OASIS eXtensible Access Control Markup Language (XACML)

Committee Specification 1.0, 6 November 2002

Document identifier: cs-xacml-specification-1.0.doc

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

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

Editors:

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

Contributors:

Anne Anderson, Sun Microsystems
Bill Parducci, Overxeer
Carlisle Adams, Entrust
Daniel Engovatov, CrossLogix
Don Flinn, Quadrasis
Ernesto Damiani, University of Milan
James MacLean, Affinitex
Hal Lockhart, Entegrity
Ken Yagen, CrossLogix
Konstantin Beznosov, Quadrasis
Michiharu Kudo, IBM
Pierangela Samarati, University of Milan
Pirasenna Velandai Thiyagarajan, Sun Microsystems
Polar Humenn, Syracuse University
Sekhar Vajjhala, Sun Microsystems
Seth Proctor, Sun Microsystems
Steve Anderson, OpenNetworks
Suresh Damodaran, Sterling Commerce
Gerald Brose, Xtradyne

Abstract:

This specification defines an XML schema for an extensible 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

1. Introduction (non-normative)

1.1. Glossary

1.1.1 Preferred terms

Access - Performing an ***action***

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

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

Applicable policy - The set of ***policies*** and ***policy sets*** that governs ***access*** for a specific ***decision request***

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

Authorization decision - The result of evaluating ***applicable policy,*** returned by the ***PDP*** to the ***PEP.*** A function that evaluates to "Permit", "Deny", "Indeterminate" or "Not-applicable", and (optionally) a set of ***obligations***

Bag – An unordered collection of values, in which there may be duplicate values

Condition - An expression of ***predicates.*** A function that evaluates to "True", "False" or "Indeterminate"

Conjunctive sequence - a sequence of elements combined using the logical 'AND' operation

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

Context handler - The system entity that converts ***decision requests*** in the native request format to the XACML canonical form and converts ***authorization decisions*** in the XACML canonical form to the native response format

Decision – The result of evaluating a ***rule, policy*** or ***policy set***

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

Disjunctive sequence - a sequence of elements combined using the logical 'OR' operation

Effect - The intended consequence of a satisfied ***rule*** (either "Permit" or "Deny")

Environment - The set of ***attributes*** that are relevant to an ***authorization decision*** and are independent of a particular ***subject, resource*** or ***action***

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

Policy - A set of ***rules,*** an identifier for the ***rule-combining algorithm*** and (optionally) a set of ***obligations.*** May be a component of a ***policy set***

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

Policy-combining algorithm - The procedure for combining the ***decision*** and ***obligations*** 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 making ***decision requests*** and enforcing ***authorization decisions***

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

Policy set - A set of ***policies,*** other ***policy sets,*** a ***policy-combining algorithm*** and (optionally) a set of ***obligations.*** May be a component of another ***policy set***

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

Resource - Data, service or system component

Rule - A ***target***, an ***effect*** and a ***condition.*** A component of a ***policy***

Rule-combining algorithm - The procedure for combining ***decisions*** 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

1.1.2 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***.

1.2. 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 [SAML].
- The prefix `ds:` stands for the W3C XML Signature namespace [DS].
- The prefix `xs:` stands for the W3C XML Schema namespace [XS].
- The prefix `xf:` stands for the XPath query and function specification namespace [XF].

This specification uses the following typographical conventions in text: `<XACMLElement>`, `<ns:ForeignElement>`, `Attribute`, **Datatype**, `OtherCode`. Terms in ***italic bold-face*** are intended to have the meaning defined in the Glossary.

1.3. Schema organization and namespaces

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

```
urn:oasis:names:tc:xacml:1.0:policy
```

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

```
urn:oasis:names:tc:xacml:1.0:context
```

XACML data-types are defined in the following XML namespace:

```
urn:oasis:names:tc:xacml:1.0:data-type
```

The XML Signature XMLSigXSD is imported into the XACML schema and is associated with the following XML namespace:

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

2. Background (non-normative)

The "economics of scale" have driven computing platform vendors to develop products with very generalized functionality, so that they can be used in the widest possible range of situations. "Out of the box", these products have the maximum possible privilege for accessing data and executing software, so that they can be used in as many application environments as possible, including those with the most permissive security policies. In the more common case of a relatively restrictive security policy, the platform's inherent privileges must be constrained, by configuration.

The security policy of a large enterprise has many elements and many points of enforcement. Elements of policy may be managed by the Information Systems department, by Human Resources, by the Legal department and by the Finance department. And the policy may be enforced by the extranet, mail, WAN and remote-access systems; platforms which inherently implement a permissive security policy. The current practice is to manage the configuration of each point of enforcement independently in order to implement the security policy as accurately as possible. Consequently, it is an expensive and unreliable proposition to modify the security policy. And, it is virtually impossible to obtain a consolidated view of the safeguards in effect throughout the enterprise to enforce the policy. At the same time, there is increasing pressure on corporate and government executives from consumers, shareholders and regulators to demonstrate "best practice" in the protection of the information assets of the enterprise and its customers.

For these reasons, there is a pressing need for a common language for expressing security policy. If implemented throughout an enterprise, a common policy language allows the enterprise to manage the enforcement of all the elements of its security policy in all the components of its information systems. Managing security policy may include some or all of the following steps: writing, reviewing, testing, approving, issuing, combining, analyzing, modifying, withdrawing, retrieving and enforcing policy.

XML is a natural choice as the basis for the common security-policy language, due to the ease with which its syntax and semantics can be extended to accommodate the unique requirements of this application, and the widespread support that it enjoys from all the main platform and tool vendors.

2.1. Requirements

The basic requirements of a policy language for expressing information system security policy are:

- To provide a method for combining individual ***rules*** and ***policies*** into a single ***policy set*** that applies to a given action.
- To provide a method for flexible definition of the procedure by which ***rules*** and ***policies*** are combined.
- To provide a method for dealingwith multiple ***subjects*** acting in different capacities.
- To provide a method for basing an ***authorization decision*** on ***attributes*** of the ***subject*** and ***resource***.
- To provide a method for dealing with multi-valued ***attributes***.
- To provide a method for basing an ***authorization decision*** on the contents of an information ***resource***.
- To provide a set of logical and mathematical operators on ***attributes*** of the ***subject***, ***resource*** and ***environment***.
- To provide a method for handling a distributed set of ***policy*** components, while abstracting the method for locating, retrieving and authenticating the ***policy*** components.
- To provide a method for rapidly identifying the ***policy*** that applies to a given action, based upon the values of ***attributes*** of the ***subjects, resource*** and ***action***.
- To provide an abstraction-layer that insulates the policy-writer from the details of the application environment.

- To provide a method for specifying a set of actions that must be performed in conjunction with policy enforcement.

The motivation behind XACML is to express these well-established ideas in the field of access-control policy using an extension language of XML. The XACML solutions for each of these requirements are discussed in the following sections.

2.2. Rule and policy combining

The complete ***policy*** applicable to a particular ***decision request*** may be composed of a number of individual ***rules*** or ***policies***. For instance, in a personal privacy application, the owner of the personal information may define certain aspects of disclosure ***policy***, whereas the enterprise that is the custodian of the information may define certain other aspects. In order to render an ***authorization decision***, it must be possible to combine the two separate ***policies*** to form the single ***policy*** applicable to the request.

XACML defines three top-level policy elements: `<Rule>`, `<Policy>` and `<PolicySet>`. The `<Rule>` element contains a boolean expression that can be evaluated in isolation, but that is not intended to be accessed in isolation by a ***PDP***. So, it is not intended to form the basis of an ***authorization decision*** by itself. It is intended to exist in isolation only within an XACML ***PAP***, where it may form the basic unit of management, and be re-used in multiple ***policies***.

The `<Policy>` element contains a set of `<Rule>` elements and a specified procedure for combining the results of their evaluation. It is the basic unit of ***policy*** used by the ***PDP***, and so it is intended to form the basis of an ***authorization decision***.

The `<PolicySet>` element contains a set of `<Policy>` or other `<PolicySet>` elements and a specified procedure for combining the results of their evaluation. It is the standard means for combining separate ***policies*** into a single combined ***policy***.

Hinton et al [Hinton94] discuss the question of the compatibility of separate ***policies*** applicable to the same ***decision request***.

2.3. Combining algorithms

XACML defines a number of combining algorithms that can be identified by a `RuleCombiningAlgId` or `PolicyCombiningAlgId` attribute of the `<Policy>` or `<PolicySet>` elements, respectively. The ***rule-combining algorithm*** defines a procedure for arriving at an ***authorization decision*** given the individual results of evaluation of a set of ***rules***. Similarly, the ***policy-combining algorithm*** defines a procedure for arriving at an ***authorization decision*** given the individual results of evaluation of a set of ***policies***. Standard combining algorithms are defined for:

- Deny-overrides,
- Permit-overrides,
- First applicable and
- Only-one-applicable.

In the first case, if a single `<Rule>` or `<Policy>` element is encountered that evaluates to "Deny", then, regardless of the evaluation result of the other `<Rule>` or `<Policy>` elements in the ***applicable policy***, the combined result is "Deny". Likewise, in the second case, if a single "Permit" result is encountered, then the combined result is "Permit". In the case of the “First-applicable” combining algorithm, the combined result is the same as the result of evaluating the first `<Rule>`***,*** `<Policy>` or `<PolicySet>` element in the list of ***rules*** whose ***target*** is applicable to the ***decision request***. The "Only-one-applicable" ***policy-combining algorithm*** only applies to ***policies***. The result of this combining algorithm ensures that one and only one ***policy*** or ***policy set*** is applicable by virtue of their ***targets***. If no ***policy*** or ***policy set*** applies, then the result is "Not-applicable", but if more than one ***policy*** or ***policy set*** is applicable, then the result is "Indeterminate". When exactly one ***policy*** or ***policy set*** is applicable, the result of the combining algorithm is the result of evaluating the single ***applicable policy*** or ***policy set***.

Users of this specification may, if necessary, define their own combining algorithms.

2.4. Multiple subjects

Access-control policies often place requirements on the actions of more than one ***subject***. For instance, the policy governing the execution of a high-value financial transaction may require the approval of more than one individual, acting in different capacities. Therefore, XACML recognizes that there may be more than one ***subject*** relevant to a ***decision request***. An ***attribute*** called “subject-category” is used to differentiate between ***subjects*** acting in different capacities. Some standard values for this ***attribute*** are specified, and users may define additional ones.

2.5. Policies based on subject and resource attributes

Another common requirement is to base an ***authorization decision*** on some characteristic of the ***subject*** other than its identity. Perhaps, the most common application of this idea is the ***subject's*** role [RBAC]. XACML provides facilities to support this approach. ***Attributes*** of ***subjects*** may be identified by the `<SubjectAttributeDesignator>` element. This element contains a URN that identifies the ***attribute***. Alternatively, the `<AttributeSelector>` element may contain an XPath expression over the request ***context*** to identify a particular ***subject attribute*** value by its location in the ***context*** (see section 2.11 for an explanation of ***context***). XACML provides a standard way to reference the ***attributes*** defined in the LDAP series of specifications [LDAP-1, LDAP-2]. This is intended to encourage implementers to use standard ***attribute*** identifiers for some common ***subject attributes***.

Another common requirement is to base an ***authorization decision*** on some characteristic of the ***resource*** other than its identity. XACML provides facilities to support this approach. ***Attributes*** of ***resource*** may be identified by the `<ResourceAttributeDesignator>` element. This element contains a URN that identifies the ***attribute***. Alternatively, the `<AttributeSelector>` element may contain an XPath expression over the request ***context*** to identify a particular ***resource attribute*** value by its location in the ***context.***

2.6. Multi-valued attributes

The most common techniques for communicating ***attributes*** (LDAP, XPath, SAML, etc.) support multiple values per ***attribute***. Therefore, when an XACML ***PDP*** retrieves the value of a named ***attribute***, the result may contain multiple values. A collection of such values is called a ***bag***. A ***bag*** differs from a set in that it may contain duplicate values, whereas a set may not. Sometimes this situation represents an error. Sometimes the XACML ***rule*** is satisfied if any one of the ***attribute*** values meets the criteria expressed in the ***rule***.

XACML provides a set of functions that allow a policy writer to be absolutely clear about how the ***PDP*** should handle the case of multiple ***attribute*** values. These are the “higher-order” functions.

2.7. Policies based on resource contents

In many applications, it is required to base an ***authorization decision*** on data *contained in* the information ***resource*** to which ***access*** is requested. For instance, a common component of privacy ***policy*** is that a person should be allowed to read records for which he or she is the subject. The corresponding ***policy*** must contain a reference to the ***subject*** identified in the information ***resource*** itself.

XACML provides facilities for doing this when the information ***resource*** can be represented as an XML document. The `<AttributeSelector>` element may contain an XPath expression over the request ***context*** to identify data in the information ***resource*** to be used in the ***policy*** evaluation.

In cases where the information ***resource*** is not an XML document, specified ***attributes*** of the ***resource*** can be referenced, as described in Section 2.4.

2.8. Operators

Information security ***policies*** operate upon ***attributes*** of ***subjects***, the ***resource*** and the ***action*** to be performed on the ***resource*** in order to arrive at an ***authorization decision***. In the process of arriving at the ***authorization decision***, ***attributes*** of many different types may have to be compared or computed. For instance, in a financial application, a person's available credit may have to be calculated by adding their credit limit to their account balance. The result may then have to be compared with the transaction value. This sort of situation gives rise to the need for arithmetic operations on ***attributes*** of the ***subject*** (account balance and credit limit) and the ***resource*** (transaction value).

Even more commonly, a ***policy*** may identify the set of roles that are permitted to perform a particular action. The corresponding operation involves checking whether there is a non-empty intersection between the set of roles occupied by the ***subject*** and the set of roles identified in the ***policy***. Hence the need for set operations.

XACML includes a number of built-in functions and a method of adding non-standard functions. These functions may be nested to build arbitrarily complex expressions. This is achieved with the `<Apply>` element. The `<Apply>` element has an XML attribute called `FunctionId` that identifies the function to be applied to the contents of the element. Each standard function is defined for specific argument type combinations, and its return type is also specified. Therefore, type consistency of the ***policy*** can be checked at the time the ***policy*** is written or parsed. And, the types of the data values presented in the request ***context*** can be checked against the values expected by the ***policy*** to ensure a predictable outcome.

In addition to operators on numerical and set arguments, operators are defined for date, time and duration arguments.

Relationship operators (equality and comparison) are also defined for a number of data-types, including the RFC822 and X.500 name-forms, strings, URIs, etc..

Also noteworthy are the operators over boolean data types, which permit the logical combination of ***predicates*** in a ***rule***. For example, a ***rule*** may contain the statement that ***access*** may be permitted during business hours AND from a terminal on business premises.

The XACML method of representing functions borrows from MathML [MathML] and from XPath Query and Functions [XF].

2.9. Policy distribution

In a distributed system, individual ***policy*** statements may be written by several policy writers and enforced at several enforcement points. In addition to facilitating the collection and combination of independent ***policy*** components, this approach allows ***policies*** to be updated as required. XACML ***policy*** statements may be distributed in any one of a number of ways. But, XACML does not describe any normative way to do this. Regardless of the means of distribution, ***PDPs*** are expected to confirm, by examining the ***policy's*** `<Target>` element that the policy is applicable to the ***decision request*** that it is processing.

`<Policy>` elements may be attached to the information ***resources*** to which they apply, as described by Perritt [Perritt93]. Alternatively, `<Policy>` elements may be maintained in one or more locations from which they are retrieved for evaluation. In such cases, the ***applicable policy*** may be referenced by an identifier or locator closely associated with the information ***resource***.

2.10. Policy indexing

For efficiency of evaluation and ease of management, the overall security policy in force across an enterprise may be expressed as multiple independent ***policy*** components. In this case, it is necessary to identify and retrieve the ***applicable policy*** statement and verify that it is the correct one for the requested action before evaluating it. This is the purpose of the `<Target>` element in XACML.

Two approaches are supported:

1. ***Policy*** statements may be stored in a database, whose data-model is congruent with that of the `<Target>` element. The ***PDP*** should use the contents of the ***decision request*** that it is processing to form the database read command by which applicable ***policy*** statements are retrieved. Nevertheless, the ***PDP*** should still evaluate the `<Target>` element of the retrieved ***policy*** or ***policy set*** statements as defined by the XACML specification.

2. Alternatively, the ***PDP*** may evaluate the `<Target>` element from each of the ***policies*** or pol***icy sets*** that it has available to it, in the context of a particular ***decision request***, in order to identify the ***policies*** and ***policy sets*** that are applicable to that request.

The use of constraints limiting the applicability of a ***policy*** were described by Sloman [Sloman94].

2.11. Abstraction layer

PEPs come in many forms. For instance, a ***PEP*** may be part of a remote-access gateway, part of a Web server or part of an email user-agent, etc.. It is unrealistic to expect that all ***PEPs*** in an enterprise do currently, or will in the future, issue ***decision requests*** to a ***PDP*** in a common format. Nevertheless, a particular ***policy*** may have to be enforced by multiple ***PEPs***. It would be inefficient to force a policy writer to write the same ***policy*** several different ways in order to accommodate the format requirements of each ***PEP***. Similarly attributes may be contained in various envelope types (e.g. X.509 attribute certificates, SAML attribute assertions, etc.). Therefore, there is a need for a canonical form of the request and response handled by an XACML ***PDP***. This canonical form is called the XACML "***Context***". Its syntax is defined in XML schema.

Naturally, XACML-conformant ***PEPs*** may issue requests and receive responses in the form of an XACML ***context***. But, where this situation does not exist, an intermediate step is required to convert between the request/response format understood by the ***PEP*** and the XACML ***context*** format understood by the ***PDP***.

The benefit of this approach is that ***policies*** may be written and analyzed independent of the specific environment in which they are to be enforced.

In the case where the native request/response format is specified in XML Schema (e.g. a SAML-conformant ***PEP***), the transformation between the native format and the XACML ***context*** may be specified in the form of an Extensible Stylesheet Language Transformation [XSLT].

Similarly, in the case where the ***resource*** to which ***access*** is requested is an XML document, the ***resource*** itself may be included in, or referenced by, the request ***context***. Then, through the use of XPath expressions [XPath] in the ***policy***, values in the ***resource*** may be included in the ***policy*** evaluation.

2.12. Actions performed in conjunction with enforcement

In many applications, policies specify actions that MUST be performed, either instead of, or in addition to, actions that MAY be performed. This idea was described by Sloman [Sloman94]. XACML provides facilities to specify actions that MUST be performed in conjunction with policy evaluation in the <Obligations> element. This idea was described as a provisional action by Kudo [Kudo00]. There are no standard definitions for these actions in version 1.0 of XACML. Therefore, bilateral agreement between a ***PAP*** and the ***PEP*** that will enforce its ***policies*** is required for correct interpretation. ***PEPs*** that conform with v1.0 of XACML are required to deny ***access*** unless they understand all the `<Obligations>` elements associated with the ***applicable policy***. `<Obligations>` elements are returned to the ***PEP*** for enforcement.

3. Models (non-normative)

The data-flow model and language model of XACML are described in the following sub-sections.

3.1. Data-flow model

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

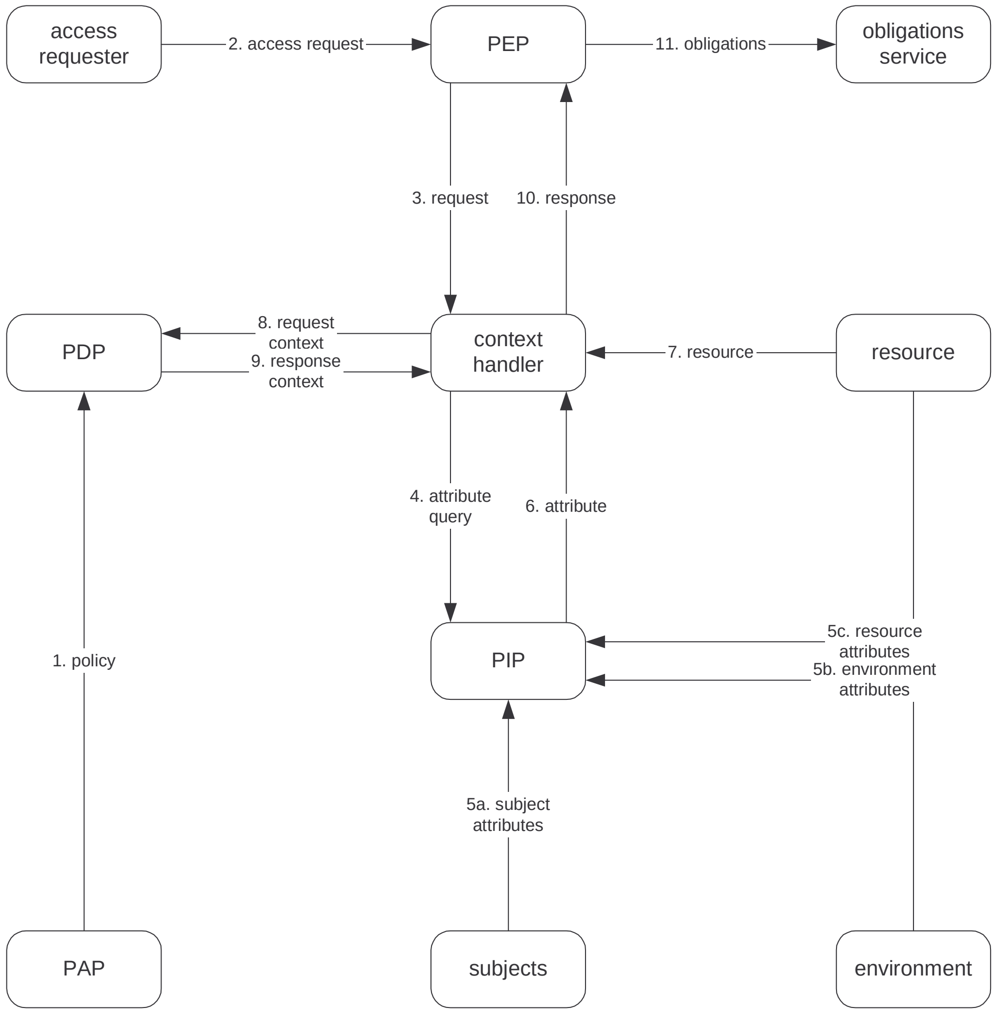

Figure 1 - Data-flow diagram

Note: some of the data-flows shown in the diagram may be facilitated by a repository. For instance, the communications between the ***context*** handler and the ***PIP*** or the communications between the ***PDP*** and the ***PAP*** 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 ***PDP***. Its ***policies*** represent the complete policy for a specified ***target***.
2. The access requester sends a request for access to the ***PEP***.
3. The ***PEP*** sends the request for ***access*** to the ***context handler*** in its native request format, optionally including ***attributes*** of the ***subjects***, ***resource*** and ***action***. The ***context handler*** constructs an XACML request ***context*** in accordance with steps 4,5,6 and 7.
4. ***Subject***, ***resource*** and ***environment attributes*** may be requested from a ***PIP***.
5. The ***PIP*** obtains the requested ***attributes***.

6. The ***PIP*** returns the requested ***attributes*** to the ***context handler***.
7. Optionally, the ***context handler*** includes the ***resource*** in the ***context***.
8. The ***context handler*** makes information about the request ***context*** available to the ***PDP***. The ***PDP*** identifies the ***policy*** applicable to the request ***context***. The ***PDP*** evaluates the ***policy***.
9. The ***PDP*** returns the response ***context*** (including the ***authorization decision***) to the ***context handler***.
10. The ***context handler*** translates the response ***context*** to the native response format of the ***PEP***. The ***context handler*** returns the response to the ***PEP***.
11. The ***PEP*** fulfills the ***obligations***.
12. (Not shown) If ***access*** is permitted, then the ***PEP*** permits ***access*** to the ***resource;*** otherwise, it denies ***access***.

3.2. XACML context

XACML is intended to be suitable for a variety of application environments. The core language is insulated from the application environment by the XACML ***context***, as shown in Figure 2, in which the scope of the XACML specification is indicated by the shaded area. The XACML ***context*** is defined in XML schema, describing a canonical representation for the inputs and outputs of the ***PDP***. ***Attributes*** referenced by an instance of XACML policy may be in the form of XPath expressions on the ***context***, or attribute designators that identify the ***attribute*** by ***subject, resource, action*** or ***environment*** and its identifier. 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.

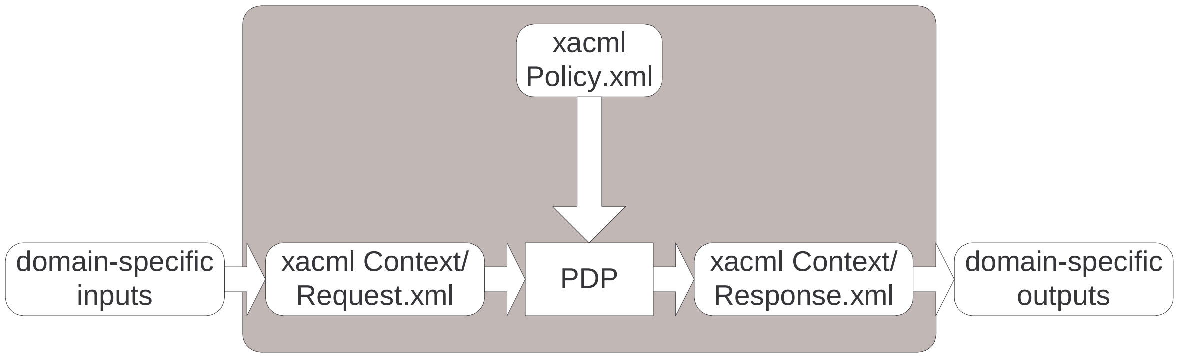

Figure 2 - XACML context

Note: The ***PDP*** may be implemented such that it uses a processed form of the XML files.

See Section 7.9 for a more detailed discussion of the request ***context***.

3.3. Policy language model

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

- ***Rule***;
- ***Policy***; and

- ***Policy set***.

These are described in the following sub-sections.

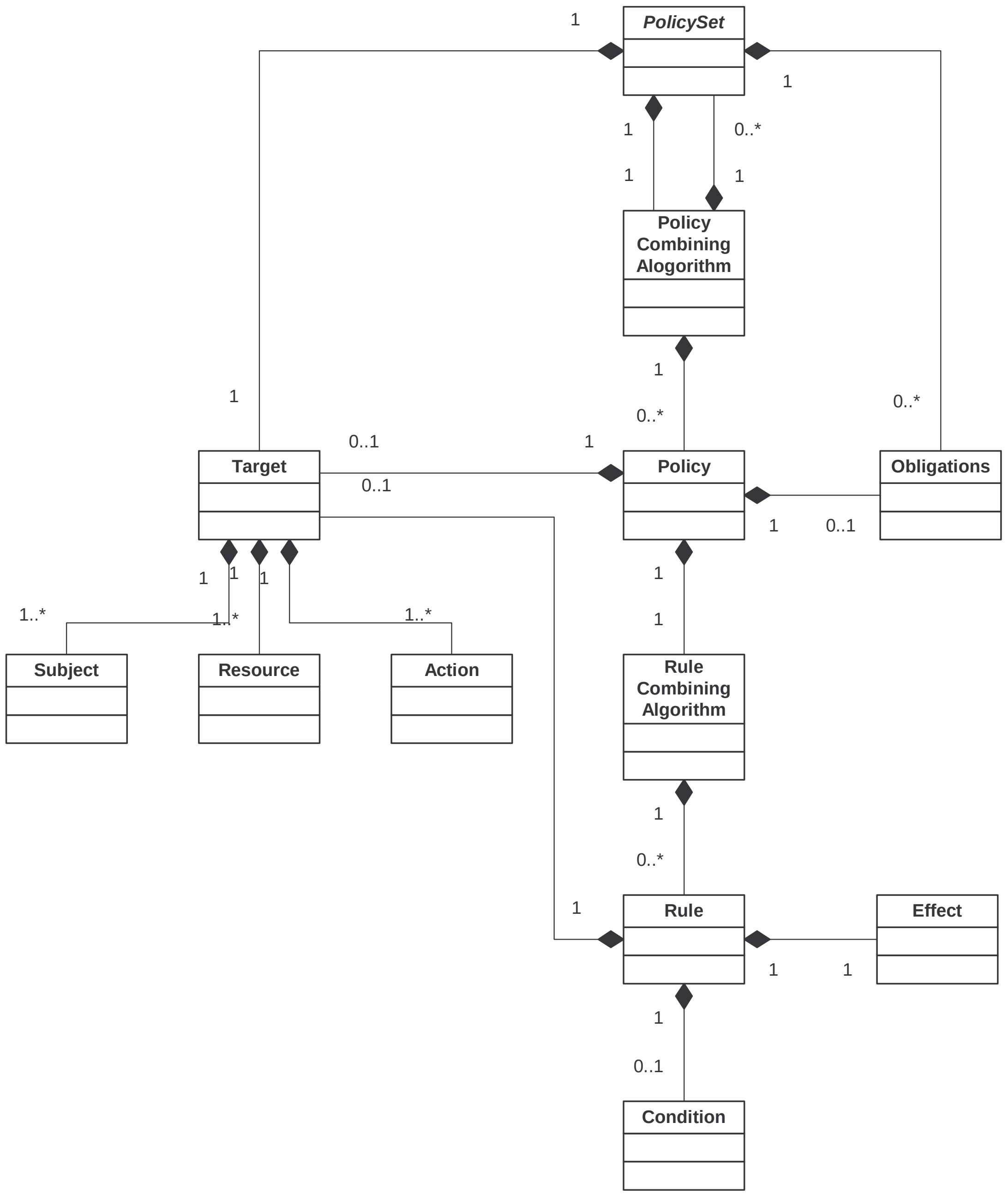

Figure 3 - Policy language model

3.3.1 Rule

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

- a ***target***;

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

These are discussed in the following sub-sections.

3.3.1.1. Rule target

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

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

to which the ***rule*** is intended to apply. The `<Condition>` element may further refine the applicability established by the ***target***. If the ***rule*** is intended to apply to all entities of a particular type, then an empty element named `<AnySubject/>`, `<AnyResource/>` or `<AnyAction/>` is used. An XACML ***PDP*** verifies that the ***subjects, resource*** and ***action*** identified in the request ***context*** are all present in the ***target*** of the ***rules*** that it uses to evaluate the ***decision request***. ***Target*** definitions are discrete, in order that applicable ***rules*** may be efficiently identified by the ***PDP***.

The `<Target>` element may be absent from a `<Rule>`. In this case, the `<Rule>` inherits its ***target*** from the parent `<Policy>` element.

Certain ***subject*** name-forms, ***resource*** name-forms and certain types of ***resource*** are internally structured. For instance, the X.500 directory name-form and RFC 822 name-form are structured ***subject*** name-forms, whereas an account number commonly has no discernible structure. UNIX file-system path-names and URIs are examples of structured ***resource*** name-forms. And an XML document is an example of a structured ***resource***.

Generally, the name of a node (other than a leaf node) in a structured name-form is also a legal instance of the name-form. So, for instance, the RFC822 name "medico.com" is a legal RFC822 name identifying the set of mail addresses hosted by the medico.com mail server. And the XPath/XPointer value "//ctx:ResourceContent/md:record/md:patient/ is a legal XPath/XPointer value identifying a node-set in an XML document.

The question arises: how should a name that identifies a set of ***subjects*** or ***resources*** be interpreted by the ***PDP***, whether it appears in a ***policy*** or a request ***context***? Are they intended to represent just the node explicitly identified by the name, or are they intended to represent the entire sub-tree subordinate to that node?

In the case of ***subjects***, there is no real entity that corresponds to such a node. So, names of this type always refer to the set of ***subjects*** subordinate in the name structure to the identified node. Consequently, non-leaf ***subject*** names should not be used in equality functions, only in match functions, such as “urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match” not “urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal” (see Appendix A).

On the other hand, in the case of ***resource*** names and ***resources*** themselves, three options exist. The name could refer to:

1. the contents of the identified node only,

2. the contents of the identified node and the contents of its immediate child nodes or

3. the contents of the identified node and all its descendant nodes.

All three options are supported in XACML.

3.3.1.2. Effect

The ***effect*** of the ***rule*** indicates the rule-writer's intended consequence of a "True" evaluation for the ***rule***. Two values are allowed: "Permit" and "Deny".

3.3.1.3. Condition

Condition represents a boolean expression that refines the applicability of the ***rule*** beyond the ***predicates*** implied by its ***target***. Therefore, it may be absent.

3.3.2 Policy

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***.

Rules are described above. The remaining components are described in the following sub-sections.

3.3.2.1. Policy target

An XACML `<PolicySet>`, `<Policy>` or `<Rule>` element contains a `<Target>` element that specifies the set of ***subjects***, ***resources*** and ***actions*** to which it applies. The `<Target>` of a `<PolicySet>` or `<Policy>` may be declared by the writer of the `<PolicySet>` or `<Policy>`, or it may be calculated from the `<Target>` elements of the `<PolicySet>`, `<Policy>` and `<Rule>` elements that it contains.

A system entity that calculates a `<Target>` in this way is not defined by XACML, but there are two logical methods that might be used. In one method, the `<Target>` element of the outer `<PolicySet>` or `<Policy>` (the "outer component") is calculated as the *union* of all the `<Target>` elements of the referenced `<PolicySet>`, `<Policy>` or `<Rule>` elements (the "inner components"). In another method, the `<Target>` element of the outer component is calculated as the *intersection* of all the `<Target>` elements of the inner components. The results of evaluation in each case will be very different: in the first case, the `<Target>` element of the outer component makes it applicable to any ***decision request*** that matches the `<Target>` element of at least one inner component; in the second case, the `<Target>` element of the outer component makes it applicable only to ***decision requests*** that match the `<Target>` elements of every inner component. Note that computing the intersection of a set of `<Target>` elements is likely only practical if the target data-model is relatively simple.

In cases where the `<Target>` of a `<Policy>` is *declared* by the ***policy*** writer, any component `<Rule>` elements in the `<Policy>` that have the same `<Target>` element as the `<Policy>` element may omit the `<Target>` element. Such `<Rule>` elements inherit the `<Target>` of the `<Policy>` in which they are contained.

3.3.2.2. Rule-combining algorithm

The ***rule-combining algorithm*** specifies the procedure by which the results of evaluating the component ***rules*** are combined when evaluating the ***policy***, i.e. the `Decision` value placed in the response ***context*** by the ***PDP*** is the value of the ***policy***, as defined by the ***rule-combining algorithm***.

See Appendix C for definitions of the normative ***rule-combining algorithms***.

3.3.2.3. 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 the response ***context***. Section 7.11 explains which ***obligations*** are to be returned.

3.3.3 Policy set

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

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

The ***target*** and ***policy*** components are described above. The other components are described in the following sub-sections.

3.3.3.1. Policy-combining algorithm

The ***policy-combining algorithm*** specifies the procedure by which the results of evaluating the component ***policies*** are combined when evaluating the ***policy set***, i.e.the `Decision` value placed in the response ***context*** by the ***PDP*** is the result of evaluating the ***policy set***, as defined by the ***policy-combining algorithm***.

See Appendix C for definitions of the normative ***policy-combining algorithms***.

3.3.3.2. Obligations

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

When a ***PDP*** evaluates a ***policy set*** containing ***obligations***, it returns certain of those ***obligations*** to the ***PEP*** in its response context. Section 7.11 explains which ***obligations*** are to be returned.

4. Examples (non-normative)

This section contains two examples of the use of XACML for illustrative purposes. The first example is a relatively simple one to illustrate the use of ***target***, ***context***, matching functions and ***subject***

attributes. The second example additionally illustrates the use of the ***rule-combining algorithm***, ***conditions*** and ***obligations***.

4.1. Example one

4.1.1 Example policy

Assume that a corporation named Medi Corp (medico.com) has an ***access control policy*** that states, in English:

> Any user with an e-mail name in the "medico.com" namespace is allowed to perform any action on any ***resource***.

An XACML ***policy*** consists of header information, an optional text description of the policy, a **target**, one or more **rules** and an optional set of **obligations**.

The header for this policy is

```
[p01]    <?xml version=1.0" encoding="UTF-8"?>
[p02]    <Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
[p03]    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[p04]    xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
[p05]    http://www.oasis-open.org/tc/xacml/1.0/cs-xacml-schema-policy-01.xsd"
[p06]    PolicyId="identifier:example:SimplePolicy1"
[p07]    RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">
```

Line [p01] is a standard XML document tag indicating which version of XML is being used and what the character encoding is.

Line [p02] introduces the XACML Policy itself.

Lines [p03-p05] are XML namespace declarations.

Line [p05] gives a URL to the schema for XACML ***policies***.

Line [p06] assigns a name to this ***policy*** instance. The name of a ***policy*** should be unique for a given ***PDP*** so that there is no ambiguity if one ***policy*** is referenced from another ***policy***.

Line [p07] specifies the algorithm that will be used to resolve the results of the various ***rules*** that may be in the ***policy***. The *deny-overrides* ***rule-combining algorithm*** specified here says that, if any ***rule*** evaluates to “Deny”, then that ***policy*** must return “Deny”. If all ***rules*** evaluate to “Permit”, then the ***policy*** must return “Permit”. The ***rule-combining algorithm***, which is fully described in Appendix C, also says what to do if an error were to occur when evaluating any ***rule***, and what to do with ***rules*** that do not apply to a particular ***decision request***.

```
[p08]    <Description>
[p09]      Medi Corp access control policy
[p10]    </Description>
```

Lines [p08-p10] provide a text description of the policy. This description is optional.

```
[p11]    <Target>
[p12]      <Subjects>
[p13]        <AnySubject/>
[p14]      </Subjects>
[p15]      <Resources>
[p16]        <AnyResource/>
[p17]      </Resources>
[p18]      <Actions>
[p19]        <AnyAction/>
```

```
[p20]     </Actions>
[p21]   </Target>
```

Lines [p11-p21] describe the ***decision requests*** to which this ***policy*** applies. If the ***subject***, ***resource*** and ***action*** in a ***decision request*** do not match the values specified in the ***target***, then the remainder of the ***policy*** does not need to be evaluated. This ***target*** section is very useful for creating an index to a set of ***policies***. In this simple example, the ***target*** section says the ***policy*** is applicable to any ***decision request***.

```
[p22]   <Rule
[p23]     RuleId= "urn:oasis:names:tc:xacml:1.0:example:SimpleRule1"
[p24]     Effect="Permit">
```

Line [p22] introduces the one and only ***rule*** in this simple ***policy***. Just as for a ***policy***, each ***rule*** must have a unique identifier (at least unique for any ***PDP*** that will be using the ***policy***).

Line [p23] specifies the identifier for this ***rule***.

Line [p24] says what ***effect*** this ***rule*** has if the ***rule*** evaluates to “True”. ***Rules*** can have an ***effect*** of either “Permit” or “Deny”. In this case, the rule will evaluate to “Permit”, meaning that, as far as this one ***rule*** is concerned, the requested ***access*** should be permitted. If a ***rule*** evaluates to “False”, then it returns a result of “Not-applicable”. If an error occurs when evaluating the ***rule***, the ***rule*** returns a result of “Indeterminate”. As mentioned above, the ***rule-combining algorithm*** for the ***policy*** tells how various ***rule*** values are combined into a single ***policy*** value.

```
[p25]     <Description>
[p26]       Any subject with an e-mail name in the medico.com domain
[p27]       can perform any action on any resource.
[p28]     </Description>
```

Lines [p25-p28] provide a text description of this ***rule***. This description is optional.

```
[p29]     <Target>
```

Line [p29] introduces the ***target*** of the ***rule***. As described above for the ***target*** of a policy, the ***target*** of a ***rule*** describes the ***decision requests*** to which this ***rule*** applies. If the ***subject***, ***resource*** and ***action*** in a ***decision request*** do not match the values specified in the ***rule target***, then the remainder of the ***rule*** does not need to be evaluated, and a value of “Not-applicable” is returned to the ***policy*** evaluation.

```
[p30]       <Subjects>
[p31]         <Subject>
[p32]           <SubjectMatch MatchId=" urn:oasis:names:tc:xacml:1.0:function:rfc822Name-
            match">
[p33]             <SubjectAttributeDesignator
[p34]               AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
[p35]               DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
[p36]             <AttributeValue
[p37]              DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">medico.com
[p38]             </AttributeValue>
[p39]           </SubjectMatch>
[p40]         </Subject>
[p41]       </Subjects>
[p42]       <Resources>
[p43]         <AnyResource/>
[p44]       </Resources>
[p45]       <Actions>
[p46]         <AnyAction/>
[p47]       </Actions>
[p48]     </Target>
```

The ***rule target*** is similar to the ***target*** of the ***policy*** itself, but with one important difference. Lines [p32-p41] do not say `<AnySubject/>`, but instead spell out a specific value that the ***subject*** in the ***decision request*** must match. The `<SubjectMatch>` element specifies a matching function in the `MatchId` attribute, a pointer to a specific ***subject attribute*** in the request ***context*** by means of the `<SubjectAttributeDesignator>` element, and a literal value of "medico.com". The matching function will be used to compare the value of the ***subject attribute*** with the literal value. Only if the match returns "True" will this ***rule*** apply to a particular ***decision request***. If the match returns "False", then this ***rule*** will return a value of "Not-applicable".

```
[p49]    </Rule>
[p50]   </xacml:Policy>
```

Line [p49] closes the ***rule*** we have been examining. In this ***rule***, all the *work* is done in the `<Target>` element. In more complex ***rules***, the `<Target>` may have been followed by a `<Condition>` (which could also be a set of ***conditions*** to be *AND*ed or *OR*ed together).

Line [p50] closes the ***policy*** we have been examining. As mentioned above, this ***policy*** has only one ***rule***, but more complex ***policies*** may have any number of ***rules***.

4.1.2 Example request context

Let's examine a hypothetical ***decision request*** that might be submitted to a ***PDP*** using the ***policy*** above. In English, the ***access*** request that generates the ***decision request*** may be stated as follows:

Bart Simpson, with e-mail name "bs@simpsons.com", wants to read his medical record at Medi Corp.

In XACML, the information in the ***decision request*** is formatted into a ***request context*** statement that looks as follows.:

```
[c01]   <?xml version="1.0" encoding="UTF-8"?>
[c02]   <Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
[c03]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[c04]   xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:context
[c05]   http://www.oasis-open.org/tc/xacml/1.0/sc-xacml-schema-context-01.xsd">
```

Lines [c01-c05] are the header for the ***request context***, and are used the same way as the header for the ***policy*** explained above.

```
[c06]    <Subject>
[c07]     <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
[c08]       DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
[c09]      <AttributeValue>bs@simpsons.com</AttributeValue>
[c10]     </Attribute>
[c11]    </Subject>
```

The `<Subject>` element contains one or more ***attributes*** of the entity making the ***access*** request. There can be multiple ***subjects***, and each ***subject*** can have multiple ***attributes***. In this case, in lines [c06-c11], there is only one ***subject***, and the ***subject*** has only one ***attribute***: the ***subject's*** identity, expressed as an e-mail name, is "bs@simpsons.com".

```
[c12]    <Resource>
[c13]     <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:ufs-path"
[c14]       DataType="http://www.w3.org/2001/XMLSchema#anyURI">
[c15]      <AttributeValue>/medico/record/patient/BartSimpson</AttributeValue>
[c16]     </Attribute>
[c17]    </Resource>
```

The `<Resource>` element contains one or more ***attributes*** of the ***resource*** to which the ***subject*** (or ***subjects***) has requested ***access***. There can be only one `<Resource>` per ***decision request***. Lines [c13-c16] contain the one ***attribute*** of the ***resource*** to which Bart Simpson has requested ***access***: the ***resource*** unix file-system path-name, which is "/medico/record/patient/BartSimpson".

```
[c18]    <Action>
[c19]     <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
[c20]       DataType="http://www.w3.org/2001/XMLSchema#string">
[c21]      <AttributeValue>read</AttributeValue>
[c22]     </Attribute>
[c23]    </Action>
```

The `<Action>` element contains one or more ***attributes*** of the ***action*** that the ***subject*** (or ***subjects***) wishes to take on the ***resource***. There can be only one ***action*** per ***decision request***. Lines [c18-c23] describe the identity of the ***action*** Bart Simpson wishes to take, which is "read".

```
[c24]   </Request>
```

Line [c24] closes the ***request context***. A more complex ***request context*** may have contained some ***attributes*** not associated with either the ***subject***, the ***resource*** or the ***action***. These would have been placed in an optional `<Environment>` element following the `<Action>` element.

The ***PDP*** processing this request ***context*** locates the ***policy*** in its policy repository. It compares the ***subject***, ***resource*** and ***action*** in the request ***context*** with the ***subjects***, ***resources*** and ***actions*** in the ***policy target***. Since the ***policy target*** matches the `<AnySubject/>`, `<AnyResource/>` and `<AnyAction/> elements`, the ***policy*** matches this ***context***.

The ***PDP*** now compares the ***subject***, ***resource*** and ***action*** in the request ***context*** with the ***target*** of the one ***rule*** in this ***policy***. The requested ***resource*** matches the `<AnyResource/>` element and the requested ***action*** matches the `<AnyAction/>` element, but the requesting subject-id ***attribute*** does not match "*@medico.com".

4.1.3 Example response context

As a result, there is no ***rule*** in this ***policy*** that returns a "Permit" result for this request. The ***rule-combining algorithm*** for the ***policy*** specifies that, in this case, a result of "Not-applicable" should be returned. The response ***context*** looks as follows:

```
[r01]        <?xml version="1.0" encoding="UTF-8"?>
[r02]        <Response xmlns="urn:oasis:names:tc:xacml:1.0:context"
[r03]        xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:context
[r04]        http://www.oasis-open.org/tc/xacml/1.0/sc-xacml-schema-context-01.xsd">
```

Lines [r01-r04] contain the same sort of header information for the response as was described above for a ***policy***.

```
[r05]         <Result>
[r06]          <Decision>Not-applicable</Decision>
[r07]         </Result>
```

The `<Result>` element in lines [r05-r07] contains the result of evaluating the ***decision request*** against the ***policy***. In this case, the result is "Not-applicable". A ***policy*** can return "Permit", "Deny", "Not-applicable" or "Indeterminate".

```
[r08]        </Response>
```

Line [r08] closes the response ***context***.

4.2. Example two

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. These illustrate a ***rule-combining algorithm***, ***conditions*** and ***obligations***.

4.2.1 Example medical record instance

The 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="http://www.medico.com/schemas/record.xsd "
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance>
   <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 http://www.w3.org/2001/XMLSchema#type="date">1992-03-
21</patientDoB>
      <patientGender
http://www.w3.org/2001/XMLSchema#type="string">male</patientGender>
      <policyNumber
http://www.w3.org/2001/XMLSchema#type="string">555555</policyNumber>
   </patient>
   <parentGuardian>
      <parentGuardianId>HS001</parentGuardianId>
      <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>
```

4.2.2 Example request context

The following example illustrates a request ***context*** to which the example ***rules*** may be applicable. It represents a request by the physician Julius Hibbert to read the patient date of birth in the record of Bartholomew Simpson.

```
[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
[03]  xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy"
[04]  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[05]  <Subject>
[06]     <Attribute AttributeId=
[07]           "urn:oasis:names:tc:xacml:1.0:subject:subject-category"
[08]           DataType="http://www.w3.org/2001/XMLSchema#string"
[09]           Issuer="www.medico.com"
[10]           IssueInstant="2001-12-17T09:30:47-05:00">
[11]        <AttributeValue>
[12]           urn:oasis:names:tc:xacml:1.0:subject:category:access-subject
[13]        </AttributeValue>
[14]     </Attribute>
[15]     <Attribute AttributeId=
[16]           "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
[17]           DataType=
[18]           "urn:oasis:names:tc:xacml:1.0.data-type:x500name"
[19]           Issuer="www.medico.com"
[20]           IssueInstant="2001-12-17T09:30:47-05:00">
```

```
[21]       <AttributeValue>CN=Julius Hibbert</AttributeValue>
[22]    </Attribute>
[23]    <Attribute AttributeId=
[24]          "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
[25]          DataType=”http://www.w3.org/2001/XMLSchema#string”
[26]          Issuer="www.medico.com"
[27]          IssueInstant="2001-12-17T09:30:47-05:00">
[28]       <AttributeValue>physician</AttributeValue>
[29]    </Attribute>
[30]    <Attribute AttributeId=
[31]       "urn:oasis:names:tc:xacml:1.0:example:attribute:physician-id"
[32]          DataType=”http://www.w3.org/2001/XMLSchema#string”
[33]          Issuer="www.medico.com"
[34]          IssueInstant="2001-12-17T09:30:47-05:00">
[35]       <AttributeValue>jh1234</AttributeValue>
[36]    </Attribute>
[37] </Subject>
[38] <Resource>
[39]    <ResourceContent>
[40]       <md:record
[41]          xmlns:md="//http:www.medico.com/schemas/record.xsd">
[42]          <md:patient>
[43]             <md:patientDoB>1992-03-21</md:patientDoB>
[44]          </md:patient>
[45]          <!-- other fields -->
[46]       </md:record>
[47]    </ResourceContent>
[48]    <Attribute AttributeId=
[49]          "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
[50]          DataType=”http://www.w3.org/2001/XMLSchema#string”>
[55]       <AttributeValue>
[56]          //medico.com/records/bart-simpson.xml#
[57]             xmlns(md=//http:www.medico.com/schemas/record.xsd)
[58]             xpointer(/md:record/md:patient/md:patientDoB)
[59]       </AttributeValue>
[60]    </Attribute>
[61]    <Attribute AttributeId=
[62]          "urn:oasis:names:tc:xacml:1.0:resource:xpath"
[63]          DataType=”http://www.w3.org/2001/XMLSchema#string”>
[64]       <AttributeValue>
[65]          xmlns(md=http:www.medico.com/schemas/record.xsd)
[66]             xpointer(/md:record/md:patient/md:patientDoB)
[67]       </AttributeValue>
[68]    </Attribute>
[69]    <Attribute AttributeId=
[70]       ”urn:oasis:names:tc:xacml:1.0:resource:target-namespace”
[71]       DataType=”http://www.w3.org/2001/XMLSchema#string”>
[72]       <AttributeValue>
[73]          http://www.medico.com/schemas/record.xsd
[74]       </AttributeValue>
[75]    </Attribute>
[76] </Resource>
[77] <Action>
[78]    <Attribute AttributeId=
[79]          "urn:oasis:names:tc:xacml:1.0:action:action-id"
[80]          DataType=”http://www.w3.org/2001/XMLSchema#string”>
[81]       <AttributeValue>read</AttributeValue>
[82]    </Attribute>
[83] </Action>
[84] </Request>
```

[02]-[04] Standard namespace declarations.

[05]-[37] ***Subject*** attributes are placed in the `Subject` section of the `Request`. Each ***attribute*** consists of the ***attribute*** meta-data and the ***attribute*** value.

[06]-[14] Each `Subject` section must have one and only one subject-category ***attribute***. The value of this ***attribute*** describes the role that the ***subject*** plays in making the ***decision request***. The value of “`access-subject`” denotes the identity for which the request was issued.

[15]-[22] ***Subject*** `subject-id` ***attribute***.

[23]-[29] ***Subject*** `role` ***attribute***.

[30]-[36] ***Subject*** `physician-id` ***attribute***.

[38]-[69] ***Resource*** attributes are placed in the `Resource` section of the `Request`. Each ***attribute*** consists of ***attribute*** meta-data and an ***attribute*** value.

[39]-[47] ***Resource*** content. The XML document that is being requested is placed here.

[48]-[60] ***Resource*** identifier.

[56]-[58] The ***Resource*** is identified with an Xpointer expression that names the URI of the file that is accessed, the target namespace of the document, and the XPath location path to the specific element.

[61]-[68] The XPath location path in the “`resource-id`” attribute is extracted and placed in the `xpath` attribute.

[69]-[75] ***Resource*** `target-namespace` ***attribute***.

[77]-[84] ***Action attributes*** are placed in the `Action` section of the `Request`.

[78]-[82] ***Action*** identifier.

4.2.3 Example plain-language rules

The following plain-language rules are to be enforced:

Rule 1: A person may read any record for which he or she is the designated patient.

Rule 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.

Rule 3: A physician may write to any medical element for which he or she is the designated primary care physician, provided an email is sent to the patient.

Rule 4: An administrator shall not be permitted to read or write to medical elements of a patient record.

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

4.2.4 Example XACML rule instances

4.2.4.1. Rule 1

Rule 1 illustrates a simple ***rule*** with a single `<Condition>` element. The following XACML `<Rule>` instance expresses Rule 1:

```
[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Rule
```

```
[03]    xmlns="urn:oasis:names:tc:xacml:1.0:policy"
[04]    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[05]    xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
[06]    xmlns:md="http://www.medico.com/schemas/record.xsd"
[07]    RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
[08]    Effect="Permit">
[09]  <Description>
[10]    A person may read any medical record in the
[11]    http://www.medico.com/schemas/record.xsd namespace
[12]    for which he or she is a designated patient
[13]  </Description>
[14]  <Target>
[15]    <Subjects>
[16]      <AnySubject/>
[17]    </Subjects>
[18]    <Resources>
[20]      <Resource>
[21]        <!-- match document target namespace -->
[22]        <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[23]          <ResourceAttributeDesignator AttributeId=
[24]        "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[25]          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">
[26]            http://www.medico.com/schemas/record.xsd
[27]          </AttributeValue>
[28]        </ResourceMatch>
[29]        <!-- match requested xml element -->
[30]        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
node-match">
[31]          <ResourceAttributeDesignator AttributeId=
[32]            "urn:oasis:names:tc:xacml:1.0:resource:xpath"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
[33]          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeValue>
[34]        </ResourceMatch>
[35]      </Resource>
[36]    </Resources>
[37]    <Actions>
[38]      <Action>
[39]        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
[40]          <ActionAttributeDesignator AttributeId=
[41]          "urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[42]          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
[43]        </ActionMatch>
[44]      </Action>
[45]    </Actions>
[46]  </Target>
[47]  <!-- compare policy number in the document with
[48]      policy-number attribute -->
[49]  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[50]    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-
only">
[51]      <!-- policy-number attribute -->
[52]      <SubjectAttributeDesignator AttributeId=
[53]    "urn:oasis:names:tc:xacml:1.0:examples:attribute:policy-number"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[54]    </Apply>
```

```
[55]    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-
only">
[56]       <!-- policy number in the document -->
[57]       <AttributeSelector RequestContextPath=
[58]          "//md:record/md:patient/md:policyNumber"
DataType="http://www.w3.org/2001/XMLSchema#string">
[59]       </AttributeSelector>
[60]    </Apply>
[61]  </Condition>
[62] </Rule>
```

[02]-[06]. XML namespace declarations.

[07] ***Rule*** identifier.

[08]. When a ***rule*** evaluates to 'True' it emits the value of the `Effect` attribute. This value is combined with the `Effect` values of other rules according to the ***rule-combining algorithm***.

[09]-[13] Free form description of the ***rule***.

[14]-[46]. A ***rule target*** defines a set of ***decision requests*** that are applicable to the ***rule***. A decision request, such that the value of the "`urn:oasis:names:tc:xacml:1.0:resource:target-namespace`" ***resource attribute*** is equal to "http://www.medico.com/schema/records.xsd" and the value of the "`urn:oasis:names:tc:xacml:1.0:resource:xpath`" ***resource attribute*** matches the XPath expression `/md:record` and the value of the "`urn:oasis:names:tc:xacml:1.0:action:action-id`" ***action attribute*** is equal to "`read`", matches the ***target*** of this ***rule***.

[15]-[17]. The `Subjects` element may contain either a ***disjunctive sequence*** of `Subject` elements or `AnySubject` element.

[16] The `AnySubject` element is a special element that matches any ***subject*** in the request ***context***.

[18]-[36]. The `Resources` element may contain either a ***disjunctive sequence*** of `Resource` elements or `AnyResource` element.

[20]-[35] The `Resource` element encloses the ***conjunctive sequence*** of `ResourceMatch` elements.

[22]-[28] The `ResourceMatch` element compares its first and second child elements according to the matching function. A match is positive if any of the values selected by the first argument match the explicit value of the second argument. This match compares the target namespace of the requested document with the value of "http://www.medico.com/schema.records.xsd".

[22] The `MatchId` attribute names the matching function.

[23]-[24] The `ResourceAttributeDesignator` element selects the ***resource attribute*** values from the request ***context***. The ***attribute*** name is specified by the `AttributeId`. The selection result is a ***bag*** of values.

[25]-[27] Literal attribute value to match.

[30]-[34] The `ResourceMatch`. This match compares the results of two XPath expressions. The first XPath expression is the location path to the requested xml element and the second XPath expression is `/md:record`. The "xpath-node-match" function evaluates to "True" if the requested XML element is below the `/md:record` element.

[30] `MatchId` attribute names the matching function.

[31]-[32] The `ResourceAttributeDesignator` selects the ***bag*** of values for the “`urn:oasis:names:tc:xacml:1.0:xpath`” ***resource attribute***. Here, there is just one element in the ***bag***, which is the location path for the requested XML element.

[33] The literal XPath expression to match. The `md` prefix is resolved using a standard namespace declaration.

[37]-[45] The `Actions` element may contain either a ***disjunctive sequence*** of `Action` elements or an `AnyAction` element.

[38]-[44] The `Action` element contains a ***conjunctive sequence*** of `ActionMatch` elements.

[39]-[43] The `ActionMatch` element compares its first and second child elements according to the matching function. Match is positive, if any of the values selected by the first argument match explicit value of the second argument. In this case, the value of the `action-id` action attribute in the request ***context*** is compared with the value “`read`”.

[39] The `MatchId` attribute names the matching function.

[40]-[41] The `ActionAttributeDesignator` selects ***action attribute*** values from the request ***context***. The ***attribute*** name is specified by the `AttributeId`. The selection result is a ***bag*** of values. “`urn:oasis:names:tc:xacml:1.0:action:action-id`” is the predefined name for the action identifier.

[42] The ***Attribute*** value to match. This is an ***action*** name.

[49]-[61] The `Condition` element. A ***condition*** must evaluate to “True” for the ***rule*** to be applicable. This condition evaluates the truth of the statement: the `policy-number` ***subject attribute*** is equal to the policy number in the XML document.

[49] The `FunctionId` attribute of the `Condition` element names the function to be used for comparison. In this case, comparison is done with `function:string-equal`; this function takes two arguments of the “`http://www.w3.org/2001/XMLSchema#string`” type.

[50] The first argument to the `function:string-equal` in the `Condition`. Functions can take other functions as arguments. The `Apply` element encodes the function call with the `FunctionId` attribute naming the function. Since `function:string-equal` takes arguments of the “`http://www.w3.org/2001/XMLSchema#string`” type and `SubjectAttributeDesignator` selects a ***bag*** of “`http://www.w3.org/2001/XMLSchema#string`” values, “`function:string-one-and-only`” is used. This function guarantees that its argument evaluates to a ***bag*** containing one and only one “`http://www.w3.org/2001/XMLSchema#string`” element.

[52]-[53] The `SubjectAttributeDesignator` selects a ***bag*** of values for the `policy-number` ***subject attribute*** in the request ***context***.

[55] The second argument to the “`function:string-equal`” in the `Condition`. Functions can take other functions as arguments. The `Apply` element encodes function call with the `FunctionId` attribute naming the function. Since “`function:string-equal`” takes arguments of the “`http://www.w3.org/2001/XMLSchema#string`” type and the `AttributeSelector` selects a ***bag*** of “`http://www.w3.org/2001/XMLSchema#string`” values, “`function:string-one-and-only`” is used. This function guarantees that its argument evaluates to a ***bag*** containing one and only one “`http://www.w3.org/2001/XMLSchema#string`” element.

[57] The AttributeSelector element selects a ***bag*** of values from the request ***context***. The `AttributeSelector` is a free-form XPath pointing device into the request ***context***. The

`RequestContextPath` attribute specifies an XPath expression over the content of the requested XML document, selecting the policy number. Note that the namespace prefixes in the XPath expression are resolved with the standard XML namespace declarations.

4.2.4.2. Rule 2

Rule 2 illustrates the use of a mathematical function, i.e. the `<Apply>` element with `functionId` "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" to calculate date. It also illustrates the use of ***predicate*** expressions, with the `functionId` "urn:oasis:names:tc:xacml:1.0:function:and".

```
[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Rule
[03] xmlns="urn:oasis:names:tc:xacml:1.0:policy"
[04] xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
[05] xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
[06] xmlns:md="http:www.medico.com/schemas/record.xsd"
[07] RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
[08] Effect="Permit">
[09] <Description>
[10]    A person may read any medical record in the
[11]    http://www.medico.com/records.xsd namespace
[12]    for which he or she is the designated parent or guardian,
[13]    and for which the patient is under 16 years of age
[14] </Description>
[15] <Target>
[16]    <Subjects>
[17]       <AnySubject/>
[18]    </Subjects>
[19]    <Resources>
[20]       <Resource>
[21]          <!-- match document target namespace -->
[22]          <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[23]             <ResourceAttributeDesignator AttributeId=
[24]          "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
DataType=”http://www.w3.org/2001/XMLSchema#string”/>
[25]             <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>
[26]                http://www.medico.com/schemas/record.xsd
[27]             </AttributeValue>
[28]          </ResourceMatch>
[29]          <!-- match requested xml element -->
[30]          <ResourceMatch MatchId=”urn:oasis:names:tc:xacml:1.0:function:xpath-
node-match">
[31]             <ResourceAttributeDesignator AttributeId=
[32]                "urn:oasis:names:tc:xacml:1.0:resource:xpath"
DataType=”http://www.w3.org/2001/XMLSchema#string”/>
[33]             <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>/md:record</AttributeValue>
[34]          </ResourceMatch>
[35]       </Resource>
[36]    </Resources>
[37]    <Actions>
[38]       <Action>
[39]          <!-- match 'read' action -->
[40]          <ActionMatch MatchId=”urn:oasis:names:tc:xacml:1.0:function:string-
equal">
[41]             <ActionAttributeDesignator AttributeId=
[42]                "urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType=”http://www.w3.org/2001/XMLSchema#string”/>
```

```
[43]            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
[44]         </ActionMatch>
[45]       </Action>
[46]    </Actions>
[47] </Target>
[48] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
[49]    <!-- compare parent-guardian-id subject attribute with
[50]       the value in the document -->
[51]    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[52]       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
and-only">
[53]         <!-- parent-guardian-id subject attribute -->
[54]         <SubjectAttributeDesignator AttributeId=
[55]            "urn:oasis:names:tc:xacml:1.0:examples:attribute:
[56]              parent-guardian-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[57]       </Apply>
[58]       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
and-only">
[59]         <!-- parent-guardian-id element in the document -->
[60]         <AttributeSelector RequestContextPath=
[61]         "//md:record/md:parentGuardian/md:parentGuardianId"
[62]            DataType="http://www.w3.org/2001/XMLSchema#string">
[63]         </AttributeSelector>
[64]       </Apply>
[65]    </Apply>
[66]    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-
equal">
[67]       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-
only">
[68]         <EnvironmentAttributeDesignator AttributeId=
[69]         "urn:oasis:names:tc:xacml:1.0:environment:current-date"
DataType="http://www.w3.org/2001/XMLSchema#date"/>
[70]       </Apply>
[71]       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-
yearMonthDuration">
[73]         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-
and-only">
[74]            <!-- patient dob recorded in the document -->
[75]            <AttributeSelector RequestContextPath=
[76]              "//md:record/md:patient/md:patientDoB"
DataType="http://www.w3.org/2001/XMLSchema#date">
[77]            </AttributeSelector>
[78]         </Apply>
[79]         <AttributeValue DataType="xf:yearMonthDuration">
[80]            P16Y
[81]         </AttributeValue>
[82]       </Apply>
[83]    </Apply>
[84] </Condition>
[85] </Rule>
```

[02]-[47] ***Rule*** declaration and ***rule target***. See Rule 1 in section 4.2.4.1 for the detailed explanation of these elements.

[48]-[82] The `Condition` element. ***Condition*** must evaluate to "True" for the ***rule*** to be applicable. This ***condition*** evaluates the truth of the statement: the requestor is the designated parent or guardian and the patient is under 16 years of age.

[48] The `Condition` is using the "`function:and`" function. This is a boolean function that takes one or more boolean arguments (2 in this case) and performs the logical "AND" operation to compute the truth value of the expression.

[51]-[65] The truth of the first part of the condition is evaluated: The requestor is the designated parent or guardian. The `Apply` element contains a function invocation. The function name is contained in the `FunctionId` attribute. The comparison is done with "`function:string-equal`" that takes 2 arguments of "`http://www.w3.org/2001/XMLSchema#string`" type.

[52] Since "`function:string-equal`" takes arguments of the "`http://www.w3.org/2001/XMLSchema#string`" type, "`function:string-one-and-only`" is used to ensure that the ***subject attribute*** "urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id" in the request ***context*** contains one and only one value. "`Function:string-equal`" takes an argument expression that evaluates to a ***bag*** of "`http://www.w3.org/2001/XMLSchema#string`" values.

[54] Value of the ***subject attribute*** "`urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id`" is selected from the request ***context*** with the `SubjectAttributeDesignator` element. This expression evaluates to a bag of "`http://www.w3.org/2001/XMLSchema#string`" values.

[58] "`function:string-one-and-only`" is used to ensure that the ***bag*** of values selected by it's argument contains one and only one value of type "`http://www.w3.org/2001/XMLSchema#string`".

[60] The value of the `md:parentGuardianId` element is selected from the ***resource*** content with the `AttributeSelector` element. `AttributeSelector` is a free-form XPath expression, pointing into the request ***context***. The `RequestContextPath` XML attribute contains an XPath expression over the request ***context***. Note that all namespace prefixes in the XPath expression are resolved with standard namespace declarations. The `AttributeSelector` evaluates to the ***bag*** of values of type "`http://www.w3.org/2001/XMLSchema#string`".

[66]-[83] The expression: "the patient is under 16 years of age" is evaluated. The patient is under 16 years of age if the current date is less than the date computed by adding 16 to the patient's date of birth.

[66] "`function:date-less-or-equal`" is used to compute the difference of two dates.

[67] "`function:date-one-and-only`" is used to ensure that the ***bag*** of values selected by its argument contains one and only one value of type "`http://www.w3.org/2001/XMLSchema#date`".

[68]-[69] Current date is evaluated by selecting the "`urn:oasis:names:tc:xacml:1.0:environment:current-date`" ***environment attribute***.

[71] "`function:date-add-yearMonthDuration`" is used to compute the date by adding 16 to the patient's date of birth. The first argument is a "`http://www.w3.org/2001/XMLSchema#date`", and the second argument is an "`xf:yearMonthDuration`".

[73] "`function:date-one-and-only`" is used to ensure that the ***bag*** of values selected by it's argument contains one and only one value of type "`http://www.w3.org/2001/XMLSchema#date`".

[75]-[76] The `<AttributeSelector>` element selects the patient's date of birth by taking the XPath expression over the document content.

[79]-[81] Year Month Duration of 16 years.

4.2.4.3. 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 `<Policy>` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
   xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
   xmlns:md="http:www.medico.com/schemas/record.xsd"
   PolicyId="urn:oasis:names:tc:xacml:examples:policyid:3"
   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
      rule-combining-algorithm:deny-overrides">
<Description>
   Policy for any medical record in the
   http://www.medico.com/schemas/record.xsd namespace
</Description>
<Target>
   <Subjects>
      <AnySubject/>
   </Subjects>
   <Resources>
      <Resource>
         <!-- match document target namespace -->
         <ResourceMatch
MatchId=“urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <ResourceAttributeDesignator AttributeId=
         "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
DataType=”http://www.w3.org/2001/XMLSchema#string”/>
            <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>
               http://www.medico.com/schemas/record.xsd
            </AttributeValue>
         </ResourceMatch>
      </Resource>
   </Resources>
   <Actions>
      <AnyAction/>
   </Actions>
</Target>
<Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:3"
   Effect="Permit">
   <Description>
      A physician may write any medical element in a record
      for which he or she is the designated primary care
      physician, provided an email is sent to the patient
   </Description>
   <Target>
   <Subjects>
      <Subject>
         <!-- match subject group attribute -->
         <SubjectMatch MatchId=“urn:oasis:names:tc:xacml:1.0:function:string-
equal">
            <SubjectAttributeDesignator AttributeId=
   "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
DataType=”http://www.w3.org/2001/XMLSchema#string”/>
            <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>physician</AttributeValue>
         </SubjectMatch>
      </Subject>
```

```
[51]    </Subjects>
[52]    <Resources>
[53]       <Resource>
[54]          <!-- match requested xml element -->
[55]          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
node-match">
[56]             <ResourceAttributeDesignator AttributeId=
[57]                "urn:oasis:names:tc:xacml:1.0:resource:xpath"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[58]             <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">
[59]                /md:record/md:medical
[60]             </AttributeValue>
[61]          </ResourceMatch>
[62]       </Resource>
[63]    </Resources>
[64]    <Actions>
[65]       <Action>
[66]          <!-- match action -->
[67]          <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
[68]             <ActionAttributeDesignator AttributeId=
[069]      "urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[070]            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
[071]         </ActionMatch>
[072]      </Action>
[073]   </Actions>
[074]   </Target>
[075]   <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
[076]      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
and-only">
[077]         <!-- physician-id subject attribute -->
[078]         <SubjectAttributeDesignator AttributeId=
[079]            "urn:oasis:names:tc:xacml:1.0:example:
[080]               attribute:physician-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[081]      </Apply>
[082]      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
and-only">
[083]         <AttributeSelector RequestContextPath=
[084]      "//md:record/md:primaryCarePhysician/md:registrationID"
[085]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
[086]      </Apply>
[087]   </Condition>
[089]</Rule>
[090]<Obligations>
[091]   <!-- send e-mail message to the document owner -->
[092]   <Obligation ObligationId=
[093]      "urn:oasis:names:tc:xacml:example:obligation:email"
[094]      FulfillOn="Permit">
[095]      <AttributeAssignment AttributeId=
[096]      "urn:oasis:names:tc:xacml:1.0:example:attribute:mailto"
[097]         DataType="http://www.w3.org/2001/XMLSchema#string">
[098]         <AttributeSelector RequestContextPath=
[099]         "//md:/record/md:patient/md:patientContact/md:email"
[100]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
[101]      </AttributeAssignment>
[102]      <AttributeAssignment AttributeId=
[103]         "urn:oasis:names:tc:xacml:1.0:example:attribute:text"
[104]         DataType="http://www.w3.org/2001/XMLSchema#string">
```

```
[105]          <AttributeValue>
[106]             Your medical record has been accessed by:
[107]          </AttributeValue>
[108]       </AttributeAssignment>
[109]       <AttributeAssignment AttributeId=
[110]             "urn:oasis:names:tc:xacml:example:attribute:text"
[111]          DataType="http://www.w3.org/2001/XMLSchema#string">
[112]          <SubjectAttributeDesignator AttributeId=
[113]          "urn:osasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[114]       </AttributeAssignment>
[115]    </Obligation>
[116] </Obligations>
[117] </Policy>
```

[01]-[09] The `Policy` element includes standard namespace declarations as well as policy specific parameters, such as `PolicyId` and `RuleCombiningAlgId`.

[07] ***Policy*** identifier. This parameter is used for the inclusion of the `Policy` in the `PolicySet` element.

[08]-[09] ***Rule combining algorithm*** identifier. This parameter is used to compute the combined outcome of ***rule effects*** for ***rules*** that are applicable to the ***decision request***.

[10-13] Free-form description of the ***policy***.

[14]-[33] ***Policy target***. The ***policy target*** defines a set of applicable decision requests. The structure of the `Target` element in the `Policy` is identical to the structure of the `Target` element in the `Rule`. In this case, the ***policy target*** is a set of all XML documents conforming to the "http://www.medico.com/schemas/record.xsd" target namespace. For the detailed description of the `Target` element see Rule 1, section 4.2.4.1.

[34]-[89] The only `Rule` element included in this `Policy`. Two parameters are specified in the ***rule*** header: `RuleId` and `Effect`. For the detailed description of the `Rule` structure see Rule 1, section 4.2.4.1.

[41]-[74] A ***rule target*** narrows down a ***policy target***. ***Decision requests*** with the value of "`urn:oasis:names:tc:xacml:1.0:exampe:attribute:role`" ***subject attribute*** equal to "`physician`" [42]-[51], and that access elements of the medical record that "xpath-node-match" the "`/md:record/md:medical`" XPath expression [52]-[63], and that have the value of the "`urn:oasis:names:tc:xacml:1.0:action:action-id`" ***action attribute*** equal to "`read`".

[65]-[73] match the ***target*** of this ***rule***. For a detailed description of the rule target see example 1, section 4.2.4.1.

[75]-[87] The `Condition` element. For the ***rule*** to be applicable to the authorization request, ***condition*** must evaluate to True. This ***rule condition*** compares the value of the "`urn:oasis:names:tc:xacml:1.0:examples:attribute:physician-id`" ***subject attribute*** with the value of the `physician id` element in the medical record that is being accessed. For a detailed explanation of rule condition see Rule 1, section 4.2.4.1.

[90]-[116] The `Obligations` element. ***Obligations*** are a set of operations that must be performed by the ***PEP*** in conjunction with an ***authorization decision.*** An ***obligation*** may be associated with a positive or negative ***authorization decision***.

[92]-[115] The `Obligation` element consists of the `ObligationId`, the authorization decision value for which it must fulfill, and a set of attribute assignments.

[92]-[93] `ObligationId` identifies an ***obligation***. ***Obligation*** names are not interpreted by the ***PDP***.

[94] `FulfillOn` attribute defines an ***authorization decision*** value for which this ***obligation*** must be fulfilled.

[95]-[101] ***Obligation*** may have one or more parameters. The ***obligation*** parameter "`urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto`" is assigned the value from the content of the xml document.

[95-96] `AttributeId` declares "`urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto`" ***obligation*** parameter.

[97] The ***obligation*** parameter data type is defined.

[98]-[100] The ***obligation*** parameter value is selected from the content of the XML document that is being accessed with the XPath expression over request ***context***.

[102]-[108] The ***obligation*** parameter "`urn:oasis:names:tc:xacml:1.0:examples:attribute:text`" of type "`http://www.w3.org/2001/XMLSchema#string`" is assigned the literal value "`Your medical record has been accessed by:`"

[109]-[114] The ***obligation*** parameter "`urn:oasis:names:tc:xacml:1.0:examples:attribute:text`" of the "http://www.w3.org/2001/XMLSchema#strng" data type is assigned the value of the "`urn:oasis:names:tc:xacml:1.0:subject:subject-id`" ***subject attribute***.

4.2.4.4. Rule 4

Rule 4 illustrates the use of the "Deny" `Effect` value, and a `Rule` with no `Condition` element.

```
[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Rule
[03] xmlns="urn:oasis:names:tc:xacml:1.0:policy"
[04] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[05] xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
[06] xmlns:md="http:www.medico.com/schemas/record.xsd"
[07] RuleId="urn:oasis:names:tc:xacml:example:ruleid:4"
[08] Effect="Deny">
[09] <Description>
[10]    An Administrator shall not be permitted to read or write
[11]    medical elements of a patient record in the
[12]    http://www.medico.com/records.xsd namespace.
[13] </Description>
[14] <Target>
[15]    <Subjects>
[16]       <Subject>
[17]          <!-- match role subject attribute -->
[18]          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
[19]             <SubjectAttributeDesignator AttributeId=
[20]    "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[21]             <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">administrator</AttributeValue>
[22]          </SubjectMatch>
[23]       </Subject>
[24]    </Subjects>
[25]    <Resources>
```

```
[26]        <Resource>
[27]          <!-- match document target namespace -->
[28]          <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[29]            <ResourceAttributeDesignator AttributeId=
[30]    "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[31]            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">
[32]              http://www.medico.com/schemas/record.xsd
[33]            </AttributeValue>
[34]          </ResourceMatch>
[35]          <!-- match requested xml element -->
[36]          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
node-match">
[37]            <ResourceAttributeDesignator AttributeId=
[38]              "urn:oasis:names:tc:xacml:1.0:resource:xpath"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[39]            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">
[40]              /md:record/md:medical
[41]            </AttributeValue>
[42]          </ResourceMatch>
[43]        </Resource>
[44]      </Resources>
[45]      <Actions>
[46]        <Action>
[47]          <!-- match 'read' action -->
[48]          <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
[49]            <ActionAttributeDesignator AttributeId=
[50]            "urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[51]            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
[52]          </ActionMatch>
[53]        </Action>
[54]        <Action>
[55]          <!-- match 'write' action -->
[56]          <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
[57]            <ActionAttributeDesignator AttributeId=
[58]            "urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
[59]            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
[60]          </ActionMatch>
[61]        </Action>
[62]      </Actions>
[63]   </Target>
[64] </Rule>
```

[01]-[08] The `Rule` element declaration. The most important parameter here is `Effect`. See Rule 1, section 4.2.4.1 for a detailed explanation of the `Rule` structure.

[08] ***Rule*** `Effect`. Every ***rule*** that evaluates to "True" emits ***rule effect*** as its value that will be combined later on with other ***rule effects*** according to the ***rule combining algorithm***. This ***rule*** `Effect` is "Deny" meaning that according to this rule, access must be denied.

[09]-[13] Free form description of the ***rule***.

[14]-[63] ***Rule target***. The ***Rule target*** defines a set of ***decision requests*** that are applicable to the **rule**. This ***rule*** is matched by:

- a ***decision request*** with ***subject attribute*** "`urn:oasis:names:tc:xacml:1.0:examples:attribute:role`" equal to "`administrator`";
- the value of ***resource attribute*** "`urn:oasis:names:tc:xacml:1.0:resource:target-namespace`" is equal to "`http://www.medico.com/schemas/record.xsd`"
- the value of the requested XML element matches the XPath expression "`/md:record/md:medical`";
- the value of ***action attribute*** "urn:oasis:names:tc:xacml:1.0:action:action-id" is equal to "read"

See Rule 1, section 4.2.4.1 for the detailed explanation of the `Target` element.

This ***rule*** does not have a `Condition` element.

4.2.4.5. Example PolicySet

This section uses the examples of the previous sections to illustrate the process of combining ***policies***. The policy governing read access to medical elements of a record is formed from each of the four ***rules described in Section 4.2.3.*** 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 `<PolicySet>` illustrates the combined ***policies***. ***Policy*** 3 is included by reference and ***policy*** 2 is explicitly included.

```
[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <PolicySet
[03]    xmlns="urn:oasis:names:tc:xacml:1.0:policy"
[04]    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[05]    PolicySetId=
[06]       "urn:oasis:names:tc:xacml:1.0:examples:policysetid:1"
[07]    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
[071]      policy-combining-algorithm:deny-overrides"/>
[08] <Description>
[09]    Example policy set.
[10] </Description>
[11] <Target>
[12]    <Subjects>
[13]       <Subject>
[14]          <!-- any subject -->
[15]          <AnySubject/>
[16]       </Subject>
[17]    </Subjects>
[18]    <Resources>
[19]       <Resource>
[20]          <!-- any resource in the target namespace -->
[21]          <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
```

```
[22]            <ResourceAttributeDesignator AttributeId=
[23]    "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
DataType=”http://www.w3.org/2001/XMLSchema#string”/>
[24]            <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>
[25]              http://www.medico.com/records.xsd
[26]            </AttributeValue>
[27]          </ResourceMatch>
[28]        </Resource>
[29]      </Resources>
[30]      <Actions>
[31]        <Action>
[32]          <!-- any action -->
[33]          <AnyAction/>
[34]        </Action>
[35]      </Actions>
[36]  </Target>
[37]  <!-- include policy from the example 3 by reference -->
[38]  <PolicyIdReference>
[39]      urn:oasis:names:tc:xacml:1.0:examples:policyid:3
[40]  </PolicyIdReference>
[41]      <!-- policy 2 combines rules from the examples 1, 2,
[42]      and 4 is included by value. -->
[43]  <Policy
[44]      PolicyId="urn:oasis:names:tc:xacml:examples:policyid:2"
[45]      RuleCombiningAlgId=
[46]"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
[47]      <Description>
[48]        Policy for any medical record in the
[49]        http://www.medico.com/schemas/record.xsd namespace
[50]      </Description>
[51]      <Target> ... </Target>
[52]      <Rule
[53]        RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
[54]        Effect="Permit"> ... </Rule>
[55]      <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
[56]        Effect="Permit"> ... </Rule>
[57]      <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:4"
[58]        Effect="Deny"> ... </Rule>
[59]      <Obligations> ... </Obligations>
[60]  </Policy>
[61] </PolicySet>
```

[02]-[07] `PolicySet` declaration. Standard XML namespace declarations are included as well as `PolicySetId`, and ***policy combining algorithm*** identifier.

[05]-[06] `PolicySetId` is used for identifying this ***policy set*** and for possible inclusion of this ***policy set*** into another ***policy set***.

[07] ***Policy combining algorithm*** identifier. Policies in the ***policy set*** are combined according to the specified ***policy combining algorithm*** identifier when the ***authorization decision*** is computed.

[08]-[10] Free form description of the ***policy set***.

[11]-[36] `PolicySet` `Target` element defines a set of ***decision requests*** that are applicable to this `PolicySet`.

[38]-[40] `PolicyIdReference` includes ***policy*** by id.

[43]-[60] **Policy** 2 is explicitly included in this ***policy set***.

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

5.1. Element <PolicySet>

The `<PolicySet>` element is a top-level element in the XACML policy schema. `<PolicySet>` is an aggregation of other ***policy sets*** and ***policies***. ***Policy sets*** MAY be included in an enclosing `<PolicySet>` element either directly by the `<PolicySet>` element or indirectly by the `<PolicySetIdReference>` element. ***Policies*** MAY be included in an enclosing `<PolicySet>` element either directly by the <Policy> element or indirectly by the `<PolicyIdReference>` element.

If a `<PolicySet>` element contains references to other ***policy sets*** or ***policies*** in the form of URLs, then these references may be resolvable.

Policies included in the `<PolicySet>` element MUST be combined by the algorithm specified by the `PolicyCombiningAlgId` attribute.

The `<Target>` element defines the applicability of the `<PolicySet>` to ***decision requests***. If there is a match between the `<Target>` element within `<PolicySet>` and the ***request context***, then the `<PolicySet>` element MAY be used by the ***PDP*** in making its ***authorization decision***.

The `<Obligations>` element contains a set of ***obligations*** that MUST be fulfilled by the ***PEP*** in conjunction with the ***authorization decision***. If the ***PEP*** does not understand any of the ***obligations***, then it MUST act as if the ***PDP*** had returned a “Deny” ***authorization decision*** value.

```
   <xs:element name="PolicySet" type="xacml:PolicySetType"/>
   <xs:complexType name="PolicySetType">
      <xs:sequence>
         <xs:element ref="xacml:Description" minOccurs="0"/>
         <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>
         <xs:element ref="xacml:Target"/>
         <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="xacml:PolicySet"/>
            <xs:element ref="xacml:Policy"/>
            <xs:element ref="xacml:PolicySetIdReference"/>
            <xs:element ref="xacml:PolicyIdReference"/>
         </xs:choice>
         <xs:element ref="xacml:Obligations" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="PolicySetId"
type="http://www.w3.org/2001/XMLSchema#anyURI" use="required"/>
      <xs:attribute name="PolicyCombiningAlgId"
type="http://www.w3.org/2001/XMLSchema#anyURI" use="required"/>
   </xs:complexType>
```

The `<PolicySet>` element is of **PolicySetType** complex type.

The `<PolicySet>` element contains the following attributes and elements:

`PolicySetId` [Required]

> ***Policy set*** identifier. It is the responsibility of the ***PAP*** to ensure that no two ***policies*** visible to the ***PDP*** have the same identifier. This MAY be achieved by following a predefined URN or URI scheme. If the policy set identifier is in the form of a URL, then it MAY be resolvable.

`PolicyCombiningAlgId` [Required]

> The identifier of the ***policy-combining algorithm*** by which the `<PolicySet>` components MUST be combined. Standard ***policy-combining algorithms*** are listed in Appendix C. Standard ***policy-combining algorithm*** identifiers are listed in Section B.10.

<Description> [Optional]

> A free-form description of the `<PolicySet>`.

`<PolicySetDefaults>` [Optional]

> A set of default values applicable to the `<PolicySet>`. The scope of the `<PolicyDefaults>` element SHALL be the enclosing policy.

`<Target>` [Required]

> The <Target> element defines the applicability of a <PolicySet> to ***decision requests***.

> The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be computed from the <Target> elements of the referenced <Policy> elements, either as an intersection or as a union.

`<PolicySet>` [Any Number]

> A ***policy set*** component that is included in this ***policy set***.

`<Policy>` [Any Number]

> A ***policy*** component that is included in this ***policy set***.

`<PolicySetIdReference>` [Any Number]

> A reference to a `<PolicySet>` component that MUST be included in this ***policy set***. If `<PolicySetIdReference>` is a URL, then it MAY be resolvable.

`<PolicyIdReference>` [Any Number]

> A reference to a `<Policy>` component that MUST be included in this ***policy set***. If the `<PolicyIdReference>` is a URL, then it MAY be resolvable.

`<Obligations>` [Optional]

> Contains the set of `<Obligation>` elements. See Section 7.11 for a description of how the set of ***obligations*** to be returned by the ***PDP*** shall be determined.

5.2. Element <Description>

The `<Description>` element is used for a free-form description of the `<PolicySet>` element and `<Policy>` element. The `<Description>` element is of **xs:string** simple type.

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

5.3. Element <PolicySetDefaults>

The `<PolicySetDefaults>` element SHALL specify default values that apply to the `<PolicySet>` element.

```
<xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>
```

```
<xs:complexType name=”DefaultsType”>
   <xs:sequence>
      <xs:choice>
         <xs:element ref=”xacml:XPathVersion” minOccurs=”0”/>
      </xs:choice>
   </xs:sequence>
</xs:complexType>
```

`<PolicySetDefaults>` element is of **DefaultsType** complex type.

`<XPathVersion>` [Optional]

Default XPath version.

5.4. Element <XPathVersion>

The `<XPathVersion>` element SHALL specify the version of the XPath specification to be used by `<AttributeSelector>` elements.

```
<xs:element name="XPathVersion" type="xs:anyURI"/>
```

The URI for the XPath 1.0 specification is “`http://www.w3.org/TR/1999/Rec-xpath-19991116`”. The `<XPathVersion>` element is REQUIRED if the XACML policy contains `<AttributeSelector>` elements.

5.5. Element <Target>

The `<Target>` element identifies the set of ***decision requests*** that the parent element is intended to evaluate. The `<Target>` element SHALL appear as a child of `<PolicySet>`, `<Policy>` and `<Rule>` elements. It contains definitions for ***subjects***, ***resources*** and ***actions***.

The `<Target>` element SHALL contain a ***conjunctive sequence*** of `<Subjects>`, `<Resources>` and `<Actions>` elements. For the parent of the `<Target>` element to be applicable to the ***decision request***, there MUST be at least one positive match between each section of the `<Target>` element and the corresponding section of the `<xacml-context:Request>` element.

```
<xs:element name="Target" type="xacml:TargetType"/>
<xs:complexType name="TargetType">
   <xs:sequence>
      <xs:element ref="xacml:Subjects"/>
      <xs:element ref="xacml:Resources"/>
      <xs:element ref="xacml:Actions"/>
   </xs:sequence>
</xs:complexType>
```

The `<Target>` element is of **TargetType** complex type.

`<Subjects>` [Required]

Matching specification for the ***subject attributes*** in the ***context***.

`<Resources>` [Required]

Matching specification for the ***resource attributes*** in the ***context***.

`<Actions>` [Required]

Matching specification for the ***action attributes*** in the ***context***.

5.6. Element <Subjects>

The `<Subjects>` element SHALL contains a ***disjunctive sequence*** of `<Subject>` elements.

```
<xs:element name="Subjects" type="xacml:SubjectsType"/>
<xs:complexType name="SubjectsType">
  <xs:choice>
    <xs:element ref="xacml:Subject" maxOccurs="unbounded"/>
    <xs:element ref="xacml:AnySubject"/>
  </xs:choice>
</xs:complexType>
```

The `<Subjects>` element is of **SubjectsType** complex type.

`<Subject>` [One To Many, Required Choice]

See section 5.7.

`<AnySubject>` [Required Choice]

See section 5.8.

5.7. Element <Subject>

The `<Subject>` element SHALL contain a ***conjunctive sequence*** of `<SubjectMatch>` elements.

```
  <xs:element name="Subject" type="xacml:SubjectType"/>
  <xs:complexType name="SubjectType">
    <xs:sequence>
      <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

The `<Subject>` element is of **SubjectType** complex type.

`<Subject>` element contains the following elements:

`<SubjectMatch>` [One to Many]

A ***conjunctive sequence*** of individual matches of the ***subject attributes*** in the ***context*** and the embedded ***attribute*** values.

5.8. Element <AnySubject>

The `<AnySubject>` element SHALL match any ***subject attribute*** in the ***context***.

```
<xs:element name="AnySubject"/>
```

5.9. Element <SubjectMatch>

The `<SubjectMatch>` element SHALL identify a set of ***subject***-related entities by matching ***attribute*** values in the `<xacml-context:Subject>` element of the ***context*** with the embedded ***attribute*** value.

```
  <xs:element name="SubjectMatch" type="xacml:SubjectMatchType"/>
  <xs:complexType name="SubjectMatchType">
    <xs:sequence>
      <xs:choice>
```

```
            <xs:element ref="xacml:SubjectAttributeDesignator"/>
            <xs:element ref="xacml:AttributeSelector"/>
         </xs:choice>
         <xs:element ref="xacml:AttributeValue"/>
      </xs:sequence>
      <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
   </xs:complexType>
```

The `<SubjectMatch>` element is of **SubjectMatchType** complex type.

The `<SubjectMatch>` element contains the following attributes and elements:

`MatchId` [Required]

Specifies a matching function. The value of this attribute MUST be of type xs:anyURI with legal values documented in Appendix A.

`<SubjectAttributeDesignator>` [Required choice]

Identifies one or more ***attribute*** values in the `<xacml-context:Subject>` child of the `<xacml-context:Request>` element.

`<AttributeSelector>` [Required choice]

MAY be used to identify one or more ***attribute*** values in the `<xacml-context:Subject>` child of the `<xacml-context:Request>` element.

`<AttributeValue>` [Required]

Embedded ***attribute*** value.

5.10. Element <Resources>

The `<Resources>` element SHALL contain a ***disjunctive sequence*** of `<Resource>` elements.

```
   <xs:element name="Resources" type="xacml:ResourcesType"/>
   <xs:complexType name="ResourcesType">
      <xs:choice>
         <xs:element ref="xacml:Resource" maxOccurs="unbounded"/>
         <xs:element ref="xacml:AnyResource"/>
      </xs:choice>
   </xs:complexType>
```

The `<Resources>` element is of **ResourcesType** complex type.

The `<Resources>` element consists of the following elements:

`<Resource>` [One To Many, Required Choice]

See section 5.11.

`<AnyResource>` [Required Choice]

See section 5.12.

5.11. Element <Resource>

The `<Resource>` element SHALL container a ***conjunctive sequence*** of `<ResourceMatch>` elements.

```
<xs:element name="Resource" type="xacml:ResourceType"/>
<xs:complexType name="ResourceType">
  <xs:sequence>
    <xs:element ref="xacml:ResourceMatch" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

The `<Resource>` element is of **ResourceType** complex type.

The `<Resource>` element contains the following elements:

`<ResourceMatch>` [One to Many]

A ***conjunctive sequence*** of individual matches of the ***resource attributes*** in the ***context*** and the embedded ***attribute*** values.

5.12. Element <AnyResource>

The `<AnyResource>` element SHALL match any ***resource attribute*** in the ***context***.

```
<xs:element name="AnyResource"/>
```

5.13. Element <ResourceMatch>

The `<ResourceMatch>` element SHALL identify a set of ***resource***-related entities by matching ***attribute*** values in the `<xacml-context:Resource>` element of the ***context*** with the embedded ***attribute*** value.

```
<xs:element name="ResourceMatch" type="xacml:ResourceMatchType"/>
<xs:complexType name="ResourceMatchType">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="xacml:ResourceAttributeDesignator"/>
      <xs:element ref="xacml:AttributeSelector"/>
    </xs:choice>
    <xs:element ref="xacml:AttributeValue"/>
  </xs:sequence>
  <xs:attribute name="MatchId" type="xs:anyMatch" use="required"/>
</xs:complexType>
```

The `<ResourceMatch>` element is of **ResourceMatchType** complex type.

The `<ResourceMatch>` element contains the following attributes and elements:

`MatchId` [Required]

Specifies a matching function. Values of this attribute MUST be of type xs:anyURI, with legal values documented in Appendix A.

`<ResourceAttributeDesignator>` [Required Choice]

Identifies one or more ***attribute*** values in the `<xacml-context:Resource>` child of the <xacml-context:Request> element.

`<AttributeSelector>` [Required Choice]

MAY be used to identify one or more ***attribute*** values in the `<xacml-context:Resource>` child of the `<xacml-context:Request>` element.

`<AttributeValue>` [Required]

Embedded ***attribute*** value.

5.14. Element <Actions>

The `<Actions>` element SHALL contain a ***disjunctive sequence*** of `<Action>` elements.

```
<xs:element name="Actions" type="xacml:ActionsType"/>
<xs:complexType name="ActionsType">
   <xs:choice>
      <xs:element ref="xacml:Action" maxOccurs="unbounded"/>
      <xs:element ref="xacml:AnyAction"/>
   </xs:choice>
</xs:complexType>
```

The `<Actions>` element is of **ActionsType** complex type.

The `<Actions>` element contains the following elements:

`<Action>` [One To Many, Required Choice]

See section 5.15.

`<AnyAction>` [Required Choice]

See section 5.16.

5.15. Element <Action>

The `<Action>` element SHALL contain a ***conjunctive sequence*** of `<ActionMatch>` elements.

```
<xs:element name="Action" type="xacml:ActionType"/>
<xs:complexType name="ActionType">
   <xs:sequence>
      <xs:element ref="xacml:ActionMatch" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>
```

The `<Action>` element is of **ActionType** complex type.

The `<Action>` element contains the following elements:

`<ActionMatch>` [One to Many]

A ***conjunctive sequence*** of individual matches of the ***action*** attributes in the ***context*** and the embedded ***attribute*** values.

5.16. Element <AnyAction>

The `<AnyAction>` element SHALL match any ***action attribute*** in the ***context***.

```
<xs:element name="AnyAction"/>
```

5.17. Element <ActionMatch>

The `<ActionMatch>` element SHALL identify a set of ***action***-related entities by matching ***attribute*** values in the `<xacml-context:Action>` element of the ***context*** with the embedded ***attribute*** value.

```
<xs:element name="ActionMatch" type="xacml:ActionMatchType"/>
<xs:complexType name="ActionMatchType">
   <xs:sequence>
      <xs:choice>
         <xs:element ref="xacml:ActionAttributeDesignator"/>
         <xs:element ref="xacml:AttributeSelector"/>
      </xs:choice>
      <xs:element ref="xacml:AttributeValue"/>
   </xs:sequence>
   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
</xs:complexType>
```

The `<ActionMatch>` element is of **ActionMatchType** complex type.

The `<ActionMatch>` element contains the following attributes and elements:

`MatchId` [Required]

Specifies a matching function. The value of this attribute MUST be of type xs:anyURI, with legal values documented in Appendix A.

`<ActionAttributeDesignator>` [Required Choice]

Identifies one or more ***attribute*** values in the `<xacml-context:Action>` child of the `<xacml-context:Request>` element.

`<AttributeSelector>` [Required Choice]

MAY be used to identify one or more ***attribute*** values in the `<xacml-context:Action>` child of the `<xacml-context:Request>` element.

`<AttributeValue>` [Required]

Embedded ***attribute*** value.

5.18. Element <PolicySetIdReference>

The `<PolicySetIdReference>` element SHALL be used to reference a `<PolicySet>` element by id. If `<PolicySetIdReference>` is a URL, then it MAY be resolvable to the `<PolicySet>`. The mechanism for resolving a ***policy set*** reference to the corresponding ***policy set*** is implementation dependent.

```
<xs:element name="PolicySetIdReference" type="xs:anyURI"/>
```

Element `<PolicySetIdReference>` is of **xs:anyURI** simple type.

5.19. Element <PolicyIdReference>

The `<xacml:PolicyIdReference>` element SHALL be used to reference a `<Policy>` element by id. If `<PolicyIdReference>` is a URL, then it MAY be resolvable to the `<Policy>`. The mechanism for resolving a ***policy*** reference to the corresponding ***policy*** is implementation dependent.

```
<xs:element name="PolicyIdReference" type="xs:anyURI"/>
```

Element `<PolicyIdReference>` is of **xs:anyURI** simple type.

5.20. Element <Policy>

The `<Policy>` element is the smallest entity that SHALL be presented to the ***PDP*** for evaluation.

The main components of this element are the `<Target>`, `<Rule>` and `<Obligations>` elements and the `RuleCombiningAlgId` attribute.

The `<Target>` element SHALL define `<Policy>` applicability to ***decision requests***. A sequence of `<Rule>` elements SHALL specify authorizations that MUST be combined according to the `RuleCombiningAlgId` attribute. The `<Obligations>` element SHALL contain a set of ***obligations*** that MUST be discharged by the ***PDP*** in conjunction with the ***authorization decision***.

```
<xs:element name="Policy" type="xacml:PolicyType"/>
<xs:complexType name="PolicyType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>
    <xs:element ref="xacml:Target"/>
    <xs:element ref="xacml:Rule" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:Obligations" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
  <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
</xs:complexType>
```

The `<Policy>` element is of **PolicyType** complex type.

The `<Policy>` element contains the following attributes and elements:

`PolicyId` [Required]

> Policy identifier. The party assigning this identifier MUST minimize the potential of some other party reusing the same identifier. This MAY be achieved by following a predefined URN or URL scheme. It is OPTIONAL for the `PolicyId` URL to be resolvable to the corresponding `<Policy>` object.

RuleCombiningAlgId [Required]

> The identifier of the rule-combining algorithm by which the <Policy> components MUST be combined. Standard rule-combining algorithms are listed in Appendix C. Standard rule-combining algorithm identifiers are listed in Section B.10.

`<Description> [Optional]`

> A free-form description of the ***policy***.

`<PolicyDefaults>` [Optional]

> Defines a set of default values applicable to the ***policy***. The scope of the `<PolicyDefaults>` element SHALL be the enclosing policy.

`<Target>` [Required]

> The <Target> element SHALL define the applicability of a <Policy> to ***decision requests***.

The `<Target>` element MAY be declared by the creator of the `<Policy>` element, or it MAY be computed from the `<Target>` elements of the referenced `<Rule>` elements either as an intersection or as a union.

`<Rule>` [Any Number]

A sequence of authorizations that MUST be combined according to the `RuleCombiningAlgId` attribute. ***Rules*** whose `<Target>` elements match the ***decision request*** MUST be considered. ***Rules*** whose `<Target>` elements do not match the ***decision request*** MUST NOT be considered. Applicability of ***rules*** to the ***decision request*** is detailed in Appendix C.

`<Obligations>` [Optional]

A ***conjunctive sequence*** of ***obligations*** that MUST be discharged by the ***PEP*** in conjunction with the ***authorization decision***. See Section 7.11 for a description of how the set of obligations to be returned by the ***PDP*** shall be determined.

5.21. Element <Rule>

The `<Rule>` element SHALL define individual ***rules*** in the ***policy***. The main components of this element are the `<Target>` and `<Condition>` elements and the Effect attribute.

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

The `<Rule>` element is of **RuleType** complex type.

The `<Rule>` element contains the following attributes and elements:

`RuleId [Required]`

A URN identifying this ***rule***.

`Effect [Required]`

Rule effect. Values of this attribute are either “Permit” or “Deny”.

`<Description>` [optional]

A free-form description of the ***rule***.

`<Target>` [optional]

Identifies the set of ***decision requests*** that the `<Rule>` element is intended to evaluate. If this element is omitted, then the ***target*** for the `<Rule>` SHALL be defined by the enclosing `<Policy>` element. See Section 5.5 for details.

`<Condition>` [optional]

A ***predicate*** that MUST be satisfied for the ***rule*** to be assigned its `Effect` value. A ***condition*** is a boolean function over a combination of ***subject***, ***resource, action*** and ***environment attributes*** or other functions.

5.22. Simple type EffectType

The **EffectType** simple type defines the values allowed for the `Effect` attribute of the `<Rule>` element and for the `FulfillOn` attribute of the `<Obligation>` element.

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

5.23. Element <Condition>

The `<Condition>` element is a boolean function over ***subject***, ***resource***, ***action*** and ***environment attributes*** or functions of ***attributes***. If the `<Condition>` element evaluates to "True", then the enclosing `<Rule>` element is assigned its `Effect` value.

```
<xs:element name="Condition" type="xacml:ApplyType"/>
```

The `<Condition>` element is of **ApplyType** complex type.

5.24. Element <Apply>

The `<Apply>` element denotes application of a function to its arguments, thus encoding a function call. The `<Apply>` element can be applied to any combination of `<Apply>`, `<AttributeValue>`, `<SubjectAttributeDesignator>`, `<ResourceAttributeDesignator>`, `<ActionAttributeDesignator>`, `<EnvironmentAttributeDesignator>` and `<AttributeSelector>` arguments.

```
<xs:element name="Apply" type="xacml:ApplyType"/>
<xs:complexType name="ApplyType">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element ref=”xacml:Function”/>
    <xs:element ref="xacml:Apply"/>
    <xs:element ref="xacml:AttributeValue"/>
    <xs:element ref="xacml:SubjectAttributeDesignator"/>
    <xs:element ref="xacml:ResourceAttributeDesignator"/>
    <xs:element ref="xacml:ActionAttributeDesignator"/>
    <xs:element ref="xacml:EnvironmentAttributeDesignator"/>
    <xs:element ref=”xacml:SubjectAttributeIsPresent”/>
    <xs:element ref=”xacml:ResourceAttributeIsPresent”/>
    <xs:element ref=”xacml:ActionAttributeIsPresent”/>
    <xs:element ref=”xacml:EnvironmentAttributeIsPresent”/>
    <xs:element ref="xacml:AttributeSelector"/>
  </xs:choice>
  <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>
</xs:complexType>
```

The `<Apply>` element is of **ApplyType** complex type.

The `<Apply>` element contains the following attributes and elements:

`FunctionId` [Required]

The URN of a function. XACML-defined functions are described in Appendix A.

<Function> [Optional]

The name of a function that is applied to the elements of a ***bag***. See section A14.11.

`<Apply>` [Optional]

A nested function-call argument.

`<AttributeValue>` [Optional]

A literal value argument.

`<ResourceAttributeDesignator>` [Optional]

A ***resource attribute*** argument.

`<ActionAttributeDesignator>` [Optional]

An ***action attribute*** argument.

`<EnvironmentAttributeDesignator>` [Optional]

An ***environment attribute*** argument.

`<SubjectAttributeIsPresent>` [Optional]

An agrument that tests presence of the ***subject attribute***

`<ResourceAttributeIsPresent>` [Optional]

An argument that tests presence of the ***resource attribute***

`<ActionAttributeIsPresent>` [Optional]

An argument that tests presence of the ***action attribute***

`<EnvironmentAttributeIsPresent>` [Optional]

An argument that tests presence of the ***environment attribute***

`<AttributeSelector>` [Optional]

An ***attribute*** selector argument.

5.25. Element <Function>

The `Function` element SHALL be used to name a function that is applied by the higher-order ***bag*** functions to every element of a ***bag***. The higher-order ***bag*** functions are described in Section A14.11.

```
<xs:element name=”Function” type=”xacml:FunctionType”/>
<xs:complexType name=”FunctionType”>
  <xs:attribute name=”FunctionId” type=”xs:QName” use=”required”/>
</xs:complexType>
```

The `Function` element is of **FunctionType** complex type.

The `Function` element contains the following attributes:

`FunctionId` [Required]

The identifier for the function that is applied to the elements of a ***bag*** by the higher-order ***bag*** functions.

5.26. Complex type AttributeDesignatorType

The **AttributeDesignatorType** complex type is the type for elements and extensions that refer to named ***attributes***. A named ***attribute*** has specific criteria with which to match ***attributes*** within a specific part of the `<xacml-context:Request>` element. The **AttributeDesignatorType** complex type specifies the attributes used for the match criteria that are common to all named ***attributes***. Elements and extensions of the **AttributeDesignatorType** complex type MAY determine the presence of named ***attributes*** or select ***attribute values*** associated with ***attributes*** that match named ***attributes***. Elements and extensions of the **AttributeDesignatorType** SHALL NOT alter the match semantics of named ***attributes***, but MAY narrow the search space.

```
<xs:complexType name="AttributeDesignatorType">
   <xs:attribute name="AttributeId"   type="xs:anyURI"  use="required"/>
   <xs:attribute name="DataType"      type="xs:anyURI"  use="required"/>
   <xs:attribute name="Issuer"        type="xs:anyURI"  use="optional"/>
   <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"/>
</xs:complexType>
```

A named ***attribute*** SHALL match an ***attribute*** if the values of their respective `AttributeId`, `DataType` and `Issuer` attributes match. The `AttributeId` attribute MUST match, by URI equality, that of the `AttributeId` attribute of the ***attribute***. The DataType attribute MUST match, by URI equality, that of the `DataType` attribute of the same ***attribute***. If the `Issuer` attribute is supplied, it MUST match, by URI equality, the `Issuer` attribute of the same ***attribute***. If the `Issuer` attribute is not supplied, the matching of the ***attribute*** to the ***named attribute*** SHALL be governed by `AttributeId` and `DataType` attributes alone, regardless of the presence, absence, or actual value of the `Issuer` attribute.

The <`AttributeDesignatorType`> contains the following attributes:

`AttributeId` [Required]

This attribute SHALL specify the `AttributeId` with which to match the ***attribute***.

`DataType` [Required]

This attribute SHALL specify the data-type with which to match the ***attribute***.

`Issuer` [Optional]

This attribute, if supplied, SHALL specify the `Issuer` with which to match the ***attribute***.

`MustBePresent` [Optional]

This attribute governs whether the element returns “Indeterminate” in the case of the absence of the named ***attribute***. If the *named attribute* is absent and `MustBePresent` is set to “True”, then this element SHALL result in “Indeterminate”. If `MustBePresent` is not supplied, its default value SHALL be `false.`

5.27. Element <ResourceAttributeDesignator>

The `<ResourceAttributeDesignator>` element retrieves a ***bag*** of values for a *named resource attribute*. A ***resource attribute*** is an ***attribute*** that SHALL only be located within the `<Resource>` element of the `<xacml-context:Request>` element. A *named resource attribute* is a *named attribute* that matches a ***resource attribute***. A *named resource attribute* SHALL be considered *present* if there is at least one ***resource attribute*** that matches the criteria set out below. A ***resource attribute*** value is an ***attribute*** value that is contained within a ***resource attribute***.

The `<ResourceAttributeDesignator>` element SHALL return a ***bag*** of all the ***resource attribute*** values that are matched by the *named resource attribute*. The `MustBePresent` attribute governs whether this element returns an empty ***bag*** or “Indeterminate” in the case of the absence of the *named resource attribute*. If the *named resource attribute* is not present and the `MustBePresent` attribute is set to “False” (its default value) this element SHALL result in an empty ***bag***. If the *named resource attribute* is not present and the `MustBePresent` attribute is set to “True”, this element SHALL result in “Indeterminate”. Regardless of the `MustBePresent` attribute, if it cannot be determined whether the *named resource attribute* is present or not in the ***request context***, or the value of the *named resource attribute* is unavailable, then the expression SHALL evaluate to “Indeterminate”.

A *named resource attribute* SHALL match a ***resource attribute*** as per the match semantics specified in the **AttributeDesignatorType** complex type [Section 5.26]

The `<ResourceAttributeDesignator>` MAY appear in the `<ResourceMatch>` element and MAY be passed to the `<Apply>` element as an argument.

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

The `<ResourceAttributeDesignator>` element is of the **AttributeDesignatorType** complex type.

The `<ResourceAttributeDesignator>` element has the following attributes:

`AttributeId` [Required]

This attribute SHALL specify the `AttributeId` with which to match the ***resource attribute***.

`DataType` [Required]

This attribute SHALL specify the DataType with which to match the ***resource attribute***.

`Issuer` [Optional]

This attribute, if supplied, SHALL specify the `Issuer` with which to match the ***resource attribute***.

`MustBePresent` [Optional]

This attribute governs whether the `<ResourceAttributeDesignator>` element returns an empty ***bag*** or “Indeterminate” in the case of the absence of the *named resource attribute*. If the *named resource attribute* is absent and `MustBePresent` is set to “False”, then this element SHALL result in an empty ***bag***. If the *named resource attribute* is absent and `MustBePresent` is set to “True”, then this element SHALL evaluate to

"Indeterminate". If `MustBePresent` is not supplied, then its default value SHALL be "False".

5.28. Element <ActionAttributeDesignator>

The `<ActionAttributeDesignator>` element retrieves a bag of values for a ***named action attribute***. An ***action attribute*** is an ***attribute*** that SHALL only be located within the `<Action>` element of the `<xacml-context:Request>` element. A ***named action attribute*** has specific criteria (described below) with which to match an ***action attribute***. A ***named action attribute*** SHALL be considered *present*, i.e. *not absent*, if there is at least one ***action attribute*** that matches the criteria. A ***action attribute value*** is an ***attribute value*** that is contained within a ***action attribute***.

The `<ActionAttributeDesignator>` element SHALL return a bag of all the ***action attribute values*** that are matched by the ***named action attribute***. The `MustBePresent` attribute governs whether this element returns an empty bag or ***indeterminate*** in the case of the absence of the ***named action attribute***. If the ***named action attribute*** is not present and the `MustBePresent` attribute is set to `false` (its default value) this element SHALL result in an empty bag. If the ***named action attribute*** is not present and the `MustBePresent` attribute is set to `true`, this element SHALL result in ***indeterminate***. Regardless of the `MustBePresent` attribute, if it cannot be determined whether the ***named action attribute*** is present or not present in the request context, or the value of the ***named action attribute*** is unavailable, then the expression SHALL result in ***indeterminate***.

A ***named action attribute*** SHALL match a ***action attribute*** as per the match semantics specified in the **AttributeDesignatorType** complex type [Section 5.26].

The `<ActionAttributeDesignator>` MAY appear in the `<ActionMatch>` element and MAY be passed to the `<Apply>` element as an argument.

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

The `<ActionAttributeDesignator>` element is of the **AttributeDesignatorType** complex type.

The `<ActionAttributeDesignator>` element has the following attributes:

`AttributeId` [Required]

This attribute SHALL specify the `AttributeId` with which to match the ***action attribute***.

`DataType` [Required]

This attribute SHALL specify the DataType with which to match the ***action attribute***.

`Issuer` [Optional]

This attribute, if supplied, SHALL specify the `Issuer` with which to match the ***action attribute***.

`MustBePresent` [Optional]

This attribute governs the whether the `<ActionAttributeDesignator>` element returns an empty bag or ***indeterminate*** in the case of the absence of the ***named action attribute***. If the ***named action attribute*** is absent and `MustBePresent` is set to `false`, this element SHALL result in an empty bag. If the ***named action attribute*** is absent and `MustBePresent` is set to `true`, this element SHALL result in ***indeterminate***. If `MustBePresent` is not supplied, its default value SHALL be `false`.

5.29. Element <EnvironmentAttributeDesignator>

The `<EnvironmentAttributeDesignator>` element retrieves a bag of values for a ***named environment attribute***. A ***environment attribute*** is an ***attribute*** that SHALL only be located within the `<Environment>` element of the `<xacml-context:Request>` element. A ***named environment attribute*** has specific criteria (described below) with which to match a ***environment attribute***. A ***named environment attribute*** SHALL be considered *present*, i.e. *not absent*, if there is at least one ***environment attribute*** that matches the criteria. A ***environment attribute value*** is an ***attribute value*** that is contained within a ***environment attribute***.

The `<EnvironmentAttributeDesignator>` element SHALL return a bag of all the ***environment attribute values*** that are matched by the ***named environment attribute***. The `MustBePresent` attribute governs whether this element returns an empty bag or ***indeterminate*** in the case of the absence of the ***named environment attribute***. If the ***named environment attribute*** is not present and the `MustBePresent` attribute is set to `false` (its default value) this element SHALL result in an empty bag. If the ***named environment attribute*** is not present and the `MustBePresent` attribute is set to `true`, this element SHALL result in ***indeterminate***. Regardless of the `MustBePresent` attribute, if it cannot be determined whether the ***named environment attribute*** is present or not present in the request context, or the value of the ***named environment attribute*** is unavailable, then the expression SHALL result in ***indeterminate***.

A ***named environment attribute*** SHALL match a ***environment attribute*** as per the match semantics specified in the **AttributeDesignatorType** complex type [Section 5.26].

The `<EnvironmentAttributeDesignator>` MAY be passed to the `<Apply>` element as an argument.

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

The `<EnvironmentAttributeDesignator>` element is of the **AttributeDesignatorType** complex type.

The `<EnvironmentAttributeDesignator>` element has the following attributes:

`AttributeId` [Required]

This attribute SHALL specify the `AttributeId` with which to match the ***environment attribute***.

`DataType` [Required]

This attribute SHALL specify the DataType with which to match the ***environment attribute***.

`Issuer` [Optional]

This attribute, if supplied, SHALL specify the `Issuer` with which to match the ***environment attribute***.

`MustBePresent` [Optional]

This attribute governs the whether the `<EnvironmentAttributeDesignator>` element returns an empty bag or ***indeterminate*** in the case of the absence of the ***named environment attribute***. If the ***named environment attribute*** is absent and `MustBePresent` is set to `false`, this element SHALL result in an empty bag. If the ***named environment attribute*** is absent and `MustBePresent` is set to `true`, this element SHALL result in ***indeterminate***. If `MustBePresent` is not supplied, its default value SHALL be `false`.

5.30. Element <ResourceAttributeIsPresent>

The `<ResourceAttributeIsPresent>` element determines whether a ***named resource attribute*** is present. A ***resource attribute*** is an ***attribute*** that SHALL only be located within the `<Resource>` element of the `<xacml-context:Request>` element. A ***named resource attribute*** is a ***named attribute*** that matches a ***resource attribute***. A ***named resource attribute*** SHALL be considered *present*, i.e. *not absent*, if there is at least one ***resource attribute*** that matches the criteria described below.

The `<ResourceAttributeIsPresent>` element SHALL result in ***true*** if its ***named resource attribute*** is present. A result of ***true*** SHALL mean that a `<ResourceAttributeDesignator>` element for this ***named resource attribute*** SHALL return a bag consisting of at least one ***attribute value***. The `MustBePresent` attribute governs whether this element returns ***false*** or ***indeterminate*** in the case of the absence of the ***named resource attribute***. If the ***named resource attribute*** is not present and the `MustBePresent` attribute is set to `false` (its default value) this element SHALL result in ***false***. If the ***named resource attribute*** is not present and the `MustBePresent` attribute is set to `true`, this element SHALL result in ***indeterminate***. Regardless of the `MustBePresent` attribute, if it cannot be determined whether the ***named resource attribute*** is present or not present in the request context, or the value of the ***named resource attribute*** is unavailable, then the expression SHALL result in ***indeterminate***.

A ***named resource attribute*** SHALL be considered present if at least one ***resource attribute*** exists that matches the values of its corresponding `AttributeId`, `DataType`, and `Issuer` attributes as per the match semantics of the **AttributeDesignatorType** specification [Section **Error! Reference source not found.**].

The `<ResourceAttributeIsPresent>` MAY be passed to the `<Apply>` element as an argument.

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

The `<ResourceAttributeIsPresent>` element is of the **AttributeDesignatorType** complex type.

The `<ResourceAttributeIsPresent>` element has the following attributes:

`AttributeId` [Required]

This attribute SHALL specify the `AttributeId` with which to match the ***resource attribute***.

`DataType` [Required]

> This attribute SHALL specify the DataType with which to match the ***resource attribute***.

`Issuer` [Optional]

> This attribute, if supplied, SHALL specify the `Issuer` with which to match the ***resource attribute***.

`MustBePresent` [Optional]

> This attribute governs the whether the <ResourceAttributeIsPresent> element returns ***false*** or ***indeterminate*** in the case of the absence of the ***named resource attribute***. If the ***named resource attribute*** is absent and `MustBePresent` is set to `false`, this element SHALL result in ***false***. If the ***named resource attribute*** is absent and `MustBePresent` is set to `true`, this element SHALL result in ***indeterminate***. If `MustBePresent` is not supplied, its default value SHALL be `false`.

5.31. Element <ActionAttributeIsPresent>

The `<ActionAttributeIsPresent>` element determines whether a ***named action attribute*** is present. An ***action attribute*** is an ***attribute*** that SHALL only be located within the `<Action>` element of the `<xacml-context:Request>` element. A ***named action attribute*** is a ***named attribute*** that matches an ***action attribute***. A ***named action attribute*** SHALL be considered *present*, i.e. *not absent*, if there is at least one ***action attribute*** that matches the criteria below

The `<ActionAttributeIsPresent>` element SHALL result in ***true*** if its ***named action attribute*** is present. A result of ***true*** SHALL mean that a `<ActionAttributeDesignator>` element for this ***named action attribute*** SHALL return a bag consisting of at least one ***attribute value***. The `MustBePresent` attribute governs whether this element returns ***false*** or ***indeterminate*** in the case of the absence of the ***named action attribute***. If the ***named action attribute*** is not present and the `MustBePresent` attribute is set to `false` (its default value) this element SHALL result in ***false***. If the ***named action attribute*** is not present and the `MustBePresent` attribute is set to `true`, this element SHALL result in ***indeterminate***. Regardless of the `MustBePresent` attribute, if it cannot be determined whether the ***named action attribute*** is present or not present in the request context, or the value of the ***named action attribute*** is unavailable, then the expression SHALL result in ***indeterminate***.

A ***named action attribute*** SHALL be considered present if at least one ***action attribute*** exists that matches the values of its corresponding `AttributeId`, `DataType`, and `Issuer` attributes as per the match semantics of the **AttributeDesignatorType** specification [Section 5.26].

The `<ActionAttributeIsPresent>` MAY be passed to the `<Apply>` element as an argument.

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

The `<ActionAttributeIsPresent>` element is of the **AttributeDesignatorType** complex type.

The `<ActionAttributeIsPresent>` element has the following attributes:

`AttributeId` [Required]

This attribute SHALL specify the `AttributeId` with which to match the ***action attribute***.

`DataType` [Required]

This attribute SHALL specify the DataType with which to match the ***action attribute***.

`Issuer` [Optional]

This attribute, if supplied, SHALL specify the `Issuer` with which to match the ***action attribute***.

`MustBePresent` [Optional]

This attribute governs the whether the <ActionAttributeIsPresent> element returns ***false*** or ***indeterminate*** in the case of the absence of the ***named action attribute***. If the ***named action attribute*** is absent and `MustBePresent` is set to `false`, this element SHALL result in ***false***. If the ***named action attribute*** is absent and `MustBePresent` is set to `true`, this element SHALL result in ***indeterminate***. If `MustBePresent` is not supplied, its default value SHALL be `false`.

5.32. Element <EnvironmentAttributeIsPresent>

The `<EnvironmentAttributeIsPresent>` element determines whether a ***named environment attribute*** is present. An ***environment attribute*** is an ***attribute*** that SHALL only be located within the `<Environment>` element of the `<xacml-context:Request>` element. A ***named environment attribute*** is a ***named attribute*** that matches an ***environment attribute***. A ***named environment attribute*** SHALL be considered *present*, i.e. *not absent*, if there is at least one ***environment attribute*** that matches the criteria below.

The `<EnvironmentAttributeIsPresent>` element SHALL result in ***true*** if its ***named environment attribute*** is present. A result of ***true*** SHALL mean that a `<EnvironmentAttributeDesignator>` element for this ***named environment attribute*** SHALL return a bag consisting of at least one ***attribute value***. The `MustBePresent` attribute governs whether this element returns ***false*** or ***indeterminate*** in the case of the absence of the ***named environment attribute***. If the ***named environment attribute*** is not present and the `MustBePresent` attribute is set to `false` (its default value) this element SHALL result in ***false***. If the ***named environment attribute*** is not present and the `MustBePresent` attribute is set to `true`, this element SHALL result in ***indeterminate***. Regardless of the `MustBePresent` attribute, if it cannot be determined whether the ***named environment attribute*** is present or not present in the request context, or the value of the ***named environment attribute*** is unavailable, then the expression SHALL result in ***indeterminate***.

A ***named environment attribute*** SHALL be considered present if at least one ***environment attribute*** exists that matches the values of its corresponding `AttributeId`, `DataType`, and `Issuer` attributes as per the match semantics of the **AttributeDesignatorType** specification [Section 5.26].

The `<EnvironmentAttributeIsPresent>` MAY be passed to the `<Apply>` element as an argument.

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

The `<EnvironmentAttributeIsPresent>` element is of the **AttributeDesignatorType** complex type.

The `<EnvironmentAttributeIsPresent>` element has the following attributes:

`AttributeId` [Required]

> This attribute SHALL specify the `AttributeId` with which to match the ***environment attribute***.

`DataType` [Required]

> This attribute SHALL specify the DataType with which to match the ***environment attribute***.

`Issuer` [Optional]

> This attribute, if supplied, SHALL specify the `Issuer` with which to match the ***environment attribute***.

`MustBePresent` [Optional]

> This attribute governs the whether the <EnvironmentAttributeIsPresent> element returns ***false*** or ***indeterminate*** in the case of the absence of the ***named environment attribute***. If the ***named environment attribute*** is absent and `MustBePresent` is set to `false`, this element SHALL result in ***false***. If the ***named environment attribute*** is absent and `MustBePresent` is set to `true`, this element SHALL result in ***indeterminate***. If `MustBePresent` is not supplied, its default value SHALL be `false`.

5.33. Complex type SubjectAttributeDesignatorType

The **SubjectAttributeDesignatorType** complex type that extends the **AttributeDesignatorType** complex type. It is the base type for elements and extensions that refer to ***named categorized subject attributes***. A ***named categorized subject attribute*** is defined as follows:

A ***subject*** is represented by a `<Subject>` element of the <Subjects> element in the `<xacml-context:Request>` element. A ***categorized subject*** is a ***subject*** that contains a particular ***subject category attribute***. A ***subject attribute*** is an attribute located in a particular ***subject***. ***A named subject attribute*** is a ***named attribute*** for a ***subject***. A ***subject category attribute*** is the ***subject attribute*** that matches the ***named subject attribute*** with the `AttributeId` of "`urn:oasis:tc:xacml:1.0:subject:subject-category`" and the `DataType` of "`http://www.w3.org/2001/XMLSchema-instance#string`". A ***named categorized subject attribute*** is a ***named subject attribute*** for a particular ***categorized subject***.

The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType** with a `SubjectCategory` attribute. The **SubjectAttributeDesignatorType** extends the match semantics of the **AttributeDesignatorType** such that it narrows the attribute search space to the specific ***categorized subject*** such that the value of the `SubjectCategory` attribute matches by string equality the value of the subject's ***subject category attribute***.

If there are multiple ***subjects*** with the same ***subject category attribute***, then they SHALL be treated as if they were one ***categorized subject***.

Elements and extensions of the **SubjectAttributeDesignatorType** complex type determine the presence of select ***attribute values*** associated with ***named categorized subject attributes***. Elements and extensions of the **SubjectAttributeDesignatorType** SHALL NOT alter the match semantics of ***named categorized subject attributes***, but MAY narrow the search space.

```
<xs:complexType name="SubjectAttributeDesignatorType">
   <xs:complexContent>
     <xs:extension base="xacml: AttributeDesignatorType">
        <xs:attribute name="SubjectCategory"
                    type="xs:anyURI"
                    use="optional"
                    default=
             "urn:org:oasis:tc:xacml:1.0:subject-category:access-subject"/>
     </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

The <`SubjectAttributeDesignatorType`> complex type contains the following attribute in addition to the attributes of the **AttributeDesignatorType** complex type:

`SubjectCategory` [Optional]

This attribute SHALL specify the ***categorized subject*** from which to match ***named subject attributes***. If `SubjectCategory` is not supplied, its default value SHALL `urn:org:oasis:tc:xacml:1.0:subject-category`:access-subject.

5.34. Element <SubjectAttributeIsPresent>

The <SubjectAttributeIsPresent> element determines whether a ***named categorized subject attribute*** is present or not. Its match semantics are that of the **SubjectAttributeDesignatorType**.

The <`SubjectAttributeIsPresent`> element SHALL result in ***true*** if its ***named categorized subject attribute*** is present. A result of ***true*** SHALL mean that a <`SubjectAttributeDesignator`> element for the same ***named categorized subject attribute*** SHALL return a bag consisting of at least one ***attribute value***. The `MustBePresent` attribute governs whether this element returns ***false*** or ***indeterminate*** in the case of the absence of the ***named categorized subject attribute***. If the ***named categorized subject attribute*** is not present and the `MustBePresent` attribute is set to `false` (its default value) this element SHALL result in ***false***. If the ***named categorized subject attribute*** is not present and the `MustBePresent` attribute is set to `true`, this element SHALL result in ***indeterminate***. Regardless of the `MustBePresent` attribute, if it cannot be determined whether the ***named categorized subject attribute*** is present or not present in the request context, or the value of the ***named categorized subject attribute*** is unavailable, then the expression SHALL result in ***indeterminate***.

A ***named categorized subject attribute*** SHALL be considered present if at least one ***subject attribute*** exists that matches the values of its corresponding `AttributeId`, `DataType`, and `Issuer` attributes from the ***categorized subject*** as per the match semantics of the **SubjectAttributeDesignatorType** specification [Section 5.33]

The <`SubjectAttributeIsPresent`> MAY be passed to the <`Apply`> element as an argument.

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

The <`CategorizedAttributeIsPresent`> element has the following attributes:

`AttributeId` [Required]

This attribute SHALL specify the `AttributeId` with which to match the ***subject attribute*** of the ***categorized subject***.

`DataType` [Required]

This attribute SHALL specify the DataType with which to match the ***subject attribute*** of the ***categorized subject***.

`Issuer` [Optional]

This attribute, if supplied, SHALL specify the `Issuer` with which to match the ***subject attribute*** of the ***categorized subject***.

`MustBePresent` [Optional]

This attribute governs the whether the `<SubjectAttributeIsPresent>` element returns ***false*** or ***indeterminate*** in the case of the absence of the ***named categorized subject attribute***. If the ***named categorized subject attribute*** is absent and `MustBePresent` is set to `false`, this element SHALL result in ***false***. If the ***named categorized subject attribute*** is absent and `MustBePresent` is set to `true`, this element SHALL result in ***indeterminate***. If `MustBePresent` is not supplied, its default value SHALL be `false`.

5.35. Element <AttributeSelector>

The `AttributeSelector`'s `RequestContextPath` XML attribute SHALL contain a legal XPATH expression over the `<xacml-context:Request>` element. The `AttributeSelector` element evaluates to a bag of values of a single primitive type that is specified by the selector's `DataType` attribute. In the case where the XPath expression matches attributes in the request context by AttributeId, it must also match the attribute's DataType with the selector's `DataType`. In the case of using XPath 1.0, the value of the XPath expression is either a node-set, string value, numeric value, or boolean value. If the XPath 1.0 expression evaluates to a node-set, each node may consist of a string, numeric, boolean value, or a child node (i.e. structured node). In this case, each node is logically converted to string data by applying the "`string`" function defined in the XPath 1.0 specification, resulting in a sequence of string data. In the single string, numeric, or boolean value case, the value is converted to string data by applying the "`string`" function defined in the XPath 1.0 specification, resulting in a sequence of one string data element. In XPath 2.0, the result of the XPath expression is a sequence of items (where an item is an atomic value or a node) or the error value. When the error value is returned, the ***PDP*** SHOULD return "`Indeterminate`". Otherwise, each node is logically converted to a string using the `xf:string` accessor function, resulting in a sequence of string data. The resulting sequence of string data is converted to a bag of primitive values that is implied by the type system.

Support for the `<AttributeSelector>` element is OPTIONAL.

```
<xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"/>
<xs:complexType name="AttributeSelectorType">
   <xs:attribute name="RequestContextPath" type="xs:anyURI" use="required"/>
   <xs:attribute name=”DataType” type=”xs:anyURI” use=”required”/>
   <xs:attribute name=”MustBePresent” type=”xs:boolean” use=”optional”
default=”false”
</ xs:complexType>
```

The `<AttributeSelector>` element is of **AttributeSelectorType** complex type.

The `<AttributeSelector>` element has the following attributes:

`RequestContextPath` [Required]

An XPath expression into the request ***context***. There SHALL be no restriction on the XPath syntax.

`DataType` [Required]

The data type of the ***attribute***.

`MustBePresent` [Optional]

Whether or not designated ***attribute*** must be present in the ***context***.

5.36. Element <AttributeValue>

The `<AttributeValue>` element SHALL contain a literal ***attribute*** value.

```
   <xs:element name="AttributeValue" type="xacml:AttributeValueType"/>
   <xs:complexType name="AttributeValueType" mixed="true">
      <xs:sequence>
         <xs:any namespace="##any" processContents="lax" minOccurs="0" 
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>
```

The `<AttributeValue>` element is of **AttributeValueType** complex type.

The `<AttributeValue>` element has following attributes:

`DataType` [Required]

The data type of the ***attribute*** value.

5.37. Element <Obligations>

The `<Obligations>` element SHALL contain a set of `<Obligation>` elements.

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

The `<Obligations>` element is of **ObligationsType** complexType.

`<Obligation>` [One to Many]

A sequence of ***obligations***

5.38. Element <Obligation>

The `<Obligation>` element SHALL contain an identifier for the ***obligation*** and a set of attributes that form arguments of the action defined by the ***obligation***. The `FulfillOn` attribute SHALL indicate the ***effect*** for which this ***obligation*** applies.

```
   <xs:element name="Obligation" type="xacml:ObligationType"/>
   <xs:complexType name="ObligationType">
      <xs:sequence>
         <xs:element ref="xacml:AttributeAssignment" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
         <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>
      </xs:complexType>
```

The `<Obligation>` element is of **ObligationType** complexType. See Section 7.11 for a description of how the set of obligations to be returned by the PDP is determined.

The `ObligationId` [required]

> ***Obligation*** identifier. The value of the ***obligation*** identifier SHALL be interpreted by the ***PEP***.

`FulfillOn` [required]

> The ***effect*** for which this ***obligation*** applies.

`<AttributeAssignment>` [required]

> ***Obligation*** arguments assignment. The values of the ***obligation*** arguments SHALL be interpreted by the ***PEP***.

5.39. Element <AttributeAssignment>

The `<AttributeAssignment>` element SHALL contain an `AttributeId` and the corresponding ***attribute*** value. The `AttributeId` is part of ***attribute*** meta-data, and is used when the ***attribute*** cannot be referenced by its location in the `<xacml-context:Request>`. This situation may arise in an `<Obligation>` element if the ***obligation*** includes parameters.

```
   <xs:element name="AttributeAssignment" type="xacml:AttributeAssignmentType"/>
   <xs:complexType name="AttributeAssignmentType">
      <xs:complexContent>
         <xs:extension base="xacml:AttributeValueType">
            <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

The `<AttributeAssignment>` element is of **AttributeAssignmentType** complex type.

`AttributeId` [Required]

> The ***attribute*** Identifier

`DataType` [Required]

> The data type for the assigned value.

6. Context syntax (normative with the exception of the schema fragments)

6.1. Element <Request>

The `<Request>` element is a top-level element in the XACML ***context*** schema. The <Request> element is an abstraction layer used by the ***policy*** language. Any proprietary system using the XACML specification MUST transform its input into the form of an XACML ***context***`<Request>`.

The `<Request>` element consists of sections denoted by the `<Subject>`, `<Resource>`, `<Action>`, and `<Environment>` elements. There may be multiple `<Subject>` elements. Each section contains a sequence of XACML context `<Attribute>` elements associated with the ***subject***, ***resource***, ***action***, and ***environment*** respectively.

```
<xs:element name="Request" type="xacml-context:RequestType"/>
<xs:complexType name="RequestType">
   <xs:sequence>
      <xs:element ref="xacml-context:Subject" maxOccurs="unbounded"/>
      <xs:element ref="xacml-context:Resource"/>
      <xs:element ref="xacml-context:Action"/>
      <xs:element ref="xacml-context:Environment" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
```

The `<Request>` element is of **RequestType** complex type.

The `<Request>` element contains the following elements:

`<Subject>` [One to Many]

Specifies information about a ***subject*** of the request ***context*** by listing a sequence of `<Attribute>` elements associated with the ***subject***. One or more `<Subject>` elements are allowed. A ***subject*** is an entity associated with making the ***access*** request. One ***subject*** might be a human user that initiated the application from which the request is being issued. Another ***subject*** might be the application's executable code that issued this request. Another ***subject*** might be the machine on which the application is executing. Another ***subject*** might be the target entity that is to be the recipient of the resource. Attributes of each of these entities MUST be enclosed in a separate `<Subject>` element.

`<Resource>` [Required]

Specifies information about the ***resource*** for which access is being requested by listing a sequence of `<Attribute>` elements associated with the resource. It MAY

include a `<ResourceContent>` element.

`<Action>` [Required]

Specifies the requested ***action*** to be performed on the ***resource*** by listing a set of `<Attribute>` elements associated with the action.

`<Environment>` [Optional]

Contains a set of `<Attribute>` elements of the ***environment***. These `<Attribute>` elements MAY form a part of ***policy*** evaluation.

6.2. Element <Subject>

The `<Subject>` element specifies a ***subject*** of a ***decision request context*** by listing a sequence of `<Attribute>` elements associated with the ***subject***.

```
   <xs:element name="Subject" type="xacml-context:SubjectType"/>
   <xs:complexType name="SubjectType">
      <xs:sequence>
         <xs:element ref="xacml-context:Attribute" minOccurs="0" 
maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
```

The `<Subject>` element is of **SubjectType** complex type.

`<Attribute>` [Any Number]

A sequence of ***attributes*** that apply to the ***subject***.

Every `<Subject>` element MUST contain one and only one `<Attribute>` with AttributeId `“urn:oasis:names:tc:xacml:1.0:subject:subject-category”`. This ***attribute*** indicates a role that the parent `<Subject>` entity plays in making the ***access*** request. If this ***attribute*** is not present in a given `<Subject>` element, that `<Subject>` implicitly contains this ***attribute*** with the value of `“urn:oasis:names:tc:xacml:1.0:subject:subject-category:access-subject”`, indicating that the ***subject*** is the entity ultimately associated with initiating the ***access*** request. Typically, a `<Subject>` element will also contain an `<Attribute>` with AttributeId `“urn:oasis:names:tc:xacml:1.0:subject:subject-id”`, containing the identity of the ***subject*** entity.

No more than one `<Subject>` element may contain an `<Attribute>` with the given value for AttributeId `“urn:oasis:names:tc:xacml:1.0:subject:subject-category”`.

A `<Subject>` element MAY contain additional `<Attribute>` elements.

6.3. Element <Resource>

The `<Resource>` element specifies information about the ***resource*** for which access is being requested by listing a sequence of `<Attribute>` elements associated with the `resource`. It MAY include the ***resource*** content.

```
   <xs:element name="Resource" type="xacml-context:ResourceType"/>
   <xs:complexType name="ResourceType">
      <xs:sequence>
         <xs:element ref="xacml-context:ResourceContent" minOccurs="0"/>
         <xs:element ref="xacml-context:Attribute" minOccurs="0" 
maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
```

The `<Resource>` element is of **ResourceType** complex type.

The `<Resource>` element contains the following elements:

`<ResourceContent>` [Optional]

The ***resource*** content.

`<Attribute>` [Any Number]

A sequence of ***resource attributes***. The `<Resource>` element MUST contain one and only one `<Attribute>` with AttributeId "`urn:oasis:names:tc:xacml:1.0:resource:resource-id`". This ***attribute*** specifies the identity of the ***resource*** for which ***access*** is requested. The `<Resource>` element MAY contain additional `<Attribute>` elements.

6.4. Element <ResourceContent>

The `<ResourceContent>` element is a notional placeholder for the ***resource*** content. If an XACML ***policy*** references the contents of the ***resource***, then the `<ResourceContent>` element is used as the reference point.

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

The `<ResourceContent>` element is of **ResourceContentType** complex type.

The `<ResourceContent>` element allows arbitrary elements and attributes.

6.5. Element <Action>

The `<Action>` element specifies the requested ***action*** to be performed on the ***resource*** by listing a set of `<Attribute>` elements associated with the ***action***.

```
   <xs:element name="Action" type="xacml-context:ActionType"/>
   <xs:complexType name="ActionType">
      <xs:sequence>
         <xs:element ref="xacml-context:Attribute" minOccurs="0" 
maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
```

The `<Action>` element is of **ActionType** complex type.

The `<Attribute>` [Any Number]

List of ***attributes*** of the ***action*** to be performed on the ***resource***.

6.6. Element <Environment>

The `<Environment>` element contains a set of ***attributes*** of the ***environment***. These ***attributes*** MAY form part of the ***policy*** evaluation.

```
   <xs:element name="Environment" type="xacml-context:EnvironmentType"/>
   <xs:complexType name="EnvironmentType">
      <xs:sequence>
         <xs:element ref="xacml-context:Attribute" minOccurs="0" 
maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
```

The `<Environment>` element is of **EnvironmentType** complex type.

The `<Environment>` element contains the following elements:

`<Attribute>` [Any Number]

> A list of ***environment attributes***. Environment attributes are attributes that are not associated with the ***resource,*** the ***action***, or with any of the ***subjects*** of the ***access*** request.

6.7. Element <Attribute>

The `<Attribute>` element is the central abstraction of the request ***context***. It contains an ***attribute*** value and ***attribute*** meta-data. The ***attribute*** meta-data comprises the ***attribute*** identifier, the ***attribute*** issuer and the ***attribute*** issue instant. ***Attribute*** designators and ***attribute*** selectors in the ***policy*** refer to ***attributes*** by this meta-data.

```
<xs:element name="Attribute" type="xacml-context:AttributeType"/>
<xs:complexType name="AttributeType">
   <xs:sequence>
      <xs:element ref="xacml-context:AttributeValue" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
   <xs:attribute name="Issuer" type="xs:string" use="optional"/>
   <xs:attribute name="IssueInstant" type="xs:dateTime" use="optional"/>
</xs:complexType>
```

The `<Attribute>` element is of **AttributeType** complex type.

The `<Attribute>` element contains the following attributes and elements:

`AttributeId` [Required]

> ***Attribute*** identifier. A number of identifiers are reserved by XACML to denote commonly used ***attributes***.

`DataType` [Required]

> ***Attribute*** data type.

`Issuer` [Optional]

> ***Attribute*** issuer. This attribute value MAY be an x500Name that binds to a public key, or it may be some other identifier exchanged out-of-band by issuing and relying parties.

`IssueInstant` [Optional]

> The date and time at which the ***attribute*** was issued.

`<AttributeValue>` [Optional]

> At most one ***attribute*** value.

6.8. Element <AttributeValue>

The `<AttributeValue>` element contains the value of an ***attribute***.

```
<xs:element name="AttributeValue" type="xacml-context:AttributeValueType"/>
<xs:complexType name="AttributeValueType" mixed="true">
   <xs:sequence>
```

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

The `<AttributeValue>` element is of **AttributeValueType** type.

The data type of the `<AttributeValue>` MAY be specified by using the DataType attribute of the parent `<Attribute>` element.

6.9. Element <Response>

The `<Response>` element encapsulates the ***authorization decision*** returned by the ***PDP***. It includes a sequence of one or more results with one `<Result>` element per requested ***resource***. Multiple results MAY be returned when the value of the “urn:oasis:xacml:1.0:resource:scope” resource ***attribute*** in the request ***context*** is “Descendants”. Support for multiple results is OPTIONAL.

```
   <xs:element name="Response" type="xacml-context:ResponseType"/>
   <xs:complexType name="ResponseType">
      <xs:sequence>
         <xs:element ref="xacml-context:Result" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
```

The `<Response>` element is of **ResponseType** complex type.

The `<Response>` element contains the following elements:

`<Result>` [One to Many]

An authorization decision result.

6.10. Element <Result>

The `<Result>` element represents an ***authorization decision*** result for the ***resource*** specified by the `ResourceId` ***attribute***. It MAY include a set of ***obligations*** that MUST be fulfilled by the ***PEP***. If the ***PEP*** does not understand an ***obligation***, then it MUST act as if the ***PDP*** had denied ***access*** to the requested ***resource***.

```
   <xs:element name="Result" type="xacml-context:ResultType"/>
   <xs:complexType name="ResultType">
      <xs:sequence>
         <xs:element ref="xacml-context:Decision"/>
         <xs:element ref="xacml-context:Status" minOccurs="0"/>
         <xs:element ref="xacml:Obligations" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="ResourceId" type="xs:anyURI" use="optional"/>
   </xs:complexType>
```

The `<Result>` element is of **ResultType** complex type.

The `<Result>` element contains the following attributes and elements:

`ResourceId` [Optional]

The identifier of the requested ***resource***. If this attribute is omitted, then the ***resource*** identity is specified by the `“urn:oasis:names:tc:xacml:1.0:resource:resource-id”` ***resource attribute*** in the `<Request>` element.

`<Decision>` [Required]

The ***authorization decision***: “Permit”, “Deny”, “Indeterminate”, or “Not-applicable”.

`<Status>` [Optional]

Indicates whether errors occurred during evaluation of the request, and optionally, information about those errors.

`<xacml:Obligations>` [Optional]

A list of ***obligations*** that MUST be discharged by the ***PEP***. If the ***PEP*** does not understand an ***obligation***, then it MUST act as if the ***PDP*** had denied ***access*** to the requested ***resource***. See Section 7.11 for a description of how the set of obligations to be returned by the PDP is determined.

6.11. Element <Decision>

The `<Decision>` element contains the result of ***policy*** evaluation.

```
<xs:element name="Decision" type="xacml-context:DecisionType"/>
<xs:simpleType name="DecisionType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Permit"/>
    <xs:enumeration value="Deny"/>
    <xs:enumeration value="Indeterminate"/>
    <xs:enumeration value="Not-applicable"/>
  </xs:restriction>
</xs:simpleType>
```

The `<Decision>` element is of **DecisionType** simple type.

The values of the `<Decision>` element have the following meanings:

“Permit”: the requested resource access is permitted.

“Deny”: the requested resource access is denied.

“Indeterminate”: the ***PDP*** is unable to evaluate the requested ***resource access***. Reasons for such inability include: missing ***attributes***, network errors while retrieving policies, division by zero during policy evaluation, syntax errors in the request or in the policy.

“Not-applicable”: the ***PDP*** does not have any policy that applies to this request.

6.12. Element <Status>

The `<Status>` element represents the status of the ***authorization decision*** result.

```
<xs:element name="Status" type="xacml-context:StatusType"/>
<xs:complexType name="StatusType">
  <xs:sequence>
    <xs:element ref="xacml-context:StatusCode"/>
    <xs:element ref="xacml-context:StatusMessage" minOccurs="0"/>
    <xs:element ref="xacml-context:StatusDetail" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

The `<Status>` element is of **StatusType** complex type.

The `<Status>` element contains the following elements:

`<StatusCode>` [Required]

Status code.

`<StatusMessage>` [Optional]

A status message describing the status code.

`<StatusDetail>` [Optional]

Additional status information.

6.13. Element <StatusCode>

The `<StatusCode>` element contains a major status code value and an optional sequence of minor status codes.

```
  <xs:element name="StatusCode" type="xacml-context:StatusCodeType"/>
  <xs:complexType name="StatusCodeType">
    <xs:sequence>
      <xs:element ref="xacml-context:StatusCode" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Value" type="xs:QName" use="required"/>
  </xs:complexType>
```

The `<StatusCode>` element is of **StatusCodeType** complex type.

The `<StatusCode>` element contains the following attributes and elements:

`Value` [Required]

See Section B.7 for a list of values.

`<StatusCode>` [Any Number]

Minor status code. This status code qualifies its parent status code.

6.14. Element <StatusMessage>

The `<StatusMessage>` element is a free-form description of the status code.

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

The `<StatusMessage>` element is of **xs:string** type.

6.15. Element <StatusDetail>

The `<StatusDetail>` element qualifies the `<Status>` element with additional information.

```
  <xs:element name="StatusDetail" type="xacml-context:StatusDetailType"/>
  <xs:complexType name="StatusDetailType">
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

The `<StatusDetail>` element is of **StatusDetailType** complex type.

The `<StatusDetail>` element allows arbitrary xml content.

Inclusion of a `<StatusDetail>` element is optional. However, if a ***PDP*** returns one of the following XACML-defined `<StatusCode>` values and includes a `<StatusDetail>` element, then the following rules apply.

urn:oasis:names:tc:xacml:1.0:status:ok

A ***PDP*** MUST NOT return a `<StatusDetail>` element in conjunction with the “ok” status value.

urn:oasis:names:tc:xacml:1.0:status:missing-attribute

A ***PDP*** MAY choose not to return any `<StatusDetail>` information or MAY choose to return a `<StatusDetail>` element containing one or more `<xacml-context:Attribute>` elements. If the ***PDP*** includes `<AttributeValue>` elements in the `<Attribute>` element, then this indicates the acceptable values for that ***attribute***. If no `<AttributeValue>` elements are included, then this indicates the names of ***attributes*** that the ***PDP*** failed to resolve during its evaluation. The list of ***attributes*** may be partial or complete. There is no guarantee by the ***PDP*** that supplying the missing values or ***attributes*** will be sufficient to satisfy the ***policy***.

urn:oasis:names:tc:xacml:1.0:status:syntax-error

A ***PDP*** MUST NOT return a `<StatusDetail>` element in conjunction with the “syntax-error” status value. A syntax error may represent either a problem with the ***policy*** being used or with the request ***context***. The ***PDP*** MAY return a `<StatusMessage>` describing the problem.

urn:oasis:names:tc:xacml:1.0:status:processing-error

A ***PDP*** MUST NOT return `<StatusDetail>` element in conjunction with the “processing-error” status value. This status code indicates an internal problem in the ***PDP***. For security reasons, the ***PDP*** MAY choose to return no further information to the ***PEP***. In the case of a divide-by-zero error or other computational error, the ***PDP*** MAY return a `<StatusMessage>` describing the nature of the error.

7. Functional requirements (normative)

This section specifies certain functional requirements that are not directly associated with the production or consumption of a particular XACML element.

7.1. Policy enforcement point

This section describes the rquiremenst for the ***PEP***.

An application functions in the role of the ***PEP*** if it guards access to a set of ***resources*** and asks the ***PDP*** for an ***authorization decision***. The ***PEP*** MUST abide by the ***authorization decision*** in the following way:

A ***PEP*** SHALL allow access to the ***resource*** only if a valid XACML response of "Permit" is returned by the ***PDP***. The ***PEP*** SHALL deny access to the ***resource*** in all other cases. An XACML response of "Permit" SHALL be considered valid only if the ***PEP*** understands all of the ***obligations*** contained in the response.

7.2. Base policy

A ***PDP*** SHALL represent one ***policy*** or ***policy set***, called its *base policy*. This base ***policy*** MAY be a `<Policy>` element containing a `<Target>` element that matches every possible ***decision request***, or (for instance) it MAY be a `<Policy>` element containing a `<Target>` element that matches only a specific ***subject***. In such cases, the base policy SHALL form the root-node of a tree of policies connected by `<PolicyIdReference>` and `<PolicySetIdReference>` elements to all the ***rules*** that may be applicable to any ***decision request*** that the ***PDP*** is capable of evaluating.

In the case of a ***PDP*** that retrieves ***policies*** according to the ***decision request*** that it is processing, the base policy SHALL contain a `<Policy>` element containing a `<Target>` element that matches every possible ***decision request*** and a `PolicyCombiningAlgId` attribute with the value "Only-one-applicable". In other words, the ***PDP*** SHALL return an error if it retrieves policies that do not form a single tree.

7.3. Target evaluation

The ***target*** value SHALL be "Match" if the ***subjects, resource*** and ***action*** specified in the request ***context*** are all present in (i.e., within the scope of) the ***target***. Its value SHALL be "No-match" if one or more of the ***subjects, resource*** or ***action*** specified in the request ***context*** is not present in the ***target***. Its value SHALL be "Indeterminate" if any ***attribute*** value referenced in the ***target*** cannot be obtained.

7.4. Condition evaluation

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

7.5. 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 Value
"Match"	"True"	Effect
"Match"	"False"	"Not-applicable"
"Match"	"Indeterminate"	"Indeterminate"
Not "Match"	Don't care	"Not-applicable"
"Indeterminate"	Don't care	"Indeterminate"

Table 1 - Rule truth table

If the ***target*** value is "No-match" or "Indeterminate" then the ***rule*** value SHALL be "Not-applicable" or "Indeterminate", respectively, regardless of the value of the ***condition***. For these cases, therefore, the ***condition*** need not be evaluated in order to determine the ***rule*** value.

If the ***target*** value is “Match” and the ***condition*** value is “True”, then the ***effect*** specified in the ***rule*** SHALL determine the ***rule*** value.

7.6. Policy evaluation

A ***policy*** has a value that can be calculated by evaluating its contents. ***Policy*** evaluation involves separate evaluation of the ***policy's target*** and ***rules***. The ***policy*** truth table is shown in Table 2.

Target	Rule values	Policy Value
“Match”	At least one rule value is its Effect	Specified by the ***rule-combining algorithm***
“Match”	All rule values are “Not-applicable”	“Not-applicable”
“Match”	At least one rule value is “Indeterminate”	Specified by the ***rule-combining algorithm***
Not “Match”	Don’t-care	“Not-applicable”
“Indeterminate”	Don’t-care	“Indeterminate”

Table 2 - Rule truth table

A Rules value of "At-least-one-applicable" SHALL be used if the `<Rule>` element is absent, or if one or more of the ***rules*** contained in the ***policy*** is applicable to the ***decision request*** (i.e., returns a value of “Effect”; see Section 7.5). A value of “None-applicable” SHALL be used if no ***rule*** contained in the ***policy*** is applicable to the request and if no ***rule*** contained in the ***policy*** returns a value of “Indeterminate”. If no ***rule*** contained in the ***policy*** is applicable to the request but one or more ***rule*** returns a value of “Indeterminate”, then ***rules*** SHALL evaluate to "Indeterminate".

If the ***target*** value is "No-match" or “Indeterminate” then the ***policy*** value SHALL be “Not-applicable” or “Indeterminate”, respectively, regardless of the value of the ***rules***. For these cases, therefore, the ***rules*** need not be evaluated in order to determine the ***policy*** value.

If the ***target*** value is “Match” and the ***rules*** value is “At-least-one-applicable” or “Indeterminate”, then the ***rule-combining algorithm*** specified in the ***policy*** SHALL determine the ***policy*** value.

7.7. Policy Set evaluation

A ***policy set*** has a value that can be calculated by evaluating its contents. ***Policy set*** evaluation involves separate evaluation of the ***policy set's target*** and ***policies***. The ***policy set*** truth table is shown in Table 3.

Target	Policy values	Policy Set Value
Match	At least one policy value is its Effect	Specified by the ***policy-combining algorithm***
Match	All policy values are “Not-applicable”	“Not-applicable”
Match	At least one policy value is	Specified by the ***policy-combining algorithm***

	“Indeterminate”	
Not match	Don’t-care	“Not-applicable”
Indeterminate	Don’t-care	“Indeterminate”

Table 3 - Rule truth table

A ***policies*** value of "At-least-one-applicable" SHALL be used if there are no contained or referenced ***policies*** or ***policy sets***, or if one or more of the ***policies*** or ***policy sets*** contained in or referenced by the ***policy set*** is applicable to the ***decision request*** (i.e., returns a value determined by its ***rule-combining algorithm***; see Section 7.6). A value of “None-applicable” SHALL be used if no ***policy*** or ***policy set*** contained in or referenced by the ***policy set*** is applicable to the request and if no ***policy*** or ***policy set*** contained in or referenced by the ***policy set*** returns a value of “Indeterminate”. If no ***policy*** or ***policy set*** contained in or referenced by the ***policy set*** is applicable to the request but one or more ***policy*** or ***policy set*** returns a value of “Indeterminate”, then ***policies*** SHALL evaluate to "Indeterminate".

If the ***target*** value is "No-match" or “Indeterminate” then the ***policy set*** value SHALL be “Not-applicable” or “Indeterminate”, respectively, regardless of the value of the ***policies***. For these cases, therefore, the ***policies*** need not be evaluated in order to determine the ***policy set*** value.

If the ***target*** value is “Match” and the ***policies*** value is “At-least-one-applicable” or “Indeterminate”, then the ***policy-combining algorithm*** specified in the ***policy set*** SHALL determine the ***policy set*** value.

7.8. Hierarchical resources

It is often the case that a ***resource*** is organized as a hierarchy (e.g. file system, XML document). Some access requesters may request ***access*** to an entire subtree of a ***resource*** specified by a node. XACML allows the ***PEP*** (or ***context handler***) to specify whether the ***decision request*** is just for a single ***resource*** or for a subtree below the specified ***resource***. The latter is equivalent to repeating a single request for each node in the entire subtree. When a request ***context*** contains a resource attribute of type

"urn:oasis:names:tc:xacml:1.0:resource:scope"

with a value of "Immediate", or if it does not contain that ***attribute***, then the ***decision request*** SHALL be interpreted to apply to just the single ***resource*** specified by the `ResourceId` ***attribute***.

When the

"urn:oasis:names:tc:xacml:1.0:resource:scope"

attribute has the value "Children", the ***decision request*** SHALL be interpreted to apply to the specified ***resource*** and its immediate children ***resources***.

When the

"urn:oasis:names:tc:xacml:1.0:resource:scope"

attribute has the value "Descendants", the ***decision request*** SHALL be interpreted to apply to both the specified ***resource*** and all its descendant ***resources***.

In the case of "Children" and "Descendants", the ***authorization decision*** MAY include multiple results for the multiple sub-nodes in the ***resource*** sub-tree.

An XACML ***authorization response*** MAY contain multiple `<Result>` elements. In this case, the `<Status>` element SHOULD be included only in the first `<Result>` element (the remaining `<Result>` elements SHOULD NOT include the `<Status>` element).

Note that the method by which the ***PDP*** discovers whether the ***resource*** is hierarchically organized or not is outside the scope of XACML.

7.9. Attributes

Attributes are specified in the request ***context*** and are referred to in the ***policy*** by ***subject***, ***resource***, ***action*** and ***environment attribute*** designators and ***attribute*** selectors. A *named attribute* is the term used for the criteria that the specific ***subject***, ***resource***, ***action*** and ***environment attribute*** designators and selectors use to refer to ***attributes*** in the ***subject***, ***resource***, ***action*** and ***environment*** elements of the request ***context***, respectively.

7.9.1. Attribute Matching

A *named attribute* has specific criteria with which to match ***attributes*** within the ***context***. An ***attribute*** specifies `AttributeId`, `DataType` and `Issuer` attributes, and each *named attribute* also specifies `AttributeId`, `DataType` and `Issuer` attributes. A *named attribute* SHALL match an ***attribute*** if the values of their respective `AttributeId`, `DataType` and `Issuer` attributes match within their particular element, e.g. ***subject***, ***resource***, ***action*** or ***environment***, of the ***context***. The `AttributeId` attribute MUST match, by URI equality, that of the `AttributeId` attribute of the ***attribute***. The `DataType` attribute MUST match, by URI equality, that of the `DataType` attribute of the same ***attribute***. If the `Issuer` attribute is supplied, it MUST match, by URI equality, the `Issuer` attribute of the same ***attribute***. If the `Issuer` attribute is not supplied in the *named attribute*, then the matching of the ***attribute*** to the *named attribute* SHALL be governed by `AttributeId` and `DataType` attributes alone, regardless of the presence, absence, or actual value of the `Issuer` attribute. In the case of the ***attribute*** selector, the matching of the ***attribute*** to the ***named attribute*** SHALL be governed by the XPath expression, `DataType` and `Issuer` attributes.

7.9.2. Attribute Retrieval

The ***PDP*** SHALL request the values of ***attributes*** in the request ***context*** from the ***context handler***. The ***PDP*** SHALL reference the ***attributes*** as if they were in a physical request **context** document, but the ***context handler*** is responsible for obtaining and supplying the requested values. The ***context handler*** SHALL return the values of ***attributes*** that match the ***attribute*** designator or ***attribute*** selector and form them into a ***bag*** of values with the specified `DataType` attribute. If no ***attributes*** from the request ***context*** match, then the ***attribute*** SHALL be considered missing. If the ***attribute*** is missing, the `MustBePresent attribute` governs whether the ***attribute*** designator or ***attribute*** selector returns an empty ***bag*** or an ***indeterminate*** result. If `MustBePresent` is “False” (default value), then a missing attribute results in an empty ***bag***. If `MustBePresent` is “True”, then a missing ***attribute*** results in “Indeterminate”. This “Indeterminate” result SHALL be handled in accordance with the specification of the encompassing expressions, rules, policies, and policy sets. If the result is “Indeterminate”, then the `AttributeId`, `DataType` and `Issuer` of the ***attribute*** MAY be listed in the ***authorization decision*** as described in Section 7.10. However, a ***PDP*** MAY choose not to return such information for security reasons.

7.9.3. Environment Attributes

Environment attributes are listed in Section B.8. If a value for one of these ***attributes*** is supplied in the ***decision request***, then the ***context handler*** SHALL use that value. Otherwise, the ***context handler*** SHALL supply a value. For the date and time ***attributes***, the supplied value SHALL have the semantics of "date and time that apply to the ***decision request***".

7.9.4. Subject Attributes

The "subject-category" ***attribute*** is a *named attribute* with the criteria of an `AttributeId` of `"urn:oasis:names:tc:xacml:1.0:subject:subject-category"` and `DataType` attribute of `"http://www.w3.org/2001/XMLSchema#string"`. For each `<Subject>` element in the ***decision request***, if a value for the "subject-category" ***attribute*** is supplied, then the ***context handler*** SHALL use that value. Otherwise, the ***context handler*** SHALL supply the default value "`urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"`. If there is more than one "subject-category" ***attribute*** supplied in the ***decision request*** for any given `<Subject>` element, then the ***decision request*** is invalid.

7.10. Authorization decision

Given a valid XACML ***policy*** or ***policy set***, a compliant XACML ***PDP*** MUST evaluate the ***policy*** as specified in Sections 5, 6 and 4.2. The ***PDP*** MUST return a response ***context***, with one `<Decision>` element of value "Permit", "Deny", "Indeterminate" or "Not-applicable".

If the ***PDP*** cannot make a decision, then an "Indeterminate" `<Decision>` element contents SHALL be returned. The ***PDP*** MAY return a `<Decision>` element contents of "Indeterminate" with a status code of:

"urn:oasis:names:tc:xacml:1.0:missing-attribute",

signifying that more information is needed. In this case, the `<Status>` element MAY list the names and data-types of any ***attributes*** of the ***subjects*** and the ***resource*** that are needed by the ***PDP*** to refine its decision. A ***PEP*** MAY resubmit a refined request ***context*** in response to a `<Decision>` element contents of "Indeterminate" with a status code of

"urn:oasis:names:tc:xacml:1.0:missing-attribute",

by adding ***attribute*** values for the ***attribute*** names that were listed in the previous response. When the ***PDP*** returns a `<Decision>` element contents of "Indeterminate", with a status code of

"urn:oasis:names:tc:xacml:1.0:missing-attribute",

it MUST NOT list the names and data-types of any ***attribute*** of the ***subject*** or the ***resource*** for which values were supplied in the original request. Note, this requirement forces the ***PDP*** to eventually return an ***authorization decision*** of "Permit", "Deny" or "Indeterminate" with some other status code, in response to successively-refined requests.

7.11. Obligations

A ***policy*** or ***policy set*** may contain one or more ***obligations***. When such a ***policy*** or ***policy set*** is evaluated, an ***obligation*** SHALL be passed up to the next level of evaluation (the enclosing or referencing ***policy set*** or ***authorization decision***) only if the ***effect*** of the ***policy*** or ***policy set*** being evaluated matches the value of the `xacml:FulfillOn` attribute of the ***obligation***.

As a consequence of this procedure, no ***obligations*** SHALL be returned to the ***PEP*** if the ***policies*** or ***policy sets*** from which they are drawn are not evaluated, or if their evaluated result is "Indeterminate" or "Not-applicable", or if the ***decision*** resulting from evaluating the ***policy*** or ***policy set*** does not match the ***decision*** resulting from evaluating an enclosing ***policy set***.

If the ***PDP's*** evaluation is viewed as a tree of ***policy sets*** and ***policies***, each of which returns "Permit" or "Deny", then the set of ***obligations*** returned by the ***PDP*** to the ***PEP*** will include only the ***obligations*** associated with those paths where the ***effect*** at each level of evaluation is the same as the ***effect*** being returned by the ***PDP***.

A ***PEP*** that receives a valid XACML response of "Permit" with ***obligations*** SHALL be responsible for fulfilling *all* of those ***obligations***. A ***PEP*** that receives an XACML response of "Deny" with ***obligations*** SHALL be responsible for fulfilling all of the ***obligations*** that it *understands*.

8. XACML extensibility points (non-normative)

This section describes the points within the XACML model and schema where extensions can be added

8.1. Extensible XML attribute types

The following XML attributes have values that are URIs or QNames. These may be extended by the creation of new URIs or QNames associated with new semantics for these attributes.

`AttributeId,`

`AttributeValue,`

`DataType,`

`FunctionId,`

`MatchId,`

`ObligationId,`

`PolicyCombiningAlgId,`

`RuleCombiningAlgId,`

`StatusCode.`

See Section 5 for definitions of these attribute types.

8.2. Extensible XACML attribute types

The following XACML standard `AttributeIds` associated with the XACML standard element: `<Attribute>` have values that are URIs or QNames. These may be extended by the creation of new URIs or QNames associated with new semantics for these attributes.

urn:oasis:names:tc:xacml:1.0:subject:subject-category.

8.3. Structured attributes

An XACML `<AttributeValue>` element MAY contain an instance of a structured XML data-type. Section A.3 describes a number of standard techniques to identify data items within such a structured attribute. Listed here are some additional techniques that require XACML extensions.

1. For a given structured data type, a community of XACML users MAY define new attribute identifiers for each leaf sub-element of the structured data type that has a type conformant with one of the XACML-defined primitive data-types. Using these new attribute identifiers, the ***PEPs*** or ***context handlers*** used by that community of users can flatten instances of the structured data-type into a sequence of individual `<Attribute>` elements. Each such `<Attribute>` element can be compared using the XACML-defined functions. Using this method, the structured data type itself never appears in an `<AttributeValue>` element.

2. A community of XACML users MAY define a new function that can be used to compare a value of the structured data-type against some other value. This method may only be used by ***PDPs*** that support the new function.

9. Security and privacy considerations (non-normative)

This section identifies possible security and privacy compromise scenarios that should be considered when implementing an XACML-based system. The section is informative only. It is left to the implementer to decide whether these compromise scenarios are practical in their environment and to select appropriate safeguards.

9.1. Threat model

We assume here that the adversary has access to the communication channel between the XACML actors and is able to interpret, insert, delete and modify messages or parts of messages.

Additionally, an actor may use information from a former transaction maliciously in subsequent transactions. It is further assumed that ***rules*** and ***policies*** are only as reliable as the actors that create and use them. Thus it is incumbent on each actor to establish appropriate trust in the other actors upon which it relies. Mechanisms for trust establishment are outside the scope of this specification.

The messages that are transmitted between the actors in the XACML model are susceptible to attack by malicious third parties. Other points of vulnerability include the ***PEP***, the ***PDP*** and the ***PAP***. While some of these entities are not strictly within the scope of this specification, their compromise could lead to the compromise of ***access control*** enforced by the ***PEP***.

It should be noted that there are other components of a distributed system that may be compromised, such as an operating system and the domain-name system (DNS) that are outside the scope of this discussion of threat models. Compromise in these components may also lead to a policy violation.

The following sections detail specific compromise scenarios that may be relevant to an XACML system.

9.1.1. Unauthorized disclosure

XACML does not specify any inherent mechanisms for confidentiality of the messages exchanged between actors. Therefore, an adversary could observe the messages in transit. Under certain security policies, disclosure of this information is a violation. Disclosure of ***attributes*** or the types of ***decision requests*** that a ***subject*** submits may be a breach of privacy policy. In the commercial sector, the consequences of unauthorized disclosure of personal data may range from embarrassment to the custodian to imprisonment and large fines in the case of medical or financial data.

Unauthorized disclosure is addressed by confidentiality mechanisms.

9.1.2. Message replay

A message replay attack is one in which the adversary records and replays legitimate messages between XACML actors. This attack may lead to denial of service, the use of out-of-date information or impersonation.

Prevention of replay attacks requires the use of message freshness mechanisms.

Note that encryption of the message does not mitigate a replay attack since the message is just replayed and does not have to be understood by the adversary.

9.1.3. Message insertion

A message insertion attack is one in which the adversary inserts messages in the sequence of messages between XACML actors.

The solution to a message insertion attack is to use mutual authentication and a message sequence integrity mechanism between the actors. It should be noted that just using SSL mutual authentication is not sufficient. This only proves that the other party is the one identified by the subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate subject is authorized to send the message.

9.1.4. Message deletion

A message deletion attack is one in which the adversary deletes messages in the sequence of messages between XACML actors. Message deletion may lead to denial of service. However, a properly designed XACML system should not render an incorrect authorization decision as a result of a message deletion attack.

The solution to a message deletion attack is to use a message integrity mechanism between the actors.

9.1.5. Message modification

If an adversary can intercept a message and change its contents, then they may be able to alter an ***authorization decision.*** Message integrity mechanisms can prevent a successful message modification attack.

9.1.6. Not-applicable results

A result of "Not-applicable" means that the ***PDP*** did not have a policy whose target matched the information in the ***decision request***. In general, we highly recommend using a "default-deny" policy, so that when a ***PDP*** would have returned "Not-applicable", a result of "Deny" is returned instead.

In some security models, however, such as is common in many Web Servers, a result of "Not-applicable" is treated as equivalent to "Permit". There are particular security considerations that must be taken into account for this to be safe. These are explained in the following paragraphs.

If "Not-applicable" is to be treated as "Permit", it is vital that the matching algorithms used by the policy to match elements in the decision request are closely aligned with the data syntax used by the applications that will be submitting the decision request. A failure to match will be treated as "Permit", so an unintended failure to match may allow unintended access.

A common example of this is a Web Server. Commercial http responders allow a variety of syntaxes to be treated equivalently. The "%" can be used to represent characters by hex value. The URL path "/../" provides multiple ways of specifying the same value. Multiple character sets may be permitted and, in some cases, the same printed character can be represented by different binary values. Unless the matching algorithm used by the policy is sophisticated enough to catch these variations, unintended access may be permitted.

It is safe to treat "Not-applicable" as "Permit" only in a closed environment where all applications that formulate a decision request can be guaranteed to use the exact syntax expected by the policies used by the ***PDP***. In a more open environment, where decision requests may be received from applications that may use any legal syntax, it is strongly recommended that "Not-applicable" NOT be treated as "Permit" unless matching rules have been very carefully designed to match all possible applicable inputs, regardless of syntax or type variations.

9.1.7. Negative rules

A negative ***rule*** is one that is based on a ***predicate*** not being "True". If not used with care, negative ***rules*** can lead to policy violation, therefore some authorities recommend that they not be used. However, negative ***rules*** can be extremely efficient in certain cases, so XACML has chosen to include them. Nevertheless, it is recommended that they be used with care and avoided if possible.

A common use for negative ***rules*** is to deny ***access*** to an individual or subgroup when their membership in a larger group would otherwise permit them access. For example, we might want to write a ***rule*** that allows all Vice Presidents to see the unpublished financial data, except for Joe, who is only a Ceremonial Vice President and can be indiscreet in his communications. If we have complete control of the administration of ***subject attributes***, a superior approach would be to define “Vice President” and “Ceremonial Vice President” as distinct groups and then define ***rules*** accordingly. However, in some environments this approach may not be feasible. (It is worth noting in passing that, generally speaking, referring to individuals in ***rules*** does not scale well. Generally, shared ***attributes*** are preferred.)

If not used with care, negative ***rules*** can lead to policy violation in two common cases. They are: when ***attributes*** are suppressed and when the base group changes. An example of suppressed ***attributes*** would be if we have a policy that ***access*** should be permitted, *unless* the ***subject*** is a credit risk. If it is possible that the ***attribute*** of being a credit risk may be unknown to the ***PDP*** for some reason, then unauthorized ***access*** may be permitted. In some environments, the ***subject*** may be able to suppress the publication of ***attributes*** by the application of privacy controls, or the server or repository that contains the information may be unavailable for accidental or intentional reasons.

An example of a changing base group would be if there is a policy that everyone in the engineering department may change software source code, except for secretaries. Suppose now that the department was to merge with another engineering department and the intent is to maintain the same policy. However, the new department also includes individuals identified as administrative assistants, who ought to be treated in the same way as secretaries. Unless the policy is altered, they will unintentionally be permitted to change software source code. Problems of this type are easy to avoid when one individual administers all ***policies***, but when administration is distributed, as XACML allows, this type of situation must be explicitly guarded against.

9.2. Safeguards

9.2.1. Authentication

Authentication provides the means for 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.

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 an adversary could provide false or invalid ***authorization decisions***, leading to a policy violation.

It is equally important for a ***PDP*** to authenticate the identity of the ***PEP*** 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 an adversary were allowed to make unlimited requests to a ***PDP***.

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

9.2.2. Policy administration

If the contents of ***policies*** are exposed outside of the ***access control*** system, potential ***subjects*** may use this information to determine how to gain unauthorized ***access***.

To prevent this threat, the repository used for the storage of ***policies*** may itself require ***access control***. In addition, the `<Status>` element should be used to return values of missing ***attributes*** only when exposure of the identities of those ***attributes*** will not compromise security.

9.2.3. Confidentiality

Confidentiality mechanisms ensure 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. There are two areas in which confidentiality should be considered: one is confidentiality during transmission; the other is confidentiality within a `<Policy>` element.

9.2.3.1. Communication confidentiality

In some environments it is deemed good practice to treat all data within an ***access control*** system as confidential. In other environments, ***policies*** may be made freely available for distribution, inspection and audit. The idea behind keeping ***policy*** information secret is to make it more difficult for an adversary to know what steps might be sufficient to obtain unauthorized ***access***. Regardless of the approach chosen, the security of the ***access control*** system should not depend on the secrecy of the ***policy***.

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

Communications confidentiality can be provided by a confidentiality mechanism, such as SSL. Using a point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points is compromised.

9.2.3.2. Statement level confidentiality

In some cases, an implementation may want to encrypt only parts of an XACML `<Policy>` element.

The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used to encrypt all or parts of an XML document. This specification is recommended 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 ***PDP***, then a secure repository should be used to store this sensitive data.

9.2.4. Policy integrity

The XACML ***policy***, used by the ***PDP*** to evaluate the request ***context***, is the heart of the system. Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of the ***policy***. One is to ensure that `<Policy>` elements have not been altered since they were originally created by the ***PAP***. The other is to ensure that `<Policy>` elements have not been inserted or deleted from the set of ***policies***.

In many cases, both aspects can be achieved by ensuring the integrity of the actors and implementing session-level mechanisms to secure the communication between actors. The selection of the appropriate mechanisms is 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 the protected resource, it would be useful to sign the ***policy***. In these cases, the XML Signature Syntax and Processing standard from W3C is recommended to be used with XACML.

Digital signatures should only be used to ensure the integrity of the statements. Digital signatures should not be used as a method of selecting or evaluating ***policy***. That is, the ***PDP*** should not request a ***policy*** based on who signed it or whether or not it has been signed (as such a basis for selection would, itself, be a matter of policy). However, the ***PDP*** must verify that the key used to sign the ***policy*** is one controlled by the purported issuer of the ***policy***. The means to do this are dependent on the specific signature technology chosen and are outside the scope of this document.

9.2.5. Policy identifiers

Since ***policies*** can be referenced by their identifiers, it is the responsibility of the ***PAP*** to ensure that these are unique. Confusion between identifiers could lead to misidentification of the ***applicable policy***. This specification is silent on whether a ***PAP*** must generate a new identifier when a ***policy*** is modified or may use the same identifier in the modified ***policy***. This is a matter of administrative practice. However, care must be taken in either case. If the identifier is reused, there is a danger that other ***policies*** or ***policy sets*** that reference it may be adversely affected. Conversely, if a new identifier is used, these other ***policies*** may continue to use the prior ***policy***, unless it is deleted. In either case the results may not be what the ***policy*** administrator intends.

9.2.6. Trust model

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

It is worth considering the relationships between the various actors of the ***access control*** system in terms of the interdependencies that do and do not exist.

- None of the entities of the authorization system are dependent on the ***PEP***. They may collect data from it, for example authentication, but are responsible for verifying it.
- The correct operation of the system depends on the ability of the ***PEP*** to actually enforce ***policy*** decisions.
- The ***PEP*** depends on the ***PDP*** to correctly evaluate ***policies***. This in turn implies that the ***PDP*** is supplied with the correct inputs. Other than that, the ***PDP*** does not depend on the ***PEP.***
- The ***PDP*** depends on the ***PAP*** to supply appropriate policies. The ***PAP*** is not dependent on other components.

9.2.7. Privacy

It is important to be aware that any transactions that occur with respect to ***access control*** may reveal private information about the actors. For example, if an XACML ***policy*** states that certain data may only be read by ***subjects*** with “Gold Card Member” status, then any transaction in which a ***subject*** is permitted ***access*** to that data leaks information to an adversary about the ***subject's*** status. Privacy considerations may therefore lead to encryption and/or to ***access control policies*** surrounding the enforcement of XACML ***policy*** instances themselves: confidentiality-protected channels for the request/response protocol messages, protection of ***subject attributes*** in storage and in transit, and so on.

Selection and use of privacy mechanisms appropriate to 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.

10. Conformance (normative)

10.1. Introduction

The XACML specification addresses two aspects of conformance:

1.The OASIS procedure for ratification of a committee specification as an OASIS standard requires that three independent implementers attest that they are "successfully using" the committee specification, and

2. The XACML specification defines a number of functions, etc. that have somewhat specialist application, therefore they are not required to be implemented in an implementation that claims to conform with the OASIS standard.

10.2. Attestation

An implementer MAY attest to be "successfully using" the XACML committee specification provided the implementation successfully executes a set of test-cases. The test cases are hosted by Sun Microsystems and can be located from the XACML web page. The site hosting the test cases contains a full description of the test cases and how to execute them.

10.3. Conformance tables

This section lists those portions of the specification that MUST be included in an implementation of a ***PDP*** that claims to conform with XACML v1.0.

Note: "M" means mandatory-to-implement. "O" means optional.

10.3.1. Schema elements

The implementation MUST support those schema elements that are marked “M”.

Namespace	Element	M/O
xacml:Policy	Action	M
xacml:Policy	ActionAttributeDesignator	M
xacml:Policy	ActionMatch	M
xacml:Policy	Actions	M
xacml:Policy	AnyAction	M
xacml:Policy	AnyResource	M
xacml:Policy	AnySubject	M
xacml:Policy	Apply	M
xacml:Policy	AttributeAssignment	O
xacml:Policy	AttributeSelector	O
xacml:Policy	AttributeValue	M
xacml:Policy	Condition	M
xacml:Policy	Description	M
xacml:Policy	EnvironmentAttributeDesignator	M
Xacml:Policy	Function	M
xacml:Policy	Obligation	O
xacml:Policy	Obligations	O
xacml:Policy	Policy	M
xacml:Policy	PolicyDefaults	O
xacml:Policy	PolicyIdReference	M
xacml:Policy	PolicySet	M
xacml:Policy	PolicySetDefaults	O
xacml:Policy	PolicySetIdReference	M
xacml:Policy	Resource	M
xacml:Policy	ResourceAttributeDesignator	M
xacml:Policy	ResourceMatch	M
xacml:Policy	Resources	M
xacml:Policy	Rule	M
xacml:Policy	Subject	M
xacml:Policy	SubjectAttributeDesignator	M
xacml:Policy	QualifiedSubjectAttributeDesignator	M
xacml:Policy	SubjectMatch	M
xacml:Policy	Subjects	M
xacml:Policy	Target	M
xacml:Policy	XPathVersion	O
xacml:Context	Action	M

xacml:Context	Attribute	M
xacml:Context	AttributeValue	M
xacml:Context	Decision	M
xacml:Context	Environment	M
xacml:Context	Obligations	O
xacml:Context	Request	M
xacml:Context	Resource	M
xacml:Context	ResourceContent	O
xacml:Context	Response	M
xacml:Context	Result	M
xacml:Context	Status	O
xacml:Context	StatusCode	O
xacml:Context	StatusDetail	O
xacml:Context	StatusMessage	O
xacml:Context	Subject	M

10.3.2. Identifier Prefixes

The following identifier prefixes are reserved by XACML.

Identifier
urn:oasis:names:tc:xacml:1.0
urn:oasis:names:tc:xacml:1.0:conformance-test
urn:oasis:names:tc:xacml:1.0:context
urn:oasis:names:tc:xacml:1.0:example
urn:oasis:names:tc:xacml:1.0:function
urn:oasis:names:tc:xacml:1.0:policy
urn:oasis:names:tc:xacml:1.0:subject
urn:oasis:names:tc:xacml:1.0:resource
urn:oasis:names:tc:xacml:1.0:action

10.3.3. Algorithms

The implementation MUST include the rule- and policy-combining algorithms associated with the following identifiers that are marked "M".

Algorithm	M/O
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides	M
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides	M
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides	M
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides	M
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable	M
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable	M
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable	M

10.3.4. Status Codes

Implementation support for the urn:oasis:names:tc:xacml:1.0:context:status element is optional, but if the element is supported, then the following status codes must be supported and must be used in the way XACML has specified.

Identifier	M/O
urn:oasis:names:tc:xacml:1.0:status:missing-attribute	M
urn:oasis:names:tc:xacml:1.0:status:ok	M
urn:oasis:names:tc:xacml:1.0:status:processing-error	M
urn:oasis:names:tc:xacml:1.0:status:syntax-error	M

10.3.5. Attributes

The implementation MUST support the attributes associated with the following attribute identifiers as specified by XACML. The value for these attributes MUST be provided by the ***PDP***, so, unlike most other attributes, their semantics are not transparent to the ***PDP*** implementation.

Identifier	M/O
urn:oasis:names:tc:xacml:1.0:environment:current-time	M
urn:oasis:names:tc:xacml:1.0:environment:current-date	M
urn:oasis:names:tc:xacml:1.0:environment:current-dateTime	M
urn:oasis:names:tc:xacml:1.0:subject:subject-category	M

10.3.6. Identifiers

The implementation MUST use the attributes associated with the following identifiers in the way XACML has defined. This requirement pertains primarily to implementations of a ***PAP*** or ***PEP*** that use XACML, since the semantics of the attributes are transparent to the ***PDP***.

Identifier	M/O
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name	O
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address	O
urn:oasis:names:tc:xacml:1.0:resource:resource-location	O
urn:oasis:names:tc:xacml:1.0:resource:resource-id	O
urn:oasis:names:tc:xacml:1.0:resource:scope	O
urn:oasis:names:tc:xacml:1.0:resource:simple-file-name	O
urn:oasis:names:tc:xacml:1.0::action:action-id	M
urn:oasis:names:tc:xacml:1.0::action:implied-action	M
urn:oasis:names:tc:xacml:1.0:subject:authentication-method	O
urn:oasis:names:tc:xacml:1.0:subject:authentication-time	O
urn:oasis:names:tc:xacml:1.0:subject:key-info	O
urn:oasis:names:tc:xacml:1.0:subject:request-time	O
urn:oasis:names:tc:xacml:1.0:subject:session-start-time	O
urn:oasis:names:tc:xacml:1.0:subject:subject-id	O
urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier	O
urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	M
urn:oasis:names:tc:xacml:1.0:subject-category:codebase	O
urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject	O
urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject	O
urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine	O

10.3.7. Data Types

The implementation MUST support the data types associated with the following identifiers marked "M".

Data-type	M/O
http://www.w3.org/2001/XMLSchema#string	M
http://www.w3.org/2001/XMLSchema#boolean	M
http://www.w3.org/2001/XMLSchema#integer	M
http://www.w3.org/2001/XMLSchema#double	M
http://www.w3.org/2001/XMLSchema#date	M

http://www.w3.org/2001/XMLSchema#dateTime	M
http://www.w3.org/2001/XMLSchema#anyURI	M
http://www.w3.org/2001/XMLSchema#hexBinary	M
http://www.w3.org/2001/XMLSchema#base64Binary	M
http://www.w3.org/TR/xquey-operaqtors:dayTimeDuration	M
http://www.w3.org/TR/xquey-operaqtors:yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name	M
urn:oasis:names:tc:xacml:1.0:data-type:x500Name	M

10.3.8. Functions

The implementation MUST properly process those functions associated with the identifiers marked with an "M".

xmlns:function=”urn:oasis:names:tc:xacml:1.0:function”

Function	M/O
function:string-equal	M
function:boolean-equal	M
function:integer-equal	M
function:double-equal	M
function:date-equal	M
function:time-equal	M
function:dateTime-equal	M
function:anyURI-equal	M
function:x500Name-equal	M
function:rfc822name-equal	M
function:hexBinary-equal	M
function:base64Binary-equal	M
function:integer-add	M
function:double-add	M
function:integer-subtract	M
function:double-subtract	M
function:integer-multiply	M
function:double-multiply	M
function:integer-divide	M
function:double-divide	M
function:integer-mod	M
function:integer-abs	M
function:double-abs	M
function:round	M
function:floor	M
function:string-normalize-space	M
function:string-normalize-to-lower-case	M
function:double-to-integer	M
function:integer-to-double	M
function:or	M
function:and	M
function:n-of	M
function:not	M
function:present	M
function:integer-greater-than	M
function:integer-greater-than-or-equal	M
function:integer-less-than	M
function:integer-less-than-or-equal	M
function:double-greater-than	M
function:double-greater-than-or-equal	M

function:double-less-than	M
function:double-less-than-or-equal	M
function:dateTime-add-dayTimeDuration	M
function:dateTime-add-yearMonthDuration	M
function:dateTime-subtract-dayTimeDuration	M
function:dateTime-subtract-yearMonthDuration	M
function:date-add-yearMonthDuration	M
function:date-subtract-yearMonthDuration	M
function:string-greater-than	M
function:string-greater-than-or-equal	M
function:string-less-than	M
function:string-less-than-or-equal	M
function:time-greater-than	M
function:time-greater-than-or-equal	M
function:time-less-than	M
function:time-less-than-or-equal	M
function:dateTime-greater-than	M
function:dateTime-greater-than-or-equal	M
function:dateTime-less-than	M
function:dateTime-less-than-or-equal	M
function:date-greater-than	M
function:date-greater-than-or-equal	M
function:date-less-than	M
function:date-less-than-or-equal	M
function:string-one-and-only	M
function:string-bag-size	M
function:string-is-in	M
function:string-bag	M
function:boolean-one-and-only	M
function:boolean-bag-size	M
function:boolean-is-in	M
function:boolean-bag	M
function:integer-one-and-only	M
function:integer-bag-size	M
function:integer-is-in	M
function:integer-bag	M
function:double-one-and-only	M
function:double-bag-size	M
function:double-is-in	M
function:double-bag	M
function:date-one-and-only	M
function:date-bag-size	M
function:date-is-in	M
function:date-bag	M
function:dateTime-one-and-only	M
function:dateTime-bag-size	M
function:dateTime-is-in	M
function:dateTime-bag	M
function:anyURI-one-and-only	M
function:anyURI-bag-size	M
function:anyURI-is-in	M
function:anyURI-bag	M
function:hexBinary-one-and-only	M
function:hexBinary-bag-size	M
function:hexBinary-is-in	M
function:hexBinary-bag	M
function:base64Binary-one-and-only	M

function:base64Binary-bag-size	M
function:base64Binary-is-in	M
function:base64Binary-bag	M
function:dayTimeDuration-one-and-only	M
function:dayTimeDuration-bag-size	M
function:dayTimeDuration-is-in	M
function:dayTimeDuration-bag	M
function:yearMonthDuration-one-and-only	M
function:yearMonthDuration-bag-size	M
function:yearMonthDuration-is-in	M
function:yearMonthDuration-bag	M
function:x500Name-one-and-only	M
function:x500Name-bag-size	M
function:x500Name-is-in	M
function:x500Name-bag	M
function:rfc822Name-one-and-only	M
function:rfc822Name-bag-size	M
function:rfc822Name-is-in	M
function:rfc822Name-bag	M
function:any-of	M
function:all-of	M
function:any-of-any	M
function:all-of-any	M
function:any-of-all	M
function:all-of-all	M
function:map	M
function:x500Name-match	M
function:rfc822Name-match	M
function:xpath-node-count	O
function:xpath-node-equal	O
function:xpath-node-match	O

11. References

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

[Haskell] Haskell, a purely functional language. Available at **http://www.haskell.org/**

[Hinton94] Hinton, H, M, Lee,, E, S, The Compatibility of Policies, Proceedings 2nd ACM Conference on Computer and Communications Security, Nov 1994, Fairfax, Virginia, USA.

[IEEE754] IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-7653-8, IEEE Product No. SH10116-TBR

[Kudo00] Kudo M and Hada S, XML document security based on provisional authorization, Proceedings of the Seventh ACM Conference on Computer and Communications Security, Nov 2000, Athens, Greece, pp 87-96.

[LDAP-1] RFC2256, A summary of the X500(96) User Schema for use with LDAPv3, section 5, M Wahl, December 1997 **http://www.ietf.org/rfc/rfc2798.txt**

[LDAP-2] RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000 **http://www.ietf.org/rfc/rfc2798.txt**

[MathML] Mathematical Markup Language (MathML), Version 2.0, W3C Recommendation, 21 February 2001. Available at: **http://www.w3.org/TR/MathML2/**

[Perritt93]	Perritt, H. Knowbots, Permissions Headers and Contract Law, Conference on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment, April 1993. Available at: **http://www.ifla.org/documents/infopol/copyright/perh2.txt**
[RBAC]	Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th National Computer Security Conference, 1992. Available at: **http://csrc.nist.gov/rbac**
[RegEx]	XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, Appendix D. Available at: **http://www.w3.org/TR/xmlschema-0/**
[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
[SAML]	Security Assertion Markup Language available from **http://www.oasis-open.org/committees/security/#documents**
[Sloman94]	Sloman, M. Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.
[XF]	XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft 16 August 2002. Available at: **http://www.w3.org/TR/xquery-operators**
[XS]	XML Schema. Available at: **http:/www.w3.org/TR/2001/REC-xmlschema-2-20010502/**
[XPath]	XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999. Available at: **http://www.w3.org/TR/xpath**
[XSLT]	XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999. Available at: **http://www.w3.org/TR/xslt**

Appendix A. Standard data types, functions and their semantics (normative)

A.1. Introduction

This section contains a specification of the data-types and functions used in XACML to create ***predicates*** for a ***rule's condition*** and ***target*** matches.

This specification combines the various standards set forth by IEEE and ANSI for string representation of numeric values, as well as the evaluation of arithmetic functions.

This section describes the primitive data-types, ***bags*** and construction of expressions using XACML constructs. Finally, each standard function is named and its operational semantics are described.

A.2. Primitive types

Although XML instances represent all data-types as strings, an XACML ***PDP*** must reason about types of data that, while they have string representations, are not just strings. Types such as boolean, integer and double MUST be converted from their XML string representations to values that can be compared with values in their domain of discourse, such as numbers. The following primitive data-types are specified for use with XACML and have explicit data representations:

- http://www.w3.org/2001/XMLSchema#string
- http://www.w3.org/2001/XMLSchema#boolean
- http://www.w3.org/2001/XMLSchema#integer
- http://www.w3.org/2001/XMLSchema#double
- http://www.w3.org/2001/XMLSchema#date
- http://www.w3.org/2001/XMLSchema#dateTime
- http://www.w3.org/2001/XMLSchema#anyURI
- http://www.w3.org/2001/XMLSchema#hexBinary
- http://www.w3.org/2001/XMLSchema#base64Binary
- http://www.w3.org/TR/xquery-operators#dayTimeDuration
- http://www.w3.org/TR/xquery-operators#yearMonthDuration
- urn:oasis:names:tc:xacml:1.0:data-type:x500Name
- urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name

A.3. Structured types

An XACML `<AttributeValue>` element MAY contain an instance of a structured XML data-type, for example `<ds:KeyInfo>`. XACML 1.0 supports several ways for comparing such `<AttributeValue>` elements.

1. In some cases, such an `<AttributeValue>` element MAY be compared using one of the XACML string functions, such as “regexp-string-match”, described below. This requires that the structured data <AttributeValue> to be given the DataType="xsi:string". For example, a structured data type that is actually a ds:KeyInfo/KeyName would appear in the Context as:

```
<AttributeValue
 DataType="DataType="http://www.w3.org/2001/XMLSchema-instance#string">&lt;ds:KeyName&gt;jhibbert-key&lt;/ds:KeyName&gt;
</AttributeValue>
```

   In general, this method will not be adequate unless the structured data type is quite simple.

2. An `<AttributeSelector>` element MAY be used to select the value of a leaf sub-element of the structured data-type by means of an XPath expression. That value MAY then be compared using one of the supported XACML functions appropriate for its primitive data-type. This method requires support by the PDP for the optional XPath expressions feature.

3. An `<AttributeSelector>` element MAY be used to select the value of any node in the structured type by means of an XPath expression. This node MAY then be compared using one of the XPath-based functions described in Section A14.13. This method requires support by the PDP for the optional XPath expressions and XPath functions features.

A.4. Representations

An XACML ***PDP*** SHALL be capable of converting string representations into various primitive data types. For integers and doubles, XACML SHALL use the conversions described in IBM Standard Decimal Arithmetic [IBMDSA].

This document combines the various standards set forth by IEEE and ANSI for string representation of numeric values.

XACML defines two additional data-types; these are “urn:oasis:names:tc:xacml:1.0:data-type:x500Name” and “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name”. These types represent identifiers for ***subjects*** and appear in several standard applications, such as TLS/SSL and electronic mail.

The “urn:oasis:names:tc:xacml:1.0:data-type:x500Name” primitive type represents an X.500 Distinguished Name. The string representation of an X.500 distinguished name is specified in IETF RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names".[1]

1 An earlier RFC, RFC 1779 "A String Representation of Distinguished Names", is less restrictive, so xacml:x500Name uses the syntax in RFC 2253 for better interoperability.

The “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name” primitive type represents electronic mail addresses, and its string representation is specified by RFC 822.

An RFC822 name consists of a *local-part* followed by "@" followed by a *domain-part*. The *local-part* is case-sensitive, while the *domain-part* (which is usually a DNS host name) is not case-sensitive.[2]

A.5. Bags

XACML defines implicit collections of its primitive types. XACML refers to a collection of values that are of a single primitive type as a ***bag***. ***Bags*** of primitive types are needed because selections of nodes from an XML ***resource*** or XACML request ***context*** may return more than one value.

The `<AttributeSelector>` element uses an XPath expression to specify the selection of data from an XML ***resource***. The result of an XPath expression is termed a *node-set*, which contains all the leaf nodes from the XML ***resource*** that match the predicate in the XPath expression. Based on the various indexing functions provided in the XPath specification, it SHALL be implied that a resultant node-set is the collection of the matching nodes. XACML also defines the `<AttributeDesignator>` **element** to have the same matching methodology for attributes in the XACML request ***context***.

The values in a ***bag*** are not ordered, and some of the values may be duplicates. There SHALL be no notion of a ***bag*** containing ***bags***, or a ***bag*** containing values of differing types. I.e. a ***bag*** in XACML SHALL contain only values that are of the same primitive type.

A.6. Expressions

XACML specifies expressions in terms of the following elements. Each expression evaluates to one of the primitive types, or a ***bag*** of one of the primitive types. In addition, XACML defines an evaluation result of "Indeterminate", which is said to be the result of an invalid expression, or an operational error occurring during the evaluation of the expression.

XACML defines the following elements to be legal XACML expressions:

- `<AttributeValue>`
- `<SubjectAttributeDesignator>`
- `<SubjectAttributeSelector>`
- `<QualifiedSubjectAttributeDesignator>`
- `<ResourceAttributeDesignator>`
- `<ActionAttributeDesignator>`
- `<EnvironmentAttributeDesignator>`

2 According to IETF RFC822 and its successor specifications [RFC2821], case is significant in the *local-part.* However, many mail systems, as well as the IETF PKIX specification, treat the *local-part* as case-insensitive. This is considered an error by mail-system designers and is not encouraged.

- <AttributeSelector>
- <Apply>
- <Condition>
- <Function>

A.7. Element <AttributeValue>

The <AttributeValue> element SHALL represent an explicit value of a primitive type. For example:

```
<Apply FunctionId=”function:integer-equal”>
   <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>123</AttributeValue>
   <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>123</AttributeValue>
</Apply>
```

A.8. Elements <AttributeDesignator> and <AttributeSelector>

The <AttributeDesignator> and <AttributeSelector> elements SHALL evaluate to a ***bag*** of a specific primitive type. The type SHALL be inferred from the function in which it appears. Each element SHALL contain a URI or XPath expression, respectively, to identify the required ***attribute*** values. If an operational error were to occur while finding the values, the value of the element SHALL be set to "Indeterminate". If the required ***attribute*** cannot be located, then the value of the element SHALL be set to an empty ***bag*** of the inferred primitive type.

In the special case of the <QualifiedSubjectAttributeDesignator> element, the sequence of <SubjectMatch> elements SHALL be evaluated as if each <SubjectMatch> element, while applied only to one particular ***subject***, appeared in a conjunctive sequence.

A.9. Element <Apply>

XACML function calls are represented by the <Apply> element. The function to be applied is named in the FunctionId attribute of this element. The value of the <Apply> element SHALL be set to either a primitive type or a ***bag*** of a primitive type, whose type SHALL be inferred from the functionId. The arguments of a function SHALL be the values of the XACML expressions that are contained as ordered elements in an <Apply> element. The legal number of arguments within an <Apply> element SHALL depend upon the functionId.

A.10. Element <Condition>

The <Condition> element MAY appear in the <Rule> element as the premise for emitting the corresponding ***effect*** of the ***rule***. The <Condition> element has the same structure as the

<Apply> element, with the restriction that its result SHALL be of type "http://www.w3.org/2001/XMLSchema#boolean". The evaluation of the <Condition> element SHALL follow the same evaluation semantics as those of the <Apply> element.

A.11. Element <Function>

The <Function> element names a standard XACML function or an extension function in its FunctionId attribute. The <Function> element MAY be used as an argument in functions that take a function as an argument.

A.12. Matching elements

Matching elements appear in the <Target> element of ***rules***, ***policies*** and ***policy sets***. They are the following:

- <SubjectMatch>
- <ResourceMatch>
- <ActionMatch>
- <EnvironmentMatch>

These elements represent boolean expressions over ***attributes*** of the ***subject***, ***resource***, ***action*** and ***environment***, respectively.

The match elements: <SubjectMatch>, <ResourceMatch>, <ActionMatch> and <EnvironmentMatch> SHALL use functions that match two arguments, returning a result type of "xs:boolean", to perform the match evaluation. The function used for determining a match is named in the MatchId attribute of these elements. Each argument to the named function MUST match the appropriate primitive types for the <AttributeDesignator> or <AttributeSelector> element and the following explicit ***attribute*** value, such that the explicit ***attribute*** value is placed as the first argument to the function, while an element of the ***bag*** returned by the <AttributeDesignator> or <AttributeSelector> element is placed as the second argument to the function.

The XACML standard functions that may be used as a MatchId attribute value are:

function:*type*-equal

function: *type*-greater-than

function: *type*-greater-than-or-equal

function: *type*-less-than

function: *type*-less-than-or-equal

function: *type*-match

The evaluation semantics for a match is as follows. If an operational error were to occur while evaluating the <AttributeDesignator> or <AttributeSelector> element, then the result of the entire expression SHALL be "Indeterminate". If the <AttributeDesignator> or <AttributeSelector> element were to evaluate to an empty ***bag***, then the result of the expression SHALL be "False". Otherwise, the match function SHALL be applied between the explicit ***attribute*** value and each element of the ***bag*** returned from the `<AttributeDesignator>` or `<AttributeSelector>` element. If at least one of those function applications were to evaluate to "True", then the result of the entire expression SHALL be "True". Otherwise, if at least one of the function applications results in "Indeterminate", then the result SHALL be "Indeterminate". Finally, only if all function applications evaluate to "False", SHALL the result of the entire expression be "False".

A match can equivlently be expressed in a ***target*** or a ***condition***. For instance, the match expression that compares a "subject-name" starting with the name "John" can be expressed as follows:

```
<SubjectMatch MatchId="function:regexp-string-match">
   <SubjectAttributeDesignator AttributeId="subject-name"/>
   <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>
</SubjectMatch>
```

Alternatively, it can be expressed as an `<Apply>` element in the ***condition*** by using the "function:any-of" function, as follows:

```
<Apply FunctionId="function:any-of">
   <Function FunctionId="function:regexp-string-match"/>
   <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>
   <SubjectAttributeDesignator AttributeId="subject-name"/>
</Apply>
```

For the match elements: `<SubjectMatch>`, `<ResourceMatch>`, `<ActionMatch>` and `<EnvironmentMatch>` that appear in the `<Target>` element of a `<Rule>`, `<Policy>` or `<PolicySet>` the value specified by the `MatchId` attribute SHALL be restricted to the following functions:

- "function:*type*-equal" (for each primitive *type*),
- "function:regexp-string-match",
- "function:rfc822Name-match" and
- "function:x500Name-match",

and only those functions. Functions that are strictly within an extension to XACML should not appear as a value to the `MatchId` attribute in this case. Restricting the `MatchId` attribute to these functions facilitates the use of indexing to find the ***applicable policy*** for a particular ***authorization request***.

A.13. Arithmetic evaluation

IEEE 754 [IEEE 754] specifies how to evaluate arithmetic functions in a context, which specifies defaults for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all integer and double functions relying on the *Extended Default Context*, enhanced with double precision:

flags - all set to 0

trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the "division-by-zero" trap enabler, which SHALL be set to 1 precision - is set to the designated double precision

rounding - is set to round-half-even (IEEE 854 §4.1)

A.14. XACML standard functions

XACML specifies the following functions that are prefixed with the “function:” relative name space identifier.

A14.1 Equality predicates

The following functions are the *equality* functions for the various primitive types. Each function for a particular type follows a specified standard convention for that type. If an argument of one of these functions were to evaluate to "Indeterminate", then the function SHALL be set to "Indeterminate".

- string-equal

  This function SHALL take two arguments of “http://www.w3.org/2001/XMLSchema#string” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. The function SHALL return "True" if and only if the value of both of its arguments are of equal length and each string is determined to be equal byte-by-byte according to the function “integer-equal”.

- boolean-equal

  This function SHALL take two arguments of “http://www.w3.org/2001/XMLSchema#boolean” and SHALL return "True" if and only if both values are equal.

- integer-equal

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#integer” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL perform its evaluation on integers according to IEEE 754 [IEEE 754].

- double-equal

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#double” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL perform its evaluation on doubles according to IEEE 754 [IEEE 754].

- date-equal

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#date” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL perform its evaluation according to the “op:date-equal” function [XQO Section 8.3.11].

- time-equal

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#time” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL perform its evaluation according to the “op:time-equal” function [XQO Section 8.3.14].

- dateTime-equal

This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL perform its evaluation according to the “op:dateTime-equal” function [XQO Section 8.3.8].

- anyURI-equal

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#anyURI” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL perform its evaluation according to the “op:anyURI-equal” function [XQO Section 10.2.1].

- x500Name-equal

  This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It shall return “True” if and only if each Relative Distinguished Name (RDN) in the two arguments matches. Two RDNs shall be said to match if and only if the result of the following operations is “True”[3].

  1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names".

  2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute ValuePairs in that RDN in ascending order when compared as octet strings (described in ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").

  3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section 4.1.2.4 "Issuer".

- rfc822Name-equal

  This function SHALL take two arguments of type “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL determine whether two “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name” arguments are equal. An RFC822 name consists of a *local-part* followed by "@" followed by a *domain-part*. The *local-part* is case-sensitive, while the *domain-part* (which is usually a DNS host name) is not case-sensitive. Perform the following operations:

  1. Normalize the *domain*-part of each argument to lower case

  2. Compare the expressions by applying the function “function:string-equal” to the normalized arguments.

- hexBinary-equal

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#hexBinary” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return "True" if the octet sequences represented by the value of both arguments have equal length and are equal in a conjunctive, point-wise, comparison using the “function:integer-equal”. The

[3] ITU-T Rec. X.520 contains rules for matching X500 names, but these are very complex and require knowledge of the syntax of various AttributeTypes. IETF RFC 3280 contains simplified matching rules that the XACML x500Name-equal function uses.

conversion from the string representation to an octet sequence SHALL be as specified in [XS Section 8.2.15]

- base64Binary-equal

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#base64Binary” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return "True" if the octet sequences represented by the value of both arguments have equal length and are equal in a conjunctive, point-wise, comparison using the “function:integer-equal”. The conversion from the string representation to an octet sequence SHALL be as specified in [XS Section 8.2.16]

A14.2 Arithmetic functions

All of the following functions SHALL take two arguments of the specified *type*, integer or double, and SHALL return an element of integer or double type, respectively. However, the “add” functions MAY take more than two arguments. Each function evaluation SHALL proceed as specified by their logical counterparts in IEEE 754 [IEEE 754]. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate". In the case of the divide functions, if the divisor is zero, then the function SHALL evaluate to “Indeterminate”.

- integer-add

  This function MAY have two or more arguments.

- double-add

  This function MAY have two or more arguments.

- integer-subtract
- double-subtract
- integer-multiply
- double-multiply
- integer-divide
- double-divide
- integer-mod

The following functions SHALL take a single argument of the specified *type*. The round and floor functions SHALL take a single argument of type “http://www.w3.org/2001/XMLSchema#double” and return type “http://www.w3.org/2001/XMLSchema#double”. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- integer-abs
- double-abs
- round
- floor

A14.3 String conversion functions

The following functions convert between values of the XACML “http://www.w3.org/2001/XMLSchema#string” primitive types. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- string-normalize-space

  This function SHALL take one argument of type “http://www.w3.org/2001/XMLSchema#string” and SHALL normalize the value by stripping off all leading and trailing whitespace characters.

- string-normalize-to-lower-case

  This function SHALL take one argument of “http://www.w3.org/2001/XMLSchema#string” and SHALL normalize the value by converting each upper case character to its lower case equivalent.

A14.4 Numeric type conversion functions

The following functions convert between the XACML “http://www.w3.org/2001/XMLSchema#integer” and” http://www.w3.org/2001/XMLSchema#double” primitive types. In any expression in which the functions defined below are applied, if any argument while being evaluated results in "Indeterminate", the expression SHALL return "Indeterminate".

- double-to-integer

  This function SHALL take one argument of type “http://www.w3.org/2001/XMLSchema#double” and SHALL truncate its numeric value to a whole number and return an element of type “http://www.w3.org/2001/XMLSchema#integer”.

- integer-to-double

  This function SHALL take one argument of type “http://www.w3.org/2001/XMLSchema#integer” and SHALL promote its value to an element of type “http://www.w3.org/2001/XMLSchema#double” of the same numeric value.

A14.5 Logical functions

This section contains the specification for logical functions that operate on arguments of the “http://www.w3.org/2001/XMLSchema#boolean” type.

- or

  This function SHALL return "False" if it has no arguments and SHALL return "True" if one of its arguments evaluates to "True". The order of evaluation SHALL be from first argument to last. The evaluation SHALL stop with a result of "True" if any argument evaluates to "True", leaving the rest of the arguments unevaluated. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- and

  This function SHALL return "True" if it has no arguments and SHALL return "False" if one of its arguments evaluates to "False". The order of evaluation SHALL be from first argument to last. The evaluation SHALL stop with a result of "False" if any argument evaluates to

"False", leaving the rest of the arguments unevaluated. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- n-of

  The first argument to this function SHALL be of type “http://www.w3.org/2001/XMLSchema#integer”, specifying the number of the remaining arguments that MUST evaluate to "True" for the expression to be considered "True". If the first argument is 0, the result SHALL be "True". If the number of arguments after the first one is less than the value of the first argument, then the expression SHALL result in "Indeterminate". The order of evaluation SHALL be: first evaluate the integer value, then evaluate each subsequent argument. The evaluation SHALL stop and return "True" if the specified number of arguments evaluate to "True". The evaluation of arguments SHALL stop if it is determined that evaluating the remaining arguments will not satisfy the requirement. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- not

  This function SHALL take one logical argument. If the argument evaluates to "True", then the result of the expression SHALL be "False". If the argument evaluates to "False", then the result of the expression SHALL be "True". In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- present

  This function SHALL take an attribute value of type “http://www.w3.org/2001/XMLSchema#anyURI” as used as the `AttributeId` in an `<AttributeDesignator>` element. This expression SHALL return "True" if the named ***attribute*** can be located in the request ***context***, which means that an `<AttributeDesignator>` or `<AttributeSelector>` element for this named ***attribute*** will return a ***bag*** containing at least one value. If it cannot be determined that the ***attribute*** is present in the request ***context***, or the attribute cannot be evaluated, then the expression SHALL result in "Indeterminate".

A14.6 Arithmetic comparison functions

These functions form a minimal set for comparing two numbers, yielding a boolean result. They SHALL comply with the rules governed by IEEE 754 [IEEE 754]. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- integer-greater-than
- integer-greater-than-or-equal
- integer-less-than
- integer-less-than-or-equal
- double-greater-than
- double-greater-than-or-equal
- double-less-than
- double-less-than-or-equal

A14.7 Date and time arithmetic functions

These functions perform arithmetic operations with the date and time. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- dateTime-add-dayTimeDuration

  This function SHALL take two arguments, the first is of type “http://www.w3.org/2001/XMLSchema#dateTime” and the second is of type “xf:dayTimeDuration”. It SHALL return a result of “http://www.w3.org/2001/XMLSchema#dateTime”. This function SHALL return the value by adding the second argument to the first argument according to the specification of adding durations to date and time [XS Appendix E].

- dateTime-add-yearMonthDuration

  This function SHALL take two arguments, the first is a “http://www.w3.org/2001/XMLSchema#dateTime” and the second is a “xf:yearMonthDuration”. It SHALL return a result of “http://www.w3.org/2001/XMLSchema#dateTime”. This function SHALL return the value by adding the second argument to the first argument according to the specification of adding durations to date and time [XS Appendix E].

- dateTime-subtract-dayTimeDuration

  This function SHALL take two arguments, the first is a “http://www.w3.org/2001/XMLSchema#dateTime” and the second is a “xf:dayTimeDuration”. It SHALL return a result of “http://www.w3.org/2001/XMLSchema#dateTime”. If the second argument is a positive duration, then this function SHALL return the value by adding the corresponding negative duration, as per the specification [XS Appendix E]. If the second argument is a negative duration, then the result SHALL be as if the function “function:dateTime-add-dayTimeDuration” had been applied to the corresponding positive duration.

- dateTime-subtract-yearMonthDuration

  This function SHALL take two arguments, the first is a “http://www.w3.org/2001/XMLSchema#dateTime” and the second is a “xf:yearMonthDuration”. It SHALL return a result of “http://www.w3.org/2001/XMLSchema#dateTime”. If the second argument is a positive duration, then this function SHALL return the value by adding the corresponding negative duration, as per the specification [XS Appendix E]. If the second argument is a negative duration, then the result SHALL be as if the function “function:dateTime-add-yearMonthDuration” had been applied to the corresponding positive duration.

- date-add-yearMonthDuration

  This function SHALL take two arguments, the first is a “http://www.w3.org/2001/XMLSchema#date” and the second is a “xf:yearMonthDuration”. It return a result of “http://www.w3.org/2001/XMLSchema#date”. This function SHALL return the value by adding the second argument to the first argument according to the specification of adding durations to date [XS Appendix E].

- date-subtract-yearMonthDuration

  This function SHALL take two arguments, the first is a “http://www.w3.org/2001/XMLSchema#date” and the second is a “xf:yearMonthDuration”. It SHALL return a result of “http://www.w3.org/2001/XMLSchema#date”. If the second argument is a positive duration, then this function SHALL return the value by adding the corresponding negative duration, as per the specification [XS Appendix E]. If the second argument is a negative duration, then the result SHALL be as if the function “function:date-add-yearMonthDuration” had been applied to the corresponding positive duration.

A14.8 Non-numeric comparison functions

These functions perform comparison operations on two arguments of non-numerical types. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- string-greater-than

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#string” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the arguments are compared byte by byte and, after an initial prefix of corresponding bytes from both arguments that are considered equal by “function:integer-equal”, the next byte by byte comparison is such that the byte from the first argument is greater than the byte from the second argument by the use of the function “function:integer-equal”.

- string-greater-than-or-equal

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#string” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated with the logical function “function:or” with two arguments containing the functions “function:string-greater-than” and “function:string-equal” containing the original arguments

- string-less-than

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#string” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the arguments are compared byte by byte and, after an initial prefix of corresponding bytes from both arguments are considered equal by “function:integer-equal”, the next byte by byte comparison is such that the byte from the first argument is less than the byte from the second argument by the use of the function “function:integer-less-than”.

- string-less-than-or-equal

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#string” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated with the function “function:or” with two arguments containing the functions “function:string-less-than” and “function:string-equal” containing the original arguments.

- time-greater-than

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#time” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first argument is greater than the second argument according to the order relation specified for “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].

- time-greater-than-or-equal

  This function SHALL take two arguments of type “http://www.w3.org/2001/XMLSchema#time” and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is greater than or equal to the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

- time-less-than

  This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema#time" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is less than the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

- time-less-than-or-equal

  This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema#time" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is less than or equal to the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

- dateTime-greater-than

  This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is greater than the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

- dateTime-greater-than-or-equal

  This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is greater than or equal to the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

- dateTime-less-than

  This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is less than the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

- dateTime-less-than-or-equal

  This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema# dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is less than or equal to the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

- date-greater-than

  This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema#date" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is greater than the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

- date-greater-than-or-equal

  This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema#date" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is greater than or equal to the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

- date-less-than

  This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema#date" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is less than the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

- date-less-than-or-equal

  This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema#date" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is less than or equal to the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

A14.9 Bag functions

These functions operate on a ***bag*** of *type* values, where *type* is one of the primitive types. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate". Some additional conditions defined for each function below SHALL cause the expression to evaluate to "Indeterminate".

- *type*-one-and-only

  This function SHALL take an argument of a ***bag*** of *type* values and SHALL return a value of *type*. It SHALL return the only value in the ***bag***. If the ***bag*** does not have one and only one value, then the expression SHALL evaluate to "Indeterminate".

- *type*-bag-size

  This function SHALL take a ***bag*** of *type* values as an argument and SHALL return an "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the ***bag***.

- *type*-is-in

  This function SHALL take an argument of type *type* as the first argument and a ***bag*** of *type* values as the second argument. The expression SHALL evaluate to "True" if the first argument matches by the "function:type-equal" to any value in the ***bag***.

- *type*-bag

  This function SHALL take any number of arguments of a single type and return a ***bag*** of *type* values containing the values of the arguments. An application of this function to zero arguments SHALL produce an empty ***bag*** of the specified type.

A14.10 Set functions

These functions operate on ***bags*** mimicking sets by eliminating duplicate elements from a ***bag***. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- *type*-intersection

  This function SHALL take two arguments that are both a ***bag*** of *type* values. The expression SHALL return a ***bag*** of *type* values such that it contains only elements that are common between the two ***bags***, which is determined by "function:type-equal". No duplicates as determined by "function:type-equal" SHALL exist in the result.

- *type*-at-least-one-member-of

  This function SHALL take two arguments that are both a ***bag*** of *type* values. The expression SHALL evaluate to "True" if at least one element of the first argument is contained in the second argument as determined by "function:type-is-in".

- *type*-union

  This function SHALL take two arguments that are both a ***bag*** of *type* values. The expression SHALL return a ***bag*** of *type* such that it contains all elements of both ***bags***. No duplicates as determined by "function:type-equal" SHALL exist in the result.

- *type*-subset

  This function SHALL take two arguments that are both a ***bag*** of *type* values. It SHALL return "True" if the first argument is a subset of the second argument. Each argument is considered to have its duplicates removed as determined by "function:type-equal" before subset calculation.

- *type*-set-equals

  This function SHALL take two arguments that are both a ***bag*** of *type* values and SHALL return the result of applying "function:and" to the application of "function:type-subset" to the first and second arguments and the application of "function:type-subset" to the second and first arguments.

A14.11 Higher-order bag functions

This section describes functions in XACML that perform operations on ***bags*** such that functions may be applied to the ***bags*** in general.

In this section, a general-purpose functional language called Haskell [Haskell] is used to formally specify the semantics of these functions. Although the English description is adequate, a formal specification of the semantics is helpful.

For a quick summary, in the following Haskell notation, a function definition takes the form of clauses that are applied to patterns of structures, namely lists. The symbol “[]” denotes the empty list, whereas the expression “(x:xs)” matches against an argument of a non-empty list of which “x” represents the first element of the list, and “xs” is the rest of the list, which may be an empty list. We use the Haskell notion of a list, which is an ordered collection of elements, to model the XACML ***bags*** of values.

A simple Haskell definition of a familiar function “function:and” that takes a list of booleans is defined as follows:

```
and:: [Bool] -> Bool
```

```
and []        = "True"

and (x:xs)  = x && (and xs)
```

The first definition line denoted by a “::” formally describes the type of the function, which takes a list of booleans, denoted by “[Bool]”, and returns a boolean, denoted by “Bool”. The second definition line is a clause that states that the function “and” applied to the empty list is "True". The second definition line is a clause that states that for a non-empty list, such that the first element is “x”, which is a value of type Bool, the function “and” applied to x SHALL be combined with, using the logical conjunction function, which is denoted by the infix symbol “&&”, the result of recursively applying the function “and” to the rest of the list. Of course, an application of the “and” function is "True" if and only if the list to which it is applied is empty or every element of the list is "True". For example, the evaluation of the following Haskell expressions,

(and []), (and ["True"]), (and ["True","True"]), (and ["True","True","False"])

evaluate to "True", "True", "True", and "False", respectively.

In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- any-of

  This function applies a boolean function between a specific primitive value and a ***bag*** of values, and SHALL return "True" if and only if the predicate is "True" for at least one element of the ***bag***.

  This function SHALL take three arguments. The first argument SHALL be a `<Function>` element that names a boolean function that takes two arguments of primitive types. The second argument SHALL be a value of a primitive type. The third argument SHALL be a ***bag*** of a primitive type. The expression SHALL be evaluated as if the function named in the `<Function>` element is applied to the second argument and each element of the third argumane (the ***bag***) and the results are combined with “function:or”.

  In Haskell, the semantics of this operation are as follows:

```
any_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
any_of   f   a   []        = "False"
any_of   f   a   (x:xs) = (f  a  x) || (any_of  f  a  xs)
```

  In the above notation, “f” is the function name to be applied, “a” is the primitive value, and “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

  For example, the following expression SHALL return "True":

```
<Apply FunctionId=”function:any-of”>
   <Function FunctionId=”function:string-equal”/>
   <AttributeValue 
DataType=”http://www.w3.org/2001/XMLSchema#string”>Paul</AttributeValue>
   <Apply FunctionId=”function:string-bag”>
      <AttributeValue 
DataType=”http://www.w3.org/2001/XMLSchema#string”>John</AttributeValue>
      <AttributeValue 
DataType=”http://www.w3.org/2001/XMLSchema#string”>Paul</AttributeValue>
      <AttributeValue 
DataType=”http://www.w3.org/2001/XMLSchema#string”>George</AttributeValue>
      <AttributeValue 
DataType=”http://www.w3.org/2001/XMLSchema#string”>Ringo</AttributeValue>
   </Apply>
</Apply>
```

This expression is "True" because the first argument is equal to at least one of the elements of the ***bag***.

- all-of

This function applies a boolean function between a specific primitive value and a ***bag*** of values, and returns "True" if and only if the predicate is "True" for every element of the ***bag***.

This function SHALL take three arguments. The first argument SHALL be a `<Function>` element that names a boolean function that takes two arguments of primitive types. The second argument SHALL be a value of a primitive type. The third argument SHALL be a ***bag*** of a primitive type. The expression SHALL be evaluated as if the function named in the `<Function>` element were applied to the second argument and each element of the third argument (the ***bag***) and the results were combined using “function:and”.

In Haskell, the semantics of this operation are as follows:

```
all_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
all_of  f  a  []       = "False"
all_of  f  a  (x:xs) = (f  a  x) && (all_of  f  a  xs)
```

In the above notation, “f” is the function name to be applied, “a” is the primitive value, and “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

For example, the following expression SHALL evaluate to "True":

```
<Apply FunctionId=”function:all-of”>
   <Function FunctionId=”function:integer-greater”/>
   <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>10</AttributeValue>
   <Apply FunctionId=”function:integer-bag”>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>9</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>3</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>4</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>2</AttributeValue>
   </Apply>
</Apply>
```

This expression is "True" because the first argument is greater than *all* of the elements of the ***bag***.

- any-of-any

This function applies a boolean function between each element of a ***bag*** of values and each element of another ***bag*** of values, and returns "True" if and only if the predicate is "True" for at least one comparison.

This function SHALL take three arguments. The first argument SHALL be a `<Function>` element that names a boolean function that takes two arguments of primitive types. The second argument SHALL be a ***bag*** of a primitive type. The third argument SHALL be a ***bag*** of a primitive type. The expression SHALL be evaluated as if the function named in the `<Function>` element were applied between *every* element in the second argument and *every* element of the third argument (the ***bag***) and the results were combined using “function:or”. The semantics are that the result of the expression SHALL be "True" if and only if the applied predicate is "True" for *any* comparison of elements from the two ***bags***.

In Haskell, taking advantage of the “any_of” function defined above, the semantics of the “any_of_any” function are as follows:

```
any_of_any :: ( a -> b -> Bool ) -> [a ]-> [b] -> Bool
any_of_any  f  []          ys = "False"
any_of_any   f  (x:xs)  ys = (any_of  f  x  ys) || (any_of_any  f  xs  ys)
```

In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

For example, the following expression SHALL evaluate to "True":

```
<Apply FunctionId=”function:any-of-any”>
   <Function FunctionId=”function:string-equal”/>
   <Apply FunctionId=”function:string-bag”>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>Ringo</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>Mary</AttributeValue>
   </Apply>
   <Apply FunctionId=”function:string-bag”>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>John</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>Paul</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>George</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>Ringo</AttributeValue>
   </Apply>
</Apply>
```

This expression is "True" because at least one of the elements of the first ***bag***, namely “Ringo”, is equal to at least one of the string values of the second ***bag***.

- all-of-any

This function applies a boolean function between the elements of two ***bags***. The expression is "True" if and only if the predicate is "True" between each and all of the elements of the first ***bag*** collectively against at least one element of the second ***bag***.

This function SHALL take three arguments. The first argument SHALL be a `<Function>` element that names a boolean function that takes two arguments of primitive types. The second argument SHALL be a ***bag*** of a primitive type. The third argument SHALL be a ***bag*** of a primitive type. The expression SHALL be evaluated as if function named in the `<Function>` element were applied between every element in the second argument and every element of the third argument (the ***bag***) using “function:and”. The semantics are that the result of the expression SHALL be "True" if and only if the applied predicate is "True" for each element of the first ***bag*** and any element of the second ***bag***.

In Haskell, taking advantage of the “any_of” function defined in Haskell above, the semantics of the “all_of_any” function are as follows:

```
all_of_any :: ( a -> b -> Bool ) -> [a ]-> [b] -> Bool
all_of_any   f  []        ys = "False"
all_of_any   f  (x:xs)  ys = (any_of  f  x  ys) && (all_of_any f  xs  ys)
```

In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

For example, the following expression SHALL evaluate to "True":

```
<Apply FunctionId=”function:all-of-any”>
   <Function FunctionId=”function:integer-greater”/>
   <Apply FunctionId=”function:integer-bag”>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>10</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>20</AttributeValue>
   </Apply>
   <Apply FunctionId=”function:integer-bag”>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>1</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>3</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>5</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>21</AttributeValue>
   </Apply>
</Apply>
```

This expression is "True" because all of the elements of the first ***bag***, each “10” and “20”, are greater than at least one of the integer values “1”, ”3”, ”5”, ”21” of the second ***bag***.

- any-of-all

This function applies a boolean function between the elements of two ***bags***. The expression SHALL be "True" if and only if the predicate is "True" between at least one of the elements of the first ***bag*** collectively against all the elements of the second ***bag***.

This function SHALL take three arguments. The first argument SHALL be a `<Function>` element that names a boolean function that takes two arguments of primitive types. The second argument SHALL be a ***bag*** of a primitive type. The third argument SHALL be a ***bag*** of a primitive type. The expression SHALL be evaluated as if the function named in the `<Function>` element were applied between *every* element in the second argument and *every* element of the third argument (the ***bag***) and the results were combined using “function:or”. The semantics are that the result of the expression SHALL be "True" if and only if the applied predicate is "True" for *any* element of the first ***bag*** compared to *all* the elements of the second ***bag***.

In Haskell, taking advantage of the “all_of” function defined in Haskell above, the semantics of the “any_of_all” function are as follows:

```
any_of_all :: ( a -> b -> Bool ) -> [a ]-> [b] -> Bool
any_of_all  f  []        ys = "False"
any_of_all   f  (x:xs)  ys = (all_of  f  x  ys) || ( any_of_all  f  xs  ys)
```

In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

For example, the following expression SHALL evaluate to "True":

```
<Apply FunctionId=”function:any-of-all”>
   <Function FunctionId=”function:integer-greater”/>
   <Apply FunctionId=”function:integer-bag”>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>3</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>5</AttributeValue>
   </Apply>
   <Apply FunctionId=”function:integer-bag”>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>1</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>2</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>3</AttributeValue>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#integer”>4</AttributeValue>
   </Apply>
</Apply>
```

This expression is "True" because at least one element of the first ***bag***, namely “5”, is greater than all of the integer values “1”, ”2”, ”3”, ”4” of the second ***bag***.

- all-of-all

This function applies a boolean function between the elements of two ***bags***. The expression SHALL be "True" if and only if the predicate is "True" between each and all of the elements of the first ***bag*** collectively against all the elements of the second ***bag***.

This function SHALL take three arguments. The first argument SHALL be a `<Function>` element that names a boolean function that takes two arguments of primitive types. The second argument SHALL be a ***bag*** of a primitive type. The third argument SHALL be a ***bag*** of a primitive type. The expression is evaluated as if the function named in the `<Function>` element were applied between *every* element in the second argument and *every* element of the third argument (the ***bag***) and the results were combined using “function:and”. The semantics are that the result of the expression is "True" if and only if the applied predicate is "True" for *all* elements of the first ***bag*** compared to *all* the elements of the second ***bag***.

In Haskell, taking advantage of the “all_of” function defined in Haskell above, the semantics of the “all_of_all” function is as follows:

```
all_of_all :: ( a -> b -> Bool ) -> [a ]-> [b] -> Bool
all_of_all  f  []       ys = "False"
all_of_all  f  (x:xs)  ys = (all_of  f  x  ys) && (all_of_all  f  xs  ys)
```

In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

For example, the following expression SHALL evaluate to "True":

```
<Apply FunctionId="function:all-of-all">
   <Function FunctionId="function:integer-greater"/>
   <Apply FunctionId="function:integer-bag">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
   </Apply>
   <Apply FunctionId="function:integer-bag">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
   </Apply>
</Apply>
```

This expression is "True" because all elements of the first ***bag***, “5” and “6”, are each greater than all of the integer values “1”, ”2”, ”3”, ”4” of the second ***bag***.

- map

This function converts a ***bag*** of values to another ***bag*** of values.

This function SHALL take two arguments. The first function SHALL be a `<Function>` element naming a function that takes a single argument of a primitive type and returns a value of a primitive type. The second argument SHALL be a ***bag*** of a primitive type. The expression SHALL be evaluated as if the function named in the `<Function>` element were applied to each element in the ***bag*** resulting in a ***bag*** of the converted value. The result SHALL be a ***bag*** of the primitive type that is the same type that is returned by the function named in the `<Function>` element.

In Haskell, this function is defined as follows:

map:: (a -> b) -> [a] -> [b]

map f [] = []

map f (x:xs) = (f x) : (map f xs)

In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

For example, the following expression,

```
<Apply FunctionId="function:map">
   <Function FunctionId="function:string-normalize-to-lower-case">
   <Apply FunctionId="function:string-bag">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>
   </Apply>
</Apply>
```

evaluates to a ***bag*** containing “hello” and “world!”.

A14.12 Special match functions

These functions operate on various types and evaluate to “http://www.w3.org/2001/XMLSchema#boolean” based on the specified standard matching algorithm. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- regexp-string-match

  This function decides a regular expression match. It SHALL take two arguments of “http://www.w3.org/2001/XMLSchema#string” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. The first argument SHALL be a regular expression and the second argument SHALL be a general string. The function specification SHALL be that of the “xf:match” function with the arguments reversed [XF Section 6.3.15.1].

- x500Name-match

  This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It shall return “True” if and only if the first argument matches some terminal sequence of RDNs from the second argument when compared using x500Name-equal.

- rfc822Name-match

  This function SHALL take two arguments, the first is of type “http://www.w3.org/2001/XMLSchema#string” and the second is of type “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name” and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL evaluate to "True" if the first argument matches the second argument according to the following specification.

  An RFC822 name consists of a local-part followed by "@" followed by domain-part. The local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not case-sensitive.[4]

  The second argument contains a complete rfc822Name. The first argument is a complete or partial rfc822Name used to select appropriate values in the second argument as follows.

  In order to match a particular mailbox in the second argument, the first argument must specify the complete mail address to be matched. For example, if the first argument is “Anderson@sun.com”, this matches a value in the second argument of “Anderson@sun.com” and “Anderson@SUN.COM”, but not “Anne.Anderson@sun.com”, “anderson@sun.com” or “Anderson@east.sun.com”.

  In order to match any mail address at a particular domain in the second argument, the first argument must specify only a domain name (usually a DNS name). For example, if the first argument is “sun.com”, this matches a value in the first argument of “Anderson@sun.com? or “Baxter@SUN.COM”, but not “Anderson@east.sun.com”.

  In order to match any mail address in a particular domain in the second argument, the first argument must specify the desired domain-part with a leading ".". For example, if the first argument is “.east.sun.com”, this matches a value in the second argument of

4 According to IETF RFC822 and its successor specifications [RFC2821], case is significant in the *local-part.* Many mail systems, as well as the IETF PKIX specification, treat the *local-part* as case-insensitive. This anomaly is considered an error by mail-system designers and is not encouraged. For this reason, rfc822Name-match treats *local-part* as case sensitive.

"Anderson@east.sun.com" and "anne.anderson@ISRG.EAST.SUN.COM" but not "Anderson@sun.com".

A14.13 XPath-based functions

This section specifies functions that take XPath expressions for arguments. An XPath expression evaluates to a *node-set*, which is a set of XML nodes that match the expression. A node or node-set is not in the formal type system of XACML. All comparison or other operations on node-sets are performed in the isolation of the particular function specified. The XPath expressions in these functions are restricted to the XACML request ***context***. The following functions are defined:

- xpath-node-count

  This function SHALL take an “http://www.w3.org/2001/XMLSchema#string” as an argument, which SHALL be interpreted as an XPath expression, and evaluates to an “http://www.w3.org/2001/XMLSchema#integer”. The value returned from the function SHALL be the count of the nodes within the node-set that matches the given XPath expression.

- xpath-node-equal

  This function SHALL take two “http://www.w3.org/2001/XMLSchema#string” arguments, which SHALL be interpreted as XPath expressions, and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. The function SHALL return "True" if any XML node from the node-set matched by the first argument equals according to the “op:node-equal” function [XQO] any XML node from the node-set matched by the second argument.

- xpath-node-match

  This function SHALL take two “http://www.w3.org/2001/XMLSchema#string” arguments, which SHALL be interpreted as XPath expressions and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL first extend the first argument to match an XML document in a hierarchical fashion. If *a* is an XPath expression and it is specified as the first argument, it SHALL be interpreted to mean match the set of nodes specified by the enhanced XPath expression "*a* | *a*//* | *a*//@*". In other words, the expression *a* SHALL match all elements and attributes below the element specified by *a*. This function SHALL evaluate to "True" if any XML node that matches the enhanced XPath expression is equal according to “op:node-equal” [XQO] to any XML node from the node-set matched by the second argument.

A14.14 Extension functions and primitive types

Functions and primitive types are specified by string identifiers allowing for the introduction of functions in addition to those specified by XACML. This approach allows one to extend the XACML module with special functions and special primitive data types.

In order to preserve some integrity to the XACML evaluation strategy, the result of all function applications SHALL depend only on the values of its arguments. Global and hidden parameters SHALL NOT affect the evaluation of an expression. Functions SHALL NOT have side effects, as evaluation order cannot be guaranteed in a standard way.

Appendix B. 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:1.0
```

B.1. XACML namespaces

There are currently two defined XACML namespaces.

Policies are defined using this identifier.

```
urn:oasis:names:tc:xacml:1.0:policy
```

Request and response ***contexts*** are defined using this identifier.

```
urn:oasis:names:tc:xacml:1.0:context
```

XACML data-types are defined using this identifier.

```
urn:oasis:names:tc:xacml:1.0:data-type
```

B.2. Access subject categories

This identifier indicates the system entity that is directly requesting ***access***. That is, the final entity in a request chain. If ***subject*** category is not specified, this is the default value.

```
urn:oasis:names:tc:xacml:1.0:subject-category:access-subject
```

This identifier indicates the system entity that will receive the results of the request. Used when it is distinct from the access-subject.

```
urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject
```

This identifier indicates a system entity through which the ***access*** request was passed. There may be more than one. No means is provided to specify the order in which they passed the message.

```
urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject
```

This identifier indicates a system entity associated with a local or remote codebase that generated the request. Corresponding ***subject attributes*** might include the URL from which it was loaded and/or the identity of the code-signer. There may be more than one. No means is provided to specify the order they processed the request.

```
urn:oasis:names:tc:xacml:1.0:subject-category:codebase
```

This identifier indicates a system entity associated with the computer that initiated the ***access*** request. An example would be an IPsec identity.

```
urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine
```

B.3. XACML functions

This identifier is the base for all the identifiers in the table of functions. See Section A.1.

```
urn:oasis:names:tc:xacml:1.0:function
```

B.4. Data types

The following identifiers indicate useful data-types.

X.500 distinguished name

```
urn:oasis:names:tc:xacml:1.0:data-type:x500Name
```

An x500Name contains an ITU-T Rec. X.520 Distinguished Name. The valid syntax for such a name is described in IETF RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names".

RFC822 Name

```
urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name
```

An rfc822Name contains an "e-mail name". The valid syntax for such a name is described in IETF RFC 2821, Section 4.1.2, Command Argument Syntax, under the term "Mailbox".

The following data type identifiers are defined by XML Schema and XQuery.

```
http://www.w3.org/2001/XMLSchema:string
http://www.w3.org/2001/XMLSchema:boolean
http://www.w3.org/2001/XMLSchema:integer
http://www.w3.org/2001/XMLSchema:double
http://www.w3.org/2001/XMLSchema:date
http://www.w3.org/2001/XMLSchema:dateTime
http://www.w3.org/2001/XMLSchema:anyURI
http://www.w3.org/2001/XMLSchema:hexBinary
http://www.w3.org/2001/XMLSchema:base64Binary
http://www.w3.org/2002/08/xquery-functions:dayTimeDuration
http://www.w3.org/2002/08/xquery-functions:yearMonthDuration
```

B.5. Subject attributes

These identifiers indicate ***attributes*** of a ***subject***. When used, they SHALL appear within a `<Subject>` element of the request ***context***. They SHALL be accessed via a `<SubjectAttributeDesignator>`, a `<QualifiedSubjectAttributeDesignator>` or an `<AttributeSelector>` element pointing into a `<Subject>` element of the request ***context***.

At most one of each of these attributes is associated with each subject. Each attribute associated with authentication included within a single <Subject> element relates to the same authentication event.

This identifier indicates the name of the ***subject***. The default format is http://www.w3.org/2001/XMLSchema#string. To indicate other formats, use `DataType` attributes listed in B.4

```
urn:oasis:names:tc:xacml:1.0:subject:subject-id
```

This identifier indicates the ***subject*** category. “access-subject” is the default.

```
urn:oasis:names:tc:xacml:1.0:subject:subject-category
```

This identifier indicates the security domain of the ***subject***. It identifies the administrator and policy that manages the name-space in which the ***subject*** id is administered.

```
urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier
```

This identifier indicates a public key used to confirm the ***subject's*** identity.

```
urn:oasis:names:tc:xacml:1.0:subject:key-info
```

This identifier indicates the time at which the ***subject*** was authenticated.

```
urn:oasis:names:tc:xacml:1.0:subject:authentication-time
```

This identifier indicates the method used to authenticate the ***subject***.

```
urn:oasis:names:tc:xacml:1.0:subject:authentication-method
```

This identifier indicates the time at which the ***subject*** initiated the ***access*** request, according to the ***PEP***.

```
urn:oasis:names:tc:xacml:1.0:subject:request-time
```

This identifier indicates the time at which the ***subject's*** current session began, according to the ***PEP***.

```
urn:oasis:names:tc:xacml:1.0:subject:session-start-time
```

The following identifiers indicate the location where authentication credentials were activated. They are intended to support the corresponding entities from the SAML authentication statement.

This identifier indicates that the location is expressed as an IP address.

```
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address
```

This identifier indicates that the location is expressed as a DNS name.

```
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name
```

Where a suitable attribute is already defined in LDAP [LDAP-1, LDAP-2], the XACML identifier SHALL be formed by adding the ***attribute*** name to the URI of the LDAP specification. For example, the ***attribute*** name for the userPassword defined in the rfc2256 SHALL be:

```
http://www.ietf.org/rfc/rfc2256.txt#userPassword
```

B.6. Resource attributes

These identifiers indicate ***attributes*** of the ***resource***. When used, they SHALL appear within the `<Resource>` element of the request ***context***. They SHALL be accessed via a `<ResourceAttributeDesignator>` or an `<AttributeSelector>` element pointing into the `<Resource>` element of the request ***context***.

This identifier indicates the entire URI of the ***resource***.

```
urn:oasis:names:tc:xacml:1.0:resource:resource-id
```

A ***resource attribute*** used to indicate values extracted from the ***resource***.

```
urn:oasis:names:tc:xacml:1.0:resource:resource-content
```

This identifier indicates the last (rightmost) component of the file name. For example, if the URI is: "file://home/my/status#pointer", the simple-file-name is "status".

```
urn:oasis:names:tc:xacml:1.0:resource:simple-file-name
```

This identifier indicates that the ***resource*** is specified by an XPath expression.

```
urn:oasis:names:tc:xacml:1.0:resource:xpath
```

This identifier indicates a UNIX file-system path.

```
urn:oasis:names:tc:xacml:1.0:resource:ufs-path
```

This identifier indicates the scope of the ***resource***, as described in Section 7.8.

```
    urn:oasis:names:tc:xacml:1.0:resource:scope
```

The allowed value for this attribute is of type http://www.w3.org/2001/XMLSchema#string, and is either "Immediate", "Children" or "Descendants".

B.7. Action attributes

These identifiers indicate ***attributes*** of the ***action*** being rquested. When used, they SHALL appear within the `<Action>` element of the request ***context***. They SHALL be accessed via an `<ActionAttributeDesignator>` or an `<AttributeSelector>` element pointing into the `<Action>` element of the request ***context***.

```
urn:oasis:names:tc:xacml:1.0:action:action-id
```

Action namespace

```
urn:oasis:names:tc:xacml:1.0:action:action-namespace
```

Implied action. This is the value for action-id attribute when action is implied.

```
urn:oasis:names:tc:xacml:1.0:action:implied-action
```

B.8. Environment attributes

These identifiers indicate ***attributes*** of the ***environment*** within which the ***decision request*** is to be evaluated. When used, they SHALL appear within the `<Resource>` element of the request ***context***. They SHALL be accessed via an `<EnvironmentAttributeDesignator>` or an `<AttributeSelector>` element pointing into the `<Environment>` element of the request ***context***.

This identifier indicates the current time at the ***PDP***. In practice it is the time at which the request ***context*** was created.

```
urn:oasis:names:tc:xacml:1.0:environment:current-time
urn:oasis:names:tc:xacml:1.0:environment:current-date
urn:oasis:names:tc:xacml:1.0:environment:current-dateTime
```

B.9. Status codes

The following status code identifiers are defined.

This identifier indicates success.

```
urn:oasis:names:tc:xacml:1.0:status:ok
```

This identifier indicates that attributes necessary to make a policy decision were not available.

```
urn:oasis:names:tc:xacml:1.0:status:missing-attribute
```

This identifier indicates that some attribute value contained a syntax error, such as a letter in a numeric field.

```
urn:oasis:names:tc:xacml:1.0:status:syntax-error
```

This identifier indicates that an error occurred during policy evaluation. An example would be division by zero.

```
urn:oasis:names:tc:xacml:1.0:status:processing-error
```

B.10. Combining algorithms

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

```
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides
```

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

```
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides
```

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

```
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides
```

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

```
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides
```

The first-applicable rule-combining algorithm has the following value for `ruleCombiningAlgId`:

```
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable
```

The first-applicable policy-combining algorithm has the following value for `policyCombiningAlgId`:

```
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable
```

The only-one-applicable-policy policy-combining algorithm has the following value for `policyCombiningAlgId`:

```
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable-policy
```

Appendix C. Combining algorithms (normative)

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

C.1. Deny-overrides

The following specification defines the “Deny-overrides” ***rule-combining algorithm*** of a ***policy***.

> In the entire set of ***rules*** in the ***policy***, if any ***rule*** evaluates to "Deny", then the result of the ***rule*** combination SHALL be "Deny". If any ***rule*** evaluates to "Permit" and all other ***rules*** evaluate to "Not-applicable", then the result of the ***rule*** combination SHALL be "Permit". In other words, "Deny" takes precedence, regardless of the result of evaluating any of the other ***rules*** in the combination. If all ***rules*** are found to be "Not-applicable" to the ***decision request***, then the ***rule*** combination SHALL evaluate to "Not-applicable".
>
> If an error occurs while evaluating the ***target*** or ***condition*** of a ***rule*** that contains an ***effect*** value of "Deny" then the evaluation SHALL continue to evaluate subsequent ***rules***, looking for a result of "Deny". If no other ***rule*** evaluates to "Deny", then the combination SHALL evaluate to "Indeterminate".
>
> If at least one ***rule*** evaluates to "Permit", all other ***rules*** that do not have evaluation errors evaluate to "Permit" or "Not-applicable" and all ***rules*** that do have evaluation errors contain ***effects*** of "Permit", then the result of the combination SHALL be "Permit".

The following pseudo-code represents the evaluation strategy of this ***rule-combining algorithm***.

```
Decision denyOverridesRuleCombiningAlgorithm(Rule rule[])
{
   Boolean atLeastOneError  = false;
   Boolean potentialDeny    = false;
   Boolean atLeastOnePermit = false;
   for( i=0 ; i < lengthOf(rules) ; i++ )
   {
      Decision decision = evaluate(rule[i]);
      if (decision == Deny)
      {
         return Deny;
      }
      if (decision == Permit)
      {
         atLeastOnePermit = true;
         continue;
      }
      if (decision == Not-applicable)
      {
         continue;
      }
      if (decision == Indeterminate)
      {
         atLeastOneError = true;

         if (effect(rule[i]) == Deny)
         {
            potentialDeny = true;
         }
         continue;
```

```
    }
  }
  if (potentialDeny)
  {
    return Indeterminate;
  }
  if (atLeastOnePermit)
  {
    return Permit;
  }
  if (atLeastOneError)
  {
    return Indeterminate;
  }
  return Not-applicable;
}
```

The following specification defines the “Deny-overrides” ***policy-combining algorithm*** of a ***policy set***.

> In the entire set of ***policies*** in the ***policy set***, if any ***policy*** evaluates to "Deny", then the result of the ***policy*** combination SHALL be "Deny". In other words, "Deny" takes precedence, regardless of the result of evaluating any of the other ***policies*** in the ***policy set***. If all ***policies*** are found to be "Not-applicable" to the ***decision request***, then the ***policy set*** SHALL evaluate to "Not-applicable".
>
> If an error occurs while evaluating the ***target*** of a ***policy***, or a reference to a ***policy*** is considered invalid or the ***policy*** evaluation results in "Indeterminate", then the ***policy set*** SHALL evaluate to "Deny".

The following pseudo-code represents the evaluation strategy of this ***policy-combining algorithm***.

```
Decision denyOverridesPolicyCombiningAlgorithm(Policy policy[])
{
  Boolean atLeastOnePermit = false;
  for( i=0 ; i < lengthOf(policy) ; i++ )
  {
    Decision decision = evaluate(policy[i]);
    if (decision == Deny)
    {
      return Deny;
    }
    if (decision == Permit)
    {
      atLeastOnePermit = true;
      continue;
    }
    if (decision == Not-applicable)
    {
      continue;
    }
    if (decision == Indeterminate)
    {
      return Deny;
    }
  }
  if (atLeastOnePermit)
  {
    return Permit;
  }
  return Not-applicable;
}
```

Obligations of the individual ***policies*** shall be combined as described in Section 3.3.2.3.

C.2. Permit-overrides

The following specification defines the “Permit-overrides” ***rule-combining algorithm*** of a ***policy***.

> In the entire set of ***rules*** in the ***policy***, if any ***rule*** evaluates to "Permit", then the result of the ***rule*** combination SHALL be "Permit". If any ***rule*** evaluates to "Deny" and all other ***rules*** evaluate to "Not-applicable", then the ***policy*** SHALL evaluate to "Deny". In other words, "Permit" takes precedence, regardless of the result of evaluating any of the other ***rules*** in the ***policy***. If all ***rules*** are found to be "Not-applicable" to the ***decision request***, then the ***policy*** SHALL evaluate to "Not-applicable".
>
> If an error occurs while evaluating the ***target*** or ***condition*** of a ***rule*** that contains an ***effect*** of "Permit" then the evaluation SHALL continue looking for a result of "Permit". If no other ***rule*** evaluates to "Permit", then the ***policy*** SHALL evaluate to "Indeterminate".
>
> If at least one ***rule*** evaluates to "Deny", all other ***rules*** that do not have evaluation errors evaluate to "Deny" or "Not-applicable" and all ***rules*** that do have evaluation errors contain an ***effect*** value of "Deny", then the ***policy*** SHALL evaluate to "Deny".

The following pseudo-code represents the evaluation strategy of this ***rule-combining algorithm***.

```
Decision permitOverridesRuleCombiningAlgorithm(Rule rule[])
{
   Boolean atLeastOneError  = false;
   Boolean potentialPermit  = false;
   Boolean atLeastOneDeny   = false;
   for( i=0 ; i < lengthOf(rule) ; i++ )
   {
      Decision decision = evaluate(rule[i]);
      if (decision == Deny)
      {
         atLeastOneDeny = true;
         continue;
      }
      if (decision == Permit)
      {
         return Permit;
      }
      if (decision == Not-applicable)
      {
         continue;
      }
      if (decision == Indeterminate)
      {
         atLeastOneError = true;

         if (effect(rule[i]) == Permit)
         {
            potentialPermit = true;
         }
         continue;
      }
   }
   if (potentialPermit)
   {
      return Indeterminate;
   }
   if (atLeastOneDeny)
   {
      return Deny;
   }
   if (atLeastOneError)
   {
      return Indeterminate;
   }
   return Not-applicable;
}
```

The following specification defines the “Permit-overrides” ***policy-combining algorithm*** of a ***policy set***.

> In the entire set of ***policies*** in the ***policy set***, if any ***policy*** evaluates to "Permit", then the result of the ***policy*** combination SHALL be "Permit". In other words, "Permit" takes precedence, regardless of the result of evaluating any of the other ***policies*** in the ***policy set***. If all ***policies*** are found to be "Not-applicable" to the ***decision request***, then the ***policy set*** SHALL evaluate to "Not-applicable".
>
> If an error occurs while evaluating the ***target*** of a ***policy***, a reference to a ***policy*** is considered invalid or the ***policy*** evaluation results in "Indeterminate", then the ***policy set*** SHALL evaluate to "Indeterminate" provided no other ***policies*** evaluate to "Permit" or "Deny".

The following pseudo-code represents the evaluation strategy of this ***policy-combining algorithm***.

```
Decision permitOverridesPolicyCombiningAlgorithm(Policy policy[])
{
   Boolean atLeastOneError = false;
   Boolean atLeastOneDeny  = false;
   for( i=0 ; i < lengthOf(policy) ; i++ )
   {
      Decision decision = evaluate(policy[i]);
      if (decision == Deny)
      {
         atLeastOneDeny = true;
         continue;
      }
      if (decision == Permit)
      {
         return Permit;
      }
      if (decision == Not-applicable)
      {
         continue;
      }
      if (decision == Indeterminate)
      {
         atLeastOneError = true;
         continue;
      }
   }
   if (atLeastOneDeny)
   {
      return Deny;
   }
   if (atLeastOneError)
   {
      return Indeterminate;
   }
   return Not-applicable;
}
```

Obligations of the individual policies shall be combined as described in Section 3.3.2.3.

C.3. First-applicable

The following specification defines the "First-Applicable " **rule-combining algorithm** of a ***policy***.

> Each ***rule*** SHALL be evaluated in the order in which it is listed in the ***policy***. For a particular ***rule***, if the ***target*** matches and the ***condition*** evaluates to "True", then the evaluation of the ***policy*** SHALL halt and the corresponding ***effect*** of the ***rule*** SHALL be the result of the evaluation of the ***policy*** (i.e. "Permit" or "Deny"). For a particular ***rule*** selected in the evaluation, if the ***target*** evaluates to "False" or the ***condition*** evaluates to "False", then the next ***rule*** in the order SHALL be evaluated. If no further ***rule*** in the order exists, then the ***policy*** SHALL evaluate to "Not-applicable".

> If an error occurs while evaluating the ***target*** or ***condition*** of a ***rule,*** then the evaluation SHALL halt, and the ***policy*** shall evaluate to "Indeterminate", with the appropriate error status.

The following pseudo-code represents the evaluation strategy of this ***rule-combining algorithm***.

```
Decision firstApplicableEffectRuleCombiningAlgorithm(Rule rule[])
{
   for( i = 0 ; i < lengthOf(rule) ; i++ )
   {
      Decision decision = evaluate(rule[i]);
      if (decision == Deny)
      {
         return Deny;
      }
      if (decision == Permit)
      {
         return Permit;
      }
      if (decision == Not-applicable)
      {
         continue;
      }
      if (decision == Indeterminate)
      {
         return Indeterminate;
      }
   }
   return Not-applicable;
}
```

The following specification defines the “First-applicable” ***policy-combining algorithm*** of a ***policy set***.

> Each ***policy*** is evaluated in the order that it appears in the ***policy set***. For a particular ***policy***, if the ***target*** evaluates to "True" and the ***policy*** evaluates to a determinate value of "Permit" or "Deny", then the evaluation SHALL halt and the ***policy set*** SHALL evaluate to the ***effect*** value of that ***policy***. For a particular ***policy***, if the ***target*** evaluate to "False", or the ***policy*** evaluates to "Not-applicable", then the next ***policy*** in the order SHALL be evaluated. If no further ***policy*** exists in the order, then the ***policy set*** SHALL evaluate to "Not-applicable".

If an error occurs while evaluating the ***target*** or the ***policy***, or a reference to a ***policy*** is considered invalid, then the evaluation SHALL continue looking for an ***applicable policy***, if no ***applicable policy*** is found, then the ***policy set*** SHALL evaluate to "Indeterminate".

If an error were to occur when evaluating the ***target***, or when evaluating a specific ***policy***, the reference to the ***policy*** is considered invalid, or the ***policy*** itself evaluates to "Indeterminate", then the evaluation of the ***policy-combining algorithm*** shall halt, and the ***policy set*** shall evaluate to "Indeterminate" **with an appropriate error status**.

The following pseudo-code represents the evaluation strategy of this ***policy-combination algorithm***.

```
Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy policy[])
{
    for( i = 0 ; i < lengthOf(policy) ; i++ )
    {
        Decision decision = evaluate(policy[i]);
        if(decision == Deny)
        {
            return Deny;
        }
        if(decision == Permit)
        {
            return Permit;
        }
        if (decision == Not-applicable)
        {
            continue;
        }
        if (decision == Indeterminate)
        {
            return Indeterminate;
        }
    }
    return Not-applicable;
}
```

Obligations of the individual policies shall be combined as described in Section 3.3.2.3

C.4. Only-one-applicable

The following specification defines the “Only-one-applicable" ***policy-combining algorithm*** of a ***policy set***.

In the entire set of policies in the ***policy set***, if no ***policy*** is considered applicable by virtue of their ***targets***, then the result of the policy combination algorithm SHALL be "Not-applicable". If more than one policy is considered applicable by virtue of their ***targets***, then the result of the policy combination algorithm SHALL be "Indeterminate".

If only one ***policy*** is considered applicable by evaluation of the ***policy targets***, then the result of the ***policy-combining algorithm*** SHALL be the result of evaluating the ***policy***.

If an error occurs while evaluating the ***target*** of a ***policy***, or a reference to a ***policy*** is considered invalid or the ***policy*** evaluation results in "Indeterminate, then the ***policy set*** SHALL evaluate to "Indeterminate".

The following pseudo-code represents the evaluation strategy of this policy combining algorithm.

```
Decision onlyOneApplicablePolicyPolicyCombiningAlogrithm(Policy policy[])
```

```
{
  Boolean          atLeastOne     = false;
  Policy           selectedPolicy = null;
  ApplicableResult appResult;

  for ( i = 0; i < lengthOf(policy) ; i++ )
  {
     appResult = isApplicable(policy[I]);

     if ( appResult == Indeterminate )
     {
         return Indeterminate;
     }
     if( appResult == Applicable )
     {
         if ( atLeastOne )
         {
             return Indeterminate;
         }
         else
         {
             atLeastOne     = true;
             selectedPolicy = policy[i];
         }
     }
     if ( appResult == NotApplicable )
     {
         continue;
     }
  }
  if ( atLeastOne )
  {
      return evaluate(selectedPolicy);
  }
  else
  {
      return NotApplicable;
  }
}
```

Appendix D. Acknowledgments

The following individuals were voting members of the XACML committee at the time that this version of the specification was issued:

Affinitex James MacLean JMaclean@affinitex.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
Quadrasis Don Flinn Don.Flinn@hitachisoftware.com
Quadrasis Konstantin Beznosov konstantin.beznosov@quadrasis.com
OpenNetwork Steve Andersen sanderson@opennetwork.com
Overxeer Bill Parducci bill.parducci@overxeer.com
Overxeer Simon Godik simon.godik@overxeer.com
IBM Michiharu Kudo kudo@jp.ibm.com
Self Polar Humenn polar@syr.edu
Sterling Commerce Suresh Damodaran Suresh_Damodaran@stercomm.com
Sun Microsystems Anne Anderson Anne.Anderson@Sun.com
Sun Microsystems Pirasenna Velandai Thiyagarajan Pirasenna.Thiyagarajan@Sun.com
Xtradyne Gerald Brose Gerald.Brose@xtradyne.com

Appendix E. Revision history

Rev	Date	By whom	What
V1.0	6 Nov 2002	XACML Technical Committee	First committee specification.

Appendix F. Notices