Interoperable Access Control Policies: A XACML and RIF Demonstration

Fatih Turkmen[1,2], Lalana Kagal[2], and Bruno Crispo[1]

[1] University of Trento, Italy
[2] CSAIL, MIT, USA

Abstract. eXtensible Access Control Markup Language (XACML), an OASIS standard language for the specification of access control rules, has been widely deployed in many Web-based systems. However, many domains still use their custom solutions to manage authorizations. This makes collaboration between and integration over applications and domains using disparate policy language difficult and requires prior negotiation and agreement between them. Rule Interchange Format (RIF) is an interlingua being developed at W3C to allow the exchange of rules between rule systems. We propose to express XACML as RIF in order to enable XACML policy rules to be understood by any RIF based system. In this paper, we present the design of our translator from/to XACML to/from RIF by mapping XACML constructs to RIF. Our translator will enable the exchange of RIF encoded XACML rules among different policy systems.

1 Introduction

Service compositions especially in cross-enterprise collaborations bring many costs along with the benefits. The parties in such a cooperation aim to keep the additional cost of cooperative operation at minimum. Among other discrepancies security systems produce many inconsistencies [1–3]. These include the operational issues such as the interfaces and standards employed in daily execution of business processes and operations [1]. One such a barrier is the representation of authorization policies for company assets (e.g. services, hardware resources, databases) in one enterprise being incompatible with the other/s.

For access control (i.e. authorization) specification many languages have been proposed and employed. XACML [2], an OASIS standard, is the predominant language for the specification of access control policies. It has found a wide usage and deployment in both academia and industry because of its simplicity and expressivity. However, there are many organizations employing different languages in their authorization systems. Some of these languages include Ponder2 [4], Rei [5], PRIME [6], AIR [7], although a comprehensive list can be found in [8]. As it is, in many cases, very difficult to change the authorization systems, making

[1] http://searchsecurity.techtarget.com/tip/0,289483,sid14_gci1260582,00.html
[2] http://www.oasis-open.org/committees/xacml/

them interoperable might serve as a cost-effective solution for collaborative businesses. For interoperability of access control policies, there is a need of a means that will transit an access control rule from one representation to the other.

RIF promises the exchange of rules in a standard but more importantly semantically preserving way. RIF Working Group aims to reach a common set of rule features that will support majority of rule systems.

In this paper, we present our attempt to make XACML interoperable with other access control languages having different semantic models. Specifically, we elucidate the details of our RIF translator that gets a XACML policy as input and produces the corresponding RIF-PRD representation. XACML specification has been written in an informal language so that

2 Paper Contribution and Organization

The contributions of our paper can be summarized as the followings: By translating XACML into RIF, we enable the interoperability of XACML based systems with non-standard custom solutions. Translation of a policy encoded in a standard representation dedicated for interoperability , RIF-PRD, will enable the analysis of a policy very easy by further translating it into another form. By using our translator a policy encoded in RIF-PRD will be translated into XACML, a widely used standard access control language. Our analysis show useful information about the translation process of a rule language to RIF by eliciting the steps and constraints of RIF.

We believe that our translation work between XACML and RIF will lead to a new research direction dedicated to rule exchange and interoperability in access control. This way we illustrate how wide spectrum regulatory XACML policies can be exchanged between custom systems. Our very first motivation is summarized in Figure 1 in which RIF acts a bridge between different policy languages.

3 eXtensible Access Control Language (XACML)

XACML is an XML-based access control language endorsed by OASIS. It has been widely deployed in many Web-based systems and there are many implementations supporting its use. The third version of XACML is expected to be announced at the end of this year with some extensions modifications according to received feedbacks.

Syntactically, an XACML Policy is composed of following top level elements: PolicySet, Policy, Combining Algorithms, Target, Rules and Obligations. A PolicySet composed of Policies or other PolicySets is the topmost element of an XACML policy. It wraps all Policies together with a Policy Combining Algorithm that resolves conflicts among policy decisions. Policy Combining Algorithms include Permit-overrides, Deny-Overrides, First-applicable, Ordered-permit-overrides, Ordered-deny-overrides and Only-one-applicable. A Policy is a

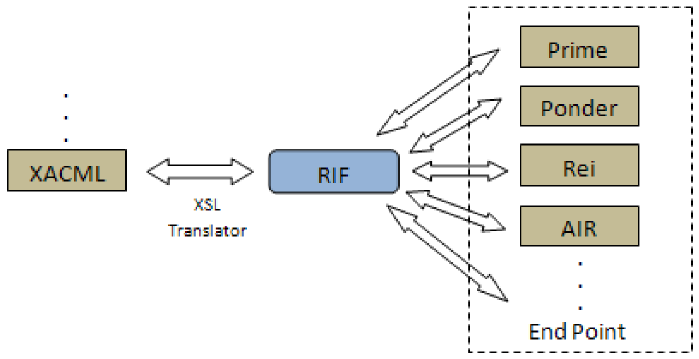

Fig. 1. RIF Use Case

container for XACML Rules that are atomic decision elements on the given Requests. The decisions made by each individual rule is combined with a Rule Combining algorithm. Rule Combining Algorithms are the with Policy-Combining Algorithms except Only-one-applicable which is not suported.

Policy Target is used to seek the applicability of a policy for the given request. It includes the quadruple *Subjects/Resources/Actions/Environments* with different number of *Subject*, *Resource*, *Action* and *Environment* (abbreviated as S/R/A/E what follows) elements inside. Each *(S/R/A/E)* contains one or more *(S/R/A/E)* Match elements (e.g. SubjectMatch, ResourceMatch) defining relevant attributes of the element they are contained. The following code snippet shows a small fraction of an XACML Match element.

```
<SubjectMatch MatchId="rfc822Name-match">
<AttributeValue"> med.example.com </AttributeValue>
<SubjectAttributeDesignator AttributeId="subj-id" DataType="rfc822Name"/>
</SubjectMatch>
```

Rules are cores of XACML Policies. A Policy can contain different number of *Rules* combined with a *Combining Algorithm.* A rule specifies a logical expression stating the conditions a request must satisfy for the rule to be applicable (e.g. a request issued between 8am and 8pm is applicable) and a decision in case the conditions are satisfied. It constitutes of a *Target* similar to Policy Target and a *Condition.* Conditions describe further constraints on S/R/A/E elements by additional attribute value requirements. A rule has an *Effect* of either Permit or Deny if it is applicable for a given Request.

3.1 Access Decisions in XACML

The component making access control decisions is called Policy Decision Point (PDP) in XACML. Figure 3 illustrates the evaluation structure of XACML

```
PolicySet ::= (PolicySet)* PolicyCombiningAlgId  Target  (Policy)* Obligations?
PolicyCombiningAlgId ::=  (“deny-overrides” | “permit-overrides” |
            “first-applicable” | “only-one-applicable” | “ordered-deny-overrides”
            | “ordered-permit-overrides”)
Target ::= Subjects?  Resources?  Actions?  Environments?
Policy ::= RuleCombiningAlgId  Target  ( … | (Rule)+)  Obligations?
RuleCombiningAlgId ::= URI (“deny-overrides” | “permit-overrides” |
            “first-applicable” | “ordered-deny-overrides”
            | “ordered-permit-overrides”)
Rule ::= RuleId  (“Permit” | “Deny”)  Description?  Target?  Condition?
Obligation ::= ObligationId  (“Permit” | “Deny”)  (AttributeAssignment)*
Subject ::= (AttributeValue (SubjectAttributeDesignator | AttributeSelector))+
Resource ::= (AttributeValue (ResourceAttributeDesignator | AttributeSelector))+
Action ::= (AttributeValue (ActionAttributeDesignator | AttributeSelector))+
Environment ::= (AttributeValue (EnvironmentAttributeDesignator
            | AttributeSelector))+
Expression ::= Apply | AttributeSelector | AttributeValue | Function
            | VariableReference | (Expression | AttributeDesignatorType)
            | (Expression | AttributeDesignatorType) | (Expression  |
            (AttributeDesignatorType SubjectCategory?)) | (Expression |
            AttributeDesignatorType)
Condition ::= Expression
AttributeAssignment ::= AttributeId AttributeValue
AttributeValue ::= Expression | (Any)* DataType
Apply ::= FunctionId (Expression)*
AttributeSelector ::= RequestContextPath  DataType MustBePresent?
```

Fig. 2. EBNF of XACML Policies

Policies/PolicySets. When an XACML Request arrives, PDP is provided relevant attribute values and follows the below steps for a decision: 1) Applicability check of the top level element (e.g. PolicySet, Policy) through Target - Request matching, 2) If top level applicability succeeds, Rule applicability is checked via Rule Target - Request matching, 3) According to given Combining Algorithm, the applicable Rule Effect is propagated to Policy level, 4) If the top level element is a PolicySet then the Policy decision is propagated to the PolicySet level according to Policy Combining Algorithm.

In XACML, applicability check through Target and Request matching is done through equality of ids (i.e. value identifiers with AttributeSelector and AttributeDesignator) and the values given in the Target section and in the Request. The matching process results in three states: "Match", "No-match", "Indeterminate". If a Match state is caught then Rule Effect (i.e. either Permit or Deny decision) is propagated to Policy level. Apart from two conventional cases, XACML provides two informative results from a Rule, Policy or PolicySet evaluation: NotApplicable and Indeterminate. "NotApplicable" is returned when the Target-Request matching falls into "No-match" state or a Rule Condition is not satisfied. The result ”Indeterminate” is given in the existence of several applicable policies with Only-one-applicable Combining Algorithm or an error during the evaluation.

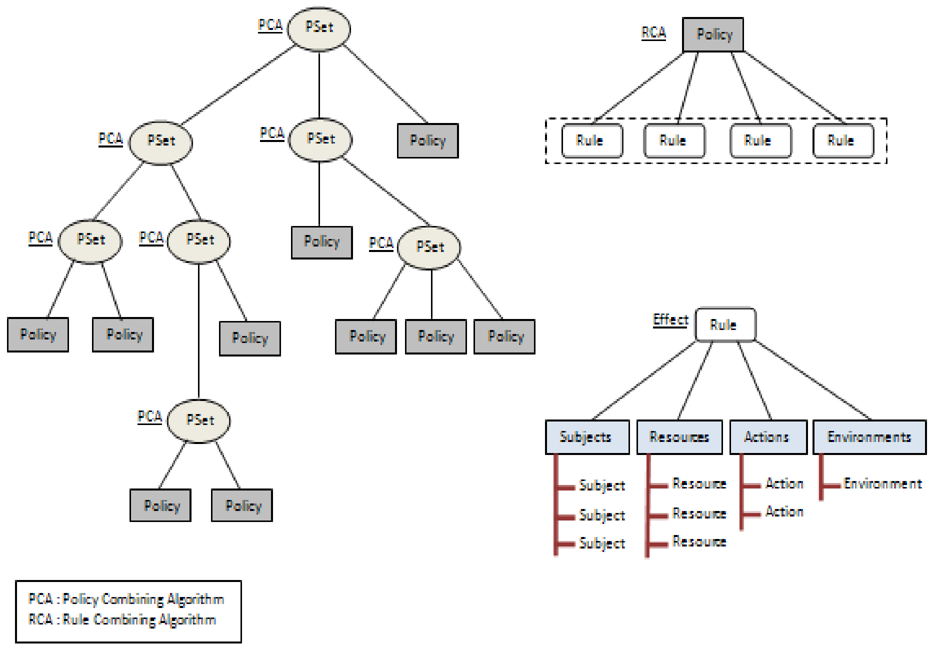

Fig. 3. XACML Evaluation Structure

4 Rule Interchange Format (RIF)

RIF is a World Wide Web Consortium (W3C) standard for the exchange of rules [3]. RIF Working Group envisions a generic platform for rule exchange among different parties in a semantically preserving way. RIF currently has three dialects: Basic Logic Dialect (referred as BLD), Production Rules Dialect (PRD), RIF-Core Dialect. BLD is a logic based (i.e. Horn rules) dialect with equality and a standard first-order semantics. It has extensions to support F-Logic and XML-Schema datatypes. PRD is used for the representation of production rules forming the structure $\{condition \rightarrow action\}$ where conditions are the antecedents of *if* clauses and actions are *then* clauses mainly used for fact related operations (e.g. assertion, modification and retraction) on the knowledge base (KB). RIF-Core is a common subset of BLD and PRD dialects. In addition to these three dialects, RIF specification provides a framework (i.e. Framework of

[3] http://www.w3.org/2005/rules/wiki/RIF_Working_Group

Logic Dialects) for defining new logic based dialects. RIF allows the definition of new dialects by restricting or extending some features of the framework. The specification also provides an integration and compatibility model between RIF and RDF/OWL making RIF Web-aware.

One of the early RIF papers, [9] demonstrates several use cases of RIF with some feature analysis. It showcases BLD usage in different platforms and lists some of the features not available such as negation and aggregation. In addition to those listed items, we have found that RIF in general doesn't have a mechanism for the interchange of metadata for the rules exchanged. We believe that RIF can be a strong standard with the support of a component/mechanism for the transport of metadata as some languages have extensions for the specification of metadata inside the rule language itself (e.g. XACML).

During the writing of the paper, RIF Working Group was about to officially announce RIF as a W3C recommendation (circa September 2009). In our translation we tried to stick RIF-Core constructs as it is the minimal rule interchange dialect and supported by the majority of rule platforms with the least amount of effort. However, RIF-Core and RIF-BLD do not support negation. Because of negation feature, our translation falls into PRD. For that reason we followed PRD syntax in our demonstrations of translation in the paper although any feature apart from negation can be easily represented in BLD. What follows in the next section is a brief discussion of RIF-Core dialect and its relation to PRD and BLD.

4.1 Common Subset of PRD and BLD: RIF-Core

PRD enables the exchange of production rules. Production rules form an "if" *Condition* "then" *Action* structure where Condition represents logical expressions for the Actions to be performed. PRD provides 5 atomic actions: *Assert* to assert new knowledge (frame or fact) to the base, *Retract* to cancel knowledge (frame or fact) from the base, *Retract Object* to cancel all facts of a given frame object, *Modify* to replace all the values of an object and *Execute* to invoke an externally defined action. Differently from logic-based dialects where the operational semantics are left to rule developer, PRD provides a default operational semantics along with conflict resolution techniques for rule invocation.

BLD has a similar syntax to Datalog with a first-order semantics. It has several extensions to support additional features such as International Resource Identifiers (IRI) and frames of F-Logic [10]. BLD specification describes two languages used to serialize BLD expressions in XML: Condition language and Rule Language. Condition language used to specify premises of BLD rules while Rule language is used to represent the production of the rule at a high level.

RIF-Core is the minimum set of syntax and semantics required for rule interchange by RIF. It is a common subset of BLD and PRD. Syntactically, RIF-Core is subset of BLD such that it has Datalog-like rules and further extensions for frames of F-Logic and IRIs. Alike with the rest of RIF dialects, RIF-Core specification provides two syntaxes: presentation syntax to describe the concepts by combining mathematical symbols with English sentences and XML syntax

as the serialization of the given EBNF. Apart from non-safe rules (using some built-in functions) all RIF-Core rules can be converted to BLD and PRD. As RIF-Core is a subset of BLD, the translation from/to RIF-Core from/to BLD is very easy. PRD specification also provides a methodology for the conversion of PRD documents into RIF-Core as long as the document complies with the following conditions:
Condition formulas do not contain negation: Negation is not available both in RIF-Core and BLD.
Rule action blocks do not contain action variable declarations: No frame object declarations in the form ?Var1(?Var2 Func(?Var3)) and frame slot values *(?value o[?s → ?value])*. The simple method for converting RIF-Core documents into PRD is to assert RIF-Core conclusions as facts to the knowledge base. For conversion from PRD to RIF-Core the following symbols must be eliminated from PRD document: *##*, *such that*, *Not*, *INeg*, *Do*, *Assert*,*Retract*, *Modify*, *Execute*, and *New*.

5 Mapping Analysis

The very first issue we had to tackle was deciding which dialect to use for the translation. In brief, we tried to answer whether XACML is a production system or a logic-based system in representation. RIF-Core, as a subset of different dialects of RIF, proves that there are many common features between logic based dialects and production systems. However the main difference between logic dialects and production rule dialects lies in the operational semantics of the language represented in RIF. PRD provides a set of operational semantics together with conflict resolution strategies. However, rich set of features offered by PRD makes it difficult to be mapped to many of the existing systems.

However, BLD and RIF-Core do not support negation and user-defined functions. Because of variable semantics on many platforms, negation has been kept out of BLD. For that reason, we decided to use PRD with a minimal set of features that will capture the semantics of XACML. In this way, we are convinced that our mapping can be easily specified in RIF-Core if negation is omitted. Besides, we observed that characteristically XACML's operational semantics have a close similarity to that of PRD. For example, XACML does not yield only Permit or Deny but also gives two informative results: NotApplicable and Indeterminate. Besides, environmental changes injected through parameters can affect the access decisions. Especially when considering XACML used to specify a Usage Control policy [11] a forward chaining approach is required.

To verify the semantical correctness of our mapping, we developed a proof-of-concept implementation in SWI-Prolog [4] available at [5].

5.1 Requirements

Apart from which RIF dialect to use, our analysis revealed two important issues:

[4] http://www.swi-prolog.org/
[5] http://dit.unitn.it/~turkmen

- XACML concepts to be addressed in the mapping are: PolicySet, Policy, Match, Target, Rule, Combining Algorithm, Condition and Obligation. We have kept Obligations out of scope for our initial work.
- There are different ways of representing XACML rules and policies in RIF during a rule exchange. We call our representation as Policy or PolicySet wise representation in which rules and policies are encoded as a whole. Each section of a policy and a rule accompanied with the corresponding values in the form of facts is mapped to RIF.
- The knowledge in KB must have a logical grouping to reflect the associations between entities. For example, a Match must be associated with a Target of a Rule, or a Rule must be associated with a Policy or finally a Policy must be associated with a PolicySet. A XACML entity relationship can be summarized as follows: $Rules \rightarrow Policies \rightarrow PolicySets \rightarrow Policies \rightarrow Rules$
- We assume that there are no external Policy/PolicySet references in an XACML PolicySet or they have been resolved prior to translation. Alternatively, they can be converted into OWL ontologies as described in [12]. After converting XACML policies into OWL ontologies, they can be referenced by using RIF Web Ontology Language(OWL)/Resource Description Framework (RDF) compatability. RIF-OWL/RDF compatibility allows external vocabulary references from a RIF document.

The motivations behind our way of representation are the incorporated semantics of RIF and characteristics of XACML. The former required us to capture Ruleset concept of rule based systems. The latter motivation allowed us doing easy association between a rule as a micro entity and a policy as a macro entity. At a higher scale, we could associate a Policy as a micro entity to a PolicySet as a macro entity without adding further semantics during the translation.

6 XACML2RIF

After obtaining the requirements listed in Section 5.1, we followed an approach very similar to the presentation of XACML elements in XACML v2.0 specification for mapping. At the very core, we have a Target element acting as a precondition for Rules, Policies and PolicySets. Further we have Rules combined with a given Rule Combining Algorithm in case they are applicable. At the very top level we have PolicySets where all decisions brought up from Policies are merged with a Policy Combining Algorithm. Apart from Only-one-applicable Policy Combining Algorithm, the semantics of Combining Algorithms are the same for Policies and Rules. For that reason, we present only Rule Combining Algorithms from which Policy Combining Algorithms can be easily obtained by changing Rule entity to Policy.

6.1 Target Evaluation

The single entity we use for matching is a Match (i.e. SubjectMatch, ResourceMatch, ActionMatch and EnvironmentMatch) element. In the knowledge base, a

Match element is represented as *_Match(PolicyNumber, RuleNumber, _MatchNumber, AttributeID, AttributeValue)* where; PolicyNumber and RuleNumber identifies the Policy and Rule respectively that _Match element belongs to; _MatchNumber represents the set of Match elements defining a single S/R/A/E element (any of them) and finally AttributeID and AttributeValue represent the actual attribute id/value pairs to be used in matching. In addition to these Match elements, we have an additional fact added to RIF representation to refer to number of Match elements of each type. The below code snippet represents a single Match element. A Match group formula is available in four types by replacing the underscore with Subject, Resource, Action and Environment. Besides, the negation of above group formula produces "No-Match" case where $(?Rule[_Match(\rightarrow "No-Match"]))$ is asserted to the base.

```
(*Match*)
Forall ?MatchNumber such that(
  If Or( Exists ?RequestAttId ?RequestAttVal ?_Match(
           ?_Match[PolicyNumber->?Policy]
           ?_Match[RuleNumber->?Rule]
           ?_Match[MatchNumber->?MatchNumber]
           ?_Match[AttributeId->?RequestAttId]
                 ?_Match[AttributeValue->?RequestAttVal]
         )
       )
   Then Do (Assert(?Rule[_->"Match"])))

  If Not(?Rule[_->"Match"])
  Then Do (Assert(?Rule[_->"No-Match"])
```

In XACML, Match elements have a MatchId attribute defining the function (e.g. string-equal or integer-less-than) to be used for the comparison between attribute ids and values of the Policy. There are 6 possible functions to be used for matching between attributes. However, we included only equality predicates for demonstration as the rest of functions can be easily addressed via PRD built-in predicates. For the whole Rule Target section we have conjunctions of each individual Target entity (i.e. *(S/R/A/E)*) generated from Match elements. This assures that the XACML Request attributes match at least one set of Match elements composing an S/R/A/E entity as illustrated below.

```
(*RuleTarget*)
  If And( ?Rule[Subject->"Match"]
          ?Rule[Resource->"Match"]
          ?Rule[Action->"Match"]
          ?Rule[Environment->"Match"]
     )
  Then Do (Assert(?Rule[Target->"Match"]))

If Not(?Rule[Target->"Match"])
Then Do (Assert(?Rule[Target->"NotApplicable"]))
```

The final check whether $?Rule[Target \rightarrow "Match"]$ or $?Rule[Target \rightarrow "NotApplicable"]$ is asserted reveals "Indeterminate" case with negation. Accordingly, $?Rule[Target \rightarrow "Indeterminate"]$ is asserted to KB.

In our mapping, we distinguish between Targets of Policy, PolicySet and Rule although they are the same in functionality. The reason to address them separately lies on being able to detect the reason of NotApplicable cases. Addressing them separately enables us understand whether the Policy Target or a specific Rule Target (referred with Rule index) is not matched .

6.2 Condition

XACML Conditions put further constraints on the applicability of a Rule against a matched Request with Target. Conditions present a set of (AttributeId, AttributeValue) associated with any of S/R/A/E elements described in Target section or in the Request. [12] does not fully support XACML built-in functions claiming the impossibility of having a complete and sound reasoning procedure for all of them. In our translation we rely on RIF Data Types and Built-in (RIF-DTB) functions to support XACML functions. The majority of XACML Condition functions can be easily addressed in PRD by using RIF built-ins.

After eliminating Condition specific attributes XACML Condition is equal to an Expression nonterminal:
Expression ::= (FunctionId (Expression)*) | **AttributeSelector** |
AttributeValue | **Function** | **VariableReference** | **(S/R/A/E)Designator**

Attribute Selectors are very similar to AttributeDesignators except they work with XPath expressions. They specify the attributes they require in the form of XPath expressions. VariableReferences can be thought as a wrapper for AttributeSelector and AttributeDesignators. They enable to address them via simple referencing. For that reason, they can be handled in the same way with AttributeSelectors and AttributeDesignators. Except the below code snippet where "string-equal" function is specified Condition matching very similar to Match elements..

```
Forall ?ConNumber(
  If Exists ?RequestAttId ?RequestAttVal(
      Condition[AttributeValue->?value]
      External(pred:string-equal(?value ?RequestAttVal)))
   Then Do (Assert(?Rule[Condition->"Satisfied"])
  ))

If Not(?Rule[Condition->"Satisfied"])
Then Do (Assert(?Rule[Condition->"Not-Satisfied"])
```

6.3 XACML Functions/Predicates

XACML has a really rich set of standard functions and predicates ($\tilde{9}0$) based on XPath functions. Supporting all these functions require a significant amount of effort in a rule based system. However, RIF also has a rich set of functions based on XPath and probably causing rule system developers postpone RIF support in their systems. From XACML to RIF translation point of view, rich function support of RIF allows us only concentrate on the mismatched or unsupported XACML functions. Based on XACML v2.0 specification, Table 1 provides a categorized view of XACML functions and their correspondence in RIF.

Function/Predicate Type	Not Available in RIF
Equality Predicates	X500Name, rfc822Name, hexBinary, base64Binary equality functions
Arithmetic functions	Integer-abs (Absolute Value function)
String conversion functions	String-normalize-space
Numeric data-type conversion functions	-
Logical functions	n-of
Numeric comparison functions	-
Date and time arithmetic functions	-
Non-numeric comparison functions	-
String functions	-
Bag functions	All
Set functions	All
Higher-order bag functions	All
Regular-expression-based functions	All
Special match functions	All
XPath-based functions	All

Table 1. XACML Condition Functions

As Table 1 shows, there are some functions not available in RIF specification. However, our analysis revealed that we can map these functions with the existing functions and extensions.

6.4 Rule Evaluation

Rule evaluation results in four cases depending on the *Effect* given in the rule: Permit, Deny, NotApplicable, Indeterminate. The procedure for handling Deny and Permit is the same. For that reason a Rule evaluation result is obtained from three formulas: *DecisionWithEffect*, *NotApplicableRule* and *IndeterminateRule*. DecisionWithEffect is the conjunction of Target and Condition evaluation, and the check for Rule Effect whether it is "Permit" or "Deny".

```
(*decisionWEffect*)
If Exists ?Effect (
     And( ?Rule[Target->"Match"]
          ?Rule[Condition->"Satisfied"]
          ?Rule[Effect->?Effect]))
Then Do (Assert(?Rule[Decision->?Effect]))
```

NotApplicable formula verifies whether the Rule is applicable for the Request by checking Rule Target and Condition. If Rule Target is "NotApplicable" or it is "Applicable" but Condition is "NotSatisfied" then the Rule yields "NotApplicable". Indeterminate formula covers the rest of the cases where Rule decision is neither Permit or Deny nor NotApplicable.

After addressing Target section of individual Rules and their decisions, we use the following formula to obtain a decision from a Rule for a given Request.

```
(*RuleEvaluation*)
Forall ?Rule (   (*_Match*)    (*RuleTarget*)
 if Exists ?Result(  ?Rule[Decision->?Result]  ))
```

7 Mapping XACML Policies to RIF

XACML Policies are composed of a Policy Target, Rules and a Combining Algorithm to resolve possible conflicts among the Rule decisions. As a Policy decision depends on the given Combining Algorithm, we needed to address each algorithm with a corresponding formula (predicate). However a Policy Target is a prerequisite for the Policy Rule evaluation elicited in the previous section. The RIF mapping of Policy Target is very similar to Rule Target mapping as demonstrated before. When addressing Policy Combining Algorithms it preconditions (i.e. conjunction) the evaluation of Policy Rules. For that reason, we have defined a dedicated predicate for Policy Target that makes use of Rule Target evaluation predicates with an exceptional case: Rule number (i.e. ?Rule) is set to 0. The same idea also applies for a PolicySet Target where Policy number (e.g. ?Policy) is set to 0.

In the following sections, we present how a decision is generated from a Policy by providing relevant predicates for Rule matching and conflict resolution.

7.1 Matching a Rule

Among the standard Combining Algorithm formulas we have a common formula (i.e. *match_rule*) used to find first applicable Rule for a given Request. As the Rules are ordered according to their order in the Policy (through associations), it is straight forward to enforce the ordering of Rules. This extremely eases the translation of *First-applicable* and Ordered-permit/deny-overrides algorithms.

To match a Rule we use a predicate called rule_result that asserts Rule evaluation *Result* for the encoded Request. *rule_result* can be considered as the wrapping of all Rule related formulas presented before. The following formula that propagates the first matched Rule of a Policy.

```
match_rule(Policy,Rule,TotalRuleNumber,Result,WhichRule) :-
             (rule_result(Policy,Rule,Result),
             (Result==deny; Result==permit),WhichRule is Order,!);
             (New is Rule + 1, New =< TotalRuleNumber, WhichRule is New,
             match_rule(Policy,New,TotalRuleNumber,Result,WhichRule)).
```

7.2 Policy/Rule Combining Algorithms

At the Policy level, XACML has 5 Rule Combining Algorithms. After defining Policy Target similar to Rule Target, we explain and address each of Combining Algorithms with the relevant predicates.

First-applicable: According to the order given in the Policy, First-Applicable rule is picked among the rules. The following formula using *match_rule* predicate maps the First-applicable algorithm. Note that, it makes use of *ruleNum* predicate to obtain the number of Rules.

```
first_applicable(P,R,Result) :-  policy_target(P,Match),
                                  Match == match, ruleNum(P,Num),
                                  match_rule(P,R,Num,Result,WhichRule).
```

Permit-overrides: If among the many rules, one applicable rule yields Permit then the result would be Permit. If there is not any Permit but at least one Deny and the rest is NotApplicable then the result would be Deny. In the case of all rules NotApplicable, the result would be NotApplicable. *permit_overrides* predicate relies on another predicate that checks whether Policy Target matches the Request. The same predicate, *not_applicable_or_indeterminate*, allows us to propagate NotApplicable or Indeterminate results of a Policy.

```
permit_overrides(Policy,Rule,Result,Match) :-
       not(not_applicable_or_indeterminate(Policy, Match)),
       ruleNum(Policy, Num),
       match_rule(Policy, Rule, Num, Result, WhichRule),
       (Result==permit,!;
       New is Rule + 1, permit_overrides(Policy,New,ResultNew)).
```

Deny-overrides: It is very similar to Permit-overrides algorithm except the overriding decision is "Deny". The corresponding predicate can be obtained by changing Result values to "Deny".

Ordered-(permit/deny)-overrides: Because of our representation of Match elements we did not do to spend any special efforts to address Ordered algorithms. As we associate a Match element *subjectMatch(PolicyNumber, RuleNumber, MatchNumber, AttributeId, AttributeValue)* to a Policy and a Rule with a numbering system, the ordering is enforced implicitly in the predicates.

A Policy is represented as a disjunction of Combining Algorithms. PolicySets include Policy Combining algorithms very similar to Rule Combining algorithms with an additional algorithm Only-one-applicable. Our current mapping does not support Only-one-applicable algorithm. The rest of the algorithms can be considered identical to their Rule counterparts by referring to Policies instead of Rules in the formulas.

8 Related Work

There has been many research addressing XACML. The more related work to ours lies in the formalization of XACML policies for security property and policy change-impact analysis [12–16]. Among those, [12] has done the most extensive formalization in Datalog to provide formal semantics and enable security analysis (e.g. feature analysis) by using off-the-shelf DL reasoners. However, only a subset of XACML is supported in author's formalization and his formalization relies on Datalog stratification to prevent ambiguity and policy conflicts. This causes some of the XACML semantics to be obscured by the implementation language functionality. The main difference between our approach is that we focus on making XACML semantics explicit such that they can be implemented in any (RIF supported) rule language. Finally, author's presented system implementation is not available for testing.

RIF is expected to be standardized within several months of our writing. Because of the promises given by RIF vision and underlying requirements of such a system require a lot of effort, RIF has recently become stabilized and found some usage. For that reason, RIF related research has been limited [17, 9]. However, rule interchange has been studied to some extent under RuleML umbrella and other business rule systems [18].

The main difference between related previous work and ours is although we have done formalization, we mainly focus on the interoperability of XACML. Safe translation and transition of XACML rules to other systems is the motivation of our paper. As being a W3C recommendation, RIF is an obvious choice to be used as a rule transport means. To extent of our knowledge XACML rule exchange has not been addressed in any research. When mapping XACML to RIF we tried to cover most of the cases such as condition functions and simplified mapping of conflict resolution techniques with smart fact representation.

9 Conclusions and Future Work

We have mapped a subset of XACML to RIF by using minimum set of PRD features to keep the mapping close to RIF-Core in which negation is the main difference. PRD working draft states an effort towards the compatibility between PRD and BLD. As RIF-BLD covers majority of logic based rule languages, the compatibility effort will enable the support of many logic based access control languages in RIF-PRD. Access control requirements play a critical role in collaborative businesses so their exchange as part of contracts/agreements. RIF can act as the main means with an agreed semantics for rule interchange among diverse systems.

We believe that the compatibility studies between RIF dialects play an important role in the future development of RIF. The two main branches, production rules and logic based dialects have many things in common as specified in RIF-Core. However, support of meta data transfer between rule endpoints about the exchanged rulesets can improve RIF usage. Although RIF has not been designed for reasoning on translated rulesets, there is a need of a mechanism or tool that will help verifying the correctness of translation syntax and semantics to some level.

As a future work, we are planning to extend our work to support all Combining Algorithms (e.g. Only-one-applicable and user defined combining algorithms), user defined Condition functions and Obligations. We want to showcase our mapping in a real world scenario where a XACML ruleset is exchanged with a totally different rule system. Lastly, RIF specification mentions about the preservation of semantics during a translation. The basic question tried to be answered is whether the translated Ruleset is semantically equivalent to the original Ruleset. We are planning to do some safety analysis and semantics preservation checks on the translated XACML rules.

References

1. Security and trust management in supply chains, Ramesh Kolluru and Paul H. Meredith, Information Management and Computer Security Journal, 2001.
2. Policy-Based Collaboration: Moving to an Enterprisewide Framework for Working and Communicating with Confidence, 2008, Cisco Systems
3. Access Control for Cross-Organisational Web Service Composition, Michael Menzel and Christian Wolter and Christoph Meinel, Journal of Information Assurance and Security, 2007
4. http://ponder2.net/
5. http://rei.umbc.edu/
6. https://www.prime-project.eu/prime_products/
7. http://dig.csail.mit.edu/TAMI/2008/12/AIR/
8. http://www.w3.org/Policy/pling/wiki/PolicyLangReview#PRIME_Languages
9. Please Pass the Rules: A Rule Interchange Demonstration, Gary Hallmark and Christian de Sainte Marie and Marcos Didonet Del Fabro and Patrick Albert and Adrian Paschke, RuleML, 2008
10. Logical foundations of object-oriented and frame-based languages, Michael Kifer, Georg Lausen, James Wu. Journal of ACM, July 1995, pp. 741–843
11. Towards usage control models: beyond traditional access control, Jaehong Park and Ravi Sandhu, Symposium on Access Control Models and Technologies, 2002, Monterey, California, USA
12. Logic-based Framework For Web Access Control Policies, Vladimir Kolosvki, Phd Thesis, 2008
13. Automated Verification of Access Control Policies Using a SAT Solver, Graham Hughes and Tevfik Bultan, Journal on Software Tools for Technology Transfer (STTT), 2008.
14. The Formal Semantics of XACML, Polar Humenn, 2003
15. Verification and Change-Impact Analysis of Access-Control Policies, Kathi Fisler and Shriram Krishnamurthi and Leo A. Meyerovich and Michael Carl Tschantz, ICSE, 2005
16. Reasoning about XACML policies using CSP, Jery Bryans, Workshop on Secure Web Services, 2005, New York, USA
17. Rule Interchange Format: The Framework, Michael Kifer, Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web, 2008
18. Rule Modeling and Interchange, Proceedings of the Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Adrian Giurca and Gerd Wagner, 2008