A Proposal For

XACML Extension Model and Its Schema

December 24, 2001

Author: Michiharu Kudo

This document proposes an XACML Extension Model that defines what portion of the XACML specification is a core and to what extent the XACML specification can be extended. Based on this proposal, XACML policy administrators can represent much broader access control policies by extending the core portion of the XACML specification. This extension model is designed to support an XACML extensibility property stated in the XACML charter. This proposal is based on the current language proposal document [2] but includes several modifications.

1. Glossary

(Terms described here are described for further communication in TC. Definitions may change.)

XACML Core - XACML Core represents a mandatory set of the XACML specification that must be supported in every XACML system. ***XACML Core*** consists of ***XACML Core Semantics*** and ***XACML Core Schema***.

XACML Core Grant Policy - A policy that ***XACML Core Semantics*** supports. The semantics is defined as “if one or more rule(s) holds, then access is grant, otherwise access is denied.”

XACML Core Schema - A set of the XACML policy primitives that consists of applicablePolicy, policy, rule, precondition, postCondition, and other useful primitives such as principal and resource. (Appendix A shows a proposed XACML Core Schema.)

XACML Core Semantics - The semantics that determines the meaning of access control policies defined in the ***XACML Core***. One instance of the XACML Core Semantics is ***XACML Core Grant Policy***.

XACML Extension - XACML Extension represents a class of extensible XACML access control policies and semantics. XACML Extension (?MUST, SHOULD, MAY) support ***XACML Core***. XACML Extension consists of ***XACML Extension Semantics*** and ***XACML Extension Schema***.

XACML Extension Model - XACML Extension Model defines what portion of XACML specification is a core and to what extent the XACML specification can be extended.

XACML Extension Schema - Any XML schema that can be defined as an extension of ***XACML Core Schema***. XACML Extension Schema is defined by policy administrators. An extended schema MAY differ from another extended schema. (Appendix B shows several extension examples.)

XACML Extension Semantics - Semantics that determines the meaning of the user-defined access control policies. XACML Extension Semantics is defined by policy administrators. An extended semantic basis MAY differ from another extended semantic basis.

2. XACML Extension Model

Figure 1 shows an XACML Extension Model.

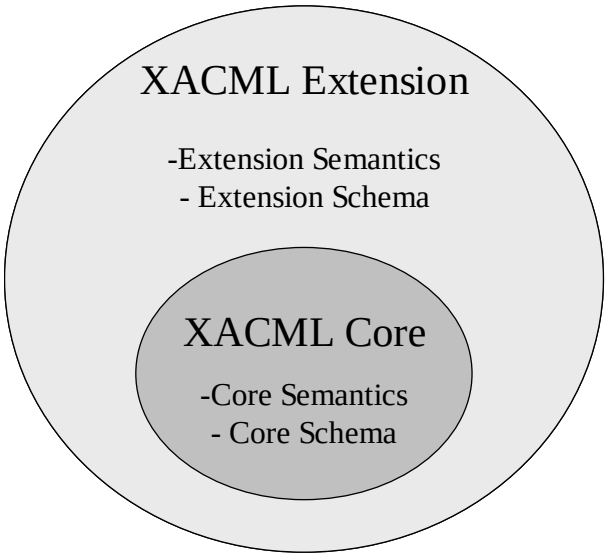

Figure 1. XACML Extension Model

An XACML Core is a part of the XACML specification that consists of an XACML Core Semantics and an XACML Core Schema. The XACML Core itself CANNOT be extended. The XACML Core MAY not describe any specific algorithm that implements the XACML Core Semantics. Algorithms for the XACML Core depend on each implementation.

An XACML Extension represents a class of extensible XACML-based access control policies.The XACML Extension consists of an XACML Extension Semantics and an XACML Extension Schema. The XACML Specification does not define any specific extension instance but only defines a framework how to extend the XACML Core. Each policy instance extended from the XACML Core is an instance of the XACML Extension. By this extension model, any policy

administrator who needs to specify local access control policies can define the semantics and the schema of their policies that conform to the XACML Specification. For the purpose of the XACML conformance, the algorithm implemented by each policy administrator (?MUST, SHOULD, MAY) support the XACML Core Semantics in addition to their local semantics. This restriction guarantees that every XACML system supports at least the semantcs of the XACML Core. The extension schema MUST be an extension of the XACML Core Schema.

2.1 Overview of XACML Extensibility Property

The figure 2 shows the overview of the XACML Core and its extensibility property proposed in this document.

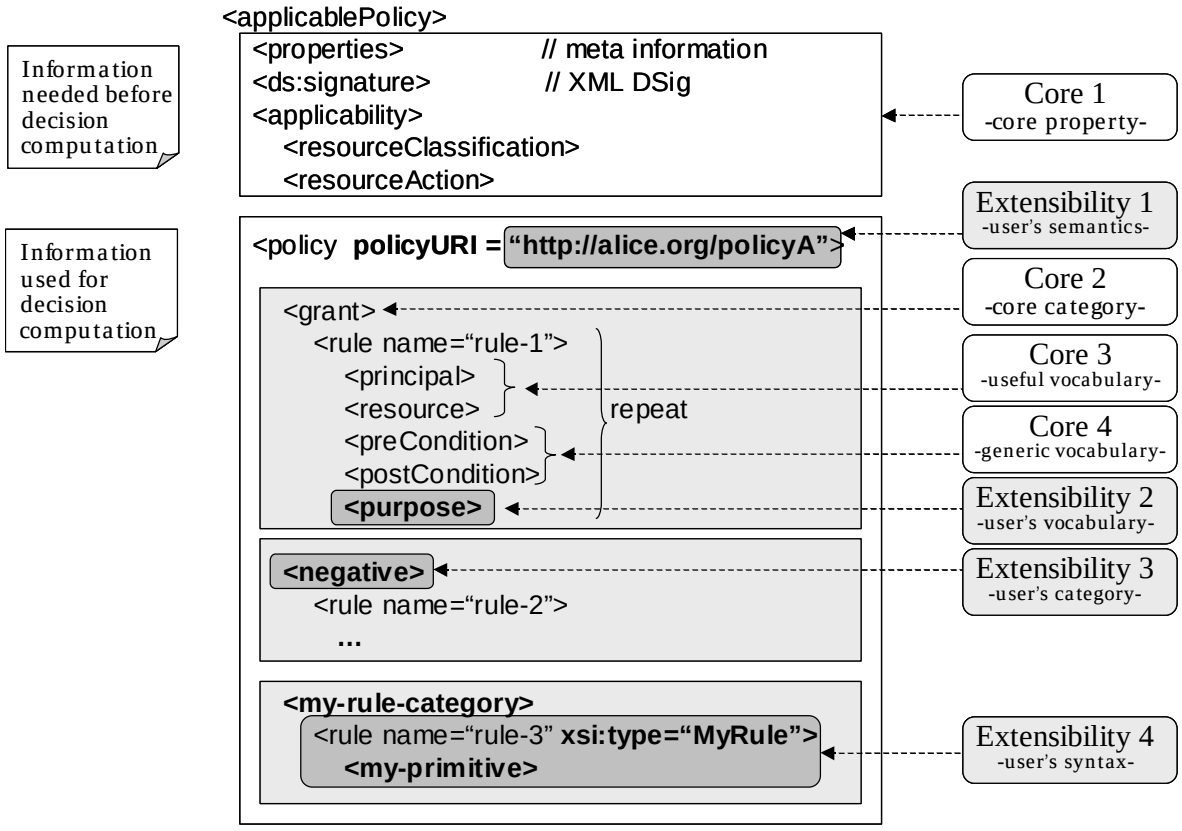

Figure 2. Overview of XACML Extensibility Property

2.2. Design Principles

Design principles of the XACML Extension Model is the following:

1. The XACML Core stringently defines the semantics and the schema of access control policy rules.

2. The XACML Core represents common functionalities described in the XACML Use Case Summary document [1] and mailing-list discussion in proper, concise, and integral manner.

3. The XACML Extension provides a maximum extensibility with the XACML enough for as yet unknown features. For this purpose, we assume a model of how should policies be extended. This model avoids ad hoc extensions that might be done by users.

4. The extension to the XACML Core Schema is realized by the extensible functions of XML Schema. It is desirable to apply flexible object-oriented design scheme.

2.3. XACML Core

The XACML Core consists of the XACML Core Semantics and the XACML Core Schema.

The XACML Core Semantics basically corresponds to the one described in the current XACML language proposal [2]. The grant-based policy is stated as “if one or more rule(s) holds, then access is grant, otherwise access is denied.” The semantic basis of each rule is represented as a set of Boolean expressions such as equality and inequality. This intuitively means that if a set of Boolean expressions of the specific rule holds, then that rule holds. If at least one rule in the policy holds, then the PDP determines that access is grant. If none of the rule holds, then the PDP determines that access is denied. We call this semantics an XACML Core Grant Policy. The XACML Core only supports the XACML Core Grant Policy because most of the access control policies described in the XACML Use Case Document can be described using only the XACML Core Grant Policy. This decision is made to address the second design principle of the XACML Extension Model.

For the XACML Core Schema, we definitely need more schema primitives than the current draft defines, because almost all the policies included in the XACML Use Case Summary document consist of the equalities on principal attributes and resource attributes. A kind of the access-triple syntax that consists of equality conditions of principal attributes and resource attributes make the access control policy look more proper, concise, and familiar (we assume that an action classifiction is specified in the applicability element.) The above primitives are created to address this issue. Since the meaning of the added primitives is defined based on the Boolean semantics, it does not destroy the XACML Core Semantics that corresponds to the semantic basis defined in the current language proposal.

2.4. XACML Extension

The XACML Extension consists of an XACML Extension Semantics and an XACML Extension Schema. The XACML Specification does not define any specific extension instance but only defines a framework how to extend the XACML Core. It would be nice to include useful extension examples.

The XACML Extension Semantics and Schema consists of the following extension points.

- XACML Extension Semantics
  1. User-defined semantics extension (mandatory)
  2. User-defined algorithm extension (optional)

- XACML Extension Schema
  1. Rule category extension
  2. Rule extension
  3. Parameter restriction
  4. Function/Predicate extension
  5. Macro extension

The XACML Extension Semantics allows policy administrators to define the meaning of their local access control policy. It consists of a mandatory user-defined semantics extension and an optional algorithm extension. The user-defined semantic basis is identified by a unique URI. While there is no need to specify algorithm information, policy administrators can specify implementation specific information identified by a unique URI. Such flexibility satisfies the third design principle.

The XACML Extension Schema allows policy administrators to define the extended format of their local access control policy. It consists of a rule category extension, a rule extension, a parameter restriction, function/predicate extension, and a macro extension. This flexibility satisfies the third design principle. We explain each extension point below.

1. The rule category extension is a first extensible point of the XACML Schema. The rule category means a primitive element of the user-defined semantic basis and it allows policy administrators to specify several semantic primitives under the policy element. Intuitively speaking, the name of the rule category corresponds to a returned value of the policy evaluation. For instance, the XACML Core uses the “grant” element for the default rule category because “grant” is the result of the policy evaluation if at least one rule holds. In other access control policies, “positive” and “negative” rule categories can represent the policy that allows policy administrators to specify an explicit negative permission in the rule in addition to the positive permission. In this case, the rule category could be defined as “positive” and “negative”. More concrete examples are described in Appendix B.2, B.3, and B.4. Note that the semantics of the extended rule category must be unambiguously handled by the user-defined algorithm.

2. The rule extension is the second extensible point of the XACML Schema. The rule extension means that policy administrators can add new elements in the rule element. For instance, the XACML Core allows only “principal”, “resource”, “preCondition”, and “postCondition” elements in the rule element. Using the rule extension, policy administrators can add arbitrary elements such as “codeSource” and “purpose” in the rule element. This extension allows policy administrators to use a local vocabulary of their policy domain. It could greatly improve readability of the policy as well. Note that the semantics of the added elements in the rule element must be unambiguously handled by the user-defined algorithm.

3. The parameter restriction is the third extensible point of the XACML Schema. The parameter restriction allows policy administrators to specify this kind of restrictions on parameters in the

XACML Extension Schema. The notion of the parameter restriction is represented by XML Schema Derivation function. (this proposal needs concrete examples for this extension)

4. The function and/or predicate extension are the fourth extensible point of XACML Schema. It is defined in the current language proposal.

5. The macro extension is also mentioned in the current language proposal.

2.5 Rule Priority

When policy administrators think of the semantics of their access control policy rules, there is a case that a priority is assigned to each rule to solve conflicts if multiple rules return different decision values such as a positive and a negative. The XACML Extension does not provide any functionality for implementing this kind of semantic basis. Instead, the XACML Core supports more fundamental way to realize such semantics. In the XACML Core, each rule can have a name attribute. Using this attribute, the user-defined algorithm can use the rule name to map to the rule priorities locally defined in their algorithm.

2.6 Policy Property

Basically it is necessary to support user-defined meta-information about the policy. A few examples of such meta-information would be 1) a list of the events when the access control policy can be legally bypassed (this addresses the “breaking the class” requirement in the Clinical Record use case), 2) a list of the exceptions that access control policy may return to the PDP, and 3) a condition when and how the access control policy description should be digitally signed. They are so application-specific that the contents of the meta-information are outside the scope of the XACML Core. The XACML Core only provides a space for describing such meta-information. In this proposal, a property element is added under the applicablePolicy element. Policy administrators can put any information under the property element. The notion of this “any” information is represented by “any” element placeholder of the XML Schema.

2.7 XACML Extension Suite

A XACML Extension Suite represents a notion of application-specific reusable policy components. For example, if there is a useful set of policy components that works on a tree-based resource hierarchy, people may use the whole extension components for their specific target domain. A set of those components can be called XACML Extension Suite for AAA.

2.8 Interoperability

For the XACML Core, the interoperability is defined as “every XACML Core implementation must output the same access decision in response to the same access request and environment values

based on the same access control policy rules." Since the semantics and the schema are stringently defined and no extension is allowed, the interoperability can be easily achieved.

For the XACML Extension, the situation is the opposite to the XACML Core because policy administrators can extend the semantics of the policy as well as the schema of the policy. However, two implementations A and B outputs the same access decision in response to the same access request and environment values based on the same access control policy rules provided, the implementations of A and B have the same value for the policy URI attribute specified in the policy element and they shares the same XACML Extension schema.

3. XACML Schema Representation

We present each XACML schema primitive.The proposed XACML Schema is designed particularly for supporting the extensible points in the schema described in the Section 2.4. For example, the rule category extension is realized in the Rule Category element. The rule extension is realized in the Rule element.

3.1 Applicable Policy

Applicable policy is identical to the one in the current language proposal except for the property element described in the Section 2.6.

3.2 Policy

The policy element is an aggregation point of rule categories. This element determines a semantics (and optionally an algorithm) and a schema for every rule described under this element. The semantics of the policy is specified by a policyURI attribute. When the policyURI attribute is omitted, the default XACML Core Grant Policy is assumed. The algorithm is optionally specified by an algoURI attribute. The policy schema and its exact location can be identified using a namespace definition and location definition of the XML Schema. The following schema defines the policy element.

```
<xs:element name="policy" type="Policy"/>
<xs:complexType name="Policy">
   <xs:sequence>
      <xs:element name="ruleCategory" minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="policyURI" type="xs:anyURI" default="http://www.xacml.org/grantpolicy"/>
   <xs:attribute name="algoURI" type="xs:anyURI"/>
</xs:complexType>
```

A sample policy instance is described below:

```
<policy policyURI="http://www.xacml.com/grantpolicy" algoURI="http://www.xacml.com/algo"/>
```

3.3 Rule Category

The rule category element is an important extension point of the XACML Schema as well as an aggregation point of one or more rules. This element symbolically groups a set of rules that belongs to a specific rule category such as “grant”, “positive”, “negative” and “onlyif”. The following schema defines the rule category element.

```
<xs:element name="ruleCategory" type="RuleCategory" abstract="true"/>
<xs:complexType name="RuleCategory">
  <xs:sequence>
    <xs:element name="rule" type="Rule" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

Note that the rule category element is defined as “abstract”. Thus, policy administrators MUST substitute this element if they need to extend the semantics of the policies. Thus, the notion of the rule category is represented by XML Schema Element Substitiongroup function. Concrete extension examples are described in Appendix B.2, B.3, and B.4. If the policy administrator uses the XACML Core Grant Policy (XACML default rule category), no substitution is required. The XACML Core has a pre-defined rule category “grant”.

```
<xs:element name="grant" type="RuleCategory" substitutionGroup="ruleCategory"/>
```

The XACML Core Semantics defines the meaning of this “grant” rule category as “if at least one rule in the policy holds, then the PDP determines that access is grant. if none of the rule holds, then the PDP determines that access is denied.” If policy administrators need to modify the semantics of this grant policy, the grant rule category element MUST not be used. A sample policy instance is described below:

```
<policy policyURI=”http://www.xacml.com/grantpolicy” />
<grant>
  <rule name=”rule-1” >
    <preCondition>…
    …
  </rule>
</grant>
```

If the policy administrator does not need to extend the XACML Core Grant Policy but wants to extend the XACML Core Schema by adding a new element e.g. “codesource”, it is required to substitute the rule element. The rule element is described in the next section.

3.4 Rule

The rule element is an extension point of XACML Schema as well as an aggregation point of a set of rule primitives. This element groups a set of rule primitives such as “principal” and “preCondition” elements. The pre-defined rule elements consist of “principal”, “resource”, “preCondition”, and “postcondition”. The following schema defines the rule element:

```
<xs:element name="rule" type="Rule"/>
<xs:complexType name="Rule">
  <xs:sequence>
    <xs:element name="principal" type="PreconditionAlias" minOccurs="0" maxOccurs="1"/>
    <xs:element name="resource" type="PreconditionAlias" minOccurs="0" maxOccurs="1"/>
    <xs:element name="preCondition" type="PreCondition" minOccurs="0" maxOccurs="1"/>
    <xs:element name="postCondition" type="PostCondition" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
```

If you need to add a new element, you MUST derive a new rule element by substituting the Rule type. The notion of the rule extension is represented by XML Schema Type Substitution function. The derived rule can contain the added element as well as all the pre-defined elements. Each pre-defined primitive is optional. The example below shows how to derive the new rule that contains “purpose” element:

```
<xs:complexType name="PrivacyRule">
  <xs:complexContent>
    <xs:extension base="Rule">
      <xs:sequence>
        <xs:element name="purpose" type="Purpose”/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

A sample policy instance is described below:

```
<policy policyURI=”http://www.privacy.com/” algoURI=”http://www.privacy.com/algo”/>
<grant>
  <rule name=”rule-1” xsi:type=”PrivacyRule”>
    <purpose>fulfilment</purpose>
    <resource>…
    <preCondition>…
  </rule>
</grant>
```

A concrete example is described in Appendix B.1.

3.5 Principal

This elememt is created to address the second design principle of the XACML Extension Model. The principal element specifies conditions on the principal using simplified Boolean expression that supports the conjunctive and/or disjunctive formula of equalities and/or inequalities. Note that this limitation is not intrinsic, but rather derived from the use cases described in the XACML Use Case Summary document and mailing list discussions. Almost all the policies can be specified only using these simplified Boolean expressions. The semantics of this element is defined as a subset of the semantics of the preCondition element. The preCondition is capable of specifying more complicated conditions if the policy administrator needs to write them. The following schema defines the principal element.

```
<xs:element name="principal" type="PreconditionAlias" minOccurs="0" maxOccurs="1">
<xs:complexType name="PreconditionAlias">
   <xs:sequence>
   <xs:element name="simpleLogicalOperator" type="SimpleLogicalOperator" minOccurs=”0” maxOccurs=”unbounded”/>
   </xs:sequence>
</xs:complexType>

<xs:element name=”simpleLogicalOperator” type=”SimpleLogicalOperator” abstract=”true”/>
<xs:complexType name="SimpleLogicalOperator">
  <xs:sequence>
   <xs:element name="simpleExpression" type="SimpleExpression" minOccurs=”0” maxOccurs=”unbounded”/>
  </xs:sequence>
</xs:complexType>
<xs:element name="and" type="SimpleLogicalOperator" substitutionGroup="simpleLogicalOperator"/>
<xs:element name="or" type="SimpleLogicalOperator" substitutionGroup="simpleLogicalOperator"/>

<xs:element name=”simpleExpression” type=”SimpleExpression” abstract=”true”/>
<xs:complexType name="SimpleExpression">
  <xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string"/>
  <xs:attribute name="value" type="xs:string"/>
</xs:complexType>
<xs:element name="equality" type="SimpleExpression" substitutionGroup="simpleExpression"/>
<xs:element name="inequality" type="SimpleExpression" substitutionGroup="simpleExpression"/>
```

A sample principal expression is described below:

```
<principal>
   <and>
      <equality type=”saml/Attribute/AttributeName/Role” value=”InternalUser”/>
      <equality type=”saml/Attribute/AttributeName/Role” value=”Manager”/>
   </and>
</principal>
```

The above policy means that the principal should be an internal user and a manager at the same time.

3.6 Resource

The resource element specifies one or more simplified Boolean expressions of the resource condition. The element definition is similar to that of the principal element. A sample resource expression is:

```
<resource>
   <or>
      <equality type=”environment/targetXML“ value=”//confidential”/>
      <equality type=”environment/targetXML” value=”//secret”/>
   </or>
</resource>
```

The above policy means that the target XML element is confidential or secret at any level of the target XML resource.

3.7 Pre-condition

Precondition is identical to the one in the current language proposal with minor modifications.

3.8 Post-condition

The semantics of the postcondition is not defined in the XACML Schema because people has not yet been familiar with this notion. However, two use cases in [1] definitely require the postcondition. Thus, the XACML Schema defines just a space for this notion as an optional element called post condition that can contain any elements. The definition is:

```
<xs:element name="postCondition" type="PostCondition" />
<xs:complexType name="PostCondition">
   <xs:sequence>
   <xs:any namespace="##any" processContents="skip"/>
   </xs:sequence>
</xs:complexType>
```

References

[1] XACML Use Case Summary, http://lists.oasis-open.org/archives/xacml/200110/msg00073.html

[2] XACML Language Proposal v0.7, http://lists.oasis-open.org/archives/xacml/200111/msg00039.html

[3] XACL, http://www.trl.ibm.com/projects/xml/xacl/index.htm and http://alphaworks.ibm.com/tech/xmlsecuritysuite

[4] An Access Control Model for Data Archives, http://sansone.crema.unimi.it/~samarati/Papers/sec01.ps

[5] Sushil Jajodia, Pierangela Samarati, and V. S. Subrahmanian, “A Logical Language for Expressing Authorizations,” IEEE Security and Privacy, 1997.

[6] J2SE Use Case, http://lists.oasis-open.org/archives/xacml/200112/msg00045.html

Appendix A - XACML Schema

A.1 XACML Core Schema

The following schema defines the XACML Core Schema. Since many schema definitions are overlapping with the current lanaguage proposal, we omit several type definitions such as PreCondision type.

```
<xs:element name="applicablePolicy">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="properties" type=”Properties” minOccurs="0" maxOccurs="1”/>
      <!—ds:signature element may be located here -->
      <xs:element name="applicability" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="resourceClassification" type="xs:anyURI"/>
            <xs:element name="resourceAction" type="saml:Actions" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="policy" type="Policy"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="Properties">
  <xs:sequence>
  <xs:any namespace="##any" processContents="skip"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="policy" type="Policy"/>
<xs:complexType name="Policy">
  <xs:sequence>
    <xs:element name="ruleCategory" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="policyURI" type="xs:anyURI" default=”http://www.xacml.org/grantpolicy"/>
  <xs:attribute name="algoURI" type="xs:anyURI"/>
</xs:complexType>

<xs:element name="ruleCategory" type="RuleCategory" abstract="true"/>
<xs:complexType name="RuleCategory">
  <xs:sequence>
    <xs:element name="rule" type="Rule" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="grant" type="RuleCategory" substitutionGroup="ruleCategory"/>

<xs:element name="rule" type="Rule"/>
<xs:complexType name="Rule">
  <xs:sequence>
    <xs:element name="principal" type="PreconditionAlias" minOccurs="0" maxOccurs="1"/>
    <xs:element name="resource" type="PreconditionAlias" minOccurs="0" maxOccurs="1"/>
    <xs:element name="preCondition" type="PreCondition" minOccurs="0" maxOccurs="1"/>
```

```
        <xs:element name="postCondition" type="PostCondition" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>

<xs:element name="postCondition" type="PostCondition" />
<xs:complexType name="PostCondition">
    <xs:sequence>
    <xs:any namespace="##any" processContents="skip"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="PreconditionAlias">
    <xs:sequence>
    <xs:element name="simpleLogicalOperator" type="SimpleLogicalOperator" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:element name="simpleLogicalOperator" type="SimpleLogicalOperator" abstract="true"/>
<xs:complexType name="SimpleLogicalOperator">
  <xs:sequence>
    <xs:element name="simpleExpression" type="SimpleExpression" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="and" type="SimpleLogicalOperator" substitutionGroup="simpleLogicalOperator"/>
<xs:element name="or" type="SimpleLogicalOperator" substitutionGroup="simpleLogicalOperator"/>

<xs:element name="simpleExpression" type="SimpleExpression" abstract="true"/>
<xs:complexType name="SimpleExpression">
  <xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string"/>
  <xs:attribute name="value" type="xs:string"/>
</xs:complexType>
<xs:element name="equality" type="SimpleExpression" substitutionGroup="simpleExpression"/>
<xs:element name="inequality" type="SimpleExpression" substitutionGroup="simpleExpression"/>
```

Appendix B - XACML Extension Examples

B.1 J2SE Policy

J2SE policy is based on the use case description posted on the XACML mailing-list [6]. Their requirements are:

1. There must be a way in the policy language to express both a signer as well as a principal within a rule. It has been suggested that one of the existing XACML attributes - environment, resource, principal attributes could be used. Of the three the most logical one seems to be principal attribute. But overloading of a principal attribute for both principals and signers makes the authorization rules less clear.

2. There must be a way to express CodeSource (i.e. URL from where the code originated from and/or the certificates used to sign the code).

This example shows a solution to the above requirements by adding three new primitives "codesource", "signer", and "permission" elements in the rule element. The following schema extension shows how to extend XACML Core Schema:

```
<xs:complexType name="J2SE">
  <xs:complexContent>
    <xs:extension base="Rule">
      <xs:sequence>
        <xs:element name="codesource" type="Codesource"/>
        <xs:element name="signer" type="Signer">
        <xs:element name="permission" type="Permission">
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The example below shows a J2SE policy instance.

```
<policy policyURI="http://www.j2se.com/" algoURI="http://www.j2se.com/algo"/>
<grant>
  <rule name="rule-1" xsi:type="J2SE">
    <codesource>file:c:/programs/myprogram.jar</codesource>
    <principal>com.j2se.com.J2SEPrincipal</principal>
    <signer>ABC.com</signer>
    <permission>javax.security.auth.AuthPermission "doAs"</permission>
  </rule>
</grant>
```

(Since we assume here that the semantics of the J2SE policy is identical to the XACML Core Grant Policy, there is no need to extend the rule category element. If not, a proper rule category must be defined.)

B.2 Restriction-based Policy

An access control policy that allows a restriction-based control is described in [4]. It allows policy administrators to specify "onlyif" rule. Since it needs a different semantic basis than the XACML Core Grant Policy, the rule category element must be substituted. The example below shows a substitution.

```
<xs:element name="onlyif" type="RuleCategory" substitutionGroup="ruleCategory"/>
```

Since the "onlyif" rule category needs an extended rule syntax, the rule element must also be substituted. The example below shows a substitution.

```
<xs:complexType name="OnlyIf">
  <xs:complexContent>
    <xs:extension base="Rule">
      <xs:sequence>
        <xs:element name="restrictionCondition" type="PreCondition"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
```

```
</xs:complexType>
```

The example below shows a policy instance of the “onlyif” rule category.

```
<policy policyURI=”http://www.onlyif.com/” algoURI=”http://www.onlyif.com/algo”/>
<onlyif>
  <rule name=”rule-2” xsi:type=”OnlyIf”>
    <restrictionCondition>citizenship is ’UK’</restrictionCondition>
    <principal>…
    <resource>…
    <preCondition>…
  </rule>
</onlyif>
```

B.3 Policy for XML Resources

A fine-graind access control policy specification language for XML resource is proposed in [3]. It allows policy administrators to specify “positive” and “negative” permissions. Final access decision is determined by using user-defined meta policies such as a denial-takes precedence policy. The XACML Extension allows policy administrators to write such policy by substituting the rule category element. The example below shows the substitution.

```
<xs:element name="positive" type="RuleCategory" substitutionGroup="ruleCategory"/>
<xs:element name="negative" type="RuleCategory" substitutionGroup="ruleCategory"/>
```

The example below shows an policy instance.

```
<policy policyURI=”http://www.xacl.com/policy/dtp” algoURI=”http://www.xacl.com/algo”/>
<positive>
  <rule name=”rule-1”>
    <principal>…
    <resource>…
    <preCondition>…
    <postCondition>…
  </rule>
</positive>
<negative>
  <rule name=”rule-2”>
    <principal>…
    <resource>…
    <preCondition>…
    <postCondition>…
  </rule>
</negative>
```

B.4 Logic-based Flexible Access Control Policy

A logic-based flexible authoriztion framework and Authorization Specification Langauge (ASL) are proposed in [5]. Multiple access control policies are described based on the semantics of the locally stratified datalog. The XACML Extension allows policy administrators to write ASL-based policies

by substituting the rule category element by “cando”, “dercando”, and “do”. We need to extend the rule element for the sign element. The example below shows the substitution.

```
<xs:element name="cando" type="RuleCategory" substitutionGroup="ruleCategory"/>
<xs:element name="dercando" type="RuleCategory" substitutionGroup="ruleCategory"/>
<xs:element name="do" type="RuleCategory" substitutionGroup="ruleCategory"/>

<xs:complexType name="ASL">
   <xs:complexContent>
      <xs:extension base="Rule">
         <xs:sequence>
            <xs:element name="sign" type="Sign"/>
            <xs:element name="action" type="Action"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

The example below shows an policy instance of this rule category.

```
<policy policyURI="http://www.asl.com/" algoURI="http://www.asl.com/algo"/>
<cando>
   <rule name="cando(Alice, //secret, +r) :- conditionA." xsi:type="ASL">
      <principal>Alice</principal>
      <resource>//secret</resource>
      <sign>+</sign>
      <action>r</action>
      <preCondition>conditionA</preCondition>
   </rule>
</cando>
<dercando>
   <rule name="dercando(X,Y,+Z) :- conditionB" xsi:type="ASL">
      <principal>X</principal>
      <resource>Y</resource>
      <sign>+</sign>
      <action>Z</action>
      <preCondition>conditionB</preCondition>
   </rule>
</dercando>
<do>
   <rule name="do(X,Y,+Z) :- conditionC." xsi:type="ASL">
      <principal>X</principal>
      <resource>Y</resource>
      <sign>+</sign>
      <action>Z</action>
      <preCondition>conditionC</preCondition>
   </rule>
</do>
```