Abstract:
This specification describes a profile for XACML 3.0 to enable it to express administration and delegation policies.

Status:
This document was last revised or approved by the eXtensible Access Control Markup Language (XACML) TC on the above date. The level of approval is also listed above. Check the “Latest Version” or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “Send A Comment” button on the Technical Committee’s web page at http://www.oasis-open.org/committees/xacml/.

Related work:
This specification is related to:
- eXtensible Access Control Markup Language (XACML) Version 3.0 Working Draft 11

Declared XML Namespace(s):
None
For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/xacml/ipr.php).

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/xacml/.
# Table of Contents

1 Introduction .................................................................................................................. 6  
1.1 Terminology ............................................................................................................. 6  
1.2 Glossary .................................................................................................................. 6  
1.3 Normative References .......................................................................................... 6  
1.4 Non-Normative References .................................................................................. 7  
2 Use Cases (non-normative) ....................................................................................... 8  
2.1 Administration/Delegation .................................................................................... 8  
2.1.1 Use case 1: Policy Administration ................................................................... 8  
2.1.2 Use case 2: Dynamic Delegation ..................................................................... 8  
2.1.3 Discussion ......................................................................................................... 8  
2.2 Only if X is permitted to do it ................................................................................ 8  
3 Solution Overview and Semantics (non-normative) .................................................. 10  
4 Processing Model ....................................................................................................... 11  
4.1 URIs ........................................................................................................................ 11  
4.2 Reserved Attribute Categories ............................................................................. 11  
4.3 Trusted policies ........................................................................................................ 11  
4.4 The context handler .............................................................................................. 11  
4.5 Administrative request generation during reduction ............................................ 12  
4.6 Policy set evaluation .............................................................................................. 12  
4.7 Forming the reduction graph ................................................................................ 13  
4.8 Reduction of “Permit” ........................................................................................... 13  
4.9 Reduction of “Deny” .............................................................................................. 13  
4.10 Reduction of “Indeterminate” ............................................................................... 14  
4.11 Maximum delegation depth ................................................................................ 14  
4.12 Obligations ............................................................................................................ 14  
5 Example (non-normative) ......................................................................................... 15  
6 Optimization (non-normative) ................................................................................... 24  
6.1 Optimization of Reduction ................................................................................... 24  
6.2 Alternative forms of delegation ........................................................................... 24  
7 Actions Other Than Create ...................................................................................... 25  
7.1 Revocation by the issuer ....................................................................................... 25  
7.2 Revocation by super administrators ................................................................... 25  
7.3 Revocation as an action under access control ..................................................... 25  
8 Security and Privacy Considerations (non-normative) ............................................ 26  
8.1 Dynamic Issuer Attributes .................................................................................... 26  
8.2 Enforcing Constraints on Delegation .................................................................... 26  
8.3 Issuer and delegate attributes .............................................................................. 27  
8.4 Denial of Service ................................................................................................... 27  
8.5 Obligations ............................................................................................................ 27  
9 Conformance .............................................................................................................. 28  
9.1 Delegation by reduction ....................................................................................... 28  
A. Acknowledgements ................................................................................................. 29  
B. Revision History ...................................................................................................... 30
Introduction

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2 Glossary

For simplicity, this document uses the term policy to include the XACML definitions for both policy and policy set.

The following terms are defined.

Access policy

A policy that governs access.

Access request

A request to determine whether access to a resource should be granted.

Administrative policy

A policy that authorizes a delegate to issue policies about constrained situations.

Administrative request

A request to determine whether a policy was issued by an authorized source.

Backward Chaining

Finding a chain of administrative and access policies beginning with an access policy, such that each policy is authorized by the next one.

Delegate

Someone authorized by an administrative policy to issue policies.

Forward Chaining

Finding a chain of administrative and access policies beginning at a trusted policy, such that each policy authorizes the next one.

Issuer

A set of attributes describing the source of a policy.

Reduction

the process by which the authority of a policy associated with an issuer is verified or discarded.

Situation

A set of properties delineated by the <Attributes> elements of an access request context.

Trusted policy

A policy without a <PolicyIssuer> element.

1.3 Normative References


1.4 Non-Normative References

None
2 Use Cases (non-normative)

This specification is intended to support the following use cases.

2.1 Administration/Delegation

2.1.1 Use case 1: Policy Administration

Policy administration controls the types of policies that individuals can create and modify. Typically, different individuals would be allowed to create policies about certain sets of resources. Alternatively, administration might be divided up by action type, subject or some other properties.

In XACML 2.0 the question of the circumstances under which policies can be created is out of scope. It essentially says that some policies exist which the PDP will use.

2.1.2 Use case 2: Dynamic Delegation

Dynamic delegation permits some users to create policies of limited duration to delegate certain capabilities to others. XACML 2.0 allows policies that say, “Mary can do something on behalf of Jack” by means of different subject-categories. But, it would be useful to allow people to generate policies on the fly that say such things as “while I am on vacation, Mary can approve requests.” This requires the ability to create policies that control the policies that can be created.

2.1.3 Discussion

In meeting these two use cases, it is NOT desirable to require either of the following to always be true:

1. Anything you can do, you can delegate to someone else to do.
2. If you can delegate something, you can always do it yourself by generating the necessary policy that applies to you.

It should be possible to create policies that enable #1 and/or #2, but they should not be "wired in."

The main difference between use cases #1 and #2 is how policies are accessed. In #1, most likely policies will be found in some repository or set of repositories. There will be some simple enforcement mechanism that says that the issuer of one policy must correspond to the person who created or modified the other policy. In #2, policies might need to be carried in application requests or accessed dynamically via some back channel. In this case, signatures, or some other such mechanism, would be used to verify the issuer’s identity.

Note that in both cases, having a policy from Fred, signed by Fred does not mean the policy will be enforced. It merely means that it will be considered as a candidate. It is still necessary to authorize Fred’s policy for it to be enforced.

It is also desirable to arrange for policy evaluation to be optimized by doing as much work prior to access time as possible. It should be possible to “flatten” policy chains to an equivalent form using whatever policies are at hand.

Support for administration/delegation should not reduce the existing functionality of XACML 2.0

2.2 Only if X is permitted to do it

Consider the common use case: Mary is the manager and approves expense reports for her department. When she is on vacation, Jack can approve expense reports.

We need a convenient way to say "Jack is allowed to do such and such, but only if Mary is allowed to do it". Mary might or might not be the issuer of this policy. In plain XACML, there is no way to do this except by duplicating the rules that apply to Mary.
In other words, we need a way to replace the access-subject in the request context with a specified subject, call the entire policy evaluation process and if the result is “Permit”, then return a value of “True.”
3 Solution Overview and Semantics (non-normative)

The purpose of the delegation model is to make it possible to express permissions about the right to issue policies and to verify issued policies against these permissions.

A policy may contain a <PolicyIssuer> element that describes the source of the policy. A missing <PolicyIssuer> element means that the policy is trusted.

A trusted policy is considered valid and its origin is not verified by the PDP.

Policies which have an issuer need to have their authority verified. The essence of the verification is that the issuer of the policy is checked against the trusted policies, directly or through other policies with issuers. During this check the right of the issuer to issue a policy about the current access request is verified.

If the authority of the policy issuer can be traced back to the trusted policies, the policy is used by the PDP, otherwise it is discarded as an unauthorized policy. The authority of the issuer depends on which access situation the current access request applies to, so a policy can be both valid and invalid depending on the access request.

Steps in the validation process are performed using a special case XACML requests, called administrative requests, which contain information about the policy issuers and the access situation.
4 Processing Model

4.1 URIs

urn:oasis:names:tc:xacml:3.0:delegation:decision

The identifier which MUST be used for the attribute indicating which type of decision is being reduced.

4.2 Reserved Attribute Categories

urn:oasis:names:tc:xacml:3.0:attribute-category:delegate

This attribute category MUST be used in administrative requests to carry the attributes of the issuer of the policy which is being reduced.

urn:oasis:names:tc:xacml:3.0:attribute-category:delegation-info

This attribute category MUST be used in administrative requests to carry information about the reduction in progress, such as the decision being reduced.

urn:oasis:names:tc:xacml:3.0:attribute-category:delegated:<anyURI>

Categories starting with this and ending with any URI MUST be used to carry information about the situation which is being reduced.

4.3 Trusted policies

In case there is no <PolicyIssuer> element in the policy or policy set, the policy or policy set MUST be trusted and no reduction of the policy will be performed.

4.4 The context handler

The attributes contained in an explicit <Attributes> element with Category “urn:oasis:names:tc:xacml:3.0:attribute-category:delegate” MAY be complemented with additional attributes by the context handler, as is the case with the other elements in the request context.

A dynamic issuer attribute is an attribute of an issuer/delegate such that the attribute value may have changed since the policy was issued. The time at which attributes are resolved is important for dynamic delegate attributes. The PDP and context handler MUST operate in either “current issuer/delegate attribute mode” or “historic issuer/delegate attribute mode” but not in both.

- Current attributes mode

  In current attribute mode, when a delegate attribute is dynamic, the value of the attribute MUST be used as it is at the time of the access request being processed.

- Historic attributes mode

  In historic attribute mode, when a delegate attribute is dynamic, the value of the attribute MUST be used as it was at the time when the policy, from which the delegate was derived, was issued.

These rules MUST apply to both attributes that appear in the <PolicyIssuer> element and the attributes that are retrieved by the context handler, which means that in case of the current attribute mode dynamic issuer attributes MUST NOT be present in the <PolicyIssuer> element.

See also the security considerations discussion related to this in section 8.1.
4.5 Administrative request generation during reduction

Reduction is the process by which the authority of policies is established. Reduction is performed as a search in a graph. This section explains how a single administrative request is created to determine an edge in the reduction graph. Reduction is always performed in the context of a request $R$, which is being evaluated against a policy set.

Given a potentially supported policy, $P$, and the request $R$, an administrative request, $A$, generated based on $R$ by the following steps:

1. The <Attributes> elements of $R$ are mapped to <Attributes> elements in $A$ according to the following:
   a. An <Attributes> element with Category equal to "urn:oasis:names:tc:xacml:3.0:attribute-category:delegate" in $R$ has no corresponding part in $A$.
   b. An <Attributes> element with Category which starts with the prefix "urn:oasis:names:tc:xacml:3.0:attribute-category:delegated:" maps to an identical <Attributes> element.
   c. An <Attributes> element with Category equal to "urn:oasis:names:tc:xacml:3.0:attribute-category:delegation-info" in $R$ has no corresponding part in $A$. (Note, a new delegation-info category is created, see point 3 below.)
   d. An <Attributes> element with any other Category maps to an <Attributes> element with the Category prefixed with "urn:oasis:names:tc:xacml:3.0:attribute-category:delegated:" and identical contents.


3. $A$ contains an <Attributes> element with Category equal to "urn:oasis:names:tc:xacml:3.0:attribute-category:delegation-info" and the following contents:
   a. An <Attribute> element with AttributeId equal to "urn:oasis:names:tc:xacml:3.0:delegation:decision", DataType equal to "http://www.w3.org/2001/XMLSchema#string", and the value equal to the decision which is being reduced, that is either "Permit" or "Deny". (See section 4.7 for explanation on how this value is set.)

4.6 Policy set evaluation

This delegation profile defines how policy sets are evaluated in the presence of policies with issuers. A PDP implementing this profile MUST perform policy set evaluation according the following process or a process that produces an identical result in all cases. Note that the regular policy set evaluation according to [XACML] is a special case of this process as long as no policy has an issuer.

The evaluation of a policy set is done as in [XACML], with the exception that the contained policies are possibly reduced and/or discarded, before combination, as defined by the following table.

<table>
<thead>
<tr>
<th>Value of evaluated policy</th>
<th>Policy Issuer</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Don’t care</td>
<td>Absent</td>
<td>The value is combined as it is.</td>
</tr>
<tr>
<td>“Permit”, “Deny” or “Indeterminate”</td>
<td>Present</td>
<td>The value is reduced as defined in sections 4.8, 4.9 and 4.10 respectively and possibly discarded before combination.</td>
</tr>
</tbody>
</table>
After the above actions have been performed, the remaining trusted policy values determine the value of the policy set as defined in [XACML].

4.7 Forming the reduction graph

The reduction process is a graph search where the nodes of the graph are the policies in a policy set and the edges represent how the policies authorize each other.

The nodes of the reduction graph are the policies of the policy set.

There are four kinds of directed edges in the graph: Types PP, PI, DP and DI.

Note (non-normative): Informally, the PP and DP edges are used to indicate whether a policy authorizes delegation of “Permit” and “Deny” respectively. The PI and DI edges are used to propagate “Indeterminate” results from administrative policies into the final result. It is important to propagate “Indeterminate” results since failing to detect an error can result in the wrong decision being implemented by the PEP.

To generate the edges of the reduction graph

1. For each ordered pair of policies in the policy set \((P_1, P_2)\), generate an administrative request \(A\) reducing “Permit” based on \(P_1\) and the request being evaluated against the policy set.
   a. Evaluate \(A\) against \(P_2\).
   b. If and only if the result is “Permit”, there is a PP edge from \(P_1\) to \(P_2\).
   c. If and only if the result is “Indeterminate”, there is a PI edge from \(P_1\) to \(P_2\).
2. For each ordered pair of policies in the policy set \((P_1, P_2)\), generate an administrative request \(A\) reducing “Deny” based on \(P_1\) and the request being evaluated against the policy set.
   a. Evaluate \(A\) against \(P_2\).
   b. If and only if the result is “Permit”, there is a DP edge from \(P_1\) to \(P_2\).
   c. If and only if the result is “Indeterminate”, there is a DI edge from \(P_1\) to \(P_2\).

4.8 Reduction of “Permit”

A policy, \(P\), which evaluated to “Permit” in the policy set, MUST be reduced as follows in this section.

Form a reduction graph as described in section 4.7.

Start a graph search from the node corresponding to the policy to be reduced. Follow only PP edges. If it is possible to reach a node which corresponds to a trusted policy, the policy \(P\) is treated as “Permit” in combination of the policy set.

If it was not possible to reach a trusted policy, do a second graph search, following PP and PI edges. If it possible to reach a trusted policy in this manner, the policy \(P\) is treated as “Indeterminate” in combination of the policy set.

If it was not possible to reach a trusted policy with either search, the policy \(P\) is discarded and not combined in the policy set.

In all graph searches, the maximum delegation depth limit MUST be checked as described in section 4.11.

In all graph searches obligations must be collected as described in section 4.12.

4.9 Reduction of “Deny”

A policy, \(P\), which evaluated to “Deny” in the policy set, MUST be reduced as follows in this section.

Form a reduction graph as described in section 4.7.
Start a graph search from the node corresponding to the policy to be reduced. Follow only DP edges. If it is possible to reach a node which corresponds to a trusted policy, the policy \( P \) is treated as “Deny” in combination of the policy set.

If it was not possible to reach a trusted policy, do a second graph search, following DP and DI edges. If it possible to reach a trusted policy in this manner, the policy \( P \) is treated as “Indeterminate” in combination of the policy set.

If it was not possible to reach a trusted policy with either search, the policy \( P \) is discarded and not combined in the policy set.

In all graph searches, the maximum delegation depth limit MUST be checked as described in section 4.11.

In all graph searches obligations must be collected as described in section 4.12.

4.10 Reduction of “Indeterminate”

A policy, \( P \), which evaluated to “Indeterminate” in the policy set, MUST be reduced as follows in this section.

Form a reduction graph as described in section 4.7.

Start a graph search from the node corresponding to the policy to be reduced. Follow PP and PI edges.

If it is possible to reach a node which corresponds to a trusted policy, the policy \( P \) is treated as “Indeterminate” in combination of the policy set.

If it was not possible to reach a trusted policy, do a second graph search, following DP and DI edges. If it possible to reach a trusted policy in this manner, the policy \( P \) is treated as “Indeterminate” in combination of the policy set.

If it was not possible to reach a trusted policy with either search, the policy \( P \) is discarded and not combined in the policy set.

In all graph searches, the maximum delegation depth limit MUST be checked as described in section 4.11.

In all graph searches obligations must be collected as described in section 4.12.

Note (non-normative): This process is designed in this way because it is important to reduce “Indeterminate” results before combining them. An unauthorized “Indeterminate” can be used as an attack by forcing the PEP into error handling, and possibly denying or allowing access depending on the bias of the PEP. Intuitively we test if the policy would be authorized if it would have been “Permit” or “Deny”. If neither a “Permit” nor a “Deny” would have been authorized, the policy is not authorized, so the “Indeterminate” is discarded.

4.11 Maximum delegation depth

A policy or policy set MAY contain an XML attribute called MaxDelegationDepth, which limits the depth of delegation which is authorized by the policy. During the searches in the reduction graph, a path MUST be aborted if the number of nodes on the path exceeds the integer value of this attribute. The node count on the path includes the initial node which is being reduced, but does not include the node corresponding to the policy with the MaxDelegationDepth attribute being checked.

4.12 Obligations

Obligations in the access policies that have been reduced and are being combined are treated exactly as in [XACML]. Administrative policies may contain obligations but the obligations apply to the access decision, not the administrative decisions. All obligations that are found in policies that are used to reduce an access policy are treated as if they would have appeared in the access policy.

Due to security concerns with obligations, a PDP MAY refuse to load a policy with an obligation it does not recognize. Also, see Section 8.5 for security considerations concerning obligations.
5 Example (non-normative)

The following example policy set is used for illustrating the processing model.

```xml
<PolicySet PolicySetId="PolicySet1"
    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
    <Target/>
    <Policy PolicyId="Policy1"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue
                            DataType="http://www.w3.org/2001/XMLSchema#string">
                            employee
                        </AttributeValue>
                            AttributeId="group"
                            DataType="http://www.w3.org/2001/XMLSchema#string"/>
                    </Match>
                </AllOf>
                <AnyOf>
                    <AllOf>
                        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                            <AttributeValue
                                DataType="http://www.w3.org/2001/XMLSchema#string">
                                printer
                            </AttributeValue>
                                AttributeId="urn:oasis:names:tc:xacml:1.0:resource:id"
                                DataType="http://www.w3.org/2001/XMLSchema#string"/>
                        </Match>
                    </AllOf>
                </AnyOf>
                <AnyOf>
                    <AllOf>
                        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                            <AttributeValue
                                DataType="http://www.w3.org/2001/XMLSchema#string">
                                print
                            </AttributeValue>
                                AttributeId="urn:oasis:names:tc:xacml:1.0:action:id"
                                DataType="http://www.w3.org/2001/XMLSchema#string"/>
                        </Match>
                    </AllOf>
                </AnyOf>
                <AnyOf>
                    <AllOf>
                        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                            <AttributeValue
                                DataType="http://www.w3.org/2001/XMLSchema#string">
                                Carol
                            </AttributeValue>
                            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:delegate"
                                AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                                DataType="http://www.w3.org/2001/XMLSchema#string"/>
                        </Match>
                    </AllOf>
                </AnyOf>
            </Target>
        </Policy>
    </Policy>
</PolicySet>
```
<Policy PolicyId="Policy2"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permitoverrides">
  <PolicyIssuer>
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>Carol</AttributeValue>
    </Attribute>
  </PolicyIssuer>
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">
            employee</AttributeValue>
            AttributeId="group"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Match>
      </AllOf>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">
              printer</AttributeValue>
              AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </Match>
        </AllOf>
      </AnyOf>
    </AnyOf>
  </Target>
  <Rule RuleId="Rule2" Effect="Permit">
    <Target/>
  </Rule>
</Policy>

<Policy PolicyId="Policy3"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permitoverrides">
  <PolicyIssuer>
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>Bob</AttributeValue>
    </Attribute>
  </PolicyIssuer>
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">
            print</AttributeValue>
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">
            Bob</AttributeValue>
            AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
  <Rule RuleId="Rule2" Effect="Permit">
    <Target/>
  </Rule>
</Policy>
The policy set contains four policies. Policy 1 is a trusted policy since it has no issuer. The target with the standard attribute categories for the subject, resource and action constrain the situation that the policy applies to. The policy could have defined additional constraints on the environment target or by conditions or by rule targets. In this case the policy allows granting policies about any situation which is an employee who prints on the printer. Since there are <Match> elements with delegated categories in the policy target, Policy 1 is an administrative policy. In this case the policy allows for Carol to create any policy which allows a situation that is also allowed by Policy 1, that is, Carol can give access to the printer to any employee. Since there is no limit on the delegation depth, Carol can also create an administrative policy over these situations.

Policy 2 is issued by Carol as is indicated by the <PolicyIssuer> element. The allowed situations are again that an employee prints on the printer. Again, since there are <Match> elements with delegated categories, Policy 2 is an administrative policy. In this case Bob is granted the right to issue policies granting access to situations that are allowed by Policy 2.

Policy 3 is issued by Mallory, as is indicated by the <PolicyIssuer> element. The <Match> elements are on non-delegated categories, so it is an access policy. It grants access to the printer for Alice. As we will see later on, this policy is unauthorized since Mallory has not been authorized to allow access for this situation (Alice accessing the printer).

Policy 4 is issued by Bob as is indicated by the <PolicyIssuer> element. There are no delegated categories, so it is an access policy. It grants access to the printer for Alice.

We start with the following example access request. The request indicates that Alice is trying to access the printer. In this case Alice is also associated with the employee group attribute.
The request is evaluated against the policies in the policy set. The request will not match the targets in Policy 1 or Policy 2 since there are no delegated categories in the request. Both Policy 3 and Policy 4 will evaluate to “Permit” since the targets match directly. This is illustrated in the following figure.

Policy 3 and Policy 4 need to be reduced since they are not trusted.

As specified in the processing model, reduction consists of two steps. First a reduction graph is built, and then the PDP searches the graph for a path to the trusted policies for each policy with an issuer. Note that this example follows the definition of the processing model and does not attempt to be efficient. An efficient PDP can mix edge creation and path searching so that only those edges which are actually needed are created. This example does not do so for simplicity and we create a full graph before we do a search.

So, we begin by creating the reduction graph. Creating the reduction graph means finding any edges between the policies in the policy set. We need to check each pair of policies for an edge (although in practice a PDP may optimize the search to find a minimum set of edges as needed to determine the result). First, consider the question whether there is any edge between Policy 4 and Policy 2:
As defined by the processing model, there is an edge if and only if the administrative request generated from policy 4 evaluates to Permit (or Indeterminate) for policy 2. So to test for an edge, we create the following administrative request, and evaluate it against Policy 2:

```xml
<Request>
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>Alice</AttributeValue>
    </Attribute>
    <Attribute
      AttributeId="group"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>employee</AttributeValue>
    </Attribute>
  </Attributes>
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>printer</AttributeValue>
    </Attribute>
  </Attributes>
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>print</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:delegate">
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>Bob</AttributeValue>
    </Attribute>
    <Attribute
      AttributeId="group"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>administrator</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:delegation-info">
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:3.0:delegation:decision"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>true</AttributeValue>
    </Attribute>
  </Attributes>
</Request>
```
The administrative request is created based on the request being evaluated against the whole policy set and the issuer of Policy 4, that is, Bob. The subject, resource and action from the access request in Listing 2 are transformed into delegated subject, resource and action in the administrative request in Listing 3 and the issuer of Policy 4 becomes the delegate of the administrative request. We perform the request with a permit decision initially.

The interpretation of the administrative request is “Is Bob allowed to create a policy that concerns access to the printer for Alice?” In this case we also filled in the attribute representing membership in the administrators group for Bob in the request context. This represents the fact that the context handler can fill in attributes in the request context. (The details of how the context handler found the administrator attribute depend on the PDP implementation and the available attribute sources in the particular implementation.)

The request will evaluate to “Permit” on Policy 2. This means that there is a PP edge from Policy 4 to Policy 2, which represents that Policy 2 authorizes Policy 4 for a “Permit” decision on the particular situation. To test for a DP edge, another administrative request is created and evaluated. This request will have the same contents as the first one, except for a “Deny” decision in the delegation-info category. (The request is not shown here. Also note that since policy 4 evaluated to “Permit”, the DP edge is not really needed, although it is specified in the definition of the graph, so this request could be skipped by an optimizing PDP.) This will also evaluate to “Permit”, so there is a DP edge as well. (It would have been possible for Policy 2 to include a condition so it would only allow a “Permit” decision, but this is not the case here.)

We have now established the edges going from Policy 4 to Policy 2. Next, we test for edges from Policy 2 to Policy 1.

To test for PP and PI edges from Policy 2 to Policy 1, the following administrative request is generated:

```xml
<Request>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>Alice</AttributeValue>
    </Attribute>
  </Attributes>
</Request>
```
Listing 4 The administrative request for detecting edges from policy 2 to policy 1.

Again, the subject, resource and action are copied from Listing 2 in to Listing 4 as delegated subject, resource and action and the issuer of policy 2, Carol, becomes the delegate of Listing 4. (In this case Carol is not a member of the administrator group so the context handler has not added such an attribute to Carol in this request.) This request and a corresponding request with a "Deny" decision evaluate to "Permit", so we have found PP and DP edges. It remains to test the remaining combinations of nodes. These tests are not shown here to conserve space, but the end result will be a graph like this:
This is the full reduction graph for the example.

The second step of the PDP is now to find paths to the trusted policies from policies 3 and 4, which were the applicable policies to the original access request. In the graph we can see that there is a PP edged path to a trusted policy for Policy 4, so the Permit from Policy 4 is combined. There is no path for Policy 3, so policy 3 is disregarded. Policy 2 is not applicable and is not trusted, so it is also discarded. Policy 1 remains since it is trusted, although it is not applicable. We have the following:

These policies are combined as usual, which in this case leads to a “Permit” for the policy set in whole.
6 Optimization (non-normative)

6.1 Optimization of Reduction

When *administrative policies* are simple and few in number, the previous process can be executed as written. However, when *policies* are numerous, preprocessing will help improve performance at access time. The following strategies may be employed.

- **Eliminate unauthorized policies**
  Eliminating *administrative policies* for which there is no chain back to the *trusted policies* will greatly reduce the processing required at access time by eliminating backtracking. This works when *policies* are drawn exclusively from a repository. When *policies* may be presented dynamically at access time, it will be useful to limit what *policies* can be presented. For example, dynamic *policies* might be restricted to being only *access policies* or either access or leaf *administrative policies*. If root *policies* can be presented dynamically, then it will not be possible to perform this processing in advance.

- **Flatten delegation chains**
  When a chain can be found from the *trusted policies* to a particular *access policy*, then a derived *trusted policy*, with the same allowed *situations* and effect value can be substituted for the original *access policy*.

- **Split policies**
  It may be possible to split a *policy* into two (or more) simpler ones. For example, when a *policy* contains a disjunctive condition, it will be equivalent to two distinct *policies* each containing one of the alternatives, with the same effect value. The benefit of doing this is that it may then be possible to eliminate or flatten one of the derived *policies*.

- **Creating graph edges only as needed**
  Typical *reduction* graphs are likely sparse, so rather than testing each pair of nodes, it may be more efficient to test for new edges as new nodes are reached with existing edges.

These optimizations may be done by *backward chaining*, *forward chaining* or both.

One of the main obstacles to performing these optimizations will be the lack of information about *situation* attributes in advance of access time so it will be possible to tell which *situation* constraint subsumes another. In particular implementations or applications the *policies* may have restricted forms, so the *situation* constraints are directly comparable or extra knowledge of attributes is available, such that comparisons between *situation* constraints can be made.

Since the *delegate* plays a particularly crucial role, and since the number of parties who are allowed to be *policy issuers* will typically be small compared to the total user population, it may be worthwhile to arrange that the authoritative source of these attributes be made available when doing optimizations.

6.2 Alternative forms of delegation

*XACML* *policies* are written in terms of attributes. This means that another way to achieve delegation, is to delegate attribute assignment, rather than XACML *policies*. Which is more efficient depends on the particular use case requirements.

For instance, if relatively few general rules can be used to express *policies*, and the requirement of delegation is to assign to whom these rules apply, delegation of attribute assignment may be more appropriate.

In contrast, for instance, if there are no general rules, and access permissions need to combine resources from many different authorities, the delegation model described in this profile may be ideal.
7 Actions Other Than Create

An administrative policy allows policies to be created by delegates. What about other operations on policies, such as Update and Delete?

Update (modify) can be treated as a Delete followed by a Create. In the case where policies are signed by the policy issuer, this is literally true.

This profile does not specify a particular model for policy deletion (revocation of policies). An implementation MAY specify a model for policy deletion and may therefore disregard policies during processing. Revoked policies MAY also be removed from the policy repository, in which case they will not be seen by the PDP.

The following sections suggest some models for revocation which MAY be used. They are all optional and other models MAY be used as well.

7.1 Revocation by the issuer

One possible revocation model which may be implemented is that the issuer of a policy is the one who is authorized to remove it. How the issuer of the revocation is authenticated and how the effect of revocation is implemented is not specified by this profile.

7.2 Revocation by super administrators

One possible revocation model which may be implemented is that super administrators of the PDP (or policy repository) may remove any policy at their discretion.

7.3 Revocation as an action under access control

One possible revocation model is that access to the policy repository is controlled by XACML (or some other policy language) and removal of a policy is an action which can be performed. In this case the policy or the policy repository is modeled as a resource and the revocation as an action.
8 Security and Privacy Considerations (non-normative)

8.1 Dynamic Issuer Attributes

In case the attributes of an issuer may change with time, the choice of the point in time used for resolving them may affect the outcome of administrative requests. The PDP MUST treat this consistently and choose to operate in either historic or current issuer attribute mode. Policy writers need to be aware of the mode in which the PDP will operate.

Also in some environments it may be problematic to resolve old attributes and/or to reliably know at which time a policy was issued without special measures such as trusted time stamp authorities.

8.2 Enforcing Constraints on Delegation

This profile allows for defining a maximum depth for delegation. Implementers and users should be aware that this constraint cannot be enforced in the strict sense. It may be possible for someone with access rights to “delegate” that access right to anyone else “off-line” by just performing any operation himself on the behalf of the other person. However, in many applications these kinds of constraints can still be useful since they limit how the policies may evolve and indicate to users what policy is, and thus probably limiting casual policy violations.

Implementers should also be aware of that if there are nested issuers in a policy set, then the delegation that goes inside the outermost issuer is not visible to the outermost level of reduction. This means that constraints on delegation depth have no effect on the nested issuers. See the following figure for an example:

---

**Access Request**
Subject: Alice
Resource: Printer

**PolicySet A**
Issuer: Trusted
Target: Any

**Policy B**
Issuer: trusted
Target:
  Delegate: Bob
  DelegatedSubject: Alice
  DelegatedResource: Printer
Maxdepth: 2

**PolicySet C**
Issuer: Bob
Target:
  Subject: Alice
  Resource: Printer

**Policy D**
Issuer: trusted
Target:
  Delegate: Carol
  DelegatedSubject: Alice
  DelegatedResource: Printer

**Policy E**
Issuer: Carol
Target:
  Subject: Alice
  Resource: Printer
During evaluation, reduction will be performed inside policy set C, where policy D will support policy E. This reduction is not visible outside policy set C. The maximum depth condition in policy B has no effect on the reduction which goes on inside policy set C. If you wish to use a maximum depth constraint, you must collect delegated policies at a single level of nesting in a policy set.

8.3 Issuer and delegate attributes

An implementation must take care to authenticate the contents of <PolicyIssuer> elements before the policies are included in the PDP. It is the responsibility of the entity issuing a policy set to verify that the attributes of all issuers of the immediately contained policies are correct. As a special case, it is the responsibility of the PDP owner to verify all issuers of the policies in the PDP at the PDP policy set level.

If the context handler provides additional attributes of delegates, naturally, the context handler must have verified their correctness.

A special case of issuer attribute verification is when the <PolicyIssuer> element is dynamically created when the policy is loaded from storage into the PDP. In this case the <PolicyIssuer> element could for instance be based on a digital signature on the policy in the storage.

8.4 Denial of Service

If an attacker can insert policies into the repository, even if the issuers of the policies would not be trusted and the policy could not be traced to a trusted source, it may be possible, depending on the implementation, for the attacker to draft policies such that there will be a lot of computation during request evaluation. This could degrade performance and result in denied or reduced service. An implementation must take this in consideration.

On case of such intensive computation is if the attacker is able to draft policies which contain complex conditional expressions.

Another identified attack is to create nested policy sets which contain policies which need to be reduced. Since creation of the reduction graph in worst case means that every policy will be evaluated twice, by nesting reduction in policy sets, the number of times the deepest policies will be evaluated will increase exponentially with the depth of the policy set nesting. Possible protections against this attack include dynamic detection of it, not accepting policies with nested policy sets which need reduction and doing reduction graph generation by forward chaining, and not evaluate those policies which are not reached from the trusted policies.

8.5 Obligations

When access policies containing obligations are combined, an obligation from a policy will be included in the result, even if there is a policy evaluating to the same result but which does not contain the obligation. In a setting with decentralized administration where policies are issued by multiple issuers, this may in some cases be undesirable behavior. Depending on the nature of the obligation an obligation could be seen as an additional restriction to the access right. By adding an obligation to a policy, one issuer can in effect restrict the authority of another issuer. In particular, by including an obligation that is intentionally unrecognizable by the PEP, one issuer can completely deny the access that another issuer has granted.

When delegated XACML is used in an application, these issues must be considered. One possible solution is to allow only certain kinds of obligations. Another solution is to allow use of obligations only in the trusted policies.
9 Conformance

9.1 Delegation by reduction

An implementation conforms to this specification if it performs evaluation of XACML as specified in sections 4 and 7 of this document. The following URI identifies this functionality:

urn:oasis:names:tc:xacml:3.0:profile:administration:reduction
A. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Anthony Nadalin, IBM
Bill Parducci
Erik Rissanen, Axiomatics AB
Hal Lockhart, Oracle
Rich Levinson, Oracle
Seth Proctor, Sun
## B. Revision History

[optional; should not be included in OASIS Standards]

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Editor</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>WD 01</td>
<td>22 Mar 2005</td>
<td>Hal Lockhart</td>
<td>Initial working draft.</td>
</tr>
<tr>
<td>WD 02</td>
<td>8 Apr 2005</td>
<td>Tim Moses</td>
<td>Added PolicyIssuerMatch to <code>&lt;Target&gt;</code> element. Added delegation depth control.</td>
</tr>
<tr>
<td>WD 03</td>
<td>20 Apr 2005</td>
<td>Tim Moses</td>
<td>Added a pseudo-code description of the processing model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added schema for the request context.</td>
</tr>
<tr>
<td>WD 04</td>
<td>22 Apr 2005</td>
<td>Tim Moses</td>
<td>Added a plain-language description of the processing model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Modified <code>&lt;PolicyIssuerMatch&gt;</code> syntax and changed name to “delegates”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Made <code>&lt;PolicyIssuer&gt;</code> mandatory and included a URI for “root”</td>
</tr>
<tr>
<td>WD 05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WD 06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WD 07</td>
<td>5 Jul 2005</td>
<td>Erik Rissanen</td>
<td>Added missing parts and corrected incorrect parts of the schema fragments.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Clarified descriptive text. Added some new definitions in the terminology list.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fixed formatting.</td>
</tr>
<tr>
<td>WD 08</td>
<td>15 Aug 2005</td>
<td>Erik Rissanen</td>
<td>Improvements of the text, figures and formatting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Improved consistency and terminology.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fill in details, simplify and improve the processing model.</td>
</tr>
<tr>
<td>WD 09</td>
<td>13 Sep 2005</td>
<td>Erik Rissanen</td>
<td>Changed the definition of “situation”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added max delegation depth to the processing model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added obligations to the processing model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added some security considerations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Changed IndirectDelegateDesignator to IndirectDelegatesCondition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added the possibility for a target to match both access and administrative requests.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Other improvements and corrections.</td>
</tr>
<tr>
<td>WD 10</td>
<td></td>
<td>Erik Rissanen</td>
<td>Removed the term <strong>untrusted issuer</strong>. It was confusing since it really meant “issuer not trusted yet”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fixed some errors in the schema fragments</td>
</tr>
</tbody>
</table>
and the example.

Removed the `<Policies>` element from the request. It will be placed in the SAML profile instead.

Added historic/current attribute modes to the normative text.

Made the effect part of the situation in order to support deny at the access level.

Misc editing and fixing.

| WD 11 | 18 Jun 2006 | Erik Rissanen | Misc editing and corrections. |
|       |             |               | Added description for the context `<Decision>` element. |
|       |             |               | Added updated description of the access permitted function. |
|       |             |               | Disallow even the trusted issuer to issue negative administrative decisions. |

| WD 12 | 25 Jul 2006 | Erik Rissanen | Corrected typos. |
|       |             |               | Added section with additions to the SAML profile of XACML. |

| WD 13 | 4 Oct 2006  | Erik Rissanen | Updated to new OASIS document template. |

| WD 14 | 5 Oct 2006  | Erik Rissanen | Fixed typos, formatting and clarified the text in multiple places. |
|       |             |               | Removed statement in solution overview which stated that the policy which the PDP starts with by definition is issued by the trusted issuer. |
|       |             |               | See issue #27 in the issues list. |
|       |             |               | Major rewrite to make use of attribute categories. |

| WD 15 | 4 Jan 2007  | Erik Rissanen | Clarified some of the text. |

| WD 16 |             | Erik Rissanen | Removed indirect delegates. |
|       |             |               | Updated XML based on new core schema. |
|       |             |               | Removed section about SAML profile (moved into an updated SAML profile document). |
|       |             |               | Removed sections about schema (moved to the core specification draft). |
|       |             |               | Improved text and presentation. |
|       |             |               | Updated processing model. |

| WD 17 |             | Erik Rissanen | Changed to a reduction algorithm which handles indeterminate. |
|       |             |               | Changed maximum depth to use a special XML attribute, rather being part of the request XACML attributes. |
|       |             |               | Removed the non-normative overview of the processing model. It was not up to date and didn’t really contribute anything beyond the examples. |
| WD 18 | 24 Aug 2007 | Erik Rissanen | Change disjunctive/conjunctive match to AnyOf/AllOf |
| WD 20 | 28 Dec 2007 | Erik Rissanen | Converted to current OASIS template. |
| WD 21 | 24 Feb 2008 | Erik Rissanen | Added normative statement which for security reasons allows the PDP refuse policies which contain unknown obligations. Rewrote section on actions other than create and included some revocation models there. Updated the access-permitted function to the new generalized attribute categories. |
| WD 22 | 4 Nov 2008 | Erik Rissanen | Moved the “access permitted” feature to the core specification. |
| WD 23 | 18 Mar 2009 | Erik Rissanen | Fix error on treatment of delegation-info in section 4.5. |
| WD 24 | 4 Apr 2009 | Erik Rissanen | Editorial cleanups Clarification of normative statements. |
| WD 25 | | Erik Rissanen | Fixed typos. |