eXtensible Access Control Markup Language (XACML) Version 1.0

OASIS Standard, 18 February 2003

Document identifier: oasis-####-xacml-1.0.pdf

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

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

Editors:

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

Committee members:

Anne Anderson, Sun Microsystems, Anne.Anderson@Sun.com
Bill Parducci, Overxeer, bill.parducci@overxeer.com
Carlisle Adams, Entrust, carlisle.adams@entrust.com
Don Flinn, Quadrasis, Don.Flinn@hitachisoftware.com
Gerald Brose, Xtradyne, Gerald.Brose@xtradyne.com
Hal Lockhart, Entegrity, hal.lockhart@entegrity.com
Konstantin Beznosov, Quadrasis, konstantin.beznosov@quadrasis.com
Michiharu Kudo, IBM, kudo@jp.ibm.com
Polar Humenn, Self, polar@syr.edu
Simon Godik, Overxeer, simon.godik@overxeer.com
Steve Andersen, OpenNetwork, sanderson@opennetwork.com
Steve Crocker, Pervasive Security Systems, steve.crocker@pervasivesec.com
Tim Moses, Entrust, tim.moses@entrust.com

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

Errata

Errata can be found at the following location:

http://www.oasis-open.org/committees/xacml/repository/errata-001.pdf

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 "NotApplicable", 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 boolean 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 boolean 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 `xacml:` stands for the XACML policy namespace.
- The prefix `xacml-context:` stands for the XACML context namespace.
- 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 XQuery 1.0 and XPath 2.0 Function and Operators 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
```

The XML Signature **[DS]** 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 particular ***decision request***.
- 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 "NotApplicable", 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 data-type combinations, and its return data-type is also specified. Therefore, data-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 the XQuery 1.0 and XPath 2.0 Functions and Operators specification **[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 ***policy 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.

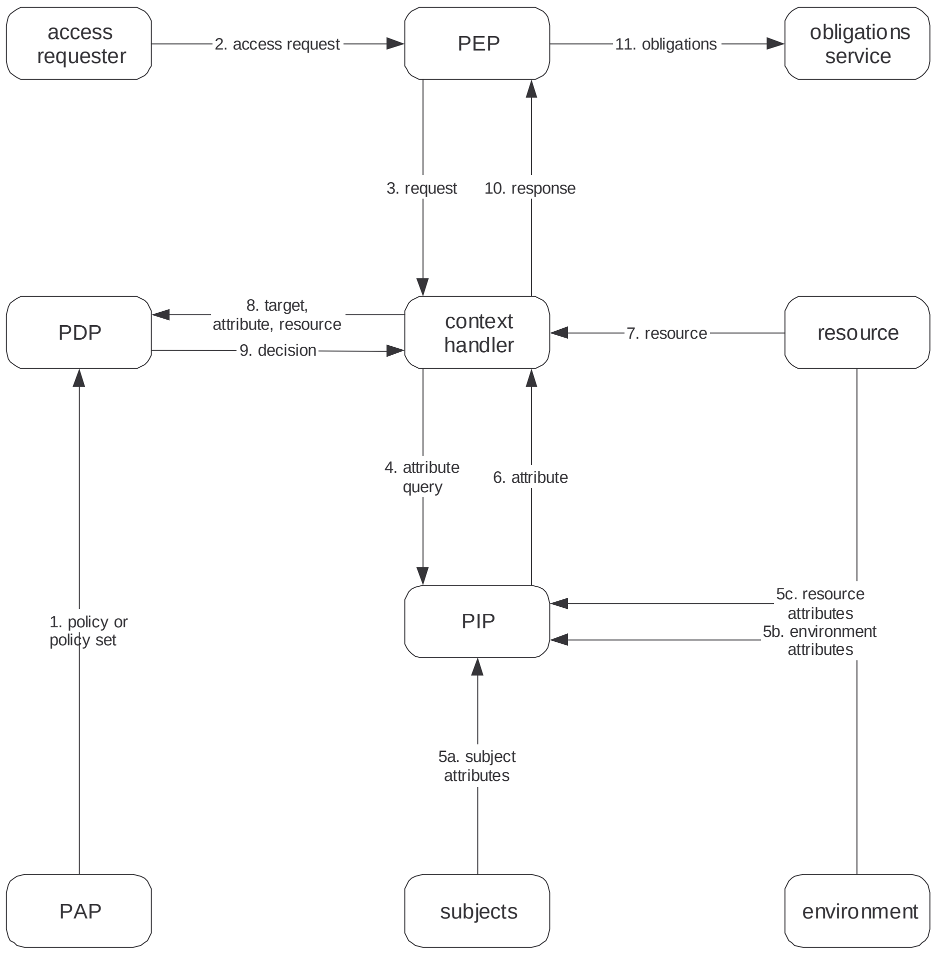

Figure 1 - Data-flow diagram

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 ***policy sets*** and make them available to the ***PDP***. These ***policies*** or ***policy sets*** 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*** sends a ***decision request,*** including the ***target,*** to the ***PDP***. The ***PDP*** identifies the ***applicable policy*** and retrieves the required ***attributes*** and (optionally) the ***resource*** from the ***context handler***. 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.

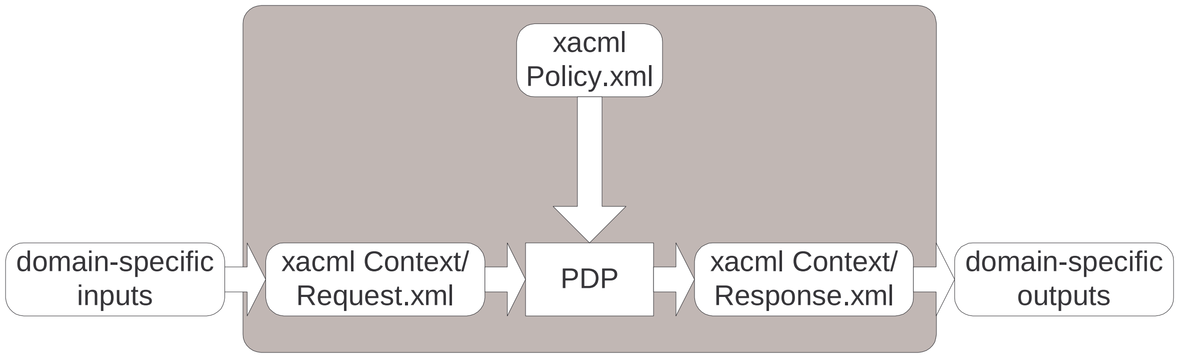

Figure 2 - XACML context

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.

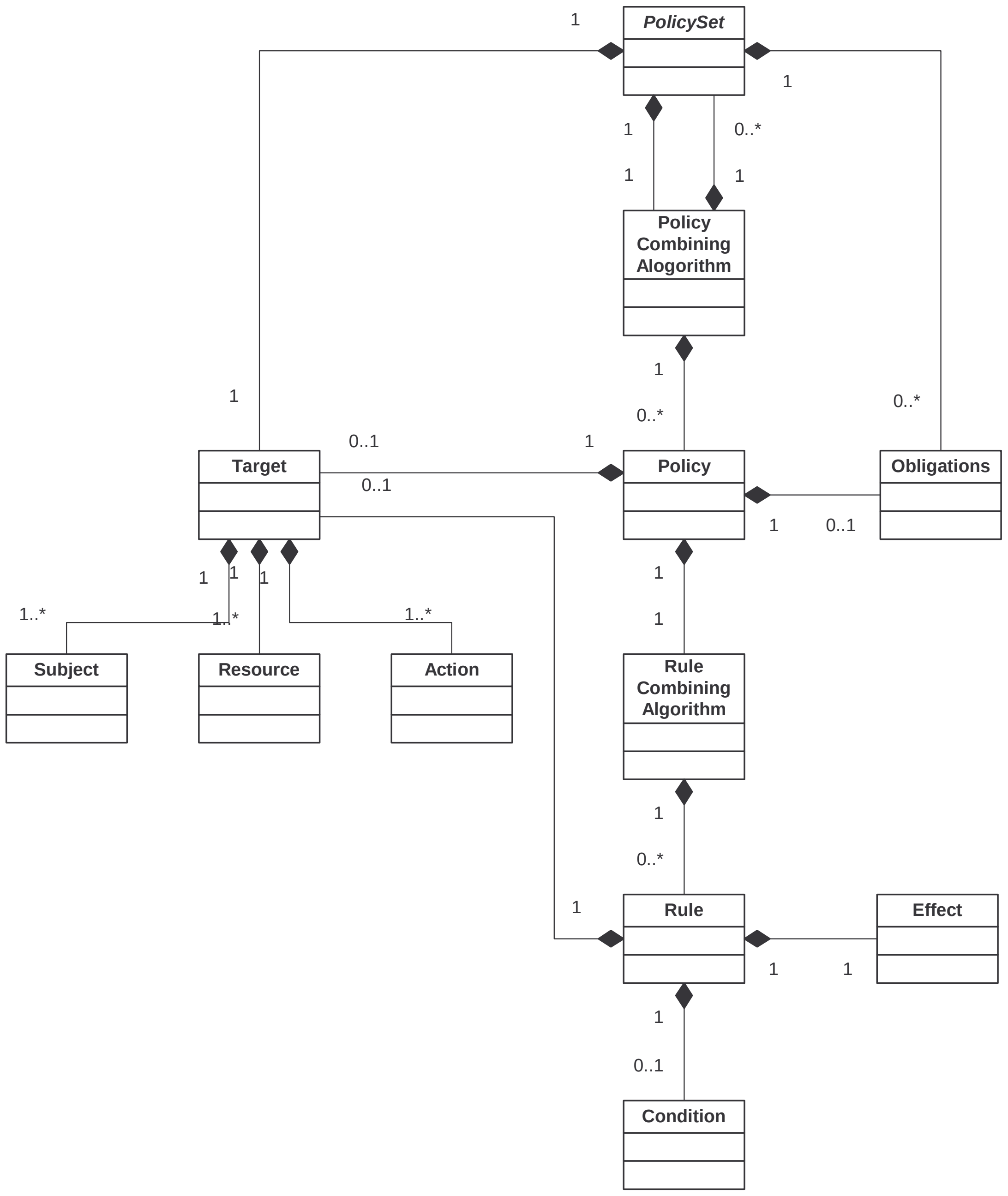

Figure 3 - Policy language model

3.3.1 Rule

A ***rule*** is the most elementary unit of ***policy***. It may exist in isolation only *within* one of the major actors of the XACML domain. In order to exchange ***rules*** between major actors, they must be encapsulated in a ***policy***. A ***rule*** can be evaluated on the basis of its contents. 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 data-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 ***target*** of the `<Rule>` is the same as that of 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">
```

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

[p02] introduces the XACML Policy itself.

[p03-p05] are XML namespace declarations.

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

[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***.

[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>
```

[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>
```

[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">
```

[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***).

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

[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 “NotApplicable”. 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>
```

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

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

[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 “NotApplicable” 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. [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 “NotApplicable”.

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

[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).

[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/cs-xacml-schema-context-01.xsd">
```

[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 [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***. [c18-c23] describe the identity of the ***action*** Bart Simpson wishes to take, which is "read".

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

[c24] closes the ***request context***. A more complex ***request context*** may have contained some ***attributes*** not associated with 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 "NotApplicable" 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/cs-xacml-schema-context-
          01.xsd">
```

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

```
[r05]      <Result>
[r06]       <Decision>NotApplicable</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 "NotApplicable". A ***policy*** can return "Permit", "Deny", "NotApplicable" or "Indeterminate".

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

[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>
      <patient-number
http://www.w3.org/2001/XMLSchema#type="string">555555</patient-number>
   </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:xsi=”http://www.w3.org/2001/XMLSchema-instance”
[04] <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject">
[05]    <Attribute AttributeId=
[06]    "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
[07]    DataType=
[08]    ”urn:oasis:names:tc:xacml:1.0.data-type:x500name”
[09]    Issuer="www.medico.com"
[10]    IssueInstant="2001-12-17T09:30:47-05:00">
[11]       <AttributeValue>CN=Julius Hibbert</AttributeValue>
[12]    </Attribute>
[13]    <Attribute AttributeId=
[14]    "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
[15]    DataType=”http://www.w3.org/2001/XMLSchema#string”
[16]    Issuer="www.medico.com"
[17]    IssueInstant="2001-12-17T09:30:47-05:00">
[18]       <AttributeValue>physician</AttributeValue>
[19]    </Attribute>
[20]    <Attribute AttributeId=
[21]       "urn:oasis:names:tc:xacml:1.0:example:attribute:physician-id"
[22]    DataType=”http://www.w3.org/2001/XMLSchema#string”
[23]    Issuer="www.medico.com"
[24]    IssueInstant="2001-12-17T09:30:47-05:00">
[25]       <AttributeValue>jh1234</AttributeValue>
[26]    </Attribute>
[27] </Subject>
[28] <Resource>
```

```
[29]     <ResourceContent>
[30]        <md:record
[31]        xmlns:md="//http:www.medico.com/schemas/record.xsd">
[32]           <md:patient>
[33]              <md:patientDoB>1992-03-21</md:patientDoB>
[34]           </md:patient>
[35]           <!-- other fields -->
[36]        </md:record>
[37]     </ResourceContent>
[38]     <Attribute AttributeId=
[39]     "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
[40]     DataType=”http://www.w3.org/2001/XMLSchema#string”>
[41]        <AttributeValue>
[42]           //medico.com/records/bart-simpson.xml#
[43]              xmlns(md=//http:www.medico.com/schemas/record.xsd)
[44]              xpointer(/md:record/md:patient/md:patientDoB)
[45]        </AttributeValue>
[46]     </Attribute>
[47]     <Attribute AttributeId=
[48]           "urn:oasis:names:tc:xacml:1.0:resource:xpath"
[49]           DataType=”http://www.w3.org/2001/XMLSchema#string”>
[50]        <AttributeValue>
[51]           xmlns(md=http:www.medico.com/schemas/record.xsd)
[52]              xpointer(/md:record/md:patient/md:patientDoB)
[53]        </AttributeValue>
[54]     </Attribute>
[55]     <Attribute AttributeId=
[56]        ”urn:oasis:names:tc:xacml:1.0:resource:target-namespace”
[57]        DataType=”http://www.w3.org/2001/XMLSchema#string”>
[58]        <AttributeValue>
[59]           http://www.medico.com/schemas/record.xsd
[60]        </AttributeValue>
[61]     </Attribute>
[62]  </Resource>
[63]  <Action>
[64]     <Attribute AttributeId=
[65]     "urn:oasis:names:tc:xacml:1.0:action:action-id"
[66]     DataType=”http://www.w3.org/2001/XMLSchema#string”>
[67]        <AttributeValue>read</AttributeValue>
[68]     </Attribute>
[69]  </Action>
[70] </Request>
```

[02]-[03] Standard namespace declarations.

[04]-[27] ***Subject*** attributes are placed in the `Subject` section of the `Request`. Each ***attribute*** consists of the ***attribute*** meta-data and the ***attribute*** value.

[04] Each `Subject` element has `SubjectCategory` xml 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.

[05]-[12] ***Subject*** `subject-id` ***attribute***.

[13]-[19] ***Subject*** `role` ***attribute***.

[20]-[26] ***Subject*** `physician-id` ***attribute***.

[28]-[62] ***Resource*** attributes are placed in the `Resource` section of the `Request`. Each ***attribute*** consists of ***attribute*** meta-data and an ***attribute*** value.

[29]-[36] ***Resource*** content. The XML document that is being requested is placed here.

[38]-[46] ***Resource*** identifier.

[47]-[61] 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.

[47]-[54] The XPath location path in the “`resource-id`” attribute is extracted and placed in the `xpath` attribute.

[55]-[61] ***Resource*** `target-namespace` ***attribute***.

[63]-[69] ***Action attributes*** are placed in the `Action` section of the `Request`.

[64]-[68] ***Action*** identifier.

4.2.3 Example plain-language rules

The following plain-language rules are to be enforced:

Rule 1: A person, identified by his or her patient number, 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]                <AttributeValue
     DataType=”http://www.w3.org/2001/XMLSchema#string”>
[24]                   http://www.medico.com/schemas/record.xsd
[25]                </AttributeValue>
[26]                <ResourceAttributeDesignator AttributeId=
[27]             "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
     DataType=”http://www.w3.org/2001/XMLSchema#string”/>
[28]             </ResourceMatch>
[29]             <!-- match requested xml element -->
[30]             <ResourceMatch
     MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
[31]                <AttributeValue
     DataType=”http://www.w3.org/2001/XMLSchema#string”>/md:record</AttributeV
     alue>
[32]                <ResourceAttributeDesignator AttributeId=
[33]                   "urn:oasis:names:tc:xacml:1.0:resource:xpath"
     DataType=”http://www.w3.org/2001/XMLSchema#string”/>
[34]             </ResourceMatch>
[35]          </Resource>
[36]        </Resources>
[37]        <Actions>
[38]          <Action>
[39]             <ActionMatch
     MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[40]                <AttributeValue
     DataType=”http://www.w3.org/2001/XMLSchema#string”>read</AttributeValue>
[41]                <ActionAttributeDesignator AttributeId=
[42]                "urn:oasis:names:tc:xacml:1.0:action:action-id"
     DataType=”http://www.w3.org/2001/XMLSchema#string”/>
[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:patient-number/text()"
           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 the value of the first argument matches any of the values selected by 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]-[25] Literal attribute value to match.

[26]-[27] 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.

[30]-[34] The `ResourceMatch`. This match compares the results of two XPath expressions. The first XPath expression is `/md:record` and the second XPath expression is the location path to the requested xml element. 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] The literal XPath expression to match. The `md` prefix is resolved using a standard namespace declaration.

[32]-[33] 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.

[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 the value of the first argument matches any of the values selected by 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] The ***Attribute*** value to match. This is an ***action*** name.

[41]-[42] 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.

[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 `patient-number` ***subject attribute*** is equal to the patient-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 `urn:oasis:names:tc:xacml:1.0:function:string-equal`; this function takes two arguments of the "`http://www.w3.org/2001/XMLSchema#string`" data-type.

[50] The first argument to the `urn:oasis:names:tc:xacml:1.0: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 `urn:oasis:names:tc:xacml:1.0:function:string-equal` takes arguments of the "`http://www.w3.org/2001/XMLSchema#string`" data-type and `SubjectAttributeDesignator` selects a ***bag*** of "`http://www.w3.org/2001/XMLSchema#string`" values, "`urn:oasis:names:tc:xacml:1.0: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 "`urn:oasis:names:tc:xacml:1.0: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 "`urn:oasis:names:tc:xacml:1.0:function:string-equal`" takes arguments of the "`http://www.w3.org/2001/XMLSchema#string`" data-type and the `AttributeSelector` selects a ***bag*** of "`http://www.w3.org/2001/XMLSchema#string`" values, "`urn:oasis:names:tc:xacml:1.0: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".

```
<?xml version="1.0" encoding="UTF-8"?>
<Rule
 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"
 RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
 Effect="Permit">
 <Description>
    A person may read any medical record in the
    http://www.medico.com/records.xsd namespace
    for which he or she is the designated parent or guardian,
    and for which the patient is under 16 years of age
 </Description>
 <Target>
    <Subjects>
       <AnySubject/>
    </Subjects>
    <Resources>
       <Resource>
          <!-- match document target namespace -->
          <ResourceMatch
 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
             <AttributeValue
 DataType=”http://www.w3.org/2001/XMLSchema#string”>
                http://www.medico.com/schemas/record.xsd
             </AttributeValue>
             <ResourceAttributeDesignator AttributeId=
          "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
 DataType=”http://www.w3.org/2001/XMLSchema#string”/>
          </ResourceMatch>
          <!-- match requested xml element -->
          <ResourceMatch
 MatchId=”urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
             <AttributeValue
 DataType=”http://www.w3.org/2001/XMLSchema#string”>/md:record</AttributeV
 alue>
             <ResourceAttributeDesignator AttributeId=
                "urn:oasis:names:tc:xacml:1.0:resource:xpath"
 DataType=”http://www.w3.org/2001/XMLSchema#string”/>
          </ResourceMatch>
       </Resource>
    </Resources>
    <Actions>
       <Action>
          <!-- match 'read' action -->
          <ActionMatch
 MatchId=”urn:oasis:names:tc:xacml:1.0:function:string-equal">
             <AttributeValue
 DataType=”http://www.w3.org/2001/XMLSchema#string”>read</AttributeValue>
             <ActionAttributeDesignator AttributeId=
                "urn:oasis:names:tc:xacml:1.0:action:action-id"
 DataType=”http://www.w3.org/2001/XMLSchema#string”/>
          </ActionMatch>
       </Action>
    </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/text()"
[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/text()"
     DataType="http://www.w3.org/2001/XMLSchema#date">
[77]             </AttributeSelector>
[78]          </Apply>
[79]          <AttributeValue DataType="http://www.w3.org/TR/2002/WD-xquery-
     operators-20020816#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 "`urn:oasis:names:tc:xacml:1.0: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 "`urn:oasis:names:tc:xacml:1.0:function:string-equal`" that takes 2 arguments of "`http://www.w3.org/2001/XMLSchema#string`" data-type.

[52] Since "`urn:oasis:names:tc:xacml:1.0:function:string-equal`" takes arguments of the "`http://www.w3.org/2001/XMLSchema#string`" data-type, "`urn:oasis:names:tc:xacml:1.0: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. "`urn:oasis:names:tc:xacml:1.0: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] "`urn:oasis:names:tc:xacml:1.0: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 data-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 data-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] "`urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal`" is used to compute the difference of two dates.

[67] "`urn:oasis:names:tc:xacml:1.0: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 data-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] "`urn:oasis:names:tc:xacml:1.0: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 "`http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration`".

[73] "`urn:oasis:names:tc:xacml:1.0: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 data-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">
            <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>
               http://www.medico.com/schemas/record.xsd
            </AttributeValue>
            <ResourceAttributeDesignator AttributeId=
         "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
DataType=”http://www.w3.org/2001/XMLSchema#string”/>
         </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">
            <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>physician</AttributeVa
lue>
            <SubjectAttributeDesignator AttributeId=
   "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
DataType=”http://www.w3.org/2001/XMLSchema#string”/>
         </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]             <AttributeValue
     DataType=”http://www.w3.org/2001/XMLSchema#string”>
[57]                /md:record/md:medical
[58]             </AttributeValue>
[59]             <ResourceAttributeDesignator AttributeId=
[60]                "urn:oasis:names:tc:xacml:1.0:resource:xpath"
     DataType=”http://www.w3.org/2001/XMLSchema#string”/>
[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]             <AttributeValue
     DataType=”http://www.w3.org/2001/XMLSchema#string”>write</AttributeValue>
[069]            <ActionAttributeDesignator AttributeId=
[070]      "urn:oasis:names:tc:xacml:1.0:action:action-id"
     DataType=”http://www.w3.org/2001/XMLSchema#string”/>
[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/text()"
[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 data-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#string" 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]             <AttributeValue
                 DataType=”http://www.w3.org/2001/XMLSchema#string”>administrato
                 r</AttributeValue>
[20]             <SubjectAttributeDesignator AttributeId=
[21]             "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
                 DataType=”http://www.w3.org/2001/XMLSchema#string”/>
[22]          </SubjectMatch>
[23]       </Subject>
[24]    </Subjects>
[25]    <Resources>
[26]       <Resource>
[27]          <!-- match document target namespace -->
```