OASIS eXtensible Access Control Markup Language (XACML)

Working Draft 15, 12 July 2002

Document identifier: draft-xacml-specification-15.doc

Location: http://www.oasis-open.org/committees/xacml/docs/

Send comments to: xacml-comment@lists.oasis-open.org

Editors:

Simon Godik, Simon Godik (simon@godik.com)
Tim Moses, Entrust (tim.moses@entrust.com)

Contributors:

Anne Anderson, Sun Microsystems
Bill Parducci, Bill Parducci
Carlisle Adams, Entrust
Daniel Engovatov, Crosslogix
Don Flinn, Hitachi
Ernesto Damiani, University of Milan
James MacLean, Affinitex
Hal Lockhart, Entegrity
Ken Yagen, Crosslogix
Konstantin Besnozov, Hitachi
Michiharu Kudo, IBM, Japan
Pierangela Samarati, University of Milan
Polar Humenn, Syracuse University
Sekhar Vajjhala, Sun Microsystems
Gerald Brose, Xtradyne

Abstract:

This specification defines an XML schema for a common access-control policy language.

Status:

This version of the specification is a working draft of the committee. As such, it is expected to change prior to adoption as an OASIS standard.

If you are on the xacml@lists.oasis-open.org list for committee members, send comments there. If you are not on that list, subscribe to the xacml-comment@lists.oasis-open.org list and send comments there. To subscribe, send an email message to xacml-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

Table of contents

163

1 Glossary (non-normative)

1.1 Preferred terms

Access - Performing an ***action***

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

Action - An operation on a ***resource***

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

Attribute - Characteristic of a ***subject***, ***resource, action or environment*** that may be referenced in a ***predicate***

Authorization decision - The result of evaluating an ***applicable policy.*** A function that evaluates to "permit, deny or indeterminate", and (optionally) a set of ***obligations***

Condition - An expression of ***predicates.*** A function that evaluates to "true or false"

Context - The canonical representation of ***decision request*** and ***authorization decision***

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

Effect - The intended consequence of a satisfied ***condition*** (either permit or deny)

Environment - The set of ***attributes*** that are independent of a particular ***subject, resource*** or ***action***

Information request - The request by a ***PDP*** to a ***PIP*** for ***attributes***

Obligation - An action specified in a ***policy*** or ***policy set*** that should be performed in conjunction with the issuance of an ***authorization decision***

Policy - A set of ***rules*** and an identifier for the ***rule-combining algorithm***

Policy administration point (PAP) - The system entity that creates a ***policy*** or ***policy set***

Policy-combining algorithm - The procedure for combining the ***target***, ***obligations*** and ***rules*** from multiple ***policies***

Policy decision point (PDP) - The system entity that evaluates ***applicable policy*** and renders an ***authorization decision***

Policy enforcement point (PEP) - The system entity that performs ***access control***, by enforcing ***authorization decisions***

Policy information point (PIP) - The system entity that acts as a source of ***attribute*** values

P*olicy retrieval point (PRP)* - The system entity that locates and retrieves ***applicable policy*** for a particular ***decision request***

Policy set - A set of ***policies*** and other ***policy sets*** and a ***policy-combining algorithm***

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

Resource - Data, service or system component

Rule - A ***target***, an ***effect*** and a set of ***conditions***

Rule-combining algorithm - The procedure for combining the ***target***, ***effect*** and ***conditions*** from multiple ***rules***

Subject - An actor whose ***attributes*** may be referenced by a ***predicate***

Target - The set of ***decision requests***, identified by definitions for ***resource***, ***subject*** and ***action***, that a ***rule***, ***policy*** or ***policy set*** is intended to evaluate

Target mapping - The process of confirming that a ***rule***, ***policy*** or ***policy set*** is applicable to an authorization ***decision request***

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: ***group*** and ***role***.

In place of the terms: ***privilege***, ***permission, authorization***, ***entitlement*** and ***right***, we use the term ***rule.***

The term ***object*** is also in common use, but we use the term ***resourc***e in this specification.

Requestors and ***initiators*** are covered by the term ***subject***.

2 Introduction (non-normative)

2.1 Background

The modern enterprise is pervaded by information systems and devices. Economies of scale have driven vendors to provide increasingly general-purpose solutions that must be configured to address the specific needs of each situation in which they are applied. This leads to constantly increasing complexity and configurability. Furthermore, the devices and systems may be distributed widely in a global enterprise. The task of analyzing and controlling system and device configuration in a consistent manner across an entire enterprise is an enormous challenge, compounded by the fact that, even when systems and devices support configuration by a remote console, there is no common interface standard. Consequently, it is becoming increasingly difficult for an enterprise to obtain a consolidated view of the policy in effect across its many and diverse systems and devices or to enforce a single policy that affects many of those devices and systems.

The objective of XACML is to address this need by defining a language capable of expressing policy statements for a wide variety of information systems and devices

The approach taken by XACML is to draw together long-established techniques for access-control and then to extend a platform-independent language (XML) with suitable syntax and semantics for expressing those techniques in the form of policy statements.

XACML exploits long-established techniques, such as:

- Combining independent rules to form a single policy.
- Combining independent policies, optionally from different policy-writers, to form a single policy set.
- The parameterization of the algorithm to be used for combining rules and policies.
- Attaching an indication of the set of decisions that a rule or policy is intended to render to the rule or policy.
- Defining the set of decisions that the rule or policy is intended to render in terms of the name or attributes of the subject, resource and action identified in the decision request.
- Specifying in a policy statement a set of actions that must be performed in conjunction with the rendering of a decision.
- Stating rule conditions as a logical expression of predicates of functions of attributes of the resource and/or subject.
- Providing an abstraction layer between the policy language and the environment to which it applies.
- The communication of policies, either attached to the resources they are intended to protect, or separately.

The following sections describe how to understand the rest of this specification.

2.1.1 Rule combining

Ref 5,

2.1.2 Policy combining

Ref 5, 8

2.1.3 Combining algorithm

Ref 7,

2.1.4 Decision indication

2.1.5 Names or attributes

Ref 2, 6

2.1.6 Specifying actions

Ref 1,

2.1.7 Expression of predicates

Ref 4,

2.1.8 Abstraction layer

2.1.9 Policy attachment

Ref 1, 3

2.2 References

1. Perritt; Knowbots, Headers & Contract Law; 1993.
2. Orange book
3. Trusted Network Interpretation
4. X.500 filter
5. J Moffett and M Sloman. Policy hierarchies for distributed system management. IEEE Journal on Selected areas in communications, pages 1404-1414, December 1993. Special Issue on network management.
6. R Sandhu, E Coyne, H Feinstein and C Youman. Role-based access control models. IEEE Computer, 9(2); 38-47, 1996.
7. S Jajodia, P Samarati, V S Subrahmanian and E Bertino. A unified framework for enforcing multiple access control policies. Proceedings of ACM SIGMOD, 1997
8. N Minsky, V Ungureanu. Unified support for heterogeneous distributed systems. 7th USENIX security symposium, San Antonio, Texas, January, 1998..

2.3 Notation

This specification contains schema conforming to W3C XML Schema and normative text to describe the syntax and semantics of XML-encoded policy statements.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 rfc2119:

> *"they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions)"*

These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

```
Listings of XACML schemas appear like this.
```

```
Example code listings appear like this.
```

Conventional XML namespace prefixes are used throughout the listings in this specification to stand for their respective namespaces as follows, whether or not a namespace declaration is present in the example:

- The prefix `saml:` stands for the SAML assertion namespace.
- The prefix `ds:` stands for the W3C XML Signature namespace.
- The prefix `xs:` stands for the W3C XML Schema namespace.

This specification uses the following typographical conventions in text: `<XACMLElement>`, `<ns:ForeignElement>`, `Attribute`, **Datatype**, `OtherCode`.

2.4 Schema Organization and Namespaces

The XACML policy syntax is defined in a schema associated with the following XML namespace:

```
urn:oasis:names:tc:xacml:0.15i:policy
```

The XACML context syntax is defined in a schema associated with the following XML namespace:

```
urn:oasis:names:tc:xacml:0.15i:context
```

XACML functions have the following namespace prefix.

```
urn:oasis:names:tc:xacml:0.15i:function
```

Note: The XACML namespace names are temporary and may change when XACML 1.0 is finalized.

The SAML assertion schema is imported into the XACML schema. Also imported is the schema for XML Signature XMLSigXSD, which is associated with the following XML namespace:

```
http://www.w3.org/2000/09/xmldsig#
```

3 Example (non-normative)

This section contains an example use of XACML for illustrative purposes.

3.1 Introduction to the example

This section contains an example XML document, an example request context and example XACML rules. The XML document is a medical record. Four separate rules are defined.

3.2 Example medical record instance

Following is an instance of a medical record to which the example XACML rules can be applied. The <record> schema is defined in the registered namespace administered by "//medico.com".

```
<?xml version="1.0" encoding="UTF-8"?>
<record xmlns="medico.com/records.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="medico.com/records.xsd
http://www.medico.com/schema/record.xsd">
 <patient>
   <patientName>
     <first>Bartholomew</first>
     <last>Simpson</last>
   </patientName>
   <patientContact>
     <street>27 Shelbyville Road</street>
     <city>Springfield</city>
     <state>MA</state>
     <zip>12345</zip>
     <phone>555.123.4567</phone>
     <fax/>
     <email/>
   </patientContact>
   <patientDoB xsi:type="date">1992-03-21</patientDoB>
   <patientGender xsi:type="string">male</patientGender>
   <policyNumber xsi:type="string">555555</policyNumber>
 </patient>
 <parentGuardian>
   <parentGuardianName>
     <first>Homer</first>
     <last>Simpson</last>
   </parentGuardianName>
   <parentGuardianContact>
     <street>27 Shelbyville Road</street>
     <city>Springfield</city>
     <state>MA</state>
     <zip>12345</zip>
     <phone>555.123.4567</phone>
     <fax/>
     <email>homers@aol.com</email>
   </parentGuardianContact>
 </parentGuardian>
 <primaryCarePhysician>
   <physicianName>
     <first>Julius</first>
     <last>Hibbert</last>
   </physicianName>
   <physicianContact>
     <street>1 First St</street>
     <city>Springfield</city>
     <state>MA</state>
     <zip>12345</zip>
     <phone>555.123.9012</phone>
     <fax>555.123.9013</fax>
     <email/>
   </physicianContact>
   <registrationID>ABC123</registrationID>
 </primaryCarePhysician>
 <insurer>
   <name>Blue Cross</name>
   <street>1234 Main St</street>
```

```
    <city>Springfield</city>
    <state>MA</state>
    <zip>12345</zip>
    <phone>555.123.5678</phone>
    <fax>555.123.5679</fax>
    <email/>
  </insurer>
  <medical>
    <treatment>
      <drug>
        <name>methylphenidate hydrochloride</name>
        <dailyDosage>30mgs</dailyDosage>
        <startDate>1999-01-12</startDate>
      </drug>
      <comment>patient exhibits side-effects of skin coloration and
carpal degeneration</comment>
    </treatment>
    <result>
      <test>blood pressure</test>
      <value>120/80</value>
      <date>2001-06-09</date>
      <performedBy>Nurse Betty</performedBy>
    </result>
  </medical>
</record>
```

3.3 Example authorization decision request

The following example illustrates a request context to which the example rules are intended to be applicable. It represents a request by the physician Julius Hibbert to read the patient date of birth in the record of Bartholomew Simpson. It includes an authentication assertion and an attribute assertion containing the role of the requestor.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:0.15i:context"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:identifier="urn:oasis:names:tc:xacml:identifier"
xmlns:xacml="urn:oasis:names:tc:xacml:0.15i:policy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:context
http://www.oasis-open.org/tc/xacml/v15/draft-xacml-schema-context-
15i.xsd">
  <Subject>
    <SubjectId Format="xs:string">Julius Hibbert</SubjectId>
  </Subject>
  <Resource>
    <ResourceSpecifier Format="xs:anyURI" Scope="Descendants"
ResourceId="//medico.com/record/patient[@patientName/first='Bartholo
mew'][@patientName/last='Simpson']/patientDoB"/>
  </Resource>
  <Action Namespace="">read</Action>
  <Environment>
    <EnvironmentAttribute
AttributeId="urn:oasis:names:tc:SAML:1.0:Assertion"
DataType="xs:string">
```

```
      <saml:Assertion AssertionID="64578390" Issuer="medico.com"
IssueInstant="2002-03-08T08:23:47-05:00" MajorVersion="0"
MinorVersion="28" xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis-
open.org/committees/security/docs/cs-sstc-schema-assertion-01.xsd">
         <saml:AuthenticationStatement AuthenticationInstant="2002-
03-08T08:23:45-05:00" AuthenticationMethod="http://www.oasis-
open.org/committees/security/docs/draft-sstc-core-28/password-sha1">
            <saml:Subject>
               <saml:NameIdentifier
NameQualifier="\\medico.com">Julius Hibbert</saml:NameIdentifier>
               <saml:SubjectConfirmation>
                  <saml:ConfirmationMethod>http://www.oasis-
open.org/committees/security/docs/draft-sstc-core-
24/artifact</saml:ConfirmationMethod>
               </saml:SubjectConfirmation>
            </saml:Subject>
            <saml:SubjectLocality IPAddress="217.57.95.242"/>
         </saml:AuthenticationStatement>
      </saml:Assertion>
    </EnvironmentAttribute>
    <EnvironmentAttribute
AttributeId="urn:oasis:names:tc:SAML:1.0:Assertion"
DataType="xs:string">
      <saml:Assertion MajorVersion="0" MinorVersion="28"
AssertionID="68938960" Issuer="medico.com" IssueInstant="2000-06-
15T15:02:39-05:00" xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis-
open.org/committees/security/docs/cs-sstc-schema-assertion-01.xsd">
         <saml:AttributeStatement>
            <saml:Subject>
               <saml:NameIdentifier
NameQualifier="\\medico.com">Julius Hibbert</saml:NameIdentifier>
            </saml:Subject>
            <saml:Attribute AttributeName="role"
AttributeNamespace="//medico.com">
               <saml:AttributeValue>physician</saml:AttributeValue>
            </saml:Attribute>
         </saml:AttributeStatement>
      </saml:Assertion>
    </EnvironmentAttribute>
  </Environment>
</Request>
```

3.4 Example plain-language rules

The following plain-language rules are to be enforced:

1. A person may read any record for which he or she is the designated patient.
2. A person may read any record for which he or she is the designated parent or guardian, and for which the patient is under 16 years of age.
3. A physician may write any medical element for which he or she is the designated primary care physician, provided an email is sent to the patient,
4. An administrator shall not be permitted to read or write medical elements of a patient record.

These rules may be written by different ***PAP***s, operating independently, or by a single ***PAP***.

3.5 Example XACML rule instances

3.5.1 Rule 1

Rule 1 illustrates a simple rule with a single condition. The following XACML <Rule> instance expresses Rule 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<Rule RuleId="//medico.com/rules/rule1" Effect="Permit"
xmlns="urn:oasis:names:tc:xacml:0.15i:policy"
xmlns:function="urn:oasis:names:tc:xacml:0.15i:function"
xmlns:identifier="urn:oasis:names:tc:xacml:0.15i:identifier"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:policy
http://www.oasis-open.org/tc/xacml/v15/draft-xacml-schema-policy-
15i.xsd">
 <Description>A person may read any record for which he or she is
the designated patient</Description>
 <Target>
    <Subjects MatchId="function:rfc822Name-equal"
DataType="xs:boolean">
      <AttributeDesignator
Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden
tifier:rfc822Name']" DataType="identifier:rfc822Name"/>
      <Attribute DataType="identifier:rfc822Name">@</Attribute>
    </Subjects>
    <Resources MatchId="function:string-match"
DataType="xs:boolean">
      <AttributeDesignator
Designator="//xacmlContext/Request/Resource/@ResourceURI"
DataType="xs:anyURI"/>
      <Attribute
DataType="xs:anyURI">//medico.com/record.*</Attribute>
    </Resources>
    <Actions MatchId="function:subset" DataType="xs:boolean">
      <AttributeDesignator
Designator="//xacmlContext/Action[@Namespace=]"
DataType="xs:string"/>
      <Attribute DataType="xs:string">read</Attribute>
    </Actions>
 </Target>
 <Condition FunctionId="function:string-equal"
DataType="xs:boolean">
    <AttributeDesignator
Designator="//xacmlContext/Request/Subject/SubjectId"
DataType="xs:string"/>
    <AttributeDesignator
Designator="//xacmlContext/Request/Resource/patientName"
DataType="xs:string"/>
 </Condition>
</Rule>
```

3.5.2 Rule 2

Rule 2 illustrates the use of a mathematical function, i.e. the <Minus> function to calculate age. It also illustrates the use of predicate expressions, with the <And> and <Not> elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<Rule RuleId="//medico.com/rules/rule2" Effect="Permit"
xmlns="urn:oasis:names:tc:xacml:0.15i:policy"
xmlns:function="urn:oasis:names:tc:xacml:0.15i:function"
xmlns:identifier="urn:oasis:names:tc:xacml:0.15i:identifier"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:policy
http://www.oasis-open.org/tc/xacml/v15/draft-xacml-schema-policy-
15i.xsd">
 <Description>A person may read any record for which he or she is
the designated parent or guardian, and for which the patient is
under 16 years of age</Description>
 <Target>
    <Subjects MatchId="function:rfc822Name-equal"
DataType="xs:boolean">
      <AttributeDesignator
Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden
tifier:rfc822Name']" DataType="identifier:rfc822Name"/>
      <Attribute DataType="identifier:rfc822Name">@</Attribute>
    </Subjects>
    <Resources MatchId="function:string-match"
DataType="xs:boolean">
      <AttributeDesignator
Designator="//xacmlContext/Request/Resource/@ResourceURI"
DataType="xs:anyURI"/>
      <Attribute
DataType="xs:anyURI">//medico.com/record.*</Attribute>
    </Resources>
    <Actions MatchId="function:subset" DataType="xs:boolean">
      <AttributeDesignator
Designator="//xacmlContext/Action[@Namespace=]"
DataType="xs:string"/>
      <Attribute DataType="xs:string">read</Attribute>
    </Actions>
 </Target>
 <Condition FunctionId="function:and" DataType="xs:boolean">
    <Function FunctionId="function:string-equal"
DataType="xs:boolean">
      <AttributeDesignator
Designator="//xacmlContext/Request/Subject/SubjectId"
DataType="xs:string"/>
      <AttributeDesignator
Designator="//xacmlContext/Request/Resource/guardianName"
DataType="xs:string"/>
    </Function>
    <Function FunctionId="function:dayTimeDuration-greater-than"
DataType="xs:boolean">
      <Function FunctionId="function:date-subtract"
DataType="xs:dayTimeDuration">
        <AttributeDesignator
Designator="//xacmlContext/Other/OtherAttribute/Attribute[@DataType=
'identifier:today'sDate']" DataType="xs:date"/>
        <AttributeDesignator
Designator="//xacmlContext/Request/Resource/patient/patientDoB"
DataType="xs:date"/>
      </Function>
      <Attribute DataType="xs:dayTimeDuration">16-0-0</Attribute>
    </Function>
 </Condition>
</Rule>
```

3.5.3 Rule 3

Rule 3 illustrates the use of an ***obligation***. The XACML <Rule> element syntax does not include an element suitable for carrying an ***obligation***, therefore Rule 3 has to be formatted as a <PolicyStatement> element, which is a type of SAML assertion.

```
<?xml version="1.0" encoding="UTF-8"?>
<saml:Assertion MajorVersion="0" MinorVersion="24"
AssertionID="A7F34K90" Issuer="medico.com" IssueInstant="2002-03-
22T08:23:47-05:00">
 <PolicyStatement PolicyId="//medico.com/rules/policy3"
RuleCombiningAlgId="//www.oasis-
open.org/committees/xacml/docs/ruleCombiningAlgorithms/DenyOverrides
" xmlns="urn:oasis:names:tc:xacml:0.15i:policy"
xmlns:function="urn:oasis:names:tc:xacml:0.15i:function"
xmlns:identifier="urn:oasis:names:tc:xacml:0.15i:identifier"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:policy
http://www.oasis-open.org/tc/xacml/v15/draft-xacml-schema-policy-
15i.xsd">
    <Description>A physician may write any medical element for which
he or she is the designated primary care physician, provided an
email is sent to the patient</Description>
    <Target>
      <Subjects MatchId="function:string-match"
DataType="xs:boolean">
        <AttributeDesignator
Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden
tifier:role']" DataType="xs:string"/>
        <Attribute DataType="xs:string">physician</Attribute>
      </Subjects>
      <Resources MatchId="function:string-match"
DataType="xs:boolean">
        <AttributeDesignator
Designator="//xacmlContext/Request/Resource/@ResourceURI"
DataType="xs:anyURI"/>
        <Attribute
DataType="xs:anyURI">//medico.com/record/medical.*</Attribute>
      </Resources>
      <Actions MatchId="function:subset" DataType="xs:boolean">
        <AttributeDesignator
Designator="//xacmlContext/Action[@Namespace=]"
DataType="xs:string"/>
        <Attribute DataType="xs:string">write</Attribute>
      </Actions>
    </Target>
    <RuleSet>
      <Rule RuleId="//medico.com/rules/rule3" Effect="Permit">
        <Description>A physician may write any medical element for
which he or she is the designated primary care
physician</Description>
        <Condition FunctionId="function:and"
DataType="xs:boolean">
          <Function FunctionId="function:string-equal"
DataType="xs:boolean">
            <AttributeDesignator
Designator="//xacmlContext/Request/Subject/SubjectId"
DataType="xs:string"/>
            <AttributeDesignator
Designator="//xacmlContext/Request/Resource/physicianName"
DataType="xs:string"/>
```

```
            </Function>
            <Function FunctionId="function:present"
DataType="xs:boolean">
               <AttributeDesignator
Designator="//xacmlContext/Request/Resource/patient/email"
DataType="xs:string"/>
            </Function>
         </Condition>
      </Rule>
    </RuleSet>
    <Obligations>
      <Obligation ObligationId="//medico.com/emailer"
FulfilOn="Permit">
         <AttributeDesignator
Designator="//xacmlContext/Request/Resource/patient/email"
DataType="xs:string"/>
         <AttributeAssignment DataType="xs:string"
AttributeId="//medico.com/text">Your medical record has been
accessed by:</AttributeAssignment>
         <AttributeDesignator
Designator="//xacmlContext/Request/Subject/SubjectId"
DataType="xs:string"/>
      </Obligation>
    </Obligations>
  </PolicyStatement>
</saml:Assertion>
```

3.5.4 Rule 4

Rule 4 illustrates the use of the "Deny" `effect` value, and a rule with no <`Condition`> element.

```
<?xml version="1.0" encoding="UTF-8"?>
<Rule RuleId="//medico.com/rules/rule4" Effect="Deny"
xmlns="urn:oasis:names:tc:xacml:0.15i:policy"
xmlns:function="urn:oasis:names:tc:xacml:0.15i:function"
xmlns:identifier="urn:oasis:names:tc:xacml:0.15i:identifier"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:policy
http://www.oasis-open.org/tc/xacml/v15/draft-xacml-schema-policy-
15i.xsd">
  <Description>An administrator shall not be permitted to read or
write medical elements of a patient record</Description>
  <Target>
    <Subjects MatchId="function:string-match" DataType="xs:boolean">
      <AttributeDesignator
Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden
tifier:role']" DataType="xs:string"/>
      <Attribute DataType="xs:string">adminstrator</Attribute>
    </Subjects>
    <Resources MatchId="function:anyURI-equal"
DataType="xs:boolean">
      <AttributeDesignator
Designator="//xacmlContext/Request/Resource/@ResourceURI"
DataType="xs:anyURI"/>
      <Attribute
DataType="xs:anyURI">//medico.com/record/medical.*</Attribute>
    </Resources>
    <Actions MatchId="function:subset" DataType="xs:boolean">
      <AttributeDesignator
Designator="//xacmlContext/Action[@Namespace=]"
DataType="xs:string"/>
```

```
      <Attribute DataType="xs:string">read write</Attribute>
    </Actions>
  </Target>
</Rule>
```

4 Models (non-normative)

The context and schema of XACML are described in two models. These models are: the data-flow model and the policy language model. They are described in the following sub-sections.

4.1 Data-flow model

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

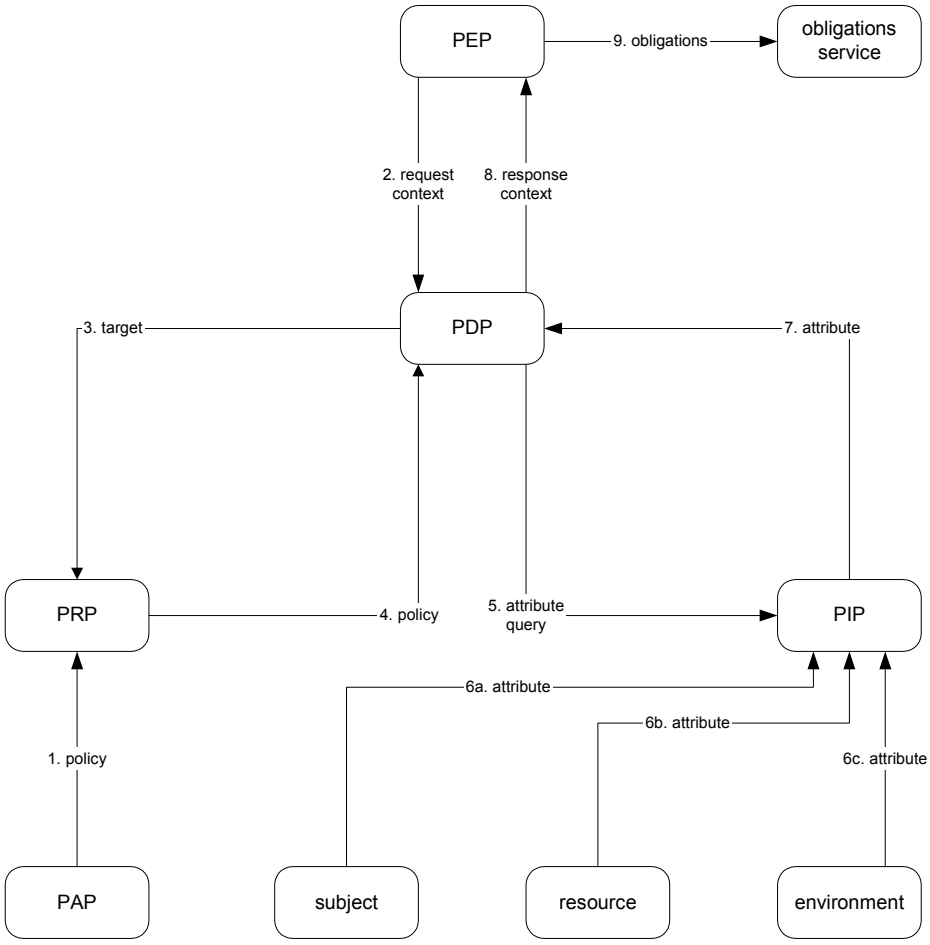

Figure 1 - Data-flow diagram

Note: some of the data-flows shown in the diagram may be facilitated by a repository. For instance, the communications between the ***PDP*** and the ***PIP*** or the communications between the ***PDP*** and the ***PRP*** or the communication between the ***PAP*** 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 by the following steps.

1. ***PAP***s write ***policies*** and make them available to the ***PRP***. From the point of view of an individual ***PAP***, its ***policies*** represent the complete policy for a particular ***target***. However, the ***PDP*** may be aware of other ***PAP***s that it considers authoritative for the same ***target***. In which case, it is the ***PDP***'s job to obtain all the ***policies*** and combine them in accordance with a ***policy-combining algorithm***. The result should be a self-consistent ***policy set***.

2. The ***PEP*** sends request ciontext to the ***PDP***, perhaps in the form of a SAML [SAML] request. The request context contains some or all of the ***attributes*** required by the ***PDP*** to render an ***authorization decision***, in accordance with ***applicable policy***. The decision request and all attributes relevant to that request are converted to an XACML input context ("xacmlContext:request") by the ***PDP*** or by another entity that it trusts to do this conversion.

3. The ***PDP*** locates and retrieves the ***policy*** applicable to the request context from the ***PRP***.

4. The ***PRP*** returns the ***applicable policy*** to the ***PDP*** in the form of an XACML <`PolicyStatement`> or <`PolicySetStatement`>. The ***PDP*** ensures that the ***input context*** is in the scope of the <`PolicyStatement`> or <`PolicySetStatement`>.

5. The ***PDP*** examines the authorization ***input context*** 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 *PIP*s, perhaps in the form of SAML requests of the attribute query type [SAML].

6. The ***PIP*** (which may be a SAML attribute authority) locates and retrieves the requested ***attributes*** from other systems by a means, and in a form, that is out of scope for this specification.

7. The ***PIP*** returns the requested ***attributes*** to the ***PDP***, perhaps in the form of SAML responses containing SAML attribute assertions. The ***PDP*** (or another trusted entity) incorporates these attribute values into the input context and evaluates the ***policy*** with respect to this input context. The result of this evaluation is a decision encoded in an output context ("xacmlContext:response") document. The response context is converted to an ***authorization decision*** protocol message by the ***PDP*** or by another entity trusted to do that conversion.

8. If the ***policy*** were to evaluate to TRUE, then the ***PDP*** returns a response context, perhaps in the form of a SAML response, to the ***PEP*** containing the "Permit" `saml:Decision` attribute and (optional) ***obligations***.

9. The ***PEP*** fulfills the ***obligations***.

The input context and output contexts are the environment-agnostic inputs/outputs for an XACML-conformant ***PDP***. For any specific environment (e.g., SAML, J2SE, CORBA) conversion processes will be needed to transform from the environment-specific inputs to the xacmlContext:request, and from the xacmlContext:response to the environment-specific outputs. These conversions may be done by the ***PDP*** or by another entity. Having them done by another entity ensures that a given ***PDP*** implementation may be deployed in any environment without modification.

4.2 XACML Context

XACML is designed to be applicable to a variety of application environments. The core language is insulated from the application environment by the XACML context. The XACML context is an XML schema describing a canonical representation for the inputs and outputs of the PDP. Attributes referenced by an instance of XACML SHALL be in the form of XPath expressions on the context. Implementations must convert between the attribute representations in the application environment (e.g., SAML, J2SE, CORBA, and so on) and the attribute representations in the XACML context. How this is achieved is outside the scope of the XACML specification. In some cases, such as

SAML, this conversion may be accomplished in an automated way through the use of an XSLT transformation.

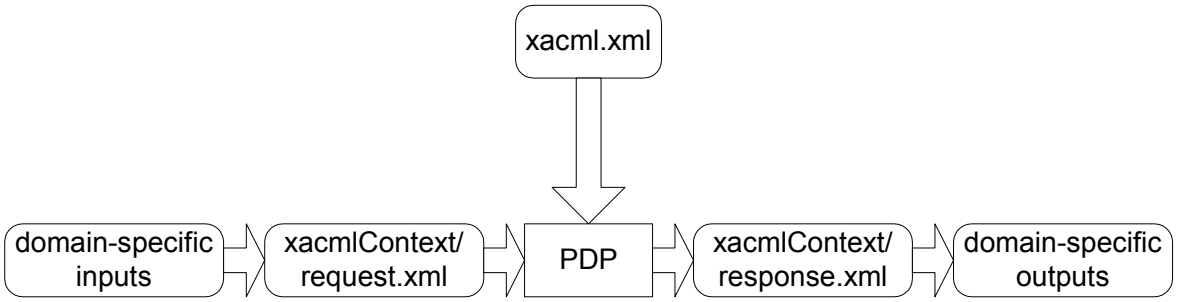

Figure 2 - Context

4.3 Policy language model

The policy language model is shown in Figure 3. The main components of the model are:

- ***Rule***;
- ***Policy statement***; and
- ***Policy set statement***.

These are described in the following sub-sections.

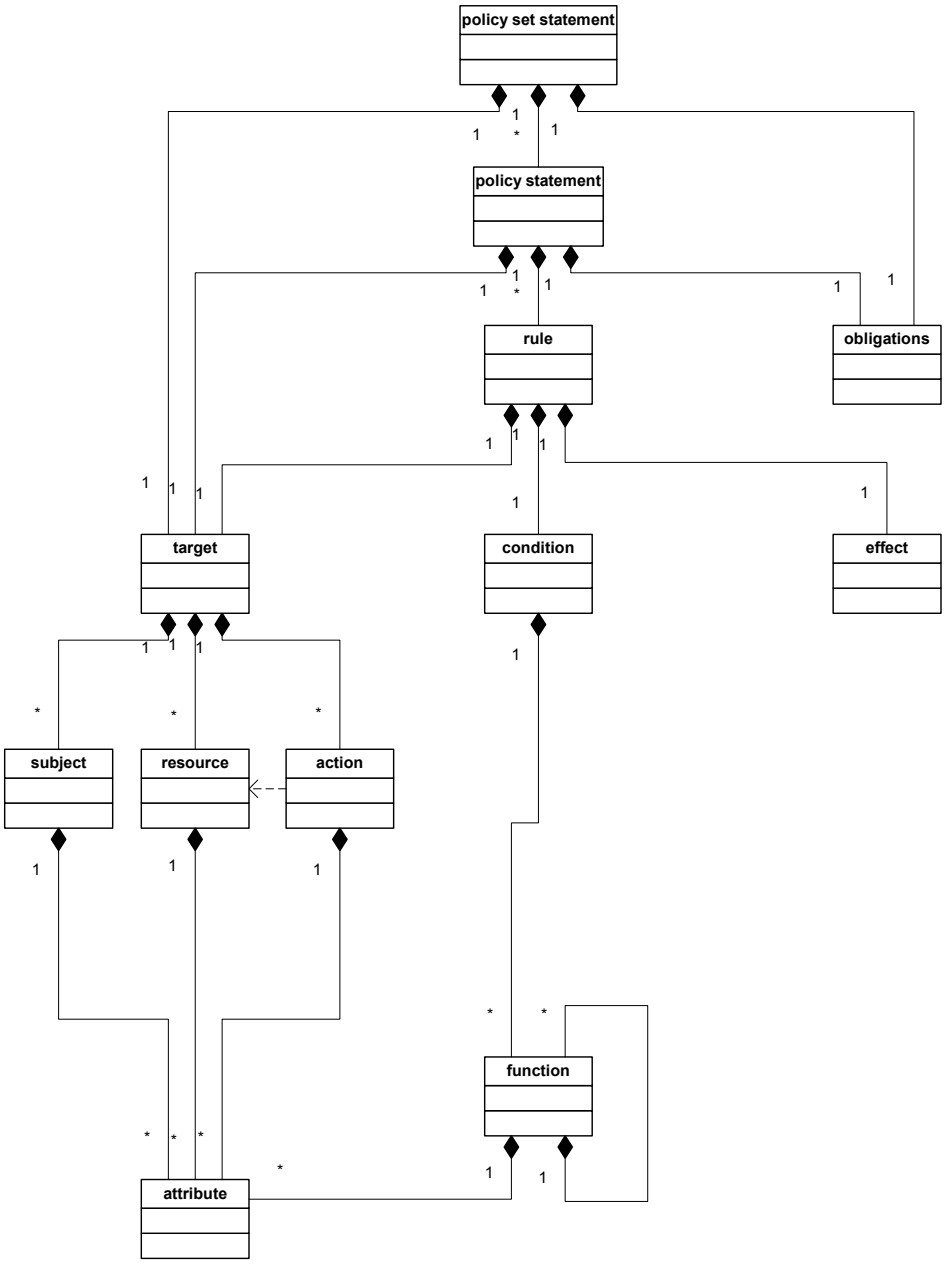

Figure 3 - Policy language model

4.3.1 Rule

The main components of a ***rule*** are:

- a ***target***;
- an ***effect***; and
- a ***condition***.

These are discussed in the following sub-sections.

4.3.1.1 Target

The ***target*** defines the set of:

- ***resource***s;
- ***subjects***; and
- ***actions***

to which the ***rule*** is intended to apply. If the ***rule*** is intended to apply to all entities of a particular type, then the ***target*** definition is the root of the applicable name space. An XACML ***PDP*** verifies that the ***resources***, ***subjects*** and ***actions*** identified in the request context are each included in the ***target*** of the ***rules*** that it uses to evaluate the ***decision request***. ***Target*** definitions are discrete, in order that they may be indexed by the ***PDP***.

4.3.1.2 Effect

The ***effect*** indicates the rule-writer's intended consequence of a true evaluation for the ***rule***. Two values are allowed: permit and deny.

4.3.1.3 Condition

Condition is a general expression of ***predicates*** of ***attributes***. It should not duplicate the exact ***predicates*** implied by the ***target***. Therefore, it may be null.

4.3.1.4 Rule evaluation

A ***rule*** has a value that can be calculated by evaluating its contents. ***Rule*** evaluation involves separate evaluation of the ***rule's target*** and ***condition***. The ***rule*** truth table is shown in Table 1.

Target	Condition	Rule
Match	True	Effect
Match	False	Not applicable
Match	Indeterminate	Indeterminate
No-match	True	Not applicable
No-match	False	Not applicable
No-match	Indeterminate	Not applicable

Table 1 - Rule truth table

The ***target*** value is Match if the ***resource***, ***subject*** and ***action*** specified in the request context are each in the ***target*** defined in the ***rule***. Otherwise, its value is No-match.

The ***condition*** value is True if the <Condition> element is null, or if it evaluates True for the ***attribute*** values supplied in, or referenced by, the request context. Its value is False if the <Condition> element evaluates False for the ***attribute*** values supplied in, or referenced by, the request context. If any ***attribute*** value referenced in the ***condition*** cannot be obtained, then the ***condition*** evaluates Indeterminate.

4.3.2 Policy statement

From the data-flow model one can see that ***rules*** are not exchanged amongst system entities. Therefore, a ***PAP*** combines ***rules*** in a ***policy***. A ***policy*** comprises four main components:

- a ***target***;
- a ***rule-combining algorithm***-identifier;
- a set of ***rules***; and
- ***obligations***.

4.3.2.1 Target

The ***target*** of a ***policy*** must include all the ***decision requests*** that it is intended to evaluate. The ***target*** may be declared by the writer of the ***policy***, or computed from the ***targets*** of its component ***rules***.

If the ***target*** of the ***policy statement*** is computed from the ***targets*** of the component ***rules***, two approaches are permitted:

- the ***target*** of the ***policy*** may be the union of the ***target*** definitions for ***resource***, ***subject*** and ***action*** that are contained in the component ***rule***s; or
- the ***target*** of the ***policy*** may be the intersection of the ***target*** definitions for ***resource***, ***subject*** and ***action*** that are contained in the component ***rules***.

In the former case, the ***target*** may be omitted from the individual ***rules***, and the ***targets*** from the component ***rules*** must be included in the form of ***conditions*** in their respective ***rules***. As an example, the following ***rule target*** and ***condition*** may be merged in a single ***condition***.

```
<Target>
 <Subjects MatchId="function:rfc822Name-equal"
DataType="xs:boolean">
    <AttributeDesignator
Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden
tifier:rfc822Name']" DataType="identifier:rfc822Name"/>
    <Attribute DataType="identifier:rfc822Name">@</Attribute>
 </Subjects>
 <Resources MatchId="function:string-match" DataType="xs:boolean">
    <AttributeDesignator
Designator="//xacmlContext/Request/Resource/@ResourceURI"
DataType="xs:anyURI"/>
    <Attribute
DataType="xs:anyURI">//medico.com/record.*</Attribute>
 </Resources>
 <Actions MatchId="function:subset" DataType="xs:boolean">
    <AttributeDesignator
Designator="//xacmlContext/Action[@Namespace=]"
DataType="xs:string"/>
    <Attribute DataType="xs:string">read</Attribute>
 </Actions>
</Target>
```

```
<Condition FunctionId="function:string-equal" DataType="xs:boolean">
 <AttributeDesignator
Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden
tifier:patientName']" DataType="xs:string"/>
```

```
 <AttributeDesignator
Designator="//xacmlContext/Request/Resource/patientName"
DataType="xs:string"/>
</Condition>
```

Following is the merged condition.

```
<Condition FunctionId="function:and" DataType="xs:boolean">
 <Function FunctionId="function:string-match"
DataType="xs:boolean">
    <AttributeDesignator
Designator="//xacmlContext/Request/Resource/@ResourceURI"
DataType="xs:anyURI"/>
    <Attribute
DataType="xs:anyURI">//medico.com/record.*</Attribute>
 </Function>
 <Function FunctionId="function:subset" DataType="xs:boolean">
    <AttributeDesignator
Designator="//xacmlContext/Action[@Namespace=]"
DataType="xs:string"/>
    <Attribute DataType="xs:string">read</Attribute>
 </Function>
 <Function FunctionId="function:string-equal"
DataType="xs:boolean">
    <AttributeDesignator
Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden
tifier:patientName']" DataType="xs:string"/>
    <AttributeDesignator
Designator="//xacmlContext/Request/Resource/patientName"
DataType="xs:string"/>
 </Function>
</Condition>
```

In the case where the ***policy target*** is computed as the intersection of the ***targets*** of the individual ***rules***, the ***target***s may be omitted from the individual ***rules***.

In the case that a ***rule target*** is present, the ***rule*** is evaluated according to the truth table of Table 1.

4.3.2.2 Rule-combining algorithm

The ***rule-combining algorithm*** specifies the algorithm by which the results of evaluating the component ***rules*** are combined, when evaluating the ***policy***.

The result of evaluating the ***policy*** is defined by the ***rule-combining algorithm***. In the case that the ***PDP*** uses a ***policy*** to determine its response to a ***decision request***, the `saml:Decision` value is the value of the ***policy***, as defined by the ***rule-combining algorithm***.

See Section 8.5 for an example of a ***rule-combining algorithm***.

4.3.2.3 Obligations

The XACML <`Rule`> syntax does not contain an element suitable for carrying ***obligations***, therefore, if required in a ***policy***, ***obligations*** must be added by the writer of the ***policy***.

When a ***PDP*** evaluates a ***policy*** containing ***obligations***, it returns certain of those ***obligations*** to the ***PEP*** in its response context. The ***obligations*** that it returns to the ***PEP*** are those whose `xacml:FulfilOn` attributes have the same value as the result of evaluating the ***policy***.

4.3.2.4 Example policy statement

This section uses the example of Section 3 to illustrate the process of combining rules. The policy governing read access to medical elements of a record is formed from each of the four rules. In plain language, the combined rule is:

- Either the requestor is the patient; or
- the requestor is the parent or guardian and the patient is under 16; or
- the requestor is the primary care physician and a notification is sent to the patient; and
- the requestor is not an administrator.

The following XACML <PolicyStatement> illustrates the combined rules. Rules 1 and 4 are included by reference, rule 2 is included as a digest, and rule 3 is explicitly included.

```
<?xml version="1.0" encoding="UTF-8"?>
<saml:Assertion MajorVersion="0" MinorVersion="28"
AssertionID="A7F34K90" Issuer="medico.com" IssueInstant="2002-03-
22T08:23:47-05:00">
 <PolicyStatement PolicyId="//medico.com/rules/policy5"
RuleCombiningAlgId="urn:oasis:names:tc:XACML:identifier:ruleCombinin
gAlgorithms:denyOverrides"
xmlns="urn:oasis:names:tc:xacml:0.15i:policy"
xmlns:function="urn:oasis:names:tc:xacml:0.15i:function"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:policy
D:\MYDOCU~1\Standards\XACML\v15\draft-xacml-schema-policy-15i.xsd">
    <Target>
      <Subjects MatchId="function:superset" DataType="xs:boolean">
        <AttributeDesignator
Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden
tifier:role']" DataType="xs:string"/>
        <Attribute DataType="xs:string"></Attribute>
      </Subjects>
      <Resources MatchId="function:anyURI-equal"
DataType="xs:boolean">
        <AttributeDesignator
Designator="//xacmlContext/Request/Resource/@ResourceURI"
DataType="xs:anyURI"/>
        <Attribute
DataType="xs:anyURI">//medico.com/record/medical.*</Attribute>
      </Resources>
      <Actions MatchId="function:subset" DataType="xs:boolean">
        <AttributeDesignator
Designator="//xacmlContext/Action[@Namespace=]"
DataType="xs:string"/>
        <Attribute DataType="xs:string">read</Attribute>
      </Actions>
    </Target>
    <RuleSet>
      <RuleDesignator>
        <RuleId>//medico.com/rules/rule1</RuleId>
      </RuleDesignator>
      <RuleDesignator>
        <RuleDigest
Base64Digest="H7jiE0+jwkn63k/JhB3+D9aI4V3J9z/o0"/>
      </RuleDesignator>
      <Rule RuleId="//medico.com/rules/rule3" Effect="Permit">
```

```
         <Description>A physician may write any medical element for
which he or she is the designated primary care
physician</Description>
         <Condition FunctionId="function:and"
DataType="xs:boolean">
            <Function FunctionId="function:string-equal"
DataType="xs:boolean">
               <AttributeDesignator
Designator="//xacmlContext/Request/Subject/SubjectId"
DataType="xs:string"/>
               <AttributeDesignator
Designator="//xacmlContext/Request/Resource/physicianName"
DataType="xs:string"/>
            </Function>
            <Function FunctionId="function:present"
DataType="xs:boolean">
               <AttributeDesignator
Designator="//xacmlContext/Request/Resource/patient/email"
DataType="xs:string"/>
            </Function>
         </Condition>
      </Rule>
      <RuleDesignator>
         <RuleId>//medico.com/rules/rule4</RuleId>
      </RuleDesignator>
   </RuleSet>
   <Obligations>
      <Obligation ObligationId="//medico.com/emailer"
FulfilOn="Permit">
         <AttributeDesignator
Designator="//xacmlContext/Request/Resource/patient/email"
DataType="xs:string"/>
         <AttributeAssignment DataType="xs:string"
AttributeId="//medico.com/text">Your medical record has been
accessed by:</AttributeAssignment>
         <AttributeDesignator
Designator="//xacmlContext/Request/Subject/SubjectId"
DataType="xs:string"/>
      </Obligation>
   </Obligations>
 </PolicyStatement>
</saml:Assertion>
```

4.3.3 Policy set statement

A ***policy set*** comprises four main components:

- a ***target***;
- a set of ***policy statements***;
- ***obligations***; and
- a ***policy-combining algorithm***-identifier.

The ***target*** and ***policy statement*** components are to be interpreted as described above.

4.3.3.1 Obligations

The writer of a ***policy set statement*** MAY add ***obligations*** to the ***policy set***, in addition to those contained in the component ***policies*** and ***policy sets***.

4.3.3.2 Policy-combining algorithm

The ***policy-combining algorithm*** is the algorithm by which the results of evaluating the component ***policies*** are combined to form the value of the ***policy set***. In the case that the ***PDP*** uses a ***policy set*** to determine its response to a ***decision request***, the `saml:Decision` value is the result of evaluating the ***policy set***.

When a ***PDP*** evaluates a ***policy set*** containing ***obligations***, it returns certain of those ***obligations*** to the ***PEP*** in its response context. The XACML <obligation> elements that are returned to the ***PEP*** are those whose `xacml:FulfilOn` attributes have the same value as the result of evaluating the ***policy set***.

As a consequence of this procedure, no ***obligations*** are returned to the ***PEP*** if the ***policies*** from which they are drawn are not evaluated or their evaluated result is Indeterminate or Not applicable.

See Section 8.8 for an example of a ***policy-combining algorithm***.

5 Policy syntax (normative, with the exception of the schema fragments)

5.1 Element <PolicySetStatement>

The `<PolicySetStatement>` element is a top-level element in the XACML schema.

```
<xs:element name="PolicySetStatement"
type="xacml:PolicySetStatementType"/>
```

5.2 Element <PolicyStatement>

The `<PolicyStatement>` element is a top-level element in the XACML schema.

```
<xs:element name="PolicyStatement"
type="xacml:PolicyStatementType"/>
```

5.3 Element <Rule>

The `<Rule>` element is a top-level element in the XACML schema.

```
<xs:element name="Rule" type="xacml:RuleType"/>
```

5.4 Complex type PolicySetStatementType

Elements of type PolicySetStatementType extend the saml:StatementAbstractType so that they MAY be included in a `<saml:Assertion>` element. The `<saml:Assertion>` element contains some policy meta-data, such as the identity of the PAP that issued the policy set statement and the date and time at which it was issued.

The main elements of this type definition are the <`Target`>, <`PolicySet`> and <`Obligations`> elements and the `policyCombiningAlgId` attribute. The <`PolicySet`> element SHALL contain references to the set of policies that are to be combined in the policy set. The <`Target`> element MAY be declared by the creator of elements of this type, or it MAY be computed from the <`Target`> elements of the referenced <`PolicyStatement`> elements, either as an intersection or

as a union. The <Obligations> element SHALL contain the set of <Obligation> elements that MUST be discharged by the PEP. The PolicyCombiningAlgId attribute SHALL contain a identifier of the policy-combining algorithm by which the referenced <PolicyStatement> elements MUST be combined.

An instance of this type MAY be referenced by its PolicySetId attribute value.

```
 <xs:complexType name="PolicySetStatementType">
    <xs:complexContent>
       <xs:extension base="saml:StatementAbstractType">
          <xs:sequence>
             <xs:element name="Description" type="xs:string"
minOccurs="0"/>
             <xs:element name="Target" type="xacml:TargetType"/>
             <xs:element name="PolicySet" type="xacml:PolicySetType"
maxOccurs="unbounded"/>
             <xs:element name="Obligations"
type="xacml:ObligationsType" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="PolicySetId" type="xs:anyURI"
use="required"/>
          <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI"
use="required"/>
       </xs:extension>
    </xs:complexContent>
 </xs:complexType>
```

5.5 Complex type PolicyStatementType

Elements of type PolicyStatementType extend the saml:StatementAbstractType so that they MAY be included in a <saml:Assertion> element. The <saml:Assertion> element contains some policy meta-data, such as the identity of the PAP that issued the policy statement and the date and time at which it was issued.

The main elements of this type definition are the <Target>, <RuleSet> and <Obligations> elements and the RuleCombiningAlgId attribute. The <RuleSet> element SHALL contain references to the <Rule> elements that are to be combined in a policy. The <Target> element MAY be declared by the creator of elements of this type, or it MAY be computed from the <Target> elements of the referenced <Rule> elements, either as an intersection or as a union. The <Obligations> element SHALL contain the set of <Obligation> elements that MUST be discharged by the PEP. The RuleCombiningAlgId attribute SHALL contain a reference to the rule-combining algorithm by which the <Rule> elements MUST be combined.

An instance of this type MAY be referenced by its PolicyId attribute value.

```
 <xs:complexType name="PolicyStatementType">
    <xs:complexContent>
       <xs:extension base="saml:StatementAbstractType">
          <xs:sequence>
             <xs:element name="Description" type="xs:string"
minOccurs="0"/>
             <xs:element name="Target" type="xacml:TargetType"/>
             <xs:element name="RuleSet" type="xacml:RuleSetType"
maxOccurs="unbounded"/>
             <xs:element name="Obligations"
type="xacml:ObligationsType" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="PolicyId" type="xs:anyURI"
use="required"/>
```

```
         <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI"
use="required"/>
      </xs:extension>
    </xs:complexContent>
 </xs:complexType>
```

5.6 Complex type RuleType

The main elements of this type definition are the <Target> and <Condition> elements, and the Effect attribute.

An instance of this type MAY be referenced by its RuleId attribute value.

```
 <xs:complexType name="RuleType">
    <xs:sequence>
      <xs:element name="Description" type="xs:string"
minOccurs="0"/>
      <xs:element name="Target" type="xacml:TargetType"
minOccurs="0"/>
      <xs:element name="Condition" type="xacml:ConditionType"
minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="RuleId" type="xs:anyURI" use="required"/>
    <xs:attribute name="Effect" type="xacml:EffectType"
use="required"/>
 </xs:complexType>
```

5.7 Complex type EffectType

This type definition defines the values allowed for the ***effect*** of a ***rule*** and the circumstances under which an ***obligation*** must be performed.

```
 <xs:simpleType name="EffectType">
    <xs:restriction base="xacmlContext:DecisionType">
      <xs:enumeration value="Permit"/>
      <xs:enumeration value="Deny"/>
    </xs:restriction>
 </xs:simpleType>
```

5.8 Complex type TargetType

Elements of this type identify the set of decision requests of type xacmlContext:RequestTYpe that the parent element is intended to evaluate. It contains definition for subjects, resources and actions. If the subject, resource and action identified in the request context match the definitions in this element, then the policy MAY be used to evaluate the request. All matches MUST be satisfied. If one or more element in the context satisfies each match, then the match is satisfied.

```
 <xs:complexType name="TargetType">
    <xs:sequence>
      <xs:element name="Subjects" type="xacml:MatchType"
maxOccurs="unbounded"/>
      <xs:element name="Resources" type="xacml:MatchType"
maxOccurs="unbounded"/>
      <xs:element name="Actions" type="xacml:MatchType"
maxOccurs="unbounded"/>
    </xs:sequence>
 </xs:complexType>
```

5.9 Complex type MatchType

Elements of type MatchType identify a set of entities by matching values in the context with values embedded in the policy. The <xacml:AttributeDesignator> element identifies one or more values in the <xacmlContext:Request> element. It MUST contain a URI that is a legal XPath expression over the <xacmlContext:Request>. The <xacml:Attribute> MUST contain a literal value. The types of the two attributes MUST be compatible with the function identified by the MatchId attribute.

```
 <xs:complexType name="MatchType">
    <xs:sequence>
      <xs:element ref="xacml:AttributeDesignator"/>
      <xs:element ref="xacml:Attribute"/>
    </xs:sequence>
    <xs:attribute name="MatchId" type="xacml:MatchIdType"/>
    <xs:attribute name="DataType" type="xs:anyURI"
fixed="xs:boolean"/>
 </xs:complexType>
```

5.10 Complex type ObligationsType

Elements of type ObligationsType contain a set of <xacml:Obligation> elements.

```
 <xs:complexType name="ObligationsType">
    <xs:sequence>
      <xs:element name="Obligation" type="xacml:ObligationType"
maxOccurs="unbounded"/>
    </xs:sequence>
 </xs:complexType>
```

5.11 Complex type ObligationType

Elements of type ObligationType contain an identifier for the obligation and a set of attributes that form arguments of the action defined by the obligation. The FulfilOn attribute indicates the decision value for which this obligation must be fulfilled.

```
 <xs:complexType name="ObligationType">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="xacml:AttributeDesignator"/>
      <xs:element name="AttributeAssignment"
type="xacml:AttributeAssignmentType"/>
    </xs:choice>
    <xs:attribute name="ObligationId" type="xs:anyURI"
use="required"/>
    <xs:attribute name="FulfilOn" type="xacml:EffectType"
use="required"/>
 </xs:complexType>
```

5.12 Element <Function>

<Function> elements reference a function of type FunctionType.

```
 <xs:element name="Function" type="xacml:FunctionType"/>
```

5.13 Complex type ConditionType

Elements of type ConditionType are functions whose return type is boolean.

```
 <xs:complexType name="ConditionType">
    <xs:complexContent>
       <xs:restriction base="xacml:FunctionType">
          <xs:choice maxOccurs="unbounded">
             <xs:element ref="xacml:Function"/>
             <xs:element ref="xacml:Attribute"/>
             <xs:element ref="xacml:AttributeDesignator"/>
          </xs:choice>
          <xs:attribute name="ConditionId"
type="xacml:ConditionIdType" use="required"/>
          <xs:attribute name="DataType" type="xs:anyURI"
fixed="xs:boolean"/>
       </xs:restriction>
    </xs:complexContent>
 </xs:complexType>
```

5.14 Complex type FunctionType

Elements of type FunctionType define a function. Xacml-defined functions are described in the accompanying table. Function definitions may take any combination of `<Function>`, `<Attribute>` and `<AttributeDesignator>` as arguments. In addition, the function's return type MUST be included in the `DataType` attribute.

```
 <xs:complexType name="FunctionType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element ref="xacml:Function"/>
       <xs:element ref="xacml:Attribute"/>
       <xs:element ref="xacml:AttributeDesignator"/>
    </xs:choice>
    <xs:attribute name="FunctionId" type="xs:anyURI"
use="required"/>
    <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
    <!-- Legal types for the first and subsequent operands are
defined in the accompanying table -->
 </xs:complexType>
```

5.15 Element <Attribute>

`<Attribute>` elements contain a literal attribute value.

```
 <xs:element name="Attribute" type="xacml:AttributeType"/>
```

5.16 Complex type AttributeType

Elements of type AttributeType contain a literal attribute value. The type of the value MUST be contained in the `DataType` attribute. The attribute MAY be of one of the xml:schema embedded types. Alternatively, it MAY be of a structured type defined in some other namespace.

```
 <xs:complexType name="AttributeType">
    <xs:complexContent>
       <xs:extension base="xs:anyType">
          <xs:attribute name="DataType" type="xs:anyURI"
use="required"/>
       </xs:extension>
    </xs:complexContent>
 </xs:complexType>
```

5.17 Element <AttributeDesignator>

<AttributeDesignator> elements reference an attribute value in <xacmlContext:Request> by means of an XPath expression. The expected type of the attribute MUST be included in the attributeDesignator.

```
 <xs:element name="AttributeDesignator"
type="xacml:AttributeDesignatorType"/>
```

5.18 Complex type AttributeDesignatorType

Elements of type AttributeDesignatorType reference an attribute value in <xacmlContext:Request>.

```
 <xs:complexType name="AttributeDesignatorType">
    <xs:attribute name="Designator" type="xs:anyURI"/>
    <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
    <!-- Designator must be a legal XPath expression over
xacmlContext:Request -->
 </xs:complexType>
```

5.19 Complex type AttributeAssignmentType

Elements of type AttributeAssignmentType contain attribute contents and an AttributeId. The AttributeId is part of attribute meta-data, and is used when an attribute cannot be referenced by its location in <xacmlContext:Request>. This situation may arise in <Obligation>.

```
 <xs:complexType name="AttributeAssignmentType">
    <xs:complexContent>
       <xs:extension base="xacml:AttributeType">
          <xs:attribute name="AttributeId" type="xs:anyURI"/>
       </xs:extension>
    </xs:complexContent>
 </xs:complexType>
```

5.20 Complex type PolicySetType

Elements of type PolicySetType identify a set of policies. Members of the set MAY be identified by any of the following: reference to a <PolicySetStatement> by its PolicySetId attribute, reference to a <PolicyStatement> by its PolicyId attribute, inclusion of a <PolicySetStatement>, inclusion of a <PolicyStatement>, inclusion of a <saml:Assertion> containing a <PolicySetStatement> or inclusion of a <saml:Assertion> containing a <PolicyStatement>. The referenced policies MUST be combined as defined by the policy-combining algorithm identified by the PolicyCombiningAlgId attribute in the parent <PolicySetStatement>.

```
 <xs:complexType name="PolicySetType">
    <xs:choice maxOccurs="unbounded">
       <xs:element name="PolicySetId" type="xs:anyURI"/>
       <xs:element name="PolicyId" type="xs:anyURI"/>
       <xs:element ref="xacml:PolicySetStatement"/>
       <xs:element ref="xacml:PolicyStatement"/>
       <xs:element name="PolicySetAssertion"
type="saml:AssertionType"/>
       <xs:element name="PolicyAssertion"
type="saml:AssertionType"/>
    </xs:choice>
```

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

5.21 Complex type RuleSetType

Elements of type RuleSetType SHALL contain a set of <Rule> or <RuleDesignator> elements.

```
 <xs:complexType name="RuleSetType">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="xacml:Rule"/>
      <xs:element name="RuleDesignator"
type="xacml:RuleDesignatorType"/>
    </xs:choice>
 </xs:complexType>
```

5.22 Complex type RuleDesignatorType

Elements of type RuleDesignatorType SHALL designate a rule by identifier or by digest.

```
 <xs:complexType name="RuleDesignatorType">
    <xs:sequence>
      <xs:element name="RuleId" type="xs:anyURI" minOccurs="0"/>
      <xs:element name="RuleDigest" minOccurs="0">
        <xs:complexType>
          <xs:attribute name="DigestAlgId" type="xs:string"
default="SHA-1"/>
          <xs:attribute name="Base64Digest" type="xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
 </xs:complexType>
```

6 Function names and legal type combinations

6.1 Functions

The table in this section lists the combinations of datatypes for which the various functions. For each function name, the table indicates the valid combination of datatypes and the datatype of the result.

Function Name	Function DataType	First operand DataType	Remaining operands DataType	Operation
function:integer-add	xs:integer	xs:integer	xs:integer	A + B
function:decimal-add	xs:decimal	xs:decimal	xs:decimal	A + B
function:add-yearMonthDuration-to-date	xs:date	xs:date	xs:yearMonthDuration	A + B
function:add-yearMonthDuration-to-date	xs:date	xs:date	xs:dayTimeDuration	A + B
function:add-dayTimeDuration-	xs:time	xs:time	xs:dayTimeDur	A + B

to-time			ation	
function:add-yearMonthDuration-to-dateTime	xs:dateTime	xs:datetime	xs:yearMonthDuration	A + B
function:add-dayTimeDuration-to-dateTime	xs:dateTime	xs:datetime	xs:dayTimeDuration	A + B
function:add-yearMonthDurations	xs:yearMonthDuration	xs:yearMonthDuration	xs:yearMonthDuration	A + B
function:add-dayTimeDurations	xs:dayTimeDuration	xs:dayTimeDuration	xs:dayTimeDuration	A + B
function:integer-subtract	xs:integer	xs:integer	xs:integer	A - B
function:decimal-subtract	xs:decimal	xs:decimal	xs:decimal	A - B
function:date-subtract	xs:dayTimeDuration	xs:date	xs:date	A - B (only two operands allowed)
function:subtract-yearMonthDuration-from-date	xs:date	xs:date	xs:yearMonthDuration	A - B
function:subtract-dayTimeDuration-from-date	xs:date	xs:date	xs:dayTimeDuration	A - B
function:time-subtract	xs:dayTimeDuration	xs:time	xs:time	A - B (only two operands allowed)
function:subtract-dayTimeDuration-from-time	xs:time	xs:time	xs:dayTimeDuration	A - B
function:datetime-subtract	xs:dayTimeDuration	xs:datetime	xs:datetime	A - B (only two operands allowed)
function:subtract-yearMonthDuration-from-dateTime	xs:dateTime	xs:datetime	xs:yearMonthDuration	A - B
function:subtract-dayTimeDuration-from-dateTime	xs:dateTime	xs:datetime	xs:dayTimeDuration	A - B
function:subtract-yearMonthDurations	xs:yearMonthDuration	xs:yearMonthDuration	xs:yearMonthDuration	A - B
function:subtract-dayTimeDurations	xs:dayTimeDuration	xs:dayTimeDuration	xs:dayTimeDuration	A - B
function:integer-multiply	xs:integer	xs:integer	xs:integer	A * B
function:decimal-multiply	xs:decimal	xs:decimal	xs:decimal	A * B
function:multiply-	xs:yearMonthD	xs:yearMonthD	xs:decimal	A * B

yearMonthDurations	uration	uration		
function:multiply-dayTimeDurations	xs:dayTimeDuration	xs:dayTimeDuration	xs:decimal	A * B
function:integer-divide	xs:integer	xs:integer	xs:integer	A div B
function:decimal-divide	xs:decimal	xs:decimal	xs:decimal	A div B
function:divide-yearMonthDurations	xs:yearMonthDuration	xs:yearMonthDuration	xs:decimal	A div B
function:divide-dayTimeDurations	xs:dayTimeDuration	xs:dayTimeDuration	xs:decimal	A div B
function:integer-mod	xs:integer	xs:integer	xs:integer	A mod B
function:decimal-mod	xs:decimal	xs:decimal	xs:decimal	A mod B
function:round	xs:integer	xs:decimal		
function:integer	xs:integer	xs:decimal		
function:decimal	xs:decimal	xs:integer		
function:integer-equal	xs:boolean	xs:integer	xs:integer	A eq B
function:decimal-equal	xs:boolean	xs:decimal	xs:decimal	A eq B
function:boolean-equal	xs:boolean	xs:boolean	xs:boolean	A eq B
function:string-equal	xs:boolean	xs:string	xs:string	A eq B
function:rfc822Name-equal	xs:boolean	Identifier:rfc822Name	Identifier:rfc822Name	A eq B
function:x500Name-equal	xs:boolean	Identifier:x500Name	Identifier:x500Name	A eq B
function:date-equal	xs:boolean	xs:date	xs:date	A eq B
function:time-equal	xs:boolean	xs:time	xs:time	A eq B
function:datetime-equal	xs:boolean	xs:dateTime	xs:dateTime	A eq B
function:yearMonthDuration-equal	xs:boolean	xs:yearMonthDuration	xs:yearMonthDuration	A eq B
function:dayTimeDuration-equal	xs:boolean	xs:dayTimeDuration	xs:dayTimeDuration	A eq B
function:gregorian-equal	xs:boolean	Gregorian	Gregorian	A eq B
function:hex-binary-equal	xs:boolean	xs:hexBinary	xs:hexBinary	A eq B
function:base64-binary-equal	xs:boolean	xs:base64Binary	xs:base64Binary	A eq B

function:anyURI-equal	xs:boolean	xs:anyURI	xs:anyURI	A eq B
function:QName-equal	xs:boolean	xs:QName	xs:QName	A eq B
function:NOTATION-equal	xs:boolean	xs:NOTATION	xs:NOTATION	A eq B
function:numeric-not-equal	xs:boolean	numeric	numeric	A ne B
function:boolean-not-equal	xs:boolean	xs:boolean	xs:boolean	A ne B
function:string-not-equal	xs:boolean	xs:string	xs:string	A ne B
function:date-not-equal	xs:boolean	xs:date	xs:date	A ne B
function:time-not-equal	xs:boolean	xs:time	xs:time	A ne B
function:datetime-not-equal	xs:boolean	xs:dateTime	xs:dateTime	A ne B
function:yearMonthDuration-not-equal	xs:boolean	xs:yearMonthDuration	xs:yearMonthDuration	A ne B
function:dayTimeDuration-not-equal	xs:boolean	xs:dayTimeDuration	xs:dayTimeDuration	A ne B
function:gregorian-not-equal	xs:boolean	Gregorian	Gregorian	A ne B
function:hex-binary-not-equal	xs:boolean	xs:hexBinary	xs:hexBinary	A ne B
function:base64-binary-not-equal	xs:boolean	xs:base64Binary	xs:base64Binary	A ne B
function:anyURI-not-equal	xs:boolean	xs:anyURI	xs:anyURI	A ne B
function:QName-not-equal	xs:boolean	xs:QName	xs:QName	A ne B
function:NOTATION-not-equal	xs:boolean	xs:NOTATION	xs:NOTATION	A ne B
function:integer-greater-than	xs:boolean	xs:integer	xs:integer	A gt B
function:decimal-greater-than	xs:boolean	xs:decimal	xs:decimal	A gt B
function:boolean-greater-than	xs:boolean	xs:boolean	xs:boolean	A gt B
function:string-greater-than	xs:boolean	xs:string	xs:string	A gt B
function:date-greater-than	xs:boolean	xs:date	xs:date	A gt B
function:time-greater-than	xs:boolean	xs:time	xs:time	A gt B
function:datetime-greater-than	xs:boolean	xs:dateTime	xs:dateTime	A gt B
function:yearMonthDuration-greater-than	xs:boolean	xs:yearMonthDuration	xs:yearMonthDuration	A gt B
function:dayTimeDuration-greater-than	xs:boolean	xs:dayTimeDuration	xs:dayTimeDuration	A gt B
function:integer-less-than	xs:boolean	xs:integer	xs:integer	A lt B

function:decimal-less-than	xs:boolean	xs:decimal	xs:decimal	A lt B
function:boolean-less-than	xs:boolean	xs:boolean	xs:boolean	A lt B
function:string-less-than	xs:boolean	xs:string	xs:string	A lt B
function:date-less-than	xs:boolean	xs:date	xs:date	A lt B
function:time-less-than	xs:boolean	xs:time	xs:time	A lt B
function:datetime-less-than	xs:boolean	xs:dateTime	xs:dateTime	A lt B
function:yearMonthDuration-less-than	xs:boolean	xs:yearMonthDuration	xs:yearMonthDuration	A lt B
function:dayTimeDuration-less-than	xs:boolean	xs:dayTimeDuration	xs:dayTimeDuration	A lt B
function:integer-greater-than-or-equal	xs:boolean	xs:integer	xs:integer	A ge B
function:decimal-greater-than-or-equal	xs:boolean	xs:decimal	xs:decimal	A ge B
function:string-greater-than-or-equal	xs:boolean	xs:string	xs:string	A ge B
function:date-greater-than-or-equal	xs:boolean	xs:date	xs:date	A ge B
function:time-greater-than-or-equal	xs:boolean	xs:time	xs:time	A ge B
function:datetime-greater-than-or-equal	xs:boolean	xs:dateTime	xs:dateTime	A ge B
function:yearMonthDuration-greater-than-or-equal	xs:boolean	xs:yearMonthDuration	xs:yearMonthDuration	A ge B
function:dayTimeDuration-greater-than-or-equal	xs:boolean	xs:dayTimeDuration	xs:dayTimeDuration	A ge B
function:integer-less-than-or-equal	xs:boolean	xs:integer	xs:integer	A le B
function:decimal-less-than-or-equal	xs:boolean	xs:decimal	xs:decimal	A le B
function:numeric-less-than-or-equal	xs:boolean	xs:string	xs:string	A le B
function:date-less-than-or-equal	xs:boolean	xs:date	xs:date	A le B
function:time-less-than-or-equal	xs:boolean	xs:time	xs:time	A le B
function:datetime-less-than-or-equal	xs:boolean	xs:dateTime	xs:dateTime	A le B

function:yearMonthDuration-less-than-or-equal	xs:boolean	xs:yearMonthDuration	xs:yearMonthDuration	A le B
function:dayTimeDuration-less-than-or-equal	xs:boolean	xs:dayTimeDuration	xs:dayTimeDuration	A le B
function:string-match	xs:boolean	xs:string	xs:string	
function:and	xs:boolean	xs:boolean	xs:boolean	A & B
function:or	xs:boolean	xs:boolean	xs:boolean	A | B
function:ordered-or	xs:boolean	xs:boolean	xs:boolean	A | B
function:n-of	xs:boolean	numeric	xs:boolean	
function:not	xs:boolean	xs:boolean		(only one operand allowed)
function:present	xs:boolean	xs:anyURI		
function:subset	xs:boolean	xs:list	xs:list	
function:superset	xs:boolean	xs:list	xs:list	
function:non-null-set-intersection	xs:boolean	xs:list	xs:list	

7 Context syntax (normative, with the exception of the schema fragments)

7.1 Element <Request>

The `<Request>` element is a top-level element in the XACML context schema.

```
<xs:element name="Request" type="xacmlContext:RequestType"/>
```

7.2 Element <Response>

The `<Resonse>` element is a top-level element in the XACML context schema.

```
<xs:element name="Response" type="xacmlContext:ResponseType"/>
```

7.3 Complex type RequestType

Elements of type RequestType contain the data required by the PDP in order to render a decision. This includes information about the subjects, resource and actions, as well as environmental information that pertains to none of these.

```
<xs:complexType name="RequestType">
  <xs:sequence>
```

```
      <xs:element name="Subject" type="xacmlContext:SubjectType"
maxOccurs="unbounded"/>
      <xs:element name="Resource"
type="xacmlContext:ResourceType"/>
      <xs:element name="Action" type="xacmlContext:ActionType"
maxOccurs="unbounded"/>
      <xs:element name="Environment"
type="xacmlContext:EnvironmentType" minOccurs="0"/>
    </xs:sequence>
 </xs:complexType>
```

7.4 Complex type ResponseType

Elements of type ResponseType contain one or more results of a policy evlaution.

```
 <xs:complexType name="ResponseType">
    <xs:sequence>
      <xs:element name="Result" type="xacmlContext:ResultType"
maxOccurs="unbounded"/>
    </xs:sequence>
 </xs:complexType>
```

7.5 Complex type ResultType

Elements of type ResultType contain information related to a single decision, including the value of the decision, the resource to which it relates, and any obligations and advice associated with the decision.

```
 <xs:complexType name="ResultType">
    <xs:sequence>
      <xs:element name="Decision"
type="xacmlContext:DecisionType"/>
      <xs:element name="ResourceId" type="xs:string"
minOccurs="0"/>
      <xs:element name="Obligations" type="xacml:ObligationsType"
minOccurs="0"/>
      <xs:element name="Advice" type="xacmlContext:AdviceType"
minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
 </xs:complexType>
```

7.6 Complex type SubjectType

Elements of type SubjectType identify a subject of a request context by means of an identifier or a key. Optionally, attributes of the subject MAY be provided and information relating to the PEP's authentication of the subject MAY be supplied.

```
 <xs:complexType name="SubjectType">
    <xs:sequence>
      <xs:choice minOccurs="0">
        <xs:element name="SubjectId"
type="xacmlContext:SubjectIdType"/>
        <xs:element ref="ds:KeyInfo"/>
      </xs:choice>
      <xs:element name="SubjectAttribute"
type="xacmlContext:AttributeType" minOccurs="0"
maxOccurs="unbounded"/>
```

```
      <xs:element name="AuthenticationInfo"
type="xacmlContext:AuthenticationInfoType" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="SubjectCategory" type="xs:anyURI"
default="identifier:AccessSubject"/>
 </xs:complexType>
```

7.7 Complex type SubjectIdType

Elements of type SubjectIdType contain information that identifies a subject. The identifier itself is a string. However, `Format` and `Qualifier` attributes are included to assist with the interpretation of the string. The `Format` attribute indicates the name-form of the identifier and hence the function by which it MUST be matched. (Note: why not call this "DataType", to be consistent throughout?) The qualifier indicates the security or administrative domain that qualifies the name of the subject. It provides a means to federate names from disparate user stores without collision. (Note: Why isn't this a name, with an accompanying DataType? Why isn't it a list of names?).

```
 <xs:complexType name="SubjectIdType">
   <xs:simpleContent>
     <xs:extension base="xs:string">
       <xs:attribute name="Format" type="xs:anyURI"
use="optional"/>
       <xs:attribute name="Qualifier" type="xs:string"
use="optional"/>
     </xs:extension>
   </xs:simpleContent>
 </xs:complexType>
```

7.8 Complex type AuthenticationInfoType

Elements of this type contain information related to the PEP's authentication of the subject.

```
 <xs:complexType name="AuthenticationInfoType">
   <xs:attribute name="Method" type="xs:anyURI" use="optional"/>
   <xs:attribute name="Instant" type="xs:dateTime" use="optional"/>
 </xs:complexType>
```

7.9 Complex type AttributeType

Elements of this type contain an attribute and attribute meta-data. It extends the xacml definition of attribute with an `AttributeId`, and Issuer `identity` and an `IssueInstant`.

```
 <xs:complexType name="AttributeType">
   <xs:complexContent>
     <xs:extension base="xacml:AttributeType">
       <xs:attribute name="AttributeId" type="xs:anyURI"
use="required"/>
       <xs:attribute name="Issuer" type="xs:anyURI"
use="optional"/>
       <xs:attribute name="IssueInstant" type="xs:dateTime"
use="optional"/>
     </xs:extension>
   </xs:complexContent>
 </xs:complexType>
```

7.10 Complex type ResourceType

Elements of this type contain information about the resource for which access is being requested. It MAY contain any combination of `<ResourceSpecifier>`, `<ResourceContent>` and `<ResourceAttribute>` elements. If present, the `<ResourceAttribute>` elements contain a an attribute of the resource.

```
 <xs:complexType name="ResourceType">
    <xs:sequence>
       <xs:element name="ResourceSpecifier"
type="xacmlContext:ResourceSpecifierType" minOccurs="0"/>
       <xs:element name="ResourceContent"
type="xacmlContext:ResourceContentType" minOccurs="0"/>
       <xs:element name="ResourceAttribute"
type="xacmlContext:AttributeType" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
 </xs:complexType>
```

7.11 Complex type ResourceSpecifierType

Elements of this type SHALL contain a ResourceId. This is in the form of a string. Interpretation of the string depends upon the value of the `Format` attribute. (Note: Perhaps the format attribute should be required). The scope attribute is used in the case where the resource is structured as a hierarchy. It indicates which part of the resource the decision request applies to.

```
 <xs:complexType name="ResourceSpecifierType">
    <xs:attribute name="Format" type="xs:anyURI" use="optional"/>
    <xs:attribute name="Scope"
type="xacmlContext:SpecifierScopeType" use="optional"/>
    <xs:attribute name="ResourceId" type="xs:string"
use="required"/>
 </xs:complexType>
```

7.12 Complex type SpecifierScopeType

Elements of this type indicate which part of a resource a decision request applies to. The value Immediate indicates the request applies just to the node of the resource identified by the ResourceId in the parent element. The Children value indicates that the request applies to the node identified in the parent element and its immediate children. The Descendants value indicates that the request applies to the node identified in the parent element and all its descendants.

```
 <xs:simpleType name="SpecifierScopeType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="Immediate"/>
       <xs:enumeration value="Children"/>
       <xs:enumeration value="Descendants"/>
    </xs:restriction>
 </xs:simpleType>
```

7.13 Complex type ResourceContentType

Elements of this type contain the resource to which access is requested.

```
 <xs:complexType name="ResourceContentType">
    <xs:sequence>
       <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
```

```
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>
```

7.14 Complex type ActionType

Elements of type ActionType contain a specification of the requested actions (Note: should this be of type xs:list? It would mean that an individual action could not contain whitespace).

```
  <xs:complexType name="ActionType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="Namespace" type="xs:anyURI"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
```

7.15 Complex type DecisionType

Elements of type DecisionType contain the result of policy evaluation.

```
  <xs:simpleType name="DecisionType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Permit"/>
      <xs:enumeration value="Deny"/>
      <xs:enumeration value="Indeterminate"/>
    </xs:restriction>
  </xs:simpleType>
```

7.16 Complex type EnvironmentType

Elements of type EnvironmentType contain a set of attributes of the environment. These attributes MAY form part of policy evaluation.

```
  <xs:complexType name="EnvironmentType">
    <xs:sequence>
      <xs:element name="EnvironmentAttribute"
type="xacmlContext:AttributeType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

7.17 Complex type AdviceType

Elements of type AdviceType contain information that MAY be used by the PEP. (Note: if we don't have a specific use for this, why don't we leave it out in this version? Users of the specification will still be able to extend the response schema to include advice, if they have a definite need for it).

```
  <xs:complexType name="AdviceType">
    <xs:sequence>
      <xs:any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="AdviceId" type="xs:anyURI"/>
  </xs:complexType>
</xs:schema>
```

8 XACML identifiers (normative)

This section defines standard identifiers for commonly-used entities. All XACML-defined identifiers have the common base:

```
urn:oasis:names:tc:XACML:identifier
```

8.1 Access Subject

The identifier for the system entity that is requesting access.

```
urn:oasis:names:tc:xacml:identifier:AccessSubject
```

8.2 Time of day

8.3 Attributes

XACML-defined attributes are represented by an element of type <saml:AttributeDesignatorType>. It has two attributes: AttributeNamespace and AttributeName. All XACML-defined attributes have the following value for AttributeNamespace:

```
urn:oasis:names:tc:XACML:identifier:attributes/
```

8.3.1 Role

```
urn:oasis:names:tc:XACML:identifier:attributes/role
```

8.3.2 RFC822 Name

RFC822 name attributes have the following value of AttributeName:

```
urn:oasis:names:tc:XACML:identifier:rfc822Name
```

8.3.3 X.500 distinguished name

X.500 distinguished name attributes have the following value of AttributeName:

```
urn:oasis:names:tc:XACML:identifier:x500Name
```

8.3.4 Unix file-system path

UNIX file-system path attributes have the following value of AttributeName:

```
urn:oasis:names:tc:XACML:identifier:attribute:UFS
```

8.3.5 Uniform resource identifier

Uniform resource identifier attributes have the following value of AttributeName:

```
urn:oasis:names:tc:XACML:identifier:attribute:URI
```

8.4 Authentication locality

8.5 Deny-overrides rule-combining algorithm

The deny-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

```
urn:oasis:names:tc:XACML:identifier:ruleCombiningAlgorithms:denyOverrides
```

8.6 Deny-overrides policy-combining algorithm

The deny-overrides policy-combining algorithm has the following value for policyCombiningAlgId:

```
urn:oasis:names:tc:XACML:identifier:policyCombiningAlgorithms:denyOverrides
```

8.7 Permit-overrides rule-combining algorithm

The permit-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

```
urn:oasis:names:tc:XACML:identifier:ruleCombiningAlgorithms:permitOverrides
```

8.8 Permit-overrides policy-combining algorithm

The permit-overrides policy-combining algorithm has the following value for policyCombiningAlgId:

```
urn:oasis:names:tc:XACML:identifier:policyCombiningAlgorithms:permitOverrides
```

9 Combining algorithms (normative)

This section contains a description of the rule-combining and policy-combining algorithms specified by XACML.

9.1 Deny-overrides

The following is a specification for the "deny-overrides" rule-combining algorithm. The identifier for this algorithm is given in Section 8.5.

> In the entire set of rules to be evaluated, if any of the rules evaluates to “deny”, then the rule combination is defined to evaluate to “deny” (that is, “deny” takes precedence, regardless of how many rules evaluate to “permit”, and causes the whole combination to return “deny”). Any rule that evaluates to “indeterminate” (that is, its return status cannot be determined for any reason) has the same effect as a “deny” in that it causes the combination to return “deny”. Finally, if none of the rules are found to be applicable to the request, the rule combination returns “notApplicable”.

What follows is a pseudocode representation of how the above specification MAY be implemented. This is provided for illustrative and explanatory purposes.

```
effect policy(rule[]){
 atLeastOnePermit = false;
 for( i=0; i<=noOfRules; i++ ){
   if(rule[i] == deny){
     return(deny);
```

```
    }
    if(rule[i] == indeterminate){
       return(deny);
    }
    if(rule[i] == permit){
       atLeastOnePermit = true;
    }
  }
  if atLeastOnePermit {
     return(permit);
  }
  else{
     return(notApplicable);
  }
}
```

The following is a specification for the "deny-overrides" policy-combining algorithm. The identifier for this algorithm is given in Section 7.6.

> In the entire set of policies to be evaluated, if any of the policies evaluates to “deny”, then the policy combination is defined to evaluate to “deny” (that is, “deny” takes precedence, regardless of how many policies evaluate to “permit”, and causes the whole combination to return “deny”). Any policy that evaluates to “indeterminate” (that is, its return status cannot be determined for any reason) has the same effect as a “deny” in that it causes the combination to return “deny”. Finally, if none of the policies are found to be applicable to the request, the policy combination returns “notApplicable”.

What follows is a pseudocode representation of how the above specification MAY be implemented. This is provided for illustrative and explanatory purposes.

```
effect policySet(policy[]){
  atLeastOnePermit = false;
  for( i=0; i<=noOfPolicies; i++ ){
     if(policy[i] == deny){
        return(deny);
     }
     if(policy[i] == indeterminate){
        return(deny);
     }
     if(policy[i] == permit){
        atLeastOnePermit = true;
     }
  }
  if atLeastOnePermit {
     return(permit);
  else{
     return(notApplicable);
  }
}
```

Obligations of the individual policies SHALL be combined as described in Section 4.3.2.3.

9.2 Permit-overrides

The following is a specification for the "permit-overrides" rule-combining algorithm. The identifier for this algorithm is given in Section 8.7.

> In the entire set of rules to be evaluated, if any of the rules evaluates to "permit", then the rule combination is defined to evaluate to "permit" (that is, "permit" takes precedence, regardless of how many rules evaluate to "deny" or "indeterminate", and causes the whole combination to return "permit"). If all of the rules found to be applicable to the request

evaluate to "deny" or "indeterminate", then the rule combination is defined to evaluate to "deny". If none of the rules is found to be applicable to the request, the rule `combination returns "notApplicable".`

What follows is a pseudocode representation of how the above specification MAY be implemented. This is provided for illustrative and explanatory purposes.

```
effect policy(rule[]) {
    atLeastOneDenyOrIndeterminate = false;
    for( i=0; i<=noOfRules; i++) {
        if (rule[i] == permit) {
            return(permit);
        }
        if (rule[i] == indeterminate) {
            atLeastOneDenyOrIndeterminate = true;
        }
        if (rule[i] == deny) {
            atLeastOneDenyOrIndeterminate = true;
        }
    }
    if atLeastOneDenyOrIndeterminate {
        return(deny);
    } else {
        return(notApplicable);
    }
}
```

The following is a specification for the "permit-overrides" policy-combining algorithm. The identifier for this algorithm is given in Section 8.8.

In the entire set of policies to be evaluated, if any of the policies evaluates to "permit", then the policy combination is defined to evaluate to "permit" (that is, "permit" takes precedence, regardless of how many policies evaluate to "deny" or "indeterminate", and causes the whole combination to return "permit"). If all of the policies found to be applicable to the request evaluate to "deny" or "indeterminate", then the policy combination is defined to evaluate to "deny". If none of the policies is found to be applicable to the request, the policy combination returns "notApplicable".

What follows is a pseudocode representation of how the above specification MAY be implemented. This is provided for illustrative and explanatory purposes.

```
effect policySet(policy[]) {
    atLeastOneDenyOrIndeterminate = false;
    for( i=0; i<=noOfPolicies; i++) {
        if (policy[i] == permit) {
            return(permit);
        }
        if (policy[i] == indeterminate) {
            atLeastOneDenyOrIndeterminate = true;
        }
        if (policy[i] == deny) {
            atLeastOneDenyOrIndeterminate = true;
        }
    }
    if atLeastOneDenyOrIndeterminate {
        return(deny);
    } else {
        return(notApplicable);
    }
}
```

Obligations of the individual policies SHALL be combined as described in Section 4.3.2.3.

10 Profiles (normative but not mandatory to implement)

10.1 XACML

Describes subsets of XACML appropriate to general classes of problem

10.2 SAML

Describes the subset of SAML that is relevant to XACML

We need to specify SAML status codes for situations specific to XACML, such as:

- ***PDP*** has no policy for the requested target
- ***PDP*** cannot retrieve the required attributes

A compliant SAML-based PDP MUST reply to a SAML Authorization Decision Request with a SAML Authorization Decision in accordance with operational semantics of the PDP stated in Section 10.1.

The following XSLT defines the transformation from a saml:AuthorizationDecision request to the xacml request context. (Note: This has not been updated in accordance with the latest context schema.)

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
xmlns:saml='http://www.oasis-
open.org/committees/security/docs/draft-sstc-schema-assertion-
28.xsd' xmlns:samlp="http://www.oasis-
open.org/committees/security/docs/draft-sstc-schema-protocol-
28.xsd">

<xsl:template match="samlp:Request">
  <Request>
    <xsl:apply-templates
select="samlp:AuthorizationDecisionQuery/saml:Subject"/>
    <xsl:apply-templates
select="samlp:AuthorizationDecisionQuery/saml:Action"/>
    <xsl:apply-templates select="samlp:AuthorizationDecisionQuery"
mode="Resource"/>
  </Request>
</xsl:template>

<xsl:template match="saml:NameIdentifier">
  <xsl:element name="Subject">
    <xsl:attribute name="SubjectCategory">

<xsl:text>urn:oasis:names:tc:xacml:identifiers:AccessSubject</xsl:te
xt>
    </xsl:attribute>
    <xsl:element name="SubjectId">
      <xsl:if test="@NameQualifier">
        <xsl:attribute name="NameQualifier">
          <xsl:value-of select="@NameQualifier"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:if test="@Format">
```

```
        <xsl:attribute name="Format">
          <xsl:value-of select="@Format"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:value-of select="."/>
    </xsl:element>
    <xsl:apply-templates select="//saml:Evidence" mode="Evidence"/>
  </xsl:element>
</xsl:template>

<xsl:template match="saml:SubjectConfirmation">
  <xsl:element name="SubjectAttribute">
    <xsl:element name="SubjectMetaData">
      <xsl:attribute name="Name">
        <xsl:text>SubjectConfirmation</xsl:text>
      </xsl:attribute>
    </xsl:element>
    <xsl:element name="AttributeValue">
      <xsl:copy-of select="."/>
    </xsl:element>
  </xsl:element>
</xsl:template>

<xsl:template match="saml:Evidence" mode="Evidence">
  <xsl:element name="SubjectAttribute">
    <xsl:element name="SubjectMetaData">
      <xsl:attribute name="Name">
        <xsl:text>Evidence</xsl:text>
      </xsl:attribute>
    </xsl:element>
    <xsl:element name="AttributeValue">
      <xsl:copy-of select="."/>
    </xsl:element>
  </xsl:element>
</xsl:template>

<xsl:template match="samlp:AuthorizationDecisionQuery"
mode="Resource">
  <xsl:element name="Resource">
    <xsl:element name="ResourceSpecifier">
      <xsl:attribute name="ResourceURI">
        <xsl:value-of select="@Resource"/>
      </xsl:attribute>
    </xsl:element>
  </xsl:element>
</xsl:template>

<xsl:template match="saml:Action">
  <Action>
    <xsl:if test="@Namespace">
      <xsl:value-of select='@Namespace'/>
      <xsl:text>/</xsl:text>
    </xsl:if>
    <xsl:value-of select="."/>
  </Action>
</xsl:template>

</xsl:stylesheet>
```

10.3 XML Digital Signature

Describes how XACML instances shall be integrity-protected in the case where XML DSig is used. ***PAP***s MAY sign XACML <policyStatement> elements. When a ***PAP*** combines <policyStatement> elements, it MAY sign the resulting <policySetStatement> element.

10.4 LDAP

The `<policyStatement>` and `<policySetStatement>` elements MAY be distributed from the PAP to the PDP by means of an LDAP repository. In this case, conformant implementations SHALL behave as described in this section.

10.4.1 Directory information tree (DIT)

The `<xacml:target>` element conforms to a data model. XACML does not specify the ***target*** data model, but it MUST be agreed between the ***PAP*** and the ***PDP***. The data model MUST be semi-hierarchical. That is, it MUST have one or more disjoint trees for ***resources*** and/or ***subjects***. ***Actions*** are leaf nodes of the ***resource*** node to which they apply (see Figure 4). Each level in the tree is identified with an ***attribute*** name. A "path" in the tree is a list of attribute-name/value pairs linking a node to the root. The form of a ***target*** is a set of paths, one or more for each tree in the data model.

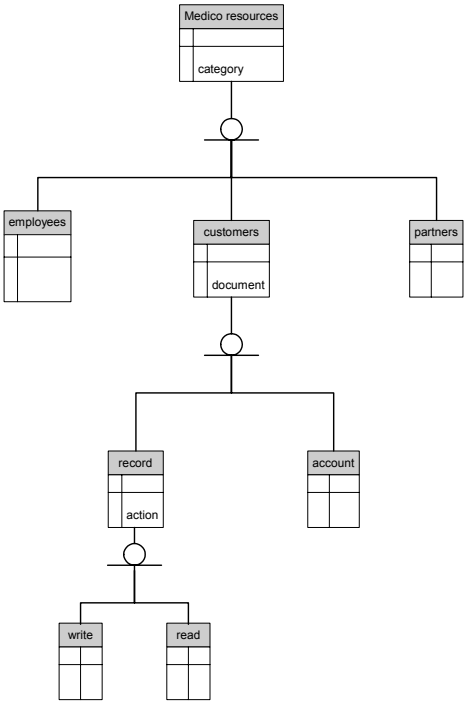

Figure 4 - Medico Inc "resource" tree

Figure 4 gives an example of a resource tree. One path in this tree is defined by the following list of attribute-name/value pairs:

```
Root = Medico resources: category = customer: document = record: action =
read
```

The Directory Information Tree of the repository SHALL be congruent with that of the ***target*** data model. A `<policyStatement>` element shall be an LDAP attribute of the entry at the lowest

node of every path in the DIT. In our example, the policy for reading a customer record SHALL be an attribute of the entry defined by the above path. In practice, the ***policy statements*** may be referenced from these entries rather than stored at them.

A node MAY have more than one ***target*** associated with it.

An authorization ***decision request*** also specifies a set of paths by (directly or indirectly) providing ***resource***, ***subject*** and ***action attribute*** values.

A ***policy statement*** is said to be applicable to a ***decision request*** if and only if every path in the policy statement's `<target>` element is part of a path in the input context's `<Request>` element.

For instance, a ***policy statement*** whose ***target*** is:

```
Root = Medico resources: category = customer
```

is attached to that entry in the DIT and is applicable to an input context whose `<Request>` element identifies the following resource/action:

```
Root = Medico resources: category = customer: document = record: action =
read
```

10.4.2 Policy combination

When ***policy statements*** are combined in a ***policy set statement***, the ***policy set statement target*** MUST be computed, and the repository must be updated. ***Policy statements*** that conform to different ***target*** data models MUST NOT be combined.

The ***policy set statement target*** SHALL be computed by separately combining trees of the same type from each of the original ***policy statement targets***. The combination may be in the form of a union or an intersection.

A union combination retains all of the original paths. If, as the result, all possible paths containing a particular DIT node are retained, then the path may be truncated at that node.

An intersection combination retains a path from one ***target*** if and only if it includes a path from the other ***target***.

The ***policy set statement*** SHOULD be stored at the lowest node of every retained path.

Some attribute values may (themselves) have an internal tree structure (e.g. DNS names). Sub-trees of such structures SHALL be represented by a regular expression (ref). When such an ***attribute*** defines a level in a ***target*** tree, the sub-tree defined by each node at that level SHALL be attached at that node.

10.4.3 Directory schema

This directory schema defines an auxiliary object class (**xacmlPolicyInfo**) for adding XACML policy data to entries, as well as a directory attribute (**xacmlPolicyData**) to contain the policies or references to entries containing policies.

Alternatively, XACML policies may be stored in policy-specific entries and referenced from the resource, action and/or subject entries to which they relate. This schema defines a structural object class (**xacmlPolicyObject**) for defining such entries, as well as a directory attribute (**xacmlPolicyRDN**) to contain the string used to name the policy-specific entry in the directory. The **xacmlPolicyData** directory attribute is also used in these entries to contain the policies themselves.

A ***PDP*** uses an LDAP Directory User Agent (DUA) to search the resource/subject trees in the directory to find the resource, action or subject of interest and retrieve the **xacmlPolicyData** directory attribute from that entry. That attribute may contain the XACML policy or a pointer to

another directory entry that contains the XACML policy. If it contains only a pointer, the PDP must query the directory again to retrieve the **xacmlPolicyData** directory attribute from the entry related to the pointer. The content of the pointer is the value of the **xacmlPolicyRDN** directory attribute that is the final Relative Distinguished Name (RDN) for the policy entry in the directory.

It is the ***PDP's*** responsibility to confirm that the retrieved policy is applicable to the ***decision request*** (i.e., the input context) that it is processing.

10.4.4 Object Class Definitions

The following object classes are defined for the LDAP profile for XACML.

10.4.4.1 XACML Policy Info

The **xacmlPolicyInfo** object class is used in defining entries for objects that hold XACML policy information in addition to other data (e.g., as part of a resource, action, or subject entry).

```
xacmlPolicyInfo   OBJECT-CLASS  ::= {
 SUBCLASS OF   {top}
 KIND    auxiliary
 MAY CONTAIN   {xacmlPolicyData}
 ID    id-???-oc-xacmlPolicyInfo }
```

10.4.4.2 XACML Policy Object

The **xacmlPolicyObject** object class is used in defining entries for objects that hold only XACML policy information.

```
xacmlPolicyObject OBJECT-CLASS  ::= {
 SUBCLASS OF   {top}
 KIND    structural
 MUST CONTAIN  {xacmlPolicyRDN}
 MAY CONTAIN   {xacmlPolicyData}
 ID    id-???-oc-xacmlPolicyObject }
```

The **xacmlPolicyRDN** directory attribute is used to name the entry and position it in a policy subtree.

10.4.5 Attribute Definitions

The following directory attributes are defined for the LDAP profile for XACML.

10.4.5.1 XACML Policy Data

The **xacmlPolicyData** directory attribute is used to store XACML policy information.

```
xacmlPolicyData   ATTRIBUTE  ::= {
 WITH SYNTAX   XacmlPolicySyntax
 ID    id-???-at-xacmlPolicyData }
```

```
XacmlPolicySyntax ::= SEQUENCE {
 policyPointer [0]   UTF8String OPTIONAL,
```

```
 policyData [1]   UTF8String OPTIONAL
 -- at least one of the optional elements must be present-- }
```

If **policyPointer** is present, it indicates the value of the **xacmlPolicyRDN** directory attribute that is used to form the final Relative Distinguished Name (RDN) of the entry that contains the actual policy information.

If **policyData** is present, it contains the XACML `<policyStatement>` or `<policySetStatement>`.

10.4.5.2 XACML Policy RDN

The **xacmlPolicyRDN** directory attribute is used to store the name of an **xacmlPolicyObject** entry relative to its position in the directory hierarchy.

```
xacmlPolicyRDN ATTRIBUTE  ::= {
 WITH SYNTAX   UTF8String
 EQUALITY MATCHING RULE    xacmlPolicyRDNMatch
 ID    id-???-at-xacmlPolicyRDN }
```

10.4.6 Matching Rule Definitions

The **xacmlPolicyRDNMatch** matching rule compares for equality a presented value with an attribute value of type **xacmlPolicyRDN**.

```
xacmlPolicyRDNMatch MATCHING-RULE  ::= {
 SYNTAX  UTF8String
 ID    id-???-at-policyNameMatch }
```

This rule returns TRUE if the presented value is equal to the stored value of the **xacmlPolicyRDN** directory attribute.

11 Operational Model (normative)

This section describes the operational model for an XACML-based environment.

11.1 Policy Decision Point (PDP)

Given a valid XACML "policy statement" or a "policy set statement", a compliant XACML PDP MUST evaluate that statement in accordance to the semantics specified in Sections 5, 6, and 7 when applied to a specific input context. The PDP MUST return an output context, with one value of "permit", "deny", or "indeterminate". The PDP MAY return decision of "indeterminate" with an error code of "insufficient information", signifying that more information is needed. In this case, the decision MAY list the names of any attributes of the subject and the resource that are needed by the PDP to refine its decision.

Decision Convergence

A PEP MAY resubmit a refined request context in response to a decision of "indeterminate" with an error code of "insufficient information" by adding attribute values for the attribute names that are listed in the response.

When the PDP returns an decision of "indeterminate" with an error code of "insufficient information", a PDP MUST NOT list the names of any attribute of the subject or the resource of the request for which values were already supplied in the request. Note, this requirement forces the PDP to eventually return a decision of "permit", "deny", or "indeterminate" with some other reason, in response to successively-refined requests.

12 XACML extensibility points (non-normative)

Describes the points within the XACML model and schema where extensions can be added

12.1 URIs

The following XML attributes are URIs.

- Function,
- ruleCombiningAlgId,
- policyCombiningAlgId,
- saml:AttributeNameSpace and
- saml:AttributeName.

13 Security and privacy (non-normative)

This section identifies possible security and privacy vulnerabilities that should be considered when implementing an XACML-based system. This section is strictly informative. It has been left to the implementers to assess whether these vulnerabilities apply to their environment and to select the appropriate safeguards.

13.1 Authentication

Authentication here means the ability of one party in a transaction to determine the identity of the other party in the transaction. Authentication may be in one direction, or it may be bilateral[1].

Given the sensitive nature of access-control systems, it is important for a ***PEP*** to authenticate the identity of the ***PDP*** to which it sends ***decision requests***. Otherwise, there is a risk that another process could provide false or invalid ***authorization decisions*** and compromise security of the access-control system.

It is equally important for a ***PDP*** to authenticate the identity of its clients and assess the level of trust to determine what, if any, sensitive data should be passed. One should keep in mind that even simple permit or deny responses could be exploited if someone was allowed to make unlimited requests to a ***PDP***.

Many different techniques may be used to provide this authentication, such as co-located code, a private network, a VPN, or digital signatures. Authentication may also be done as part of the communication protocol used to exchange the contexts. In this case, the authentication may be performed at the message level or at the session level.

13.2 Confidentiality

Confidentiality means that the contents of a message can be read only by the desired recipients and not by anyone else who encounters the message while it is in transit[2]. There are two areas in which confidentiality should be considered: one is confidentiality during transmission; the other is confidentiality within a <policyStatement>.

13.2.1 Communication Confidentiality

All data within an access-control system should be treated as confidential. This includes the <policyStatement>, the XACML requests and responses, and any external data that may be referenced as part of the decision-making process. If someone is able to eavesdrop on the communication they may be able to understand under what circumstances access will be granted, which may allow them to impersonate a valid request.

Any security concerns or requirements related to transmitting or exchanging XACML <policyStatement> elements are outside the scope of the XACML standard. While it is often important to ensure that the integrity and confidentiality of <policyStatement> elements is maintained when they are exchanged between two parties, it is left to the implementers to determine the appropriate mechanisms for their environment.

13.2.2 Statement Level Confidentiality

In some cases, an implementation may want to encrypt only parts of an XACML policy. For instance, a ***PRP*** only needs access to the target elements in order to find the appropriate rules. The other elements could be encrypted while they are stored in a repository.

The XML Encryption Syntax and Processing standard from W3C can be used to encrypt all or parts of an XML document. This standard is recommend for use with XACML.

It should go without saying that if a repository is used to facilitate the communication of cleartext (i.e., unencrypted) policy between the ***PAP*** and the ***PRP*** or between the ***PDP*** and the ***PIP***, then a secure repository should be used to store this sensitive data.

13.3 Policy Integrity

The XACML policy, used by the ***PDP*** to evaluate the request contexts, is the heart of the system. There are two aspects in maintaining the integrity of the policy. One is to ensure that <policyStatement> elements have not been altered since they were originally written or generated by the ***PAP***. The other is to ensure that <policyStatement> elements have not been inserted or deleted from the set of policies.

In the many cases, this can be achieved by ensuring the integrity of the systems and implementing session-level techniques to secure the communication between parties. The selection of the appropriate techniques has been left to the implementers.

However, when policy is distributed between organizations to be acted on at a later time, or when the policy travels with data, it would be useful to have a digital signature of the policy included with the policy statements. In these cases, the XML Signature Syntax and Processing standard from W3C is recommended to be used with this standard. See section 8.3 **[???]** for examples of using XML digital signatures with XACML.

Digital signatures SHOULD only be used to ensure the integrity of the statements. Digital signatures SHOULD NOT be use as a method of selecting or evaluating policy. The ***PDP*** SHOULD

NOT request a rule based on who signed the rule or whether or not it had been signed (as such a basis for selection would, itself, be a matter of policy).

13.4 Elements included by reference

There is a risk that references and extensions contained within a <policystatement> may have been altered since the policy was originally created, thereby changing the intent of the <policystatement>. For instance, if a <policyStatement> were to include a rule by reference, then there is no guarantee that the rule has not been changed between the time that the policy was written and the time that it is being evaluated.

A <ruleDigest> element can be used to uniquely identify a rule. The <ruleDigest> element contains a digest of the original rule. If the rule changed, then the rule digest would also change. Therefore, if the <policyStatement> is signed or integrity-protected in some other way (so that the <ruleDigest> cannot be altered without detection), the ***PDP*** can be certain that the referenced rules have not changed since the policy was created.

Alternatively, a digital signature of the source item could be included with the reference. **[I don't see this in the schema. Can we do this?]** This technique will also allow the ***PDP*** to ensure that a rule or extension has not been altered (although integrity protection is still required on the policy itself; otherwise, the included signatures may be removed or replaced).

13.5 Trust Model

Discussions of authentication, integrity, and confidentiality mechanisms necessarily assume an underlying trust model: how can one entity come to believe that a given key is uniquely associated with a specific, identified entity so that the key can be used to encrypt data for that entity or verify signatures (or other integrity structures) from that entity? Many different types of trust model exist, including strict hierarchies, distributed authorities, the Web, the bridge, and so on.

All considerations with respect to choosing and establishing a suitable trust model for a given environment are outside the scope of XACML. Suffice it to say, however, that a trust model MUST be in place in order for any of the security mechanisms described in this section to be applied. Secure access control is not possible in any environment until a trust model appropriate for that environment has been established and implemented.

13.6 Privacy

It is important to be aware that any transactions that occur with respect to access control may reveal private information about the participants. For example, if an XACML policy states that certain data may only be read by individuals with "Gold Card Member" status, then any transaction in which an entity is given access to that data leaks information to external observers about that entity's status. Privacy considerations may therefore lead to encryption or access control policies surrounding XACML policy instances themselves, confidentiality-protected channels for the request/response protocol messages, protection of user attributes in storage and in transit, and so on.

Selection and use of privacy mechanisms appropriate for a given environment are outside the scope of XACML. The decision regarding whether, how, and when to deploy such mechanisms is left to the implementers associated with the environment.

Footnotes

1 - Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) section 4.1

2066 2 - Security and Privacy Considerations for the OASIS Security Assertion Markup Language
2067 (SAML) section 4.2

2068 14 Conformance (normative)

2069 Conformance claims MAY be made by either one of two components in the XACML model:

2070 1. An implementation of a policy administration points that produces policy statements that
2071 conform with the XACML schema; and

2072 2. An implementation of a policy decision point that produces decisions in response to a request
2073 context on the basis of XACML policy statements that conform with the XACML schema.

2074 In the current version of the specification, implementations of a policy retrieval point that produce
2075 policy statements that conform with the XACML schema by combining XACML applicable policies
2076 are treated in the same way as policy administration points, from the point of view of conformance.

2077 Policy administration points MAY claim conformance with the XACML specification provided merely
2078 that they produce schema-compliant policy statements.

2079 Policy decision points MAY claim conformance with the XACML specification provided that they
2080 correctly execute the XACML conformance test suite provided at

2081 http://www.oasis-open.org/ …

2082 XACML Test Suite

2083 The test suite comprises three directories:

2084 • Request context

2085 • Policy

2086 • Response context

2087 The input context directory contains a set of text/xml/ `xacmlContext:RequestType` files that are
2088 valid XACML input contexts.

2089 The policy directory contains precisely one XACML policy file whose target is appropriate for each
2090 of the input contexts.

2091 The output context directory contains a set of text/xml/ `xacmlContext:ResponseType` files
2092 containing the output contexts that correspond to the input contexts in the input context directory.

2093 A conformant XACML ***PDP*** implementation shall create an output context in response to each and
2094 every input context. The output contexts are linked to the corresponding input contexts by the
2095 request context ID attribute. **[There's no such thing at the moment.]**

2096 XACML implementations that target a specific application domain (e.g., SAML or J2SE) may use a
2097 tool or process that is not an integral part of the XACML implementation to convert between the
2098 XACML contexts and its private data representation.

2099 Disclaimer: Implementors SHALL NOT consider the test cases provided in the XACML
2100 conformance test suite as providing 100% test coverage. OASIS does not represent that a
2101 conformant implementation will operate correctly in all respects nor that it is fit for its purpose.

15 References

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997

[RegEx]

[LDAP]

[SAML] Security Assertion Markup Language available from http://www.oasis-open.org/committees/security/#documents

[XMLSig] D. Eastlake et al., *XML-Signature Syntax and Processing*, http://www.w3.org/TR/xmldsig-core/, World Wide Web Consortium.

[XMLSig-XSD] XML Signature Schema available from http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/xmldsig-core-schema.xsd.

Appendix A. Acknowledgments

The following individuals were voting members of the committee during the development of this specification:

Affinitex James MacLean JMaclean@affinitex.com
Self Simon Godik sgodik@crosslogix.com
Crosslogix Ken Yagen kyagen@crosslogix.com
Crosslogix Daniel Engovatov dengovatov@crosslogix.com
Entegrity Hal Lockhart hal.lockhart@entegrity.com
Entrust Carlisle Adams carlisle.adams@entrust.com
Entrust Tim Moses tim.moses@entrust.com
Hitachi Don Flinn Don.Flinn@hitachisoftware.com
Hitachi Konstantin Beznosov konstantin.beznosov@quadrasis.com
IBM Michiharu Kudoh kudo@jp.ibm.com
Self Bill Parducci bill@parducci.net
Self Polar Humenn polar@syr.edu
Sterling Commerce Suresh Damodaran Suresh_Damodaran@stercomm.com
University of Milan Pierangela Samarati samarati@pinky.crema.unimi.it
University of Milan Ernesto Damiani edamiani@crema.unimi.it
Sun Microsystems Sekhar Vajjhala sekhar.vajjhala@sun.com
Sun Microsystems Anne Anderson Anne.Anderson@Sun.com
Xtradyne Gerald Brose Gerald.Brose@xtradyne.com

Appendix B. Revision History

Rev	Date	By whom	What
V14	14 June 2002	Tim Moses	Added the XACML context schema. Added the Security and Privacy section.
V15	18 July 2002	Tim Moses	Changed the representation of <Function>

Appendix C. Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.