eXtensible Access Control Markup Language (XACML) Version 2.0

OASIS Standard, 1 Feb 2005

Document Identifier: oasis-access_control-xacml-2.0-core-spec-os

Location: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

Editor:

Tim Moses, Entrust Inc. (tim.moses@entrust.com)

Abstract:

This specification defines version 2.0 of the extensible access-control markup language.

Status:

This version of the specification is an approved OASIS Standard within the OASIS Access Control TC.

Access Control TC members should send comments on this specification to the xacml@lists.oasis-open.org list. Others may use the following link and complete the comment form: http://oasis-open.org/committees/comments/form.php?wg_abbrev=xacml.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Access Control TC web page (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml).

For any errata page for this specification, please refer to the Access Control TC web page (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml).

The non-normative errata page for this specification is located at

www.oasis-open.org/committees/access-control.

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*** (see also – ***named attribute***)

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 ***predicates*** 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 ***predicates*** 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***

Named attribute – A specific instance of an ***attribute***, determined by the ***attribute*** name and type, the identity of the ***attribute*** holder (which may be of type: ***subject***, ***resource***, ***action*** or ***environment***) and (optionally) the identity of the issuing authority

Obligation - An operation specified in a ***policy*** or ***policy set*** that should be performed by the ***PEP*** 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***. This term is defined in a joint effort by the IETF Policy Framework Working Group and the Distributed Management Task Force (DMTF)/Common Information Model (CIM) in [RFC3198]. This term corresponds to "Access Decision Function" (ADF) in [ISO10181-3].

Policy enforcement point (PEP) - The system entity that performs ***access control***, by making ***decision requests*** and enforcing ***authorization decisions***. This term is defined in a joint effort by the IETF Policy Framework Working Group and the Distributed Management Task Force (DMTF)/Common Information Model (CIM) in [RFC3198]. This term corresponds to "Access Enforcement Function" (AEF) in [ISO10181-3].

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

Type Unification - The method by which two type expressions are "unified". The type expressions are matched along their structure. Where a type variable appears in one expression it is then "unified" to represent the corresponding structure element of the other expression, be it another variable or subexpression. All variable assignments must remain consistent in both structures. Unification fails if the two expressions cannot be aligned, either by having dissimilar structure, or by having instance conflicts, such as a variable needs to represent both "xs:string" and "xs:integer". For a full explanation of ***type unification***, please see [**Hancock**].

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 2.0 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 v2.0 of XACML are required to deny ***access*** unless they understand and can discharge all of 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.

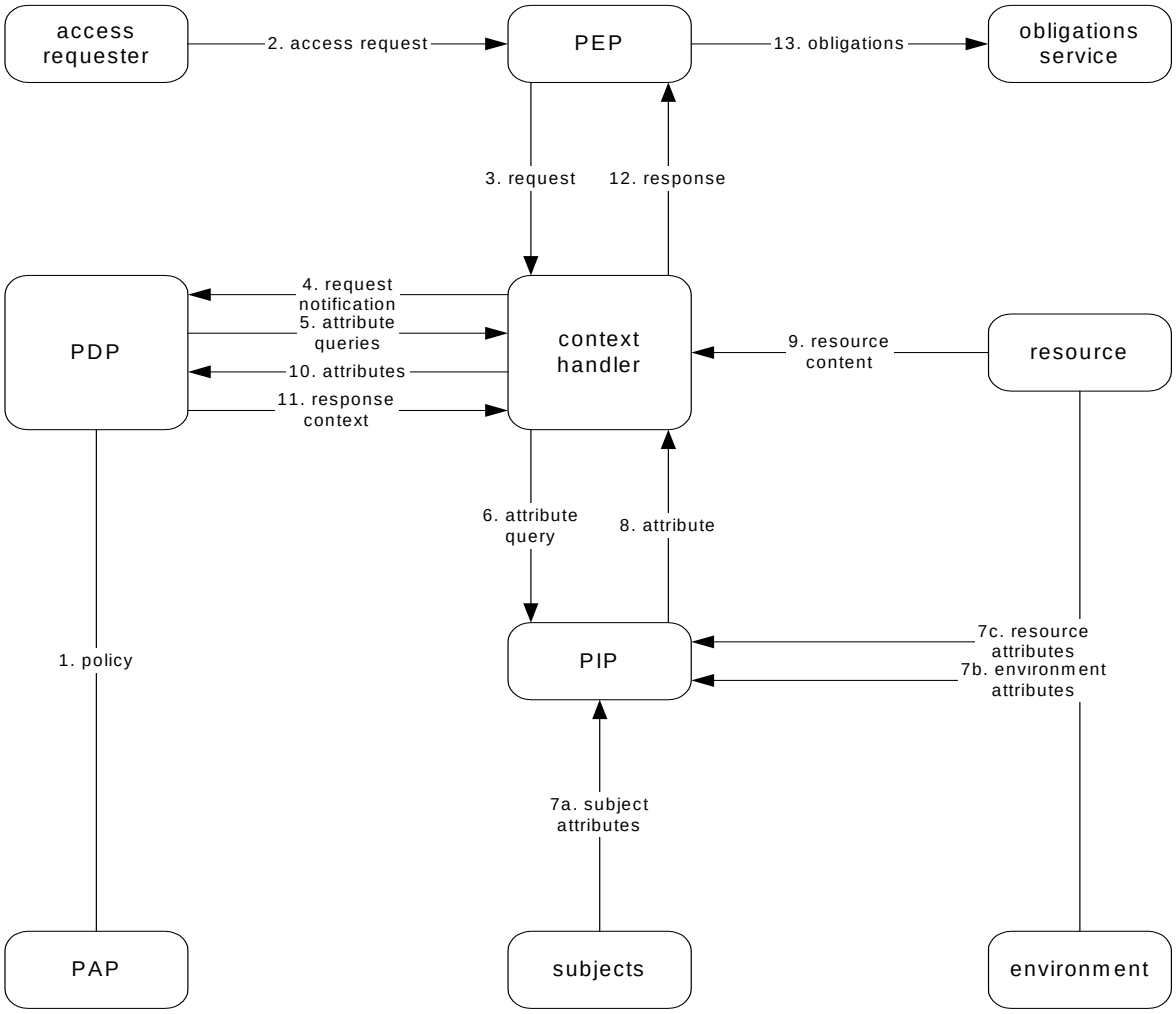

Figure 1 - Data-flow diagram

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, action*** and ***environment***.
4. The ***context handler*** constructs an XACML request ***context*** and sends it to the ***PDP***.
5. The ***PDP*** requests any additional ***subject***, ***resource, action*** and ***environment attributes*** from the ***context handler***.

6. The context handler requests the attributes from a ***PIP***.

7. The ***PIP*** obtains the requested ***attributes***.

8. The ***PIP*** returns the requested ***attributes*** to the ***context handler***.

9. Optionally, the ***context handler*** includes the ***resource*** in the ***context***.

10. The ***context handler*** sends the requested ***attributes*** and (optionally) the ***resource*** to the ***PDP***. The ***PDP*** evaluates the ***policy***.

11. The ***PDP*** returns the response ***context*** (including the ***authorization decision***) to the ***context handler***.

12. The ***context handler*** translates the response ***context*** to the native response format of the ***PEP***. The ***context handler*** returns the response to the ***PEP***.

13. The ***PEP*** fulfills the ***obligations***.

14. (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 over the ***context***, or attribute designators that identify the ***attribute*** by ***subject, resource, action*** or ***environment*** and its identifier, data-type and (optionally) its issuer. 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.

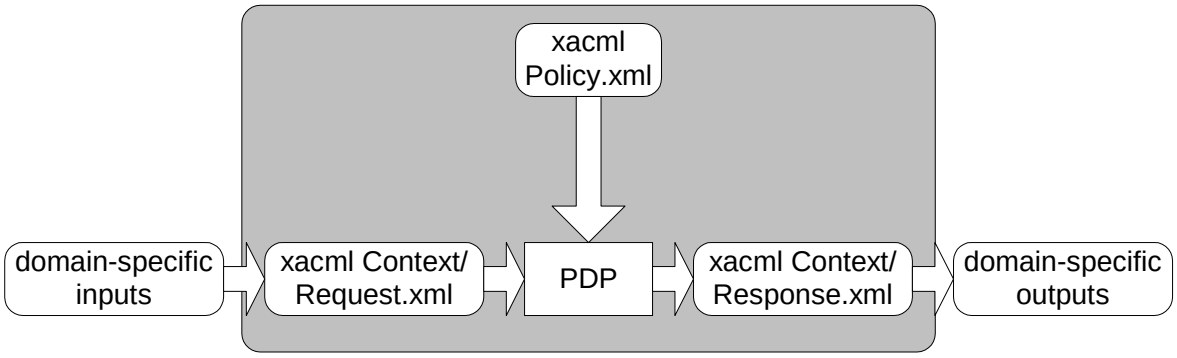

Figure 2 - XACML context

Note: The ***PDP*** is not required to operate directly on the XACML representation of a policy. It may operate directly on an alternative representation.

See Section 7.2.5 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.

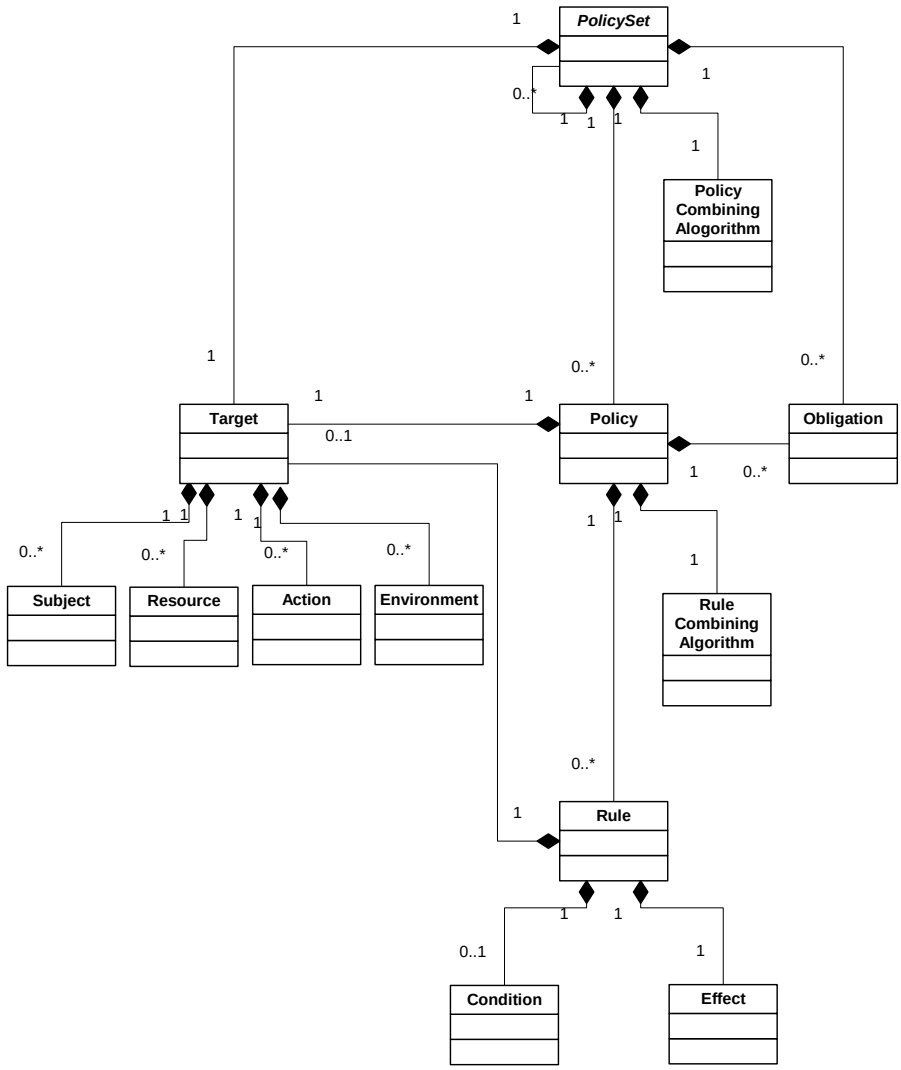

Figure 3 - Policy language model

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***;
- ***actions*** and
- ***environment***

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 the corresponding entity is omitted from the ***target***. An XACML ***PDP*** verifies that the matches defined by the ***target*** are satisfied by the ***subjects, resource, action*** and ***environment attributes*** in the request ***context***. ***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 "med.example.com" is a legal RFC822 name identifying the set of mail addresses hosted by the med.example.com mail server. And the XPath/XPointer value `//xacml-context:Request/xacml-context:Resource/xacml-context: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).

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 <Policy Set>, <Policy> or <Rule> element contains a <Target> element that specifies the set of ***subjects***, ***resources, actions*** and ***environments*** to which it applies. The <Target> of a <Policy Set> or <Policy> may be declared by the writer of the <Policy Set> or <Policy>, or it may be calculated from the <Target> elements of the <Policy Set>, <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 <Policy Set> or <Policy> (the "outer component") is calculated as the *union* of all the <Target> elements of the referenced <Policy Set>, <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***. A ***policy*** may have combining parameters that affect the operation of the ***rule-combining algorithm***.

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

3.3.2.3. Obligations

Obligations may 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 7.14 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***. A ***policy set*** may have combining parameters that affect the operation of 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 7.14 explains which ***obligations*** are to be returned.