
1
2
3
4
5
6
7
8

XACML language proposal

Bill Parducci	Bill Parducci
Carlisle Adams	Entrust
Ernesto Damiani	University of Milan
Hal Lockhart	Entegrity
Ken Yagen	Crosslogix
Michiharu Kudo	IBM, Japan
Pierangela Samarati	University of Milan
Simon Godik	CrossLogix
Tim Moses	Entrust

9
10

10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

Table of contents

1. Glossary.....	3
1.1. Preferred terms	3
1.2. Related terms	4
2. Introduction.....	4
3. Models.....	4
3.1. Data-flow model.....	5
3.2. Language model	6
3.2.1. Elements of the access control policy.....	6
3.2.2. Reserved identifiers	7
3.2.3. Principal expression	7
3.2.4. Resource expression	8
3.2.5. Action.....	9
3.2.6. Environment expression	9
3.2.7. Things to discuss	11
3.2.8. Principal/role/attribute.....	13
3.2.9. Resource/classification/attribute.....	13
3.2.10. Environment/attribute.....	14
3.2.11. Policy/action/resource/classification	14
3.2.12. Rule/pre-condition/predicate	15
3.2.13. Post-condition.....	16
3.2.14. Attribute identification	16
3.3. Administrative model	16
4. Policy syntax	18
5. Applicable policy	19
6. Policy	19
7. Rule	19
8. Pre-condition	20
8.1. And.....	20
8.2. Or.....	20
8.3. Not.....	20
8.4. Predicate.....	20
8.5. Present	21
8.6. Equality	21
8.7. Greater or equal.....	21
8.8. Less or equal.....	22
8.9. Sub-set of.....	22
8.10. Super-set of	22
8.11. Non-null Set Intersection.....	22
8.12. External function	23
8.13. Post-condition.....	23
8.14. Referenced data	23
8.15. Attribute reference.....	23
8.16. Second operand	24
8.17. Hard-coded value	24
9. References	24

56 **Chapter One**

57 **1. Glossary**

58 **1.1. Preferred terms**

59 **Access** - Performing an *action* on a *resource*

60 **Access control** - Controlling *access* in accordance with *applicable policy*

61 **Action** - Operation that may be performed on *resource*

62 **Applicable policy** - The complete set of *rules* that governs *access* for a specific
63 *resource*

64 **Attribute** - Characteristic of a *principal*, *resource* or *environment* that may be
65 referenced by a *pre-condition*

66 **Authorization decision** - The result of evaluation of *applicable policy*. A function with
67 BOOLEAN range and, optionally, a set of *post-conditions*

68 **Classification** - A set of *attributes* relevant to a *resource*

69 **Context** - The intended use of information revealed as a result of *access*.

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

71 **Environment** - The set of *attributes* that may be referenced by *pre-conditions* and that
72 are independent of a particular *principal* and *resource*

73 **Information request** - The request by the *PDP* to the *PIP* for one or more *environment*
74 *attributes*

75 **Policy** - (see **Applicable policy**)

76 **Policy conflict** - The state that exists when two or more *pre-conditions*, forming part of
77 *applicable policy*, individually yield conflicting results

78 **Policy decision point (PDP)** - The system entity that evaluates *applicable policy*

79 **Policy enforcement point (PEP)** - The system entity that performs *access control*

80 **Policy information point (PIP)** - The system entity that acts as the source of
81 *environment attributes*

82 **Policy administration point (PAP)** - The system entity that creates *applicable policy*

83 **Policy mediation point (PMP)** - The system entity that resolves *policy conflict*

84 **Policy retrieval point (PRP)** - The system entity that ensures *applicable policy* is
85 complete

86 **Post-condition** - A process specified in a *rule* that must be completed in conjunction
87 with *access*. There are two types of post-condition: an *internal post-condition* must be
88 executed by the *PDP* prior to the issuance of a "permit" response, and an *external post-*
89 *condition* must be executed by the *PEP* prior to permitting *access*

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

91 **Pre-condition** - A *predicate* or logically-combined set of *predicates*

92 **Principal** - A system entity that can be referenced by a *pre-condition*

93 **Resource** - Data, service, or system component

94 **Role** - A set of *attributes* relevant to a *principal*

95 **Rule** - The combination of a *pre-* and one or more *post-conditions*

96 **1.2. Related terms**

97 In the field of access control and authorization there are several closely related terms in
98 common use. For purposes of precision and clarity, certain of these terms are not used in
99 this specification.

100 For instance, the term *attribute* is used in place of the terms: privilege, permission, right,
101 authorization and entitlement.

102 The terms "subject" and "user" are also in common use. But, we use the term *principal*
103 in this specification.

104 The terms "object" and "target" are also in common use, but we use the term *resource* in
105 this specification.

106 While the term "group" is commonly used with a meaning that is distinct from that of
107 *role*, the distinction has no significance in the domain of XACML, therefore, the term
108 group is not used here.

109 **2. Introduction**

110 XACML specifies a mark-up language for access control policies. It is intended to be
111 used in conjunction with SAML assertions and messages.

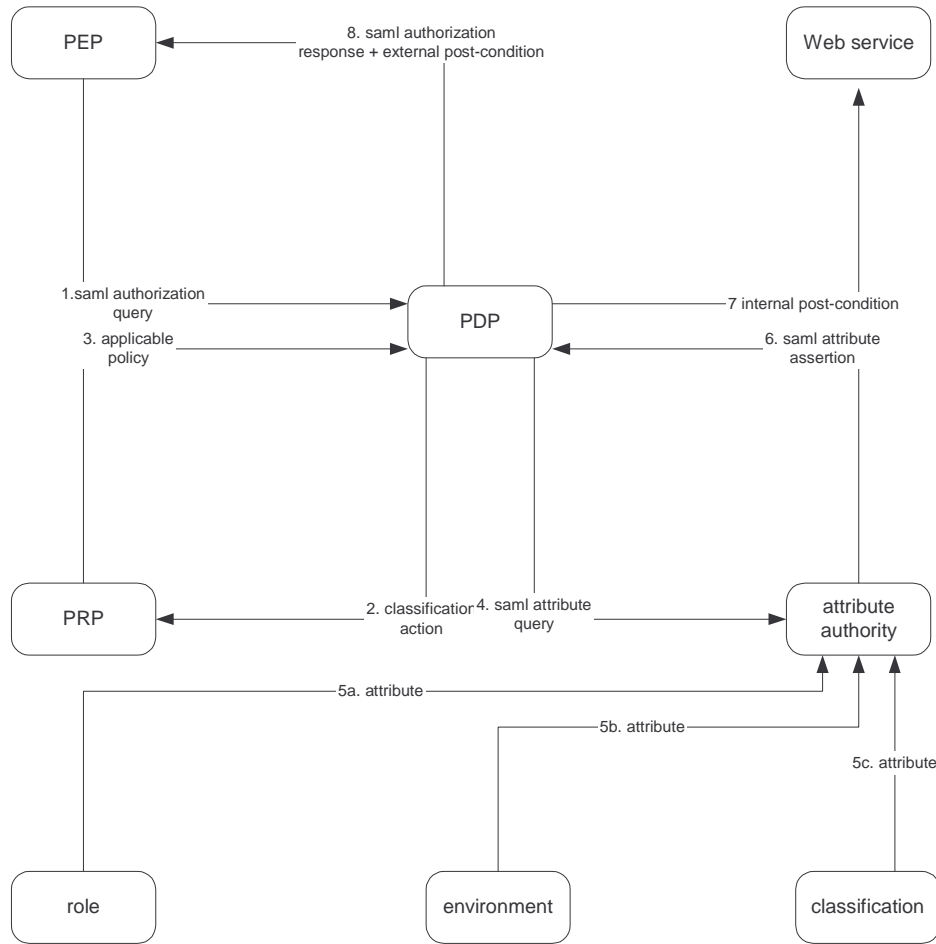
112 **3. Models**

113 *The information in this section is non-normative.*

114 The context and schema of XACML are described in three models that elaborate different
115 aspects of its operation. These models are: the data-flow model, the language model and
116 the administrative model. They are described in the following sub-sections.

117 3.1. Data-flow model

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



119

120

Figure 1 - Data-flow diagram

121 Some of the data-flows shown in the diagram may be facilitated by a repository. For
122 instance, the communications between the PDP and the attribute authority may be
123 facilitated by a repository, or the communications between the PDP and the PRP may be
124 facilitated by a repository. The XACML specification is not intended to place restrictions
125 on the location of any such repository, or indeed to prescribe a particular communication
126 protocol for any of the data-flows.

127 The model operates according to the following steps.

-
- 128 1. The PEP sends a decision request to the PDP, in the form of a SAML [SAML]
129 authorization query. The decision request contains some or all of the attributes
130 required by the PDP to render a decision, in accordance with policy.
 - 131 2. The PDP locates and retrieves the policy instance applicable to the decision request
132 from the PRP. It uses the resource classification and the requested action to identify
133 the correct policy. The means by which the PDP determines the classification of the
134 resource is out of scope for this specification. However, in the case where the
135 resource is an XML document, its classification may be an attribute of the top-level
136 element of the resource.
 - 137 3. The PRP returns the policy to the PDP in the form of an XACML instance.
 - 138 4. The PDP examines the decision request and the policy to ascertain whether it has all
139 the attribute values required to render an authorization decision. If it does not, then it
140 requests attributes from suitable attribute authorities in the form of SAML attribute
141 queries [SAML].
 - 142 5. The attribute authorities may locate and retrieve the requested attributes from other
143 systems by a means, and in a form, that is out of scope for this specification.
 - 144 6. The attribute authorities return the requested attributes to the PDP in the form of
145 SAML attribute responses containing SAML attribute assertions.
 - 146 7. The PDP evaluates the policy instance. In the case where the policy instance contains
147 internal post-conditions, the PDP executes those post-conditions.
 - 148 8. If the policy were to evaluate to TRUE, and the internal post-conditions were to
149 execute successfully, then the PDP returns an authorization decision, in the form of a
150 SAML authorization response, to the PEP containing the "permit" result code and any
151 external post-conditions.

152 **3.2. Language model**

153 3.2.1. Elements of the access control policy

154 An access control policy states regulations governing access to resources, and therefore
155 how the system should respond to requests that principals can submit. The access control
156 policy comprises of access rules stating which accesses should (or should not) be allowed
157 and, possibly, under which conditions such permissions or denials for the access apply.

158 We therefore start by identifying the various elements of the access rules. At this stage
159 we characterize the different access components with respect to their format and
160 semantics. Later we will define the precise syntax (a preliminary sketch of the syntax
161 appears in ``XACML Language").

162 An access rule can be seen as comprised of the following elements:

-
- 163 - principal expression: identifies the (dynamic set of) principals to whom the rule applies.
 - 164 - resource expression: identifies the (dynamic set of) resources to which the rule applies.
 - 165 - action expression: identifies the (dynamic set of) actions to which the rule applies.
 - 166 - environment expression: identifies system-dependent and request-dependent conditions
 - 167 to be satisfied for the rule to apply.
 - 168 - post-conditions: defines a set of actions that the access control system (PEP) must
 - 169 execute whenever a rule is applied with respect to a given access request.
 - 170 - if/only if conditional statements.....

171 3.2.2. Reserved identifiers

172 The expressiveness of the language will allow us to specify rules whose applicability will
173 depend on conditions that the principal requesting access, or the resource on which access
174 is requested, satisfy. Access rules are therefore not referred to a specific principal or a
175 specific resource but to a set of them that satisfy given conditions. To provide
176 expressiveness needed to make it possible the specification of such generic rules without
177 the need of introducing variables in the language we introduce the following reserved
178 identifiers:

- 179 - principal: is the principal presenting the request
- 180 - resource: is the resource on which access is requested
- 181 - action: is the action requested

182 3.2.3. Principal expression

183 The principal expression defines the principal, or set of principals, to which the access
184 rule applies. It is a Boolean expression evaluating SAML assertions (i.e., properties)
185 associated with the principal requesting access.

186 SAML assertions can refer to any property of the principals, including groups to which
187 the user making the request belongs or roles (privileged positions) that the user may have
188 activated and present. Groups and role management is outside the scope of the
189 authorization language, we assume information about active roles to be provided through
190 SAML assertions; we assume information about group memberships to be either
191 provided as SAML assertions or to be available at the PIP.

192 Each given assertion term (i.e., elementary component of a principal expression)
193 evaluates the value of a property associated with the requestor as is of the form

194 <SAML-assertion> <comparison operator>

195 <SAML-assertion>

196 or

197 <SAML-assertion> <comparison operator>

198 <constant-value>

199 where the comparison operator is a suitable operator (including <,=<,>=,…) depending
200 on the property type and the XPath language is used to refer to SAML assertions within
201 the specification of the assertion terms.

202 Some examples of principal expressions are as follows:

203 - principal/login_name='bob'

204 user 'bob'

205 - principal/organization='OASIS' AND principal/residence='North-Pole'

206 principals associated with OASIS and living at the North Pole

207 - principal/organization=principal/login_name

208 principals working for an organization owned by themselves.

209 Use of variables as macros (to be completed checking with Ernesto and Simon)

210 3.2.4. Resource expression

211 The resource expression is a Boolean expression of conditional terms that evaluate
212 properties of the resource. Properties appearing in these conditional terms can refer to the
213 resource content or to meta properties associated with the resource. The reserved
214 identifier 'resource' can be used in the specification of generic expression applicable to a
215 (dynamic) set of resources. Resource expressions can be also make use of the keyword
216 'principal' for specifying conditions putting in relationships properties of the resources
217 with that of the principals. Each conditional term in a resource expression is therefore of
218 the form

219 <resource-assertion> <comparison operator><SAML-assertion>

220 or

221 <resource-assertion> <comparison operator> <constant-value>

222 Reference to the meta-property of the resource or to its content

223 depends on and can make use of functions.

224 Some examples of resource expressions are as follows:

225 - resource/creation_date =< `01/01/01`

226 all resources created before January 1, 2001.

227 - resource/owner=principal/login_name

228 all resources owned by the principal making the requests

229 - resource/label=`Top Secret`

230 all resources classified as Top-Secret (note: we are not implying support for a multilevel
231 policy here)

232 3.2.5. Action

233 The action component of the access control rule defines the action, or set of actions, to
234 which the rule refers. An action is identified by a name and may have associated a set of
235 parameters. The parameters can refer to any input/output of the process.

236 Some examples of action expressions are as follows:

237 - action/withdrawal_amount =< 500,000

238 - action/data_recipient IN Doctors

239 3.2.6. Environment expression

240 It's a Boolean expressions of environmental conditions that can evaluate the system state
241 (e.g., date and time) and request parameters (e.g., location from where the principal is
242 connected).

243 IF/ONLY IF CONDITIONS

244 A policy is composed of a set of access control rules. The usual interpretation for a set of
245 rules specifying permissions is to grant all the accesses from which at least a rule is
246 satisfied. The consideration of permissions only permissions, with this interpretation may
247 result limiting in several cases. Negative access control rules (specifying denials) could
248 be used for specifying accesses that should be denied. Introduction of negative
249 authorizations introduces the problem of the different semantics that denials can carry
250 which should be properly represented with different conflict resolution criteria in the
251 model. For the time being we therefore do not consider negative authorizations.
252 Permissions only, however, result limiting. For instance, suppose we want to say that
253 only UScitizens can access a document. In a permission only scenario we could specify a
254 rule stating the permissions but the semantics of the only (meaning no one else can access
255 the document) is not supported, since the insertion of any additional rule can grant the
256 access to some noncitizens. As another example suppose we want to say that access to a

257 given document requires (beside additional conditions to be specified by the security
258 administrator) presentation of a

259 payment certificate (stated as a SAML assertion). In a permission only scenario we
260 should make sure that the payment condition is included in all the rules that apply to the
261 access. Beside being difficult to control, this would introduce complicated rules
262 (intuitively the conditions would have to be repeated in AND in every rule instead of
263 being factored out.

264 Looking at the real world cases, we often find access rules stated in restrictive form rather
265 than in the inclusive positive form just mentioned. By restrictive form we mean rules that
266 state conditions that must be satisfied for an access to be granted and such that, if at least
267 one condition is not satisfied, the access should not be granted. For instance, a rule can
268 state that "access to document-1 can be allowed only to citizens". It is easy to

269 see that such a restriction cannot be simply represented as an permission stating that
270 citizens can be authorized. In fact, while the single authorization brings the desired
271 behavior, its combination with other authorizations may not, leading the only constraint
272 to be not satisfied anymore.

273 A possible approach would be supporting two kinds of rules: restrictions and
274 authorizations. Intuitively, restrictions are useful to specify requirements of the exclusive
275 only if form stated above; while authorizations specify requirements in the traditional
276 positive if form.

277 - RESTRICTIONS: specify requirements that must all be satisfied for an access to be
278 granted. Lack to satisfy any of the requirements that apply to a given request implies the
279 request will be denied.

280 Syntactically, restrictions have the form

281 <request-description> <conditions>

282 where

283 request-description is the principal, resource, action and environment expressions

284 and

285 conditions is a Boolean expression of conditions that every request to which the
286 restriction applies must satisfy.

287 - AUTHORIZATIONS: specify permissions for the access. An access is granted if there
288 is satisfaction of at least one of the permissions that apply to the given request and no
289 restriction is violated.

290 Syntactically, authorizations have the form

291 <request-description}> <conditions}>

292 where request-description} has the same meaning as before and <conditions}> is a
293 Boolean expression of conditions whose satisfaction authorizes the access.

294 Unlike for restrictions, lack of satisfaction of a condition in an authorization simply
295 makes the authorization inapplicable but it does not imply the access will be denied. In
296 particular, access can be authorized if there is at least one authorization that applies to it
297 for which the conditions are satisfied.

298 3.2.7. Things to discuss

299 - purpose of access (discussed in the last concall), still to be inserted

300 - dynamic conditions: conditions that cannot be evaluated but can trigger procedures

301 - post-conditions

302 - content-based filtering: We should decide whether the resource expression can contain
303 conditions evaluating the resource content. The complication arises from the fact that
304 content-querying depends on the specific application/data-model/system.

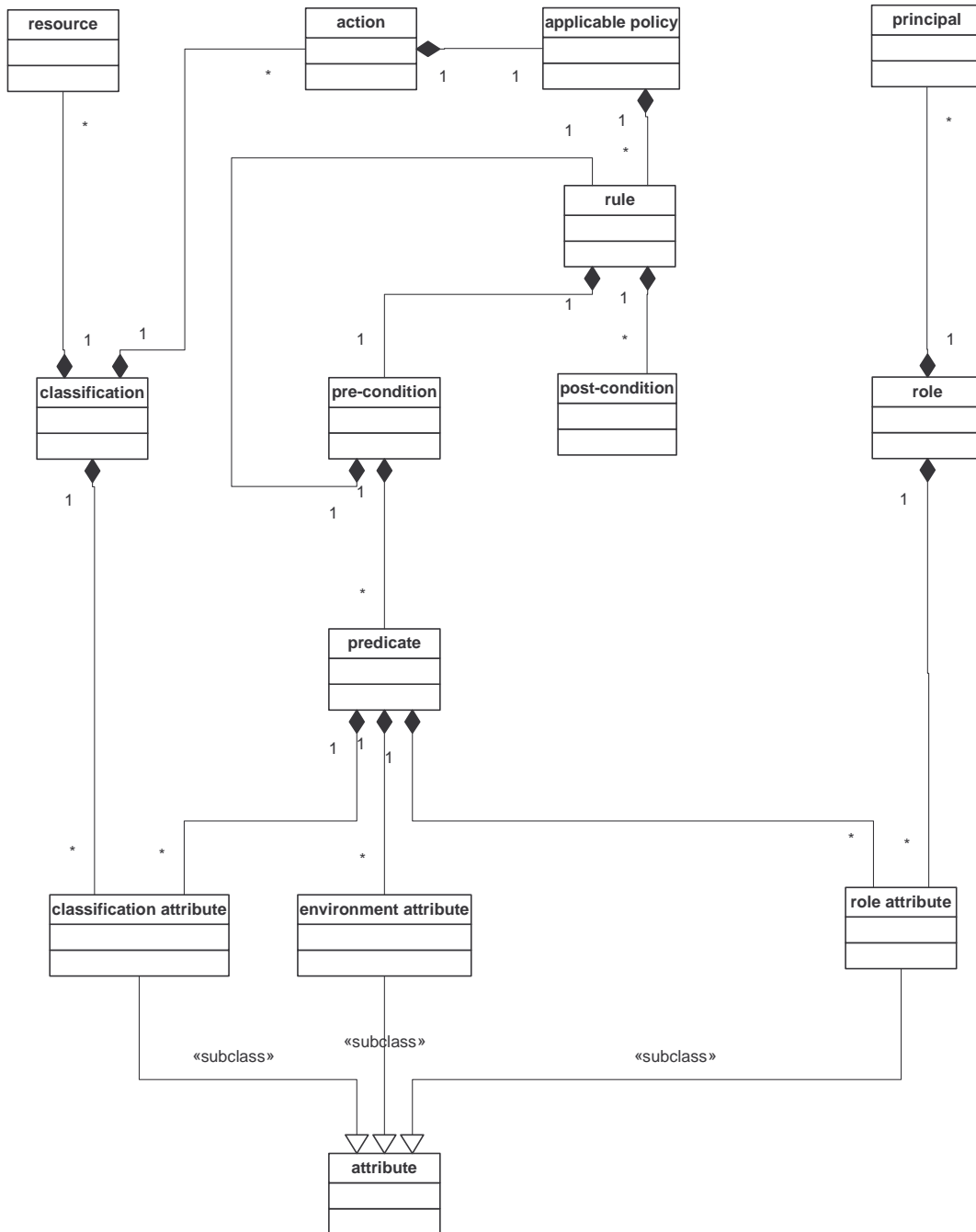
305 - attribute reference (syntactical matter): in the examples we have used naming based
306 notation to refer to parameters of an action. Should we allow (XPath permits it)
307 positional-based notation as well?

308 - examples and simplification: Now the language can seem a bit too complicated. For
309 instance, we need to say ``principal}/login_name='bob', in cases where we would
310 have just said 'bob' in traditional systems. This is however consistent with the fact
311 that for us a principal is not a user 'login-name' but it is characterized through
312 assertions. However, we could find a way to simplify expressions in some cases.

313 - dealing with unknown attributes: SAML assertions (as well as resource properties)
314 are not predefined and can change. What happens if a rule has a condition on some
315 SAML assertions that cannot be found at runtime?

316 - description of run-time behavior of access control

317 The language model is shown in Figure 2.



318

319

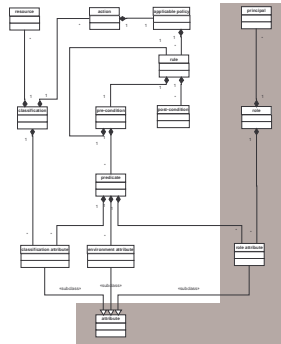
Figure 2 - Policy model

320 The various objects of the model are created by policy administrators, and may (or may
 321 not) be integrity protected using a digital signature or other integrity/authenticity
 322 mechanism. A set of objects may only be protected by the same integrity seal if they
 323 exist at the same place and the same time. Nothing in the model is intended to impose
 324 restrictions on the sequence in which the various objects are created and the combinations
 325 in which they may exist.

326 For purposes of explanation, the language model divides into six sections. These are
327 each described in the following sub-sections.

328 3.2.8. Principal/role/attribute

329 The principal/role/attribute section of the language model is shown in gray in Figure 3.



330

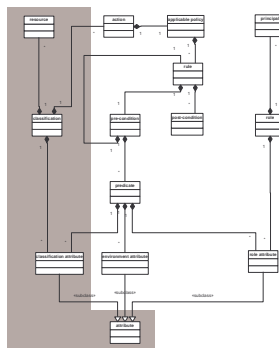
331 **Figure 3 - Principal/role/attribute section of the language model**

332 An authorization request relates to a single principal. XACML policy instances may
333 reference attributes of a particular principal, or a role of the principal. The PDP should
334 use attribute assertions to confirm that the principal occupies a required role. Both the
335 principal and the role may have attributes. For instance, the principal "Joe" may have an
336 attribute of type "role" set equal to the value "purchasing officer". Alternatively, the role
337 "purchasing officer" may have an attribute of type "signing limit" set equal to the value
338 "US\$100,000". Principal and role attributes are asserted by authorities and distributed in
339 the form of SAML attribute assertions. The PDP must check that the attribute values it
340 operates upon are asserted by suitable authorities. This operation is described in Section
341 3.2.14, below.

342 3.2.9. Resource/classification/attribute

343 The resource/classification/attribute section of the language model is shown in gray in
344 Figure 4.

345



346

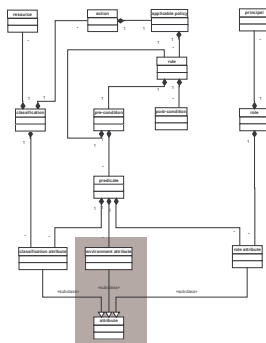
347 **Figure 4 - Resource/classification/attribute section of the language model**

348 An authorization request relates to a single resource. XACML policies may reference
349 attributes of a particular resource or a classification of the resource. The PDP must
350 confirm that the resource occupies the required classification. In the case where the
351 resource is an XML document, it may do this by examining an attribute or element within
352 the resource itself. In other cases, the PDP may use attribute assertions. Both the
353 resource and classification may have attributes. For instance, a purchase order may have
354 an attribute of type "total price" set equal to the value "US\$87,750.00". Alternatively, the
355 classification "capital equipment" may have an attribute of type "category of goods" set
356 equal to the value "computer equipment".

357 The PDP must locate and retrieve resource attributes referenced by the applicable
358 XACML policy instance. In the case where the resource is an XML document, it may do
359 this by examining an attribute or element within the resource itself. In other cases,
360 resource and classification attributes are asserted by authorities and distributed in the
361 form of SAML attribute assertions. The PDP must check that the attribute values it
362 operates upon are asserted by suitable authorities. This operation is described in Section
363 3.2.14, below.

364 3.2.10. Environment/attribute

365 The environment/attribute section of the language model is shown in gray in Figure 5.



366

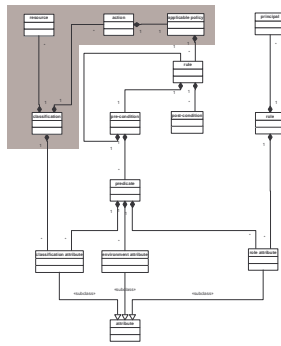
367 **Figure 5 - Environment/attribute section of the language model**

368 XACML policy instances may reference attributes that are not directly associated either
369 with the principal or the resource. These are called environment attributes. For instance,
370 the "current time of day" is an environment attribute that may be referenced by a policy
371 instance. Environment attributes are asserted by authorities and distributed in the form of
372 SAML attribute assertions. The PDP must check that the attribute values it operates upon
373 are asserted by suitable authorities. This operation is described in "Attribute
374 identification", below.

375 3.2.11. Policy/action/resource/classification

376 The policy/action/classification section of the language model is shown in gray in Figure
377 6.

378



379

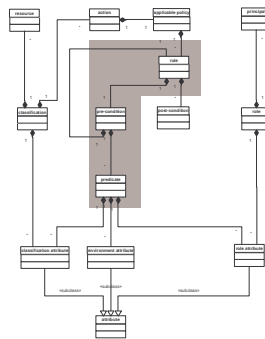
380 **Figure 6 - Policy/action/resource/classification section of the language model**

381 Policy instances are identified with a classification/action pair. In some cases the policy
 382 instance contains elements or attributes that identify the classification and action to which
 383 it is applicable. The PDP must check that the policy instance it uses to compute the
 384 authorization decision is applicable to the authorization request. It does this by verifying
 385 that the action identified in the authorization request is the same as the action identified in
 386 the policy instance, and that the resource identified in the authorization request belongs to
 387 the classification identified in the policy instance. How the PDP does this is described
 388 above.

389 **3.2.12. Rule/pre-condition/predicate**

390 The rule/pre-condition/predicate section of the language model is shown in gray in Figure
 391 7.

392



393

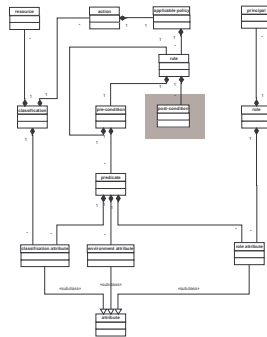
394 **Figure 7 - Rule/pre-condition/predicate section of the language model**

395 XACML policy instances are built from a logical combination of rules. Each rule
 396 comprises one pre-condition and zero or more post-conditions. A pre-condition is a
 397 logical operator or predicate. A predicate is a statement about attributes that can be
 398 verified by the PDP. If the policy instance applicable to an authorization request
 399 evaluates to TRUE, and all internal post-conditions are satisfied, then the PDP may return
 400 an authorization decision with the value TRUE to the PEP.

401 3.2.13. Post-condition

402 The post-condition section of the language model is shown in gray in Figure 8.

403



404

405 **Figure 8 - Post-condition section of the language model**

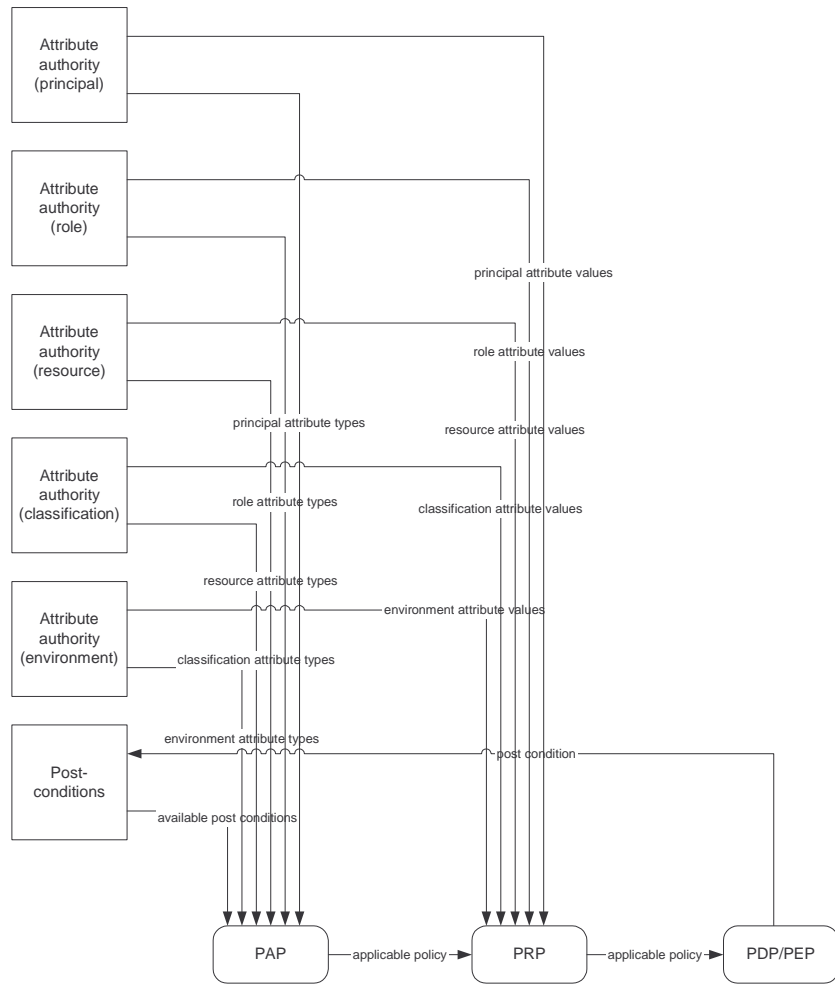
406 Post-conditions are actions specified in an XACML policy instance. Post-conditions are
407 of two types. Internal post-conditions must be successfully executed prior to returning an
408 authorization decision with the value TRUE. External post-conditions must be returned
409 by the PDP to the PEP and an authorization decision with the value TRUE may be issued
410 without confirmation that the condition has been successfully executed.

411 3.2.14. Attribute identification

412 Attribute specifiers are formed of two components: the first component identifies the
413 authority for the attribute and the second component identifies the attribute type. For
414 instance, the specifier may be an XPath expression in the form of a URI. The "host-
415 name" component of the URI identifies the authority for the attribute, and the local-path
416 component identifies the attribute type in terms of the structure of a SAML attribute
417 assertion. In the case where a suitable attribute assertion is provided by the PEP in the
418 decision request, the PDP identifies the appropriate assertion by comparing the host-name
419 in the URI with the issuer field of the assertion. In the case where no suitable assertion is
420 provided by the PEP, then the host-name component can be used to locate a suitable
421 attribute authority to which to send a SAML attribute request.

422 **3.3. Administrative model**

423 It is essential that XACML policy instances only contain references to attributes and
424 post-conditions that are accessible by the PDP or PEP. The administrative model, shown
425 in Figure 9, illustrates how this is achieved. The various SAML attribute authorities
426 involved must provide an interface by which the policy administration point can discover
427 the attribute types available from it.



428

429

430

Figure 9 - Administrative model

430 Chapter Two

431 **4. Policy syntax**

432 *The information in this section is normative, with the exception of the schema*
433 *fragments. SAML*

434

434 *Appendix A - Schema contains the normative version of the schema.*

435

436 **5. Applicable policy**

437 Applicable policy is the top-level element. It contains a description of the access to
438 which the policy applies, in the form of "resource classification" and "resource action". It
439 also contains the policy element. PDPs should use the applicability element to locate,
440 retrieve and verify the policy required for processing a particular
441 samlp:authorizationQuery. Verification means confirming that the value of the
442 resourceActions element in applicable policy is equal to the value of the saml:Actions
443 element in the samlp:authorizationQuery.

```
444 <xs:element name="applicablePolicy">
445   <xs:complexType name="scopedPolicy">
446     <xs:sequence>
447       <xs:element name="applicability" minOccurs="0" maxOccurs="unbounded">
448         <xs:complexType>
449           <xs:sequence>
450             <xs:element name="resourceClassification" type="xs:anyURI"/>
451             <xs:element name="resourceAction" type="saml:Actions" minOccurs="0"
452 maxOccurs="unbounded"/>
453           </xs:sequence>
454         </xs:complexType>
455       </xs:element>
456       <xs:element ref="policy"/>
457     </xs:sequence>
458   </xs:complexType>
459 </xs:element>
460
```

461 **6. Policy**

462 The policy element is an aggregation of rules. Rules must be combines with logical
463 operations, not merely listed.

```
464 <xs:complexType name="policy">
465   <xs:sequence>
466     <xs:element ref="rule" maxOccurs="unbounded"/>
467   </xs:sequence>
468 </xs:complexType>
```

469 **7. Rule**

470 A rule consists of a pre-condition and zero or more post-conditions. If the pre-condition
471 evaluates to TRUE and the internal post-conditions are successfully executed, then the
472 PDP should return the "permit" value in the samlp:Response/StatusCode element.
473 Otherwise, it must return the "deny" value.

```
474 <xs:element name="rule">
475   <xs:complexType>
476     <xs:sequence>
477       <xs:element name="preCondition">
478         <xs:complexType>
479           <xs:choice>
480             <xs:element ref="and"/>
481             <xs:element ref="or"/>

```

```

482         <xs:element ref="not"/>
483         <xs:element ref="predicate"/>
484     </xs:choice>
485 </xs:complexType>
486 </xs:element>
487 <xs:element ref="postCondition" minOccurs="0" maxOccurs="unbounded"/>
488 </xs:sequence>
489 </xs:complexType>
490 </xs:element>

```

491 8. Pre-condition

492 The preCondition element is a predicate or logically-combined set of predicates.

493 8.1. And

494 The "And" pre-condition evaluates to TRUE if and only if all the predicate elements that
495 it contains evaluate to TRUE.

```

496 <xs:element name="and">
497   <xs:complexType>
498     <xs:sequence>
499       <xs:element ref="rule" minOccurs="2" maxOccurs="unbounded"/>
500     </xs:sequence>
501   </xs:complexType>
502 </xs:element>

```

504 8.2. Or

505 The "Or" pre-condition evaluates to TRUE if one or more of the predicate elements that it
506 contains evaluate to TRUE.

```

507 <xs:element name="or">
508   <xs:complexType>
509     <xs:sequence>
510       <xs:element ref="rule" minOccurs="2" maxOccurs="unbounded"/>
511     </xs:sequence>
512   </xs:complexType>
513 </xs:element>

```

515 8.3. Not

516 The "Not" pre-condition evaluates to TRUE if the predicate element that it contains
517 evaluates to FALSE.

```

518 <xs:element name="not">
519   <xs:complexType>
520     <xs:sequence>
521       <xs:element ref="rule"/>
522     </xs:sequence>
523   </xs:complexType>
524 </xs:element>

```

526 8.4. Predicate

527 The predicate element contains either one of the predicates defined here, or an external
528 function.

```

529 <xs:element name="predicate">
530   <xs:complexType>

```

```

531     <xs:choice>
532         <xs:element ref="present"/>
533         <xs:element ref="equality"/>
534         <xs:element ref="greaterOrEqual"/>
535         <xs:element ref="lessOrEqual"/>
536         <xs:element ref="subsetOf"/>
537         <xs:element ref="supersetOf"/>
538         <xs:element ref="nonNullSetIntersection"/>
539         <xs:element ref="externalFunction"/>
540     </xs:choice>
541 </xs:complexType>
542 </xs:element>

```

543 **8.5. Present**

544 The Present predicate evaluates to TRUE if the element referenced by it is populated.

```

545 <xs:element name="present">
546     <xs:complexType>
547         <xs:sequence>
548             <xs:element ref="referencedData"/>
549         </xs:sequence>
550     </xs:complexType>
551 </xs:element>

```

552 **8.6. Equality**

553 The Equality predicate evaluates to TRUE if the two elements referenced by it are equal.
554 Both elements must be of the same type.

```

555 <xs:element name="equality">
556     <xs:complexType>
557         <xs:sequence>
558             <xs:element ref="referencedData"/>
559             <xs:element ref="secondOperand"/>
560         </xs:sequence>
561     </xs:complexType>
562 </xs:element>

```

563 **8.7. Greater or equal**

564 The greaterOrEqual predicate evaluates to TRUE if the first element is greater than or
565 equal to the second element. The elements must be of the same type, which may be
566 string, normalizedString, byte, unsignedByte, base64Binary, hexBinary, integer,
567 positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, int,
568 unsignedInt, long, unsignedLong, short, unsignedShort, decimal, float, double, time,
569 dateTime, duration, date, gMonth, gYear, gYearMonth, gDay, gMonthDay, Name or
570 QName.

```

571 <xs:element name="greaterOrEqual">
572     <xs:complexType>
573         <xs:sequence>
574             <xs:element ref="referencedData"/>
575             <xs:element ref="secondOperand"/>
576         </xs:sequence>
577     </xs:complexType>
578 </xs:element>

```

579

8.8. Less or equal

580 The lessOrEqual predicate evaluates to TRUE if the first element is less than or equal to
581 the second element. The elements must be of the same type, which may be string,
582 normalizedString, byte, unsignedByte, base64Binary, hexBinary, integer, positiveInteger,
583 negativeInteger, nonNegativeInteger, nonPositiveInteger, int, unsignedInt, long,
584 unsignedLong, short, unsignedShort, decimal, float, double, time, dateTime, duration,
585 date, gMonth, gYear, gYearMonth, gDay, gMonthDay, Name or QName.

586

```
587 <xs:element name="lessOrEqual">  
588   <xs:complexType>  
589     <xs:sequence>  
590       <xs:element ref="referencedData"/>  
591       <xs:element ref="secondOperand"/>  
592     </xs:sequence>  
593   </xs:complexType>  
594 </xs:element>
```

595

8.9. Sub-set of

596 The subSetOf predicate evaluates to TRUE if the value of the first element is amongst the
597 set of values referenced by the second element.

598

```
599 <xs:element name="subsetOf">  
600   <xs:complexType>  
601     <xs:sequence>  
602       <xs:element ref="referencedData"/>  
603       <xs:element ref="secondOperand" maxOccurs="unbounded"/>  
604     </xs:sequence>  
605   </xs:complexType>  
606 </xs:element>
```

606

8.10. Super-set of

607 The superSetOf predicate evaluates to TRUE if the set of values referenced by the first
608 element includes all the value(s) of the second element.

609

```
610 <xs:element name="supersetOf">  
611   <xs:complexType>  
612     <xs:sequence>  
613       <xs:element ref="referencedData"/>  
614       <xs:element ref="secondOperand" maxOccurs="unbounded"/>  
615     </xs:sequence>  
616   </xs:complexType>  
617 </xs:element>
```

617

8.11. Non-null Set Intersection

618 The nonNullSetIntersection predicate evaluates to TRUE if the set of values referenced
619 by the two elements have at least one value in common.

620

```
621 <xs:element name="nonNullSetIntersection">  
622   <xs:complexType>  
623     <xs:sequence>  
624       <xs:element ref="referencedData"/>  
625       <xs:element ref="secondOperand" maxOccurs="unbounded"/>  
626     </xs:sequence>  
627   </xs:complexType>  
628 </xs:element>
```

628

8.12. External function

629 The externalFunction element contains a definition of the interface to an external
630 function. The external function is defined as a WSDL "definition" element for a
631 "request-response" operation. The response must be a Boolean.

```
632 <xs:element name="externalFunction" type="wsdl:definitions"/>
```

633

8.13. Post-condition

634 The postCondition element contains a definition of the interface to an external function.
635 The external function is defined as a WSDL "definition" element for a "one-way"
636 operation. Internal post conditions are expected to be performed by the PDP, and a
637 "permit" statusCode must not be returned unless such conditions are successfully
638 executed. External post conditions are expected to be performed by the PEP, and they
639 must include them in the authorization decision. The PDP may return a "permit"
640 rstatusCode without confirmation that such conditions have been successfully executed.

641

```
642 <xs:element name="postCondition" type="wsdl:definitions">  
643 <xs:complexType name="">  
644 <xs:sequence>  
645 <xs:element name="internalPostCondition" type="wsdl:definitions" minOccurs="0"  
646 maxOccurs="0"/>  
647 <xs:element name="externalPostCondition" type="wsdl:definitions" minOccurs="0"  
648 maxOccurs="0"/>  
649 </xs:sequence>  
650 </xs:complexType>  
651 </xs:element>
```

652

8.14. Referenced data

653 The referencedData element contains elements for attributes of the main model entities:
654 principal, resource and environment.

```
655 <xs:element name="referencedData">  
656 <xs:complexType>  
657 <xs:choice>  
658 <xs:element name="roleAttribute" type="attributeReference"/>  
659 <xs:element name="classificationAttribute" type="attributeReference"/>  
660 <xs:element name="environmentAttribute" type="attributeReference"/>  
661 </xs:choice>  
662 </xs:complexType>  
663 </xs:element>
```

664

8.15. Attribute reference

665 The "attribute reference" element is a pointer to an attribute. The pointer is in the form of
666 a URI. It may contain an XPATH expression into a SAML attribute assertion for a
667 principal, resource or environment. If the resource is an XML document, then it may
668 contain an XPATH Expression identifying an element of the resource. If the URI does
669 not indicate a SAML assertion passed to the PDP in the samlp:authorizationQuery, then
670 the PDP should obtain the value from the attribute authority identified by the attribute
671 reference.

```
672 <xs:simpleType name="attributeReference">  
673 <xs:restriction base="xs:anyURI"/>  
674 </xs:simpleType>
```

675

8.16. *Second operand*

676 The second operand element is a choice between a referenced data element and a hard-
677 coded value.

```
678 <xs:element name="secondOperand">  
679   <xs:complexType>  
680     <xs:choice>  
681       <xs:element ref="referencedData"/>  
682       <xs:element ref="hardcodedValue"/>  
683     </xs:choice>  
684   </xs:complexType>  
685 </xs:element>
```

686

8.17. *Hard-coded value*

687 The "hard-coded value" element contains a value written directly into the policy instance.
688 Its type must be identical to that of any element with which it is paired in a predicate sub-
689 element.

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

691

9. References

692

SAML

693

693 Appendix A - Schema

```
694
695 <?xml version="1.0" encoding="UTF-8"?>
696 <xs:schema xmlns:wSDL="http://www.schemas.xmlsoap.org/wSDL/"
697 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:saml="http://www.oasis-
698 open.org/committees/security/docs/draft-sstc-scheam-assertion-19.xsd" elementFormDefault="qualified"
699 attributeFormDefault="unqualified">
700   <xs:element name="applicablePolicy">
701     <xs:complexType name="scopedPolicy">
702       <xs:sequence>
703         <xs:element name="applicability" minOccurs="0" maxOccurs="unbounded">
704           <xs:complexType>
705             <xs:sequence>
706               <xs:element name="resourceClassification" type="xs:anyURI"/>
707               <xs:element name="resourceAction" type="saml:Actions" minOccurs="0"
708 maxOccurs="unbounded"/>
709             </xs:sequence>
710           </xs:complexType>
711         </xs:element>
712         <xs:element ref="policy"/>
713       </xs:sequence>
714     </xs:complexType>
715   </xs:element>
716   <xs:complexType name="policy">
717     <xs:sequence>
718       <xs:element ref="rule" maxOccurs="unbounded"/>
719     </xs:sequence>
720   </xs:complexType>
721   <xs:element name="rule">
722     <xs:complexType>
723       <xs:sequence>
724         <xs:element name="preCondition">
725           <xs:complexType>
726             <xs:choice>
727               <xs:element ref="and"/>
728               <xs:element ref="or"/>
729               <xs:element ref="not"/>
730               <xs:element ref="predicate"/>
731             </xs:choice>
732           </xs:complexType>
733         </xs:element>
734         <xs:element ref="postCondition" minOccurs="0" maxOccurs="unbounded"/>
735       </xs:sequence>
736     </xs:complexType>
737   </xs:element>
738   <xs:element name="and">
739     <xs:complexType>
740       <xs:sequence>
741         <xs:element ref="rule" minOccurs="2" maxOccurs="unbounded"/>
742       </xs:sequence>
743     </xs:complexType>
744   </xs:element>
745   <xs:element name="or">
746     <xs:complexType>
747       <xs:sequence>
748         <xs:element ref="rule" minOccurs="2" maxOccurs="unbounded"/>
749       </xs:sequence>
750     </xs:complexType>
751   </xs:element>
752   <xs:element name="not">
753     <xs:complexType>
754       <xs:sequence>
755         <xs:element ref="rule"/>
756       </xs:sequence>
```

```

757     </xs:complexType>
758 </xs:element>
759 <xs:element name="predicate">
760   <xs:complexType>
761     <xs:choice>
762       <xs:element ref="present"/>
763       <xs:element ref="equality"/>
764       <xs:element ref="greaterOrEqual"/>
765       <xs:element ref="lessOrEqual"/>
766       <xs:element ref="subsetOf"/>
767       <xs:element ref="supersetOf"/>
768       <xs:element ref="nonNullSetIntersection"/>
769       <xs:element ref="externalFunction"/>
770     </xs:choice>
771   </xs:complexType>
772 </xs:element>
773 <xs:element name="present">
774   <xs:complexType>
775     <xs:sequence>
776       <xs:element ref="referencedData"/>
777     </xs:sequence>
778   </xs:complexType>
779 </xs:element>
780 <xs:element name="equality">
781   <xs:complexType>
782     <xs:sequence>
783       <xs:element ref="referencedData"/>
784       <xs:element ref="secondOperand"/>
785     </xs:sequence>
786   </xs:complexType>
787 </xs:element>
788 <xs:element name="greaterOrEqual">
789   <xs:complexType>
790     <xs:sequence>
791       <xs:element ref="referencedData"/>
792       <xs:element ref="secondOperand"/>
793     </xs:sequence>
794   </xs:complexType>
795 </xs:element>
796 <xs:element name="lessOrEqual">
797   <xs:complexType>
798     <xs:sequence>
799       <xs:element ref="referencedData"/>
800       <xs:element ref="secondOperand"/>
801     </xs:sequence>
802   </xs:complexType>
803 </xs:element>
804 <xs:element name="subsetOf">
805   <xs:complexType>
806     <xs:sequence>
807       <xs:element ref="referencedData"/>
808       <xs:element ref="secondOperand" maxOccurs="unbounded"/>
809     </xs:sequence>
810   </xs:complexType>
811 </xs:element>
812 <xs:element name="supersetOf">
813   <xs:complexType>
814     <xs:sequence>
815       <xs:element ref="referencedData"/>
816       <xs:element ref="secondOperand" maxOccurs="unbounded"/>
817     </xs:sequence>
818   </xs:complexType>
819 </xs:element>
820 <xs:element name="nonNullSetIntersection">
821   <xs:complexType>
822     <xs:sequence>

```

```

823         <xs:element ref="referencedData"/>
824         <xs:element ref="secondOperand" maxOccurs="unbounded"/>
825     </xs:sequence>
826 </xs:complexType>
827 </xs:element>
828 <xs:element name="externalFunction" type="wsdl:definitions"/>
829 <xs:element name="postCondition" type="wsdl:definitions">
830     <xs:complexType name="">
831         <xs:sequence>
832             <xs:element name="internalPostCondition" type="wsdl:definitions" minOccurs="0"
833 maxOccurs="0"/>
834             <xs:element name="externalPostCondition" type="wsdl:definitions" minOccurs="0"
835 maxOccurs="0"/>
836         </xs:sequence>
837     </xs:complexType>
838 </xs:element>
839 <xs:element name="referencedData">
840     <xs:complexType>
841         <xs:choice>
842             <xs:element name="roleAttribute" type="attributeReference"/>
843             <xs:element name="classificationAttribute" type="attributeReference"/>
844             <xs:element name="environmentAttribute" type="attributeReference"/>
845         </xs:choice>
846     </xs:complexType>
847 </xs:element>
848 <xs:simpleType name="attributeReference">
849     <xs:restriction base="xs:anyURI"/>
850 </xs:simpleType>
851 <xs:element name="secondOperand">
852     <xs:complexType>
853         <xs:choice>
854             <xs:element ref="referencedData"/>
855             <xs:element ref="hardcodedValue"/>
856         </xs:choice>
857     </xs:complexType>
858 </xs:element>
859 <xs:element name="hardcodedValue" type="xs:string"/>
860 </xs:schema>
861
862
863

```

863 Appendix B - Test cases

864 The text in this appendix will be replaced by normative test cases. The test cases will
865 comprise a SAML authorization request message, an XACML policy instance and the
866 resulting SAML authorization response message.

867 Authorities and assertions.

868
869 Attribute authority issues attribute assertions. Resource authority
870 issues resource assertions. Environmental authority issues environment
871 assertions. Application authority issues application-specific
872 assertions.

873
874 Attribute assertions.

```
875 <Assertion ...>  
876 <Conditions.../>  
877 <Advice.../>  
878 <AttributeStatement>  
879     <Subject .../>  
880     <Attribute AttributeNamespace="..." AttributeName="weight">  
881         <AttributeValue>100</AttributeValue>  
882     </Attribute>  
883 </AttributeStatement>  
884 </Assertion>
```

885
886 Resource assertions.

```
887 <Assertion ...>  
888 <Conditions .../>  
889 <Advice .../>  
890 <ResourceStatement>  
891     <Resource ResourceName="..." ResourceType="..."/>  
892     <Attribute AttributeNamespace="..." AttributeName="owner">  
893         <AttributeValue>superman</AttributeValue>  
894     </Attribute/>  
895     <Attribute AttributeNamespace="..." AttributeName="color">  
896         <AttributeValue>blue</AttributeValue>  
897     </Attribute>  
898 </ResourceStatement>  
899 </Assertion>
```

900
901 Request looks like this:

```
902 <Request ...>  
903     <AuthorizationQuery Resource="...">  
904         <Subject .../>  
905         <Actions .../>  
906         <Evidence .../>  
907     </AuthorizationQuery>  
908 </Request>
```

909
910
911
912
913
914
915
916
917

918 Expressions.

919 Expression consists of elementary conditions joined by logical 'and' or
920 'or' or grouped by 'paren'.

```
921  
922 <exp name="...">
```

923 <cnd test="..." />
924 <and />
925 <cnd test="..." />
926 </exp>
927
928 Name is an id-type attribute of expression element.
929 To make it more compact we can assume <and> by default:
930
931 <exp name="...">
932 <cnd test="..." />
933 <cnd test="..." />
934 </exp>
935
936 Conditions can be enclosed in parens: a and (b or c)
937
938 <exp name="...">
939 <cnd test="..." />
940 <paren>
941 <cnd test="..." />
942 <or />
943 <cnd test="..." />
944 </paren>
945 </exp>
946
947 Conditions "test" attribute is a Boolean over some xpath expression.
948
949 Here is a long form. Suppose we want to say that statement balance
950 should be over 50 dollars. Statement balance is presented as a saml
951 attribute assertion. Xpath expression is:
952 //AttributeStatement/Attribute[@AttributeNamespace='www.foo.com'][@Attri
953 buteName='balance']/AttributeValue='50'
954
955 We can put it in condition:
956 <cnd
957 test="//AttributeStatement/Attribute[@AttributeNamespace='www.foo.com']
958 @AttributeName='balance']/AttributeValue='50' />
959
960 Macros.
961 To make above condition more readable we can define macros:
962
963 <macro name="bal"
964 def="//AttributeStatement/Attribute[@AttributeNamespace='www.foo.com']
965 @AttributeName='balance']/AttributeValue"/>
966
967 '\$' sign applied to macro name denotes macro expansion. I do not know if
968 it's the best choice. We can use '#' instead or something else. We also
969 need to be able to escape macro-expansion symbol.
970
971 Then condition is:
972 <cnd test="\$bal='50' />
973
974 Macros can be reused with macros as well. For example we can define
975 namespace macro and reuse it in attribute macros:
976
977 <macro name="myns"
978 def="//AttributeStatement/Attribute[@AttributeNamespace='www.foo.com' />
979
980 <macro name="bal" def="\$myns[@AttributeName='balance']/AttributeValue"/>
981
982 Rules.
983 Rule is in the form:
984 <allow action="...">
985 <subject>
986 expression reference, or conds.

```

987     </subject>
988     <resource name="...">
989         expression reference, or conds.
990     </resource>
991     <if>
992         expression reference, or conds.
993     </if>
994 </allow>
995
996 Each component within a rule can include expression references or a set
997 of conditions.
998
999 For example we can have expression for the good customers that we want
1000 to reference in the rule:
1001
1002 <exp name="goodcust">
1003     <cnd test="$bal > `100'"/>
1004     <cnd test="$pmt > `15'"/>
1005 </exp>
1006
1007 <allow action="...">
1008     <subject>
1009         <exp name="goodcust"/>
1010     </subject>
1011     <resource .../>
1012     <if .../>
1013 </allow>
1014
1015 We can join expressions and augment them with conditions:
1016 <allow action="...">
1017     <subject>
1018         <exp name="goodcust"/>
1019         <exp name="amexholder"/>
1020         <cnd test="$birthmonth='may'"/>
1021         <cnd test="$birthyear='1861'"/>
1022     </subject>
1023     <resource .../>
1024     <if .../>
1025 </allow>
1026
1027 Reserved symbols.
1028
1029 Let Req-> refer to the current request.
1030 Subj-> refer to the attribute assertion about requestor
1031 Res-> refer to the attribute assertion about requested resource.
1032
1033 Can we write expressions for that? I'm not even sure that this is right.
1034
1035 <macro name="Req->" def="/Request/AuthorizationQuery/" />
1036
1037 <macro name="Subj->" def="//AttributeStatement/
1038 Attribute[preceding-sibling::Subject/NameIdentifier/@Name=
1039 $Req->Subject/NameIdentifier/@Name]"/>
1040
1041 <macro name="Res->"
1042 def="//ResourceStatement/
1043 Attribute[preceding-sibling::Subject/NameIdentifier/@Name=
1044 $Req->Subject/NameIdentifier/@Name]"/>
1045
1046 Roles and groups.
1047 Predicate groupmember(groupname, subject) evaluates to true if subject
1048 is a member of a group.
1049
1050

```