research and technology advances in digital libraries*, pages 140–151, Washington, DC, USA, 1997. IEEE Computer Society.

[WGL05] Gerd Wagner, Adrian Giurca, and Sergey Lukichev. R2ML: A General Approach for Marking up Rules. In F. Bry, F. Fages, M. Marchiori, and H. Ohlbach, editors, *Dagstuhl Seminar Proceedings 05371*, Principles and Practices of Semantic Web Reasoning, 2005.

[WGL06a] Gerd Wagner, Adrian Giurca, and Sergey Lukichev. A usable interchange format for rich syntax rules integrating ocl, ruleml and swrl. In *Proceedings of Reasoning on the Web*, Edinburgh, Scotland, 22 May 2006.

[WGL+06b] Gerd Wagner, Adrian Giurca, Sergey Lukichev, Grigoris Antoniou, and Mikael Berndtsson. Strelka - a visual rule modeling tool. Technical report, Munchen, Germany, April 2006.

[WGL+06c] Gerd Wagner, Adrian Giurca, Sergey Lukichev, Grigoris Antoniou, Carlos Viegas Damasio, and Norbert E. Fuchs. Language improvements and extensions. Technical report, Munchen, Germany, April 2006.

[WGL+07] Gerd Wagner, Adrian Giurca, Sergey Lukichev, Oana Nicolae, and Mircea Diaconescu. Case study 1: Userv product derby 2005. Technical report, March 2007.

[wsd] Web service semantics - wsdl-s. W3C Member Submission 7 November 2005. http://www.w3.org/Submission/WSDL-S/.

[wsm] Wsmo web service modeling ontology (wsmo). W3C Member Submission 2005. www.w3.org/Submission/WSMO.