XACML language proposal

Bill Parducci	Bill Parducci
Carlisle Adams	Entrust
Ernesto Damiani	University of Milan
Hal Lockhart	Entegrity
Ken Yagen	Crosslogix
Michiharu Kudo	IBM, Japan
Pierangela Samarati	University of Milan
Simon Godik	CrossLogix
Tim Moses	Entrust

Table of contents

Chapter One

1. Glossary

1.1. Preferred terms

Access - Performing an ***action*** on a ***resource***

Access control - Controlling ***access*** in accordance with ***applicable policy***

Action - Operation that may be performed on ***resource***

Applicable policy - The complete set of ***rules*** that governs ***access*** for a specific ***resource***

Attribute - Characteristic of a ***principal***, ***resource*** or ***environment*** that may be referenced by a ***pre-condition***

Authorization decision - The result of evaluation of ***applicable policy.*** A function with BOOLEAN range and, optionally, a set of ***post-conditions***

Classification - A set of ***attributes*** relevant to a ***resource***

Context - The intended use of information revealed as a result of ***access***.

Decision request - The request by a ***PEP*** to a ***PDP*** to render an ***authorization decision***

Environment - The set of ***attributes*** that may be referenced by ***pre-conditions*** and that are independent of a particular ***principal*** and ***resource***

Information request - The request by the ***PDP*** to the ***PIP*** for one or more ***environment attributes***

Policy - (see **Applicable policy**)

Policy conflict - The state that exists when two or more ***pre-conditions***, forming part of ***applicable policy***, individually yield conflicting results

Policy decision point (PDP) - The system entity that evaluates ***applicable policy***

Policy enforcement point (PEP) - The system entity that performs ***access control***

Policy information point (PIP) - The system entity that acts as the source of ***environment attributes***

Policy administration point (PAP) - The system entity that creates ***applicable policy***

Policy mediation point (PMP) - The system entity that resolves ***policy conflict***

Policy retrieval point (PRP) - The system entity that ensures ***applicable policy*** is complete

Post-condition - A process specified in a ***rule*** that must be completed in conjunction with ***access***. There are two types of post-condition: an ***internal post-condition*** must be executed by the ***PDP*** prior to the issuance of a "permit" response, and an ***external post-condition*** must be executed by the ***PEP*** prior to permitting ***access***

Predicate - A statement about ***attributes*** whose truth can be evaluated

Pre-condition - A ***predicate*** or logically-combined set of ***predicates***

Principal - A system entity that can be referenced by a ***pre-condition***

Resource - Data, service, or system component

Role - A set of ***attributes*** relevant to a ***principal***

Rule - The combination of a ***pre-*** and one or more ***post-conditions***

1.2. Related terms

In the field of access control and authorization there are several closely related terms in common use. For purposes of precision and clarity, certain of these terms are not used in this specification.

For instance, the term ***attribute*** is used in place of the terms: privilege, permission, right, authorization and entitlement.

The terms "subject" and "user" are also in common use. But, we use the term ***principal*** in this specification.

The terms "object" and "target" are also in common use, but we use the term ***resource*** in this specification.

While the term "group" is commonly used with a meaning that is distinct from that of ***role***, the distinction has no significance in the domain of XACML, therefore, the term group is not used here.

2. Introduction

XACML specifies a mark-up language for access control policies. It is intended to be used in conjunction with SAML assertions and messages.

3. Models

The information in this section is non-normative.

The context and schema of XACML are described in three models that elaborate different aspects of its operation. These models are: the data-flow model, the language model and the administrative model. They are described in the following sub-sections.

3.1. Data-flow model

The major actors in the XACML domain are shown in the data-flow diagram of Figure 1.

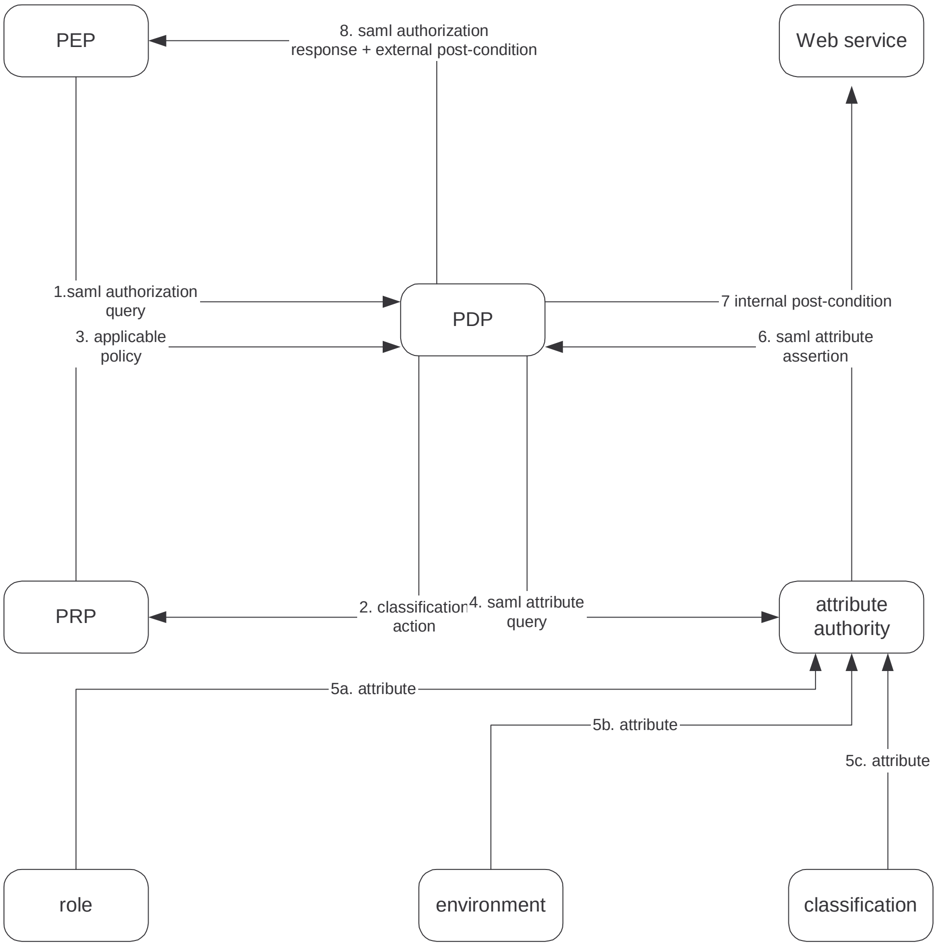


Figure 1 - Data-flow diagram

Some of the data-flows shown in the diagram may be facilitated by a repository. For instance, the communications between the PDP and the attribute authority may be facilitated by a repository, or the communications between the PDP and the PRP may be facilitated by a repository. The XACML specification is not intended to place restrictions on the location of any such repository, or indeed to prescribe a particular communication protocol for any of the data-flows.

The model operates according to the following steps.

1. The PEP sends a decision request to the PDP, in the form of a SAML [SAML] authorization query. The decision request contains some or all of the attributes required by the PDP to render a decision, in accordance with policy.

2. The PDP locates and retrieves the policy instance applicable to the decision request from the PRP. It uses the resource classification and the requested action to identify the correct policy. The means by which the PDP determines the classification of the resource is out of scope for this specification. However, in the case where the resource is an XML document, its classification may be an attribute of the top-level element of the resource.

3. The PRP returns the policy to the PDP in the form of an XACML instance.

4. The PDP examines the decision request and the policy to ascertain whether it has all the attribute values required to render an authorization decision. If it does not, then it requests attributes from suitable attribute authorities in the form of SAML attribute queries [SAML].

5. The attribute authorities may locate and retrieve the requested attributes from other systems by a means, and in a form, that is out of scope for this specification.

6. The attribute authorities return the requested attributes to the PDP in the form of SAML attribute responses containing SAML attribute assertions.

7. The PDP evaluates the policy instance. In the case where the policy instance contains internal post-conditions, the PDP executes those post-conditions.

8. If the policy were to evaluate to TRUE, and the internal post-conditions were to execute successfully, then the PDP returns an authorization decision, in the form of a SAML authorization response, to the PEP containing the "permit" result code and any external post-conditions.

3.2. Language model

3.2.1. Elements of the access control policy

An access control policy states regulations governing access to resources, and therefore how the system should respond to requests that principals can submit. The access control policy comprises of access rules stating which accesses should (or should not) be allowed and, possibly, under which conditions such permissions or denials for the access apply.

We therefore start by identifying the various elements of the access rules. At this stage we characterize the different access components with respect to their format and semantics. Later we will define the precise syntax (a preliminary sketch of the syntax appears in ``XACML Language").

An access rule can be seen as comprised of the following elements:

- principal expression: identifies the (dynamic set of) principals to whom the rule applies.

- resource expression: identifies the (dynamic set of) resources to which the rule applies.

- action expression: identifies the (dynamic set of) actions to which the rule applies.

- environment expression: identifies system-dependent and request-dependent conditions to be satisfied for the rule to apply.

- post-conditions: defines a set of actions that the access control system (PEP) must execute whenever a rule is applied with respect to a given access request.

- if/only if conditional statements.....

3.2.2. Reserved identifiers

The expressiveness of the language will allow us to specify rules whose applicability will depend on conditions that the principal requesting access, or the resource on which access is requested, satisfy. Access rules are therefore not referred to a specific principal or a specific resource but to a set of them that satisfy given conditions. To provide expressiveness needed to make it possible the specification of such generic rules without the need of introducing variables in the language we introduce the following reserved identifiers:

- principal: is the principal presenting the request

- resource: is the resource on which access is requested

- action: is the action requested

3.2.3. Principal expression

The principal expression defines the principal, or set of principals, to which the access rule applies. It is a Boolean expression evaluating SAML assertions (i.e., properties) associated with the principal requesting access.

SAML assertions can refer to any property of the principals, including groups to which the user making the request belongs or roles (privileged positions) that the user may have activated and present. Groups and role management is outside the scope of the authorization language, we assume information about active roles to be provided through SAML assertions; we assume information about group memberships to be either provided as SAML assertions or to be available at the PIP.

Each given assertion term (i.e., elementary component of a principal expression) evaluates the value of a property associated with the requestor as is of the form

<SAML-assertion> <comparison operator>

<SAML-assertion>

or

<SAML-assertion> <comparison operator>

<constant-value>

where the comparison operator is a suitable operator (including <,=<,>=,=,...) depending on the property type and the XPath language is used to refer to SAML assertions within the specification of the assertion terms.

Some examples of principal expressions are as follows:

- principal/login_name=`bob'

user `bob'

- principal/organization=`OASIS' AND principal/residence=`North-Pole'

principals associated with OASIS and living at the North Pole

- principal/organization=principal/login_name

principals working for an organization owned by themselves.

Use of variables as macros (to be completed checking with Ernesto and Simon)

3.2.4. Resource expression

The resource expression is a Boolean expression of conditional terms that evaluate properties of the resource. Properties appearing in these conditional terms can refer to the resource content or to meta properties associated with the resource. The reserved identifier `resource' can be used in the specification of generic expression applicable to a (dynamic) set of resources. Resource expressions can be also make use of the keyword `principal' for specifying conditions putting in relationships properties of the resources with that of the principals. Each conditional term in a resource expression is therefore of the form

<resource-assertion> <comparison operator><SAML-assertion>

or

<resource-assertion> <comparison operator> <constant-value>

Reference to the meta-property of the resource or to its content

depends on .... and can make use of functions.

Some examples of resource expressions are as follows:

- resource/creation_date =< `01/01/01'

  all resources created before January 1, 2001.

- resource/owner=principal/login_name

  all resources owned by the principal making the requests

- resource/label=`Top Secret'

all resources classified as Top-Secret (note: we are not implying support for a multilevel policy here)

3.2.5. Action

The action component of the access control rule defines the action, or set of actions, to which the rule refers. An action is identified by a name and may have associated a set of parameters. The parameters can refer to any input/output of the process.

Some examples of action expressions are as follows:

- action/withdrawal_amount =< 500,000

- action/data_recipient IN Doctors

3.2.6. Environment expression

It's a Boolean expressions of environmental conditions that can evaluate the system state (e.g., date and time) and request parameters (e.g., location from where the principal is connected).

IF/ONLY IF CONDITIONS

A policy is composed of a set of access control rules. The usual interpretation for a set of rules specifying permissions is to grant all the accesses from which at least a rule is satisfied. The consideration of permissions only permissions, with this interpretation may result limiting in several cases. Negative access control rules (specifying denials) could be used for specifying accesses that should be denied. Introduction of negative authorizations introduces the problem of the different semantics that denials can carry which should be properly represented with different conflict resolution criteria in the model. For the time being we therefore do not consider negative authorizations. Permissions only, however, result limiting. For instance, suppose we want to say that only UScitizens can access a document. In a permission only scenario we could specify a rule stating the permissions but the semantics of the only (meaning no one else can access the document) is not supported, since the insertion of any additional rule can grant the access to some noncitizens. As another example suppose we want to say that access to a given document requires (beside additional conditions to be specified by the security administrator) presentation of a

payment certificate (stated as a SAML assertion). In a permission only scenario we should make sure that the payment condition is included in al the rules that apply to the access. Beside being difficult o control, this would introduce complicated rules (intuitively the conditions would have to be repeated in AND in every rule instead of being factored out.

Looking at the real world cases, we often find access rules stated in restrictive form rather than in the inclusive positive form just mentioned. By restrictive form we mean rules that state conditions hat must be satisfied for an access to be granted and such hat, if at least one condition is not satisfied, the access should not be granted. For instance, a rule can state that ``access to document-1 can be allowed only to citizens''. It is easy to

see that such a restriction cannot be simply represented as an permission stating that citizens can be authorized. In fact, while the single authorization brings the desired behavior, its combination with other authorizations may not, leading the only constraint to be not satisfied anymore.

A possible approach would be supporting two kinds of rules: restrictions and authorizations. Intuitively, restrictions are useful to specify requirements of the exclusive only if form stated above; while authorizations specify requirements in the traditional positive if form.

- RESTRICTIONS: specify requirements that must all be satisfied for an access to be granted. Lack to satisfy any of the requirements that apply to a given request implies the request will be denied.

Syntactically, restrictions have the form

<request-description}> <conditions}>

where

request-description is the principal, resource, action and environment expressions

and

conditions is a Boolean expression of conditions that every request to which the restriction applies must satisfy.

- AUTHORIZATIONS: specify permissions for the access. An access is granted if there is satisfaction of at least one of the permissions that apply to the given request and no restriction is violated.

Syntactically, authorizations have the form

<request-description}> <conditions}>

where request-description} has the same meaning as before and <conditions}> is a Boolean expression of conditions whose satisfaction authorizes the access.

Unlike for restrictions, lack of satisfaction of a condition in an authorization simply makes the authorization inapplicable but it does not imply the access will be denied. In particular, access can be authorized if there is at least one authorization that applies to it for which the conditions are satisfied.

3.2.7. Things to discuss

- purpose of access (discussed in the last concall), still to be inserted

- dynamic conditions: conditions that cannot be evaluated but can trigger procedures

- post-conditions

- content-based filtering: We should decide whether the resource expression can contain conditions evaluating the resource content. The complication arises from the fact that content-querying depends on the specific application/data-model/system.

- attribute reference (syntactical matter): in the examples we have used naming based notation to refer to parameters of an action. Should we allow (XPath permits it) positional-based notation as well?

- examples and simplification: Now the language can seem a bit too complicated. For instance, we need to say ``principal}/login_name=`bob', in cases where we would have just said `bob' in traditional systems. This is however consistent with the fact that for us a principal is not a user `login-name' but it is characterized through assertions. However, we could find a way to simplify expressions in some cases.

- dealing with unknown attributes: SAML assertions (as well as resource properties) are not predefined and can change. What happens if a rule has a condition on some SAML assertions that cannot be found at runtime?

- description of run-time behavior of access control

The language model is shown in Figure 2.

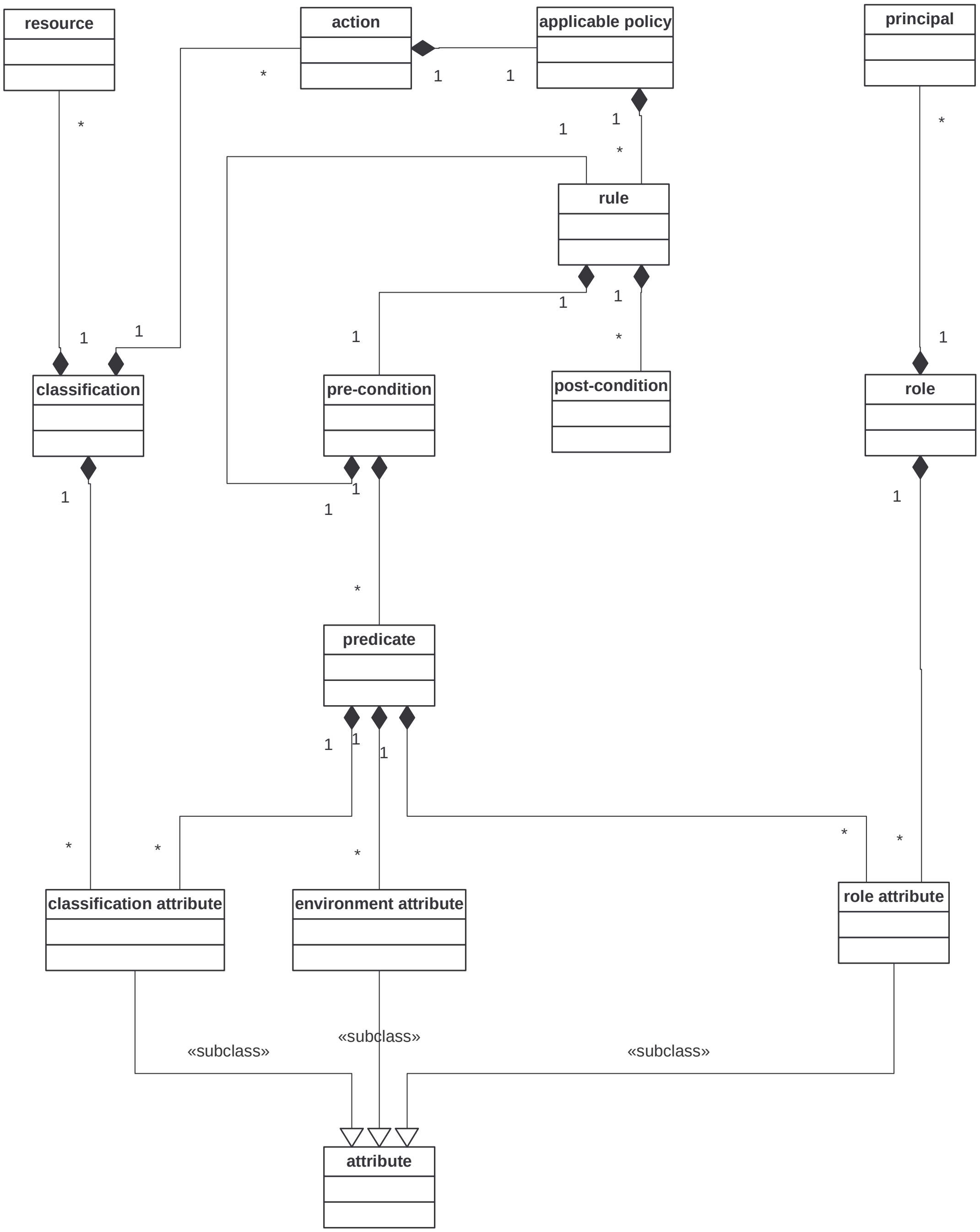


Figure 2 - Policy model

The various objects of the model are created by policy administrators, and may (or may not) be integrity protected using a digital signature or other integrity/authenticity mechanism. A set of objects may only be protected by the same integrity seal if they exist at the same place and the same time. Nothing in the model is intended to impose restrictions on the sequence in which the various objects are created and the combinations in which they may exist.

For purposes of explanation, the language model divides into six sections. These are each described in the following sub-sections.

3.2.8. Principal/role/attribute

The principal/role/attribute section of the language model is shown in gray in Figure 3.

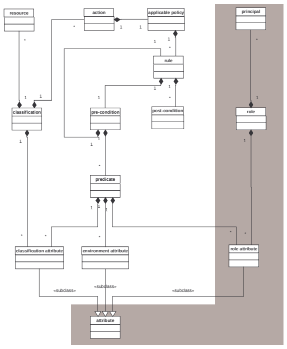


Figure 3 - Principal/role/attribute section of the language model

An authorization request relates to a single principal. XACML policy instances may reference attributes of a particular principal, or a role of the principal. The PDP should use attribute assertions to confirm that the principal occupies a required role. Both the principal and the role may have attributes. For instance, the principal "Joe" may have an attribute of type "role" set equal to the value "purchasing officer". Alternatively, the role "purchasing officer" may have an attribute of type "signing limit" set equal to the value "US$100,000". Principal and role attributes are asserted by authorities and distributed in the form of SAML attribute assertions. The PDP must check that the attribute values it operates upon are asserted by suitable authorities. This operation is described in Section 3.2.14, below.

3.2.9. Resource/classification/attribute

The resource/classification/attribute section of the language model is shown in gray in Figure 4.

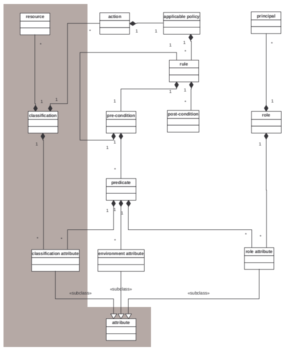


Figure 4 - Resource/classification/attribute section of the language model

An authorization request relates to a single resource. XACML policies may reference attributes of a particular resource or a classification of the resource. The PDP must confirm that the resource occupies the required classification. In the case where the resource is an XML document, it may do this by examining an attribute or element within the resource itself. In other cases, the PDP may use attribute assertions. Both the resource and classification may have attributes. For instance, a purchase order may have an attribute of type "total price" set equal to the value "US$87,750.00". Alternatively, the classification "capital equipment" may have an attribute of type "category of goods" set equal to the value "computer equipment".

The PDP must locate and retrieve resource attributes referenced by the applicable XACML policy instance. In the case where the resource is an XML document, it may do this by examining an attribute or element within the resource itself. In other cases, resource and classification attributes are asserted by authorities and distributed in the form of SAML attribute assertions. The PDP must check that the attribute values it operates upon are asserted by suitable authorities. This operation is described in Section 3.2.14, below.

3.2.10. Environment/attribute

The environment/attribute section of the language model is shown in gray in Figure 5.

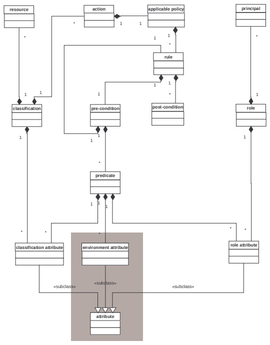


Figure 5 - Environment/attribute section of the language model

XACML policy instances may reference attributes that are not directly associated either with the principal or the resource. These are called environment attributes. For instance, the "current time of day" is an environment attribute that may be referenced by a policy instance. Environment attributes are asserted by authorities and distributed in the form of SAML attribute assertions. The PDP must check that the attribute values it operates upon are asserted by suitable authorities. This operation is described in "Attribute identification", below.

3.2.11. Policy/action/resource/classification

The policy/action/classification section of the language model is shown in gray in Figure 6.

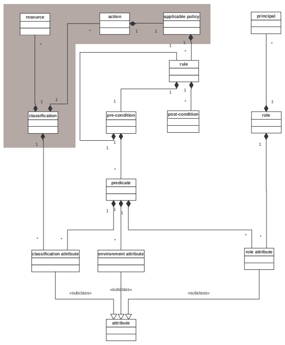


Figure 6 - Policy/action/resource/classification section of the language model

Policy instances are identified with a classification/action pair. In some cases the policy instance contains elements or attributes that identify the classification and action to which it is applicable. The PDP must check that the policy instance it uses to compute the authorization decision is applicable to the authorization request. It does this by verifying that the action identified in the authorization request is the same as the action identified in the policy instance, and that the resource identified in the authorization request belongs to the classification identified in the policy instance. How the PDP does this is described above.

3.2.12. Rule/pre-condition/predicate

The rule/pre-condition/predicate section of the language model is shown in gray in Figure 7.

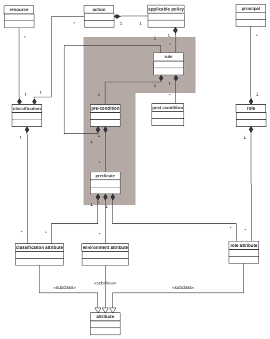


Figure 7 - Rule/pre-condition/predicate section of the language model

XACML policy instances are built from a logical combination of rules. Each rule comprises one pre-condition and zero or more post-conditions. A pre-condition is a logical operator or predicate. A predicate is a statement about attributes that can be verified by the PDP. If the policy instance applicable to an authorization request evaluates to TRUE, and all internal post-conditions are satisfied, then the PDP may return an authorization decision with the value TRUE to the PEP.

3.2.13. Post-condition

The post-condition section of the language model is shown in gray in Figure 8.

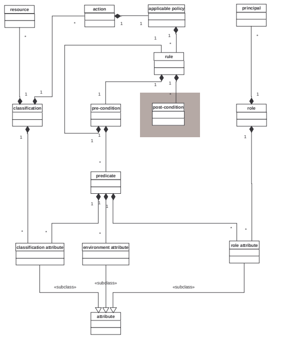


Figure 8 - Post-condition section of the language model

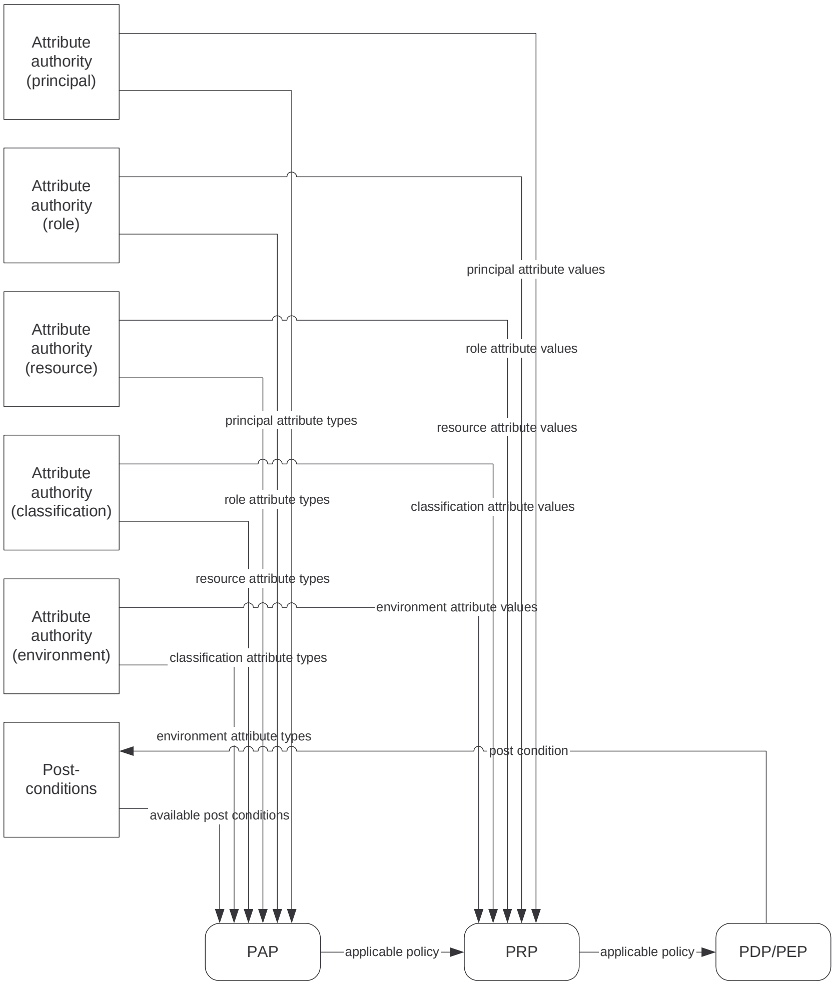
Post-conditions are actions specified in an XACML policy instance. Post-conditions are of two types. Internal post-conditions must be successfully executed prior to returning an authorization decision with the value TRUE. External post-conditions must be returned by the PDP to the PEP and an authorization decision with the value TRUE may be issued without confirmation that the condition has been successfully executed.

3.2.14. Attribute identification

Attribute specifiers are formed of two components: the first component identifies the authority for the attribute and the second component identifies the attribute type. For instance, the specifier may be an XPath expression in the form of a URI. The "host-name" component of the URI identifies the authority for the attribute, and the local-path component identifies the attribute type in terms of the structure of a SAML attribute assertion. In the case where a suitable attribute assertion is provided by the PEP in the decision request, the PDP identifies the appropriate assertion by comparing the host-name in the URI with the issuer field of the assertion. In the case where no suitable assertion is provided by the PEP, then the host-name component can be used to locate a suitable attribute authority to which to send a SAML attribute request.

3.3. Administrative model

It is essential that XACML policy instances only contain references to attributes and post-conditions that are accessible by the PDP or PEP. The administrative model, shown in Figure 9, illustrates how this is achieved. The various SAML attribute authorities involved must provide an interface by which the policy administration point can discover the attribute types available from it.


Attribute authority (principal)
Attribute authority (role)
Attribute authority (resource)
Attribute authority (classification)
Attribute authority (environment)
Post-conditions
principal attribute values
role attribute values
resource attribute values
classification attribute values
environment attribute values
principal attribute types
role attribute types
resource attribute types
classification attribute types
environment attribute types
post condition
available post conditions
PAP
applicable policy
PRP
applicable policy
PDP/PEP

428

429 **Figure 9 - Administrative model**

430

Chapter Two

4. Policy syntax

The information in this section is normative, with the exception of the schema fragments. SAML

Appendix A - Schema ***contains the normative version of the schema.***

5. Applicable policy

Applicable policy is the top-level element. It contains a description of the access to which the policy applies, in the form of "resource classification" and "resource action". It also contains the policy element. PDPs should use the applicability element to locate, retrieve and verify the policy required for processing a particular samlp:authorizationQuery. Verification means confirming that the value of the resourceActions element in applicable policy is equal to the value of the saml:Actions element in the samlp:authorizationQuery.

```
    <xs:element name="applicablePolicy">
        <xs:complexType name="scopedPolicy">
            <xs:sequence>
                <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 ref="policy"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

6. Policy

The policy element is an aggregation of rules. Rules must be combines with logical operations, not merely listed.

```
    <xs:complexType name="policy">
        <xs:sequence>
            <xs:element ref="rule" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
```

7. Rule

A rule consists of a pre-condition and zero or more post-conditions. If the pre-condition evaluates to TRUE and the internal post-conditions are successfully executed, then the PDP should return the "permit" value in the samlp:Response/StatusCode element. Otherwise, it must return the "deny" value.

```
    <xs:element name="rule">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="preCondition">
                    <xs:complexType>
                        <xs:choice>
                            <xs:element ref="and"/>
                            <xs:element ref="or"/>
                            <xs:element ref="not"/>
                            <xs:element ref="predicate"/>
                        </xs:choice>
                    </xs:complexType>
                </xs:element>
                <xs:element ref="postCondition" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

8. Pre-condition

The preCondition element is a predicate or logically-combined set of predicates.

8.1. And

The "And" pre-condition evaluates to TRUE if and only if all the predicate elements that it contains evaluate to TRUE.

```
<xs:element name="and">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="rule" minOccurs="2" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

8.2. Or

The "Or" pre-condition evaluates to TRUE if one or more of the predicate elements that it contains evaluate to TRUE.

```
<xs:element name="or">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="rule" minOccurs="2" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

8.3. Not

The "Not" pre-condition evaluates to TRUE if the predicate element that it contains evaluates to FALSE.

```
<xs:element name="not">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="rule"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

8.4. Predicate

The predicate element contains either one of the predicates defined here, or an external function.

```
<xs:element name="predicate">
    <xs:complexType>
```

```
        <xs:choice>
            <xs:element ref="present"/>
            <xs:element ref="equality"/>
            <xs:element ref="greaterOrEqual"/>
            <xs:element ref="lessOrEqual"/>
            <xs:element ref="subsetOf"/>
            <xs:element ref="supersetOf"/>
            <xs:element ref="nonNullSetIntersection"/>
            <xs:element ref="externalFunction"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

8.5. Present

The Present predicate evaluates to TRUE if the element referenced by it is populated.

```
<xs:element name="present">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="referencedData"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

8.6. Equality

The Equality predicate evaluates to TRUE if the two elements referenced by it are equal. Both elements must be of the same type.

```
<xs:element name="equality">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="referencedData"/>
            <xs:element ref="secondOperand"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

8.7. Greater or equal

The greaterOrEqual predicate evaluates to TRUE if the first element is greater than or equal to the second element. The elements must be of the same type, which may be string, normalizedString, byte, unsignedByte, base64Binary, hexBinary, integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, int, unsignedInt, long, unsignedLong, short, unsignedShort, decimal, float, double, time, dateTime, duration, date, gMonth, gYear, gYearMonth, gDay, gMonthDay, Name or Qname.

```
<xs:element name="greaterOrEqual">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="referencedData"/>
                <xs:element ref="secondOperand"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

8.8. Less or equal

The lessOrEqual predicate evaluates to TRUE if the first element is less than or equal to the second element. The elements must be of the same type, which may be string, normalizedString, byte, unsignedByte, base64Binary, hexBinary, integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, int, unsignedInt, long, unsignedLong, short, unsignedShort, decimal, float, double, time, dateTime, duration, date, gMonth, gYear, gYearMonth, gDay, gMonthDay, Name or Qname.

```
<xs:element name="lessOrEqual">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="referencedData"/>
            <xs:element ref="secondOperand"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

8.9. Sub-set of

The subSetOf predicate evaluates to TRUE if the value of the first element is amongst the set of values referenced by the second element.

```
<xs:element name="subsetOf">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="referencedData"/>
            <xs:element ref="secondOperand" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

8.10. Super-set of

The superSetOf predicate evaluates to TRUE if the set of values referenced by the first element includes all the value(s) of the second element.

```
<xs:element name="supersetOf">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="referencedData"/>
            <xs:element ref="secondOperand" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

8.11. Non-null Set Intersection

The nonNullSetIntersection predicate evaluates to TRUE if the set of values referenced by the two elements have at least one value in common.

```
<xs:element name="nonNullSetIntersection">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="referencedData"/>
            <xs:element ref="secondOperand" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

8.12. External function

The externalFunction element contains a definition of the interface to an external function. The external function is defined as a WSDL "definition" element for a "request-response" operation. The response must be a Boolean.

```
    <xs:element name="externalFunction" type="wsdl:definitions"/>
```

8.13. Post-condition

The postCondition element contains a definition of the interface to an external function. The external function is defined as a WSDL "definition" element for a "one-way" operation. Internal post conditions are expected to be performed by the PDP, and a "permit" statusCode must not be returned unless such conditions are successfully executed. External post conditions are expected to be performed by the PEP, and they must include them in the authorization decision. The PDP may return a "permit" rstatusCode without confirmation that such conditions have been successfully executed.

```

    <xs:element name="postCondition" type="wsdl:definitions">
        <xs:complexType name="">
            <xs:sequence>
                <xs:element name = "internalPostCondition" type:"wsdl definitions" minOccurs= "0"
maxOccurs="0"/>
                <xs:element name = "externalPostCondition" type:"wsdl definitions" minOccurs= "0"
maxOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

8.14. Referenced data

The referencedData element contains elements for attributes of the main model entities: principal, resource and environment.

```
    <xs:element name="referencedData">
        <xs:complexType>
            <xs:choice>
                <xs:element name="roleAttribute" type="attributeReference"/>
                <xs:element name="classificationAttribute" type="attributeReference"/>
                <xs:element name="environmentAttribute" type="attributeReference"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
```

8.15. Attribute reference

The "attribute reference" element is a pointer to an attribute. The pointer is in the form of a URI. It may contain an XPATH expression into a SAML attribute assertion for a principal, resource or environment. If the resource is an XML document, then it may contain an XPATH Expression identifying an element of the resource. If the URI does not indicate a SAML assertion passed to the PDP in the samlp:authorizationQuery, then the PDP should obtain the value from the attribute authority identified by the attribute reference.

```
    <xs:simpleType name="attributeReference">
        <xs:restriction base="xs:anyURI"/>
    </xs:simpleType>
```

8.16. Second operand

The second operand element is a choice between a referenced data element and a hard-coded value.

```
<xs:element name="secondOperand">
    <xs:complexType>
        <xs:choice>
            <xs:element ref="referencedData"/>
            <xs:element ref="hardcodedValue"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

8.17. Hard-coded value

The "hard-coded value" element contains a value written directly into the policy instance. Its type must be identical to that of any element with which it is paired in a predicate sub-element.

```
<xs:element name="hardcodedValue" type="xs:string"/>
```

9. References

SAML

Appendix A - Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:wsdl="http://www.schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:saml="http://www.oasis-
open.org/committees/security/docs/draft-sstc-scheam-assertion-19.xsd" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="applicablePolicy">
        <xs:complexType name="scopedPolicy">
            <xs:sequence>
                <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 ref="policy"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="policy">
        <xs:sequence>
            <xs:element ref="rule" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="rule">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="preCondition">
                    <xs:complexType>
                        <xs:choice>
                            <xs:element ref="and"/>
                            <xs:element ref="or"/>
                            <xs:element ref="not"/>
                            <xs:element ref="predicate"/>
                        </xs:choice>
                    </xs:complexType>
                </xs:element>
                <xs:element ref="postCondition" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="and">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="rule" minOccurs="2" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="or">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="rule" minOccurs="2" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="not">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="rule"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="predicate">
        <xs:complexType>
            <xs:choice>
                <xs:element ref="present"/>
                <xs:element ref="equality"/>
                <xs:element ref="greaterOrEqual"/>
                <xs:element ref="lessOrEqual"/>
                <xs:element ref="subsetOf"/>
                <xs:element ref="supersetOf"/>
                <xs:element ref="nonNullSetIntersection"/>
                <xs:element ref="externalFunction"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="present">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="referencedData"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="equality">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="referencedData"/>
                <xs:element ref="secondOperand"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="greaterOrEqual">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="referencedData"/>
                <xs:element ref="secondOperand"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="lessOrEqual">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="referencedData"/>
                <xs:element ref="secondOperand"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="subsetOf">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="referencedData"/>
                <xs:element ref="secondOperand" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="supersetOf">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="referencedData"/>
                <xs:element ref="secondOperand" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="nonNullSetIntersection">
        <xs:complexType>
            <xs:sequence>
```

```
                <xs:element ref="referencedData"/>
                <xs:element ref="secondOperand" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="externalFunction" type="wsdl:definitions"/>
    <xs:element name="postCondition" type="wsdl:definitions">
        <xs:complexType name="">
            <xs:sequence>
                <xs:element name = "internalPostCondition" type:"wsdl definitions" minOccurs= "0"
maxOccurs="0"/>
                <xs:element name = "externalPostCondition" type:"wsdl definitions" minOccurs= "0"
maxOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="referencedData">
        <xs:complexType>
            <xs:choice>
                <xs:element name="roleAttribute" type="attributeReference"/>
                <xs:element name="classificationAttribute" type="attributeReference"/>
                <xs:element name="environmentAttribute" type="attributeReference"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="attributeReference">
        <xs:restriction base="xs:anyURI"/>
    </xs:simpleType>
    <xs:element name="secondOperand">
        <xs:complexType>
            <xs:choice>
                <xs:element ref="referencedData"/>
                <xs:element ref="hardcodedValue"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="hardcodedValue" type="xs:string"/>
</xs:schema>
```

Appendix B - Test cases

The text in this appendix will be replaced by normative test cases. The test cases will comprise a SAML authorization request message, an XACML policy instance and the resulting SAML authorization response message.

Authorities and assertions.

Attribute authority issues attribute assertions. Resource authority issues resource assertions. Environmental authority issues environment assertions. Application authority issues application-specific assertions.

Attribute assertions.

```
<Assertion ...>
<Conditions.../>
<Advice.../>
<AttributeStatement>
      <Subject .../>
      <Attribute AttributeNamespace="..." AttributeName="weight">
               <AttributeValue>100</AttributeValue>
      </Attribute>
</AttributeStatement>
</Assertion>
```

Resource assertions.

```
<Assertion ...>
<Conditions .../>
<Advice .../>
<ResourceStatement>
      <Resource ResourceName="..." ResourceType="..."/>
      <Attribute AttributeNamespace="..." AttributeName="owner">
               <AttributeValue>superman</AttributeValue>
      <Attribute/>
      <Attribute AttributeNamespace="..." AttributeName="color">
               <AttributeValue>blue</AttributeValue>
      </Attribute>
</ResourceStatement>
</Assertion>
```

Request looks like this:

```
<Request ...>
       <AuthorizationQuery Resource="...">
                       <Subject .../>
                       <Actions .../>
                       <Evidence .../>
       </AuthorizationQuery>
</Request>
```

Expressions.
Expression consists of elementary conditions joined by logical ‘and’ or ‘or’ or grouped by ‘paren’.

```
<exp name=”…”>
```

```
      <cnd test=”…”/>
      <and/>
      <cnd test=”…”/>
</exp>
```

Name is an id-type attribute of expression element. To make it more compact we can assume <and> by default:

```
<exp name=”…”>
      <cnd test=”…”/>
      <cnd test=”…”/>
</exp>
```

Conditions can be enclosed in parens: a and (b or c)

```
<exp name=”…”>
      <cnd test=”…”/>
      <paren>
            <cnd test=”…”/>
            <or/>
            <cnd test=”…”/>
      </paren>
</exp>
```

Conditions “test” attribute is a Boolean over some xpath expression.

Here is a long form. Suppose we want to say that statement balance should be over 50 dollars. Statement balance is presented as a saml attribute assertion. Xpath expression is: //AttributeStatement/Attribute[@AttributeNamespace=’www.foo.com’][@AttributeName=’balance’]/AttributeValue=’50’

We can put it in condition:

```
<cnd
test=”//AttributeStatement/Attribute[@AttributeNamespace=’www.foo.com’][
@AttributeName=’balance’]/AttributeValue=’50’”/>
```

Macros.
To make above condition more readable we can define macros:

```
<macro name=”bal”
def=”//AttributeStatement/Attribute[@AttributeNamespace=’www.foo.com’][@
AttributeName=’balance’]/AttributeValue”/>
```

‘$’ sign applied to macro name denotes macro expansion. I do not know if it’s the best choice. We can use ‘#’ instead or something else. We also need to be able to escape macro-expansion symbol.

Then condition is:

```
<cnd test=”$bal=’50’/>
```

Macros can be reused with macros as well. For example we can define namespace macro and reuse it in attribute macros:

```
<macro name=”myns”
def=”//AttributeStatement/Attribute[@AttributeNamespace=’www.foo.com’”/>

<macro name=”bal” def=”$myns[@AttributeName=’balance’]/AttributeValue”/>
```

Rules.
Rule is in the form:

```
<allow action=”…”>
      <subject>
            expression reference, or conds.
      </subject>
      <resource name=”…”>
            expression reference, or conds.
      </resource>
      <if>
            expression reference, or conds.
      </if>
</allow>
```

Each component within a rule can include expression references or a set of conditions.

For example we can have expression for the good customers that we want to reference in the rule:

```
<exp name=”goodcust”>
      <cnd test=”$bal > ‘100’/>
      <cnd test=”$pmt > ‘15’/>
</exp>

<allow action=”…”>
      <subject>
            <exp name=”goodcust”/>
      </subject>
      <resource …/>
      <if …/>
</allow>
```

We can join expressions and augment them with conditions:

```
<allow action=”…”>
      <subject>
            <exp name=”goodcust”/>
            <exp name=”amexholder”/>
            <cnd test=”$birthmonth=’may’”/>
            <cnd test=”$birthyear=’1861’”/>
      </subject>
      <resource …/>
      <if …/>
</allow>
```

Reserved symbols.

Let Req-> refer to the current request.
Subj-> refer to the attribute assertion about requestor
Res-> refer to the attribute assertion about requested resource.

Can we write expressions for that? I’m not even sure that this is right.

```
<macro name=”Req->” def=”/Request/AuthorizationQuery/”/>

<macro name=”Subj->” def=”//AttributeStatement/
Attribute[preceding-sibling::Subject/NameIdentifier/@Name=
$Req->Subject/NameIdentifier/@Name]/”/>

<macro name=”Res->”
def=”//ResourceStatement/
Attribute[preceding-sibling::Subject/NameIdentifier/@Name=
$Req->Subject/NameIdentifier/@Name]/”/>
```

Roles and groups.
Predicate groupmember(groupname, subject) evaluates to true if subject is a member of a group.