



OASIS eXtensible Access Control Markup Language (XACML)

Working Draft 15, 12 July 2002

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

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

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

Editors:

Simon Godik, Simon Godik (simon@godik.com)

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

Contributors:

Anne Anderson, Sun Microsystems

Bill Parducci, Bill Parducci

Carlisle Adams, Entrust

Daniel Engovatov, Crosslogix

Don Flinn, Hitachi

Ernesto Damiani, University of Milan

James MacLean, Affinitex

Hal Lockhart, Entegrity

Ken Yagen, Crosslogix

Konstantin Besnozov, Hitachi

Michiharu Kudo, IBM, Japan

Pierangela Samarati, University of Milan

Polar Humenn, Syracuse University

Sekhar Vajjhala, Sun Microsystems

Gerald Brose, Xtradyne

Abstract:

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

Status:

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

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

37 Copyright © 2001, 2002 The Organization for the Advancement of Structured Information
38 Standards [OASIS]

39	Table of contents	
40	1 Glossary (non-normative)	7
41	1.1 Preferred terms	7
42	1.2 Related terms	8
43	2 Introduction (non-normative)	8
44	2.1 Background	8
45	2.1.1 Rule combining	9
46	2.1.2 Policy combining	9
47	2.1.3 Combining algorithm	9
48	2.1.4 Decision indication	9
49	2.1.5 Names or attributes	10
50	2.1.6 Specifying actions	10
51	2.1.7 Expression of predicates	10
52	2.1.8 Abstraction layer	10
53	2.1.9 Policy attachment	10
54	2.2 References	10
55	2.3 Notation	10
56	2.4 Schema Organization and Namespaces	11
57	3 Example (non-normative)	11
58	3.1 Introduction to the example	11
59	3.2 Example medical record instance	12
60	3.3 Example authorization decision request	13
61	3.4 Example plain-language rules	14
62	3.5 Example XACML rule instances	15
63	3.5.1 Rule 1	15
64	3.5.2 Rule 2	15
65	3.5.3 Rule 3	17
66	3.5.4 Rule 4	18
67	4 Models (non-normative)	19
68	4.1 Data-flow model	19
69	4.2 XACML Context	20
70	4.3 Policy language model	21
71	4.3.1 Rule	22
72	4.3.2 Policy statement	24
73	4.3.3 Policy set statement	27
74	5 Policy syntax (normative, with the exception of the schema fragments)	28

75	5.1	Element <PolicySetStatement>	28
76	5.2	Element <PolicyStatement>	28
77	5.3	Element <Rule>	28
78	5.4	Complex type PolicySetStatementType	28
79	5.5	Complex type PolicyStatementType	29
80	5.6	Complex type RuleType	30
81	5.7	Complex type EffectType	30
82	5.8	Complex type TargetType	30
83	5.9	Complex type MatchType	31
84	5.10	Complex type ObligationsType	31
85	5.11	Complex type ObligationType	31
86	5.12	Element <Function>	31
87	5.13	Complex type ConditionType	31
88	5.14	Complex type FunctionType	32
89	5.15	Element <Attribute>	32
90	5.16	Complex type AttributeType	32
91	5.17	Element <AttributeDesignator>	33
92	5.18	Complex type AttributeDesignatorType	33
93	5.19	Complex type AttributeAssignmentType	33
94	5.20	Complex type PolicySetType	33
95	5.21	Complex type RuleSetType	34
96	5.22	Complex type RuleDesignatorType	34
97	6	Function names and legal type combinations	34
98	6.1	Functions	34
99	7	Context syntax (normative, with the exception of the schema fragments)	39
100	7.1	Element <Request>	39
101	7.2	Element <Response>	39
102	7.3	Complex type RequestType	39
103	7.4	Complex type ResponseType	40
104	7.5	Complex type ResultType	40
105	7.6	Complex type SubjectType	40
106	7.7	Complex type SubjectIdType	41
107	7.8	Complex type AuthenticationInfoType	41
108	7.9	Complex type AttributeType	41
109	7.10	Complex type ResourceType	42
110	7.11	Complex type ResourceSpecifierType	42
111	7.12	Complex type SpecifierScopeType	42

112	7.13	Complex type ResourceContentType	42
113	7.14	Complex type ActionType	43
114	7.15	Complex type DecisionType	43
115	7.16	Complex type EnvironmentType	43
116	7.17	Complex type AdviceType	43
117	8	XACML identifiers (normative)	44
118	8.1	Access Subject	44
119	8.2	Time of day	44
120	8.3	Attributes	44
121	8.3.1	Role	44
122	8.3.2	RFC822 Name	44
123	8.3.3	X.500 distinguished name	44
124	8.3.4	Unix file-system path	44
125	8.3.5	Uniform resource identifier	44
126	8.4	Authentication locality	45
127	8.5	Deny-overrides rule-combining algorithm	45
128	8.6	Deny-overrides policy-combining algorithm	45
129	8.7	Permit-overrides rule-combining algorithm	45
130	8.8	Permit-overrides policy-combining algorithm	45
131	9	Combining algorithms (normative)	45
132	9.1	Deny-overrides	45
133	9.2	Permit-overrides	46
134	10	Profiles (normative but not mandatory to implement)	48
135	10.1	XACML	48
136	10.2	SAML	48
137	10.3	XML Digital Signature	50
138	10.4	LDAP	50
139	10.4.1	Directory information tree (DIT)	50
140	10.4.2	Policy combination	51
141	10.4.3	Directory schema	51
142	10.4.4	Object Class Definitions	52
143	10.4.5	Attribute Definitions	52
144	10.4.6	Matching Rule Definitions	53
145	11	Operational Model (normative)	53
146	11.1	Policy Decision Point (PDP)	53
147	12	XACML extensibility points (non-normative)	54
148	12.1	URIs	54

149	13 Security and privacy (non-normative)	54
150	13.1 Authentication	54
151	13.2 Confidentiality	55
152	13.2.1 Communication Confidentiality	55
153	13.2.2 Statement Level Confidentiality	55
154	13.3 Policy Integrity	55
155	13.4 Elements included by reference	56
156	13.5 Trust Model	56
157	13.6 Privacy	56
158	14 Conformance (normative)	57
159	15 References	58
160	Appendix A. Acknowledgments	59
161	Appendix B. Revision History	60
162	Appendix C. Notices	61
163		

165 1 Glossary (non-normative)

166 1.1 Preferred terms

167 **Access** - Performing an *action*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

206 1.2 Related terms

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

209 For instance, the term **attribute** is used in place of the terms: **group** and **role**.

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

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

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

214 2 Introduction (non-normative)

215 2.1 Background

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

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

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

231 XACML exploits long-established techniques, such as:

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

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

249 **2.1.1 Rule combining**

250 Ref 5,

251 **2.1.2 Policy combining**

252 Ref 5, 8

253 **2.1.3 Combining algorithm**

254 Ref 7,

255 **2.1.4 Decision indication**

256

257 **2.1.5 Names or attributes**

258 Ref 2, 6

259 **2.1.6 Specifying actions**

260 Ref 1,

261 **2.1.7 Expression of predicates**

262 Ref 4,

263 **2.1.8 Abstraction layer**

264

265 **2.1.9 Policy attachment**

266 Ref 1, 3

267 **2.2 References**

268 1. Perritt; Knowbots, Headers & Contract Law; 1993.

269 2. Orange book

270 3. Trusted Network Interpretation

271 4. X.500 filter

272 5. J Moffett and M Sloman. Policy hierarchies for distributed system management. IEEE Journal
273 on Selected areas in communications, pages 1404-1414, December 1993. Special Issue on
274 network management.

275 6. R Sandhu, E Coyne, H Feinstein and C Youman. Role-based access control models. IEEE
276 Computer, 9(2); 38-47, 1996.

277 7. S Jajodia, P Samarati, V S Subrahmanian and E Bertino. A unified framework for enforcing
278 multiple access control policies. Proceedings of ACM SIGMOD, 1997

279 8. N Minsky, V Ungureanu. Unified support for heterogeneous distributed systems. 7th USENIX
280 security symposium, San Antonio, Texas, January, 1998..

281 **2.3 Notation**

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

284 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",
285 "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be
286 interpreted as described in IETF RFC 2119 [rfc2119](#):

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

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

293 `Listings of XACML schemas appear like this.`

294
295 `Example code listings appear like this.`

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

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

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

304 2.4 Schema Organization and Namespaces

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

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

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

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

309 XACML functions have the following namespace prefix.

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

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

313 The SAML assertion schema is imported into the XACML schema. Also imported is the schema for
314 XML Signature [XMLSigXSD](#), which is associated with the following XML namespace:

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

316 3 Example (non-normative)

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

318 3.1 Introduction to the example

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

321 3.2 Example medical record instance

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

```
324 <?xml version="1.0" encoding="UTF-8"?>
325 <record xmlns="medico.com/records.xsd"
326 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
327 xsi:schemaLocation="medico.com/records.xsd
328 http://www.medico.com/schema/record.xsd">
329   <patient>
330     <patientName>
331       <first>Bartholomew</first>
332       <last>Simpson</last>
333     </patientName>
334     <patientContact>
335       <street>27 Shelbyville Road</street>
336       <city>Springfield</city>
337       <state>MA</state>
338       <zip>12345</zip>
339       <phone>555.123.4567</phone>
340       <fax/>
341       <email/>
342     </patientContact>
343     <patientDoB xsi:type="date">1992-03-21</patientDoB>
344     <patientGender xsi:type="string">male</patientGender>
345     <policyNumber xsi:type="string">555555</policyNumber>
346   </patient>
347   <parentGuardian>
348     <parentGuardianName>
349       <first>Homer</first>
350       <last>Simpson</last>
351     </parentGuardianName>
352     <parentGuardianContact>
353       <street>27 Shelbyville Road</street>
354       <city>Springfield</city>
355       <state>MA</state>
356       <zip>12345</zip>
357       <phone>555.123.4567</phone>
358       <fax/>
359       <email>homers@aol.com</email>
360     </parentGuardianContact>
361   </parentGuardian>
362   <primaryCarePhysician>
363     <physicianName>
364       <first>Julius</first>
365       <last>Hibbert</last>
366     </physicianName>
367     <physicianContact>
368       <street>1 First St</street>
369       <city>Springfield</city>
370       <state>MA</state>
371       <zip>12345</zip>
372       <phone>555.123.9012</phone>
373       <fax>555.123.9013</fax>
374       <email/>
375     </physicianContact>
376     <registrationID>ABC123</registrationID>
377   </primaryCarePhysician>
378   <insurer>
379     <name>Blue Cross</name>
380     <street>1234 Main St</street>
```

```

381     <city>Springfield</city>
382     <state>MA</state>
383     <zip>12345</zip>
384     <phone>555.123.5678</phone>
385     <fax>555.123.5679</fax>
386     <email/>
387 </insurer>
388 <medical>
389     <treatment>
390         <drug>
391             <name>methylphenidate hydrochloride</name>
392             <dailyDosage>30mgs</dailyDosage>
393             <startDate>1999-01-12</startDate>
394         </drug>
395         <comment>patient exhibits side-effects of skin coloration and
396 carpal degeneration</comment>
397     </treatment>
398     <result>
399         <test>blood pressure</test>
400         <value>120/80</value>
401         <date>2001-06-09</date>
402         <performedBy>Nurse Betty</performedBy>
403     </result>
404 </medical>
405 </record>

```

406 3.3 Example authorization decision request

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

```

411 <?xml version="1.0" encoding="UTF-8"?>
412 <Request xmlns="urn:oasis:names:tc:xacml:0.15i:context"
413 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
414 xmlns:identifier="urn:oasis:names:tc:xacml:identifier"
415 xmlns:xacml="urn:oasis:names:tc:xacml:0.15i:policy"
416 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
417 xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:context
418 http://www.oasis-open.org/tc/xacml/v15/draft-xacml-schema-context-
419 15i.xsd">
420     <Subject>
421         <SubjectId Format="xs:string">Julius Hibbert</SubjectId>
422     </Subject>
423     <Resource>
424         <ResourceSpecifier Format="xs:anyURI" Scope="Descendants"
425 ResourceId="//medico.com/record/patient[@patientName/first='Bartholo
426 mew'][@patientName/last='Simpson']/patientDoB"/>
427     </Resource>
428     <Action Namespace="">read</Action>
429     <Environment>
430         <EnvironmentAttribute
431 AttributeId="urn:oasis:names:tc:SAML:1.0:Assertion"
432 DataType="xs:string">

```

```

433     <saml:Assertion AssertionID="64578390" Issuer="medico.com"
434 IssueInstant="2002-03-08T08:23:47-05:00" MajorVersion="0"
435 MinorVersion="28" xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
436 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
437 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
438 xsi:schemaLocation="http://www.oasis-
439 open.org/committees/security/docs/cs-sstc-schema-assertion-01.xsd">
440     <saml:AuthenticationStatement AuthenticationInstant="2002-
441 03-08T08:23:45-05:00" AuthenticationMethod="http://www.oasis-
442 open.org/committees/security/docs/draft-sstc-core-28/password-shal">
443     <saml:Subject>
444     <saml:NameIdentifier
445 NameQualifier="\\medico.com">Julius Hibbert</saml:NameIdentifier>
446     <saml:SubjectConfirmation>
447     <saml:ConfirmationMethod>http://www.oasis-
448 open.org/committees/security/docs/draft-sstc-core-
449 24/artifact</saml:ConfirmationMethod>
450     </saml:SubjectConfirmation>
451     </saml:Subject>
452     <saml:SubjectLocality IPAddress="217.57.95.242"/>
453     </saml:AuthenticationStatement>
454     </saml:Assertion>
455     </EnvironmentAttribute>
456     <EnvironmentAttribute
457 AttributeID="urn:oasis:names:tc:SAML:1.0:Assertion"
458 DataType="xs:string">
459     <saml:Assertion MajorVersion="0" MinorVersion="28"
460 AssertionID="68938960" Issuer="medico.com" IssueInstant="2000-06-
461 15T15:02:39-05:00" xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
462 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
463 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
464 xsi:schemaLocation="http://www.oasis-
465 open.org/committees/security/docs/cs-sstc-schema-assertion-01.xsd">
466     <saml:AttributeStatement>
467     <saml:Subject>
468     <saml:NameIdentifier
469 NameQualifier="\\medico.com">Julius Hibbert</saml:NameIdentifier>
470     </saml:Subject>
471     <saml:Attribute AttributeName="role"
472 AttributeNamespace="//medico.com">
473     <saml:AttributeValue>physician</saml:AttributeValue>
474     </saml:Attribute>
475     </saml:AttributeStatement>
476     </saml:Assertion>
477     </EnvironmentAttribute>
478     </Environment>
479 </Request>

```

3.4 Example plain-language rules

The following plain-language rules are to be enforced:

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

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

489 3.5 Example XACML rule instances

490 3.5.1 Rule 1

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

```
493 <?xml version="1.0" encoding="UTF-8"?>
494 <Rule RuleId="//medico.com/rules/rule1" Effect="Permit"
495 xmlns="urn:oasis:names:tc:xacml:0.15i:policy"
496 xmlns:function="urn:oasis:names:tc:xacml:0.15i:function"
497 xmlns:identifier="urn:oasis:names:tc:xacml:0.15i:identifier"
498 xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
499 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
500 xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:policy
501 http://www.oasis-open.org/tc/xacml/v15/draft-xacml-schema-policy-
502 15i.xsd">
503   <Description>A person may read any record for which he or she is
504   the designated patient</Description>
505   <Target>
506     <Subjects MatchId="function:rfc822Name-equal"
507     DataType="xs:boolean">
508       <AttributeDesignator
509       Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden-
510       tifier:rfc822Name']" DataType="identifier:rfc822Name"/>
511       <Attribute DataType="identifier:rfc822Name">@</Attribute>
512     </Subjects>
513     <Resources MatchId="function:string-match"
514     DataType="xs:boolean">
515       <AttributeDesignator
516       Designator="//xacmlContext/Request/Resource/@ResourceURI"
517       DataType="xs:anyURI"/>
518       <Attribute
519       DataType="xs:anyURI">//medico.com/record.*</Attribute>
520     </Resources>
521     <Actions MatchId="function:subset" DataType="xs:boolean">
522       <AttributeDesignator
523       Designator="//xacmlContext/Action[@Namespace=]"
524       DataType="xs:string"/>
525       <Attribute DataType="xs:string">read</Attribute>
526     </Actions>
527   </Target>
528   <Condition FunctionId="function:string-equal"
529   DataType="xs:boolean">
530     <AttributeDesignator
531     Designator="//xacmlContext/Request/Subject/SubjectId"
532     DataType="xs:string"/>
533     <AttributeDesignator
534     Designator="//xacmlContext/Request/Resource/patientName"
535     DataType="xs:string"/>
536   </Condition>
537 </Rule>
```

538 3.5.2 Rule 2

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

```

541 <?xml version="1.0" encoding="UTF-8"?>
542 <Rule RuleId="//medico.com/rules/rule2" Effect="Permit"
543 xmlns="urn:oasis:names:tc:xacml:0.15i:policy"
544 xmlns:function="urn:oasis:names:tc:xacml:0.15i:function"
545 xmlns:identifier="urn:oasis:names:tc:xacml:0.15i:identifier"
546 xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
547 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
548 xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:policy
549 http://www.oasis-open.org/tc/xacml/v15/draft-xacml-schema-policy-
550 15i.xsd">
551   <Description>A person may read any record for which he or she is
552   the designated parent or guardian, and for which the patient is
553   under 16 years of age</Description>
554   <Target>
555     <Subjects MatchId="function:rfc822Name-equal"
556     DataType="xs:boolean">
557       <AttributeDesignator
558       Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden
559       tifier:rfc822Name']" DataType="identifier:rfc822Name"/>
560       <Attribute DataType="identifier:rfc822Name">@</Attribute>
561     </Subjects>
562     <Resources MatchId="function:string-match"
563     DataType="xs:boolean">
564       <AttributeDesignator
565       Designator="//xacmlContext/Request/Resource/@ResourceURI"
566       DataType="xs:anyURI"/>
567       <Attribute
568       DataType="xs:anyURI">//medico.com/record.*</Attribute>
569     </Resources>
570     <Actions MatchId="function:subset" DataType="xs:boolean">
571       <AttributeDesignator
572       Designator="//xacmlContext/Action[@Namespace=]"
573       DataType="xs:string"/>
574       <Attribute DataType="xs:string">read</Attribute>
575     </Actions>
576   </Target>
577   <Condition FunctionId="function:and" DataType="xs:boolean">
578     <Function FunctionId="function:string-equal"
579     DataType="xs:boolean">
580       <AttributeDesignator
581       Designator="//xacmlContext/Request/Subject/SubjectId"
582       DataType="xs:string"/>
583       <AttributeDesignator
584       Designator="//xacmlContext/Request/Resource/guardianName"
585       DataType="xs:string"/>
586     </Function>
587     <Function FunctionId="function:dayTimeDuration-greater-than"
588     DataType="xs:boolean">
589       <Function FunctionId="function:date-subtract"
590       DataType="xs:dayTimeDuration">
591         <AttributeDesignator
592         Designator="//xacmlContext/Other/OtherAttribute/Attribute[@DataType=
593         'identifier:today'sDate']" DataType="xs:date"/>
594         <AttributeDesignator
595         Designator="//xacmlContext/Request/Resource/patient/patientDoB"
596         DataType="xs:date"/>
597       </Function>
598       <Attribute DataType="xs:dayTimeDuration">16-0-0</Attribute>
599     </Function>
600   </Condition>
601 </Rule>

```


602 3.5.3 Rule 3

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

```
606 <?xml version="1.0" encoding="UTF-8"?>
607 <saml:Assertion MajorVersion="0" MinorVersion="24"
608 AssertionID="A7F34K90" Issuer="medico.com" IssueInstant="2002-03-
609 22T08:23:47-05:00">
610   <PolicyStatement PolicyId="//medico.com/rules/policy3"
611   RuleCombiningAlgId="//www.oasis-
612   open.org/committees/xacml/docs/ruleCombiningAlgorithms/DenyOverrides
613   " xmlns="urn:oasis:names:tc:xacml:0.15i:policy"
614   xmlns:function="urn:oasis:names:tc:xacml:0.15i:function"
615   xmlns:identifier="urn:oasis:names:tc:xacml:0.15i:identifier"
616   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
617   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
618   xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:policy
619   http://www.oasis-open.org/tc/xacml/v15/draft-xacml-schema-policy-
620   15i.xsd">
621     <Description>A physician may write any medical element for which
622     he or she is the designated primary care physician, provided an
623     email is sent to the patient</Description>
624     <Target>
625       <Subjects MatchId="function:string-match"
626       DataType="xs:boolean">
627         <AttributeDesignator
628         Designator="//xacmlContext/Request/Subject/Attribute[@DataTy
629         pe='identifier:role']" DataType="xs:string"/>
630         <Attribute DataType="xs:string">physician</Attribute>
631       </Subjects>
632       <Resources MatchId="function:string-match"
633       DataType="xs:boolean">
634         <AttributeDesignator
635         Designator="//xacmlContext/Request/Resource/@ResourceURI"
636         DataType="xs:anyURI"/>
637         <Attribute
638         DataType="xs:anyURI">//medico.com/record/medical.*</Attribute>
639       </Resources>
640       <Actions MatchId="function:subset" DataType="xs:boolean">
641         <AttributeDesignator
642         Designator="//xacmlContext/Action[@Namespace=]"
643         DataType="xs:string"/>
644         <Attribute DataType="xs:string">write</Attribute>
645       </Actions>
646     </Target>
647     <RuleSet>
648       <Rule RuleId="//medico.com/rules/rule3" Effect="Permit">
649         <Description>A physician may write any medical element for
650         which he or she is the designated primary care
651         physician</Description>
652         <Condition FunctionId="function:and"
653         DataType="xs:boolean">
654           <Function FunctionId="function:string-equal"
655           DataType="xs:boolean">
656             <AttributeDesignator
657             Designator="//xacmlContext/Request/Subject/SubjectId"
658             DataType="xs:string"/>
659             <AttributeDesignator
660             Designator="//xacmlContext/Request/Resource/physicianName"
661             DataType="xs:string"/>
```

```

662         </Function>
663         <Function FunctionId="function:present"
664         DataType="xs:boolean">
665             <AttributeDesignator
666             Designator="//xacmlContext/Request/Resource/patient/email"
667             DataType="xs:string"/>
668         </Function>
669     </Condition>
670 </Rule>
671 </RuleSet>
672 <Obligations>
673     <Obligation ObligationId="//medico.com/emailer"
674     FulfilOn="Permit">
675         <AttributeDesignator
676         Designator="//xacmlContext/Request/Resource/patient/email"
677         DataType="xs:string"/>
678         <AttributeAssignment DataType="xs:string"
679         AttributeId="//medico.com/text">Your medical record has been
680         accessed by:</AttributeAssignment>
681         <AttributeDesignator
682         Designator="//xacmlContext/Request/Subject/SubjectId"
683         DataType="xs:string"/>
684     </Obligation>
685 </Obligations>
686 </PolicyStatement>
687 </saml:Assertion>

```

688 3.5.4 Rule 4

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

```

690 <?xml version="1.0" encoding="UTF-8"?>
691 <Rule RuleId="//medico.com/rules/rule4" Effect="Deny"
692 xmlns="urn:oasis:names:tc:xacml:0.15i:policy"
693 xmlns:function="urn:oasis:names:tc:xacml:0.15i:function"
694 xmlns:identifier="urn:oasis:names:tc:xacml:0.15i:identifier"
695 xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
696 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
697 xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:policy
698 http://www.oasis-open.org/tc/xacml/v15/draft-xacml-schema-policy-
699 15i.xsd">
700     <Description>An administrator shall not be permitted to read or
701     write medical elements of a patient record</Description>
702     <Target>
703         <Subjects MatchId="function:string-match" DataType="xs:boolean">
704             <AttributeDesignator
705             Designator="//xacmlContext/Request/Subject/Attribute[@DataTy
706             pe='identifier:role']" DataType="xs:string"/>
707             <Attribute DataType="xs:string">adminstrator</Attribute>
708         </Subjects>
709         <Resources MatchId="function:anyURI-equal"
710         DataType="xs:boolean">
711             <AttributeDesignator
712             Designator="//xacmlContext/Request/Resource/@ResourceURI"
713             DataType="xs:anyURI"/>
714             <Attribute
715             DataType="xs:anyURI">//medico.com/record/medical.*</Attribute>
716         </Resources>
717         <Actions MatchId="function:subset" DataType="xs:boolean">
718             <AttributeDesignator
719             Designator="//xacmlContext/Action[@Namespace=]"
720             DataType="xs:string"/>

```

721
722
723
724

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

4 Models (non-normative)

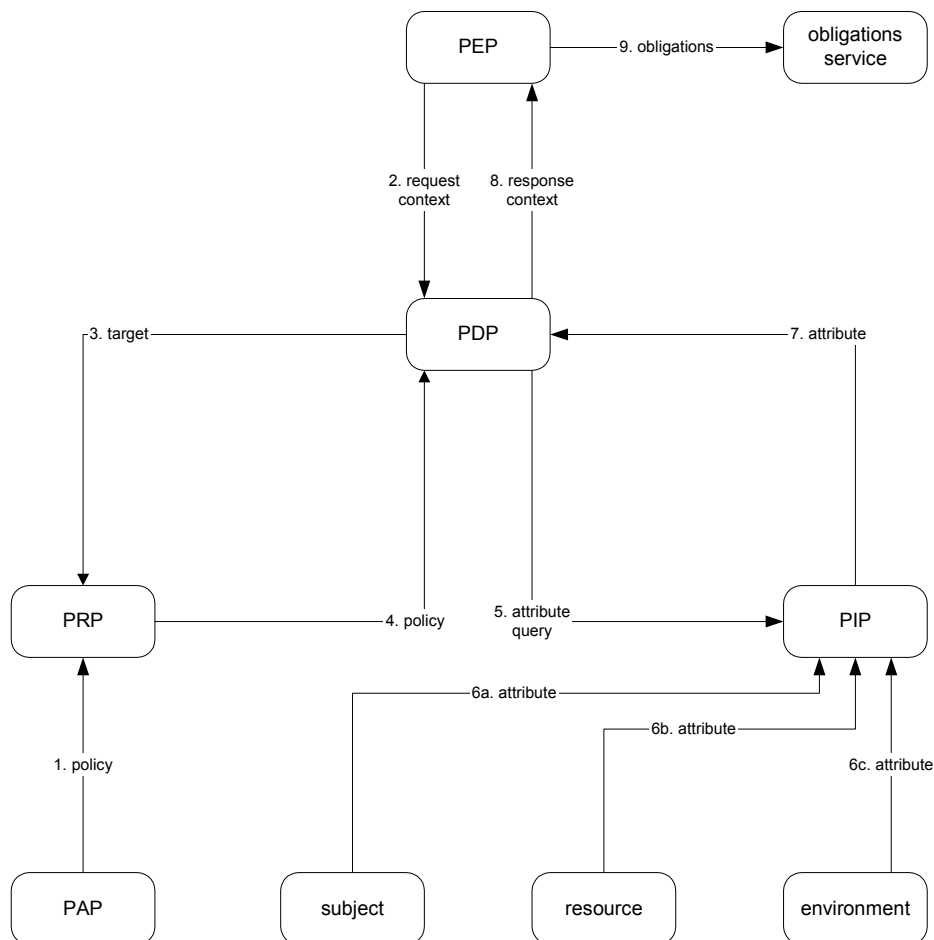
725

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

4.1 Data-flow model

728

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



730

731

Figure 1 - Data-flow diagram

732 Note: some of the data-flows shown in the diagram may be facilitated by a repository. For instance,
733 the communications between the **PDP** and the **PIP** or the communications between the **PDP** and
734 the **PRP** or the communication between the **PAP** and the **PRP** may be facilitated by a repository.
735 The XACML specification is not intended to place restrictions on the location of any such repository,
736 or indeed to prescribe a particular communication protocol for any of the data-flows.

737 The model operates by the following steps.

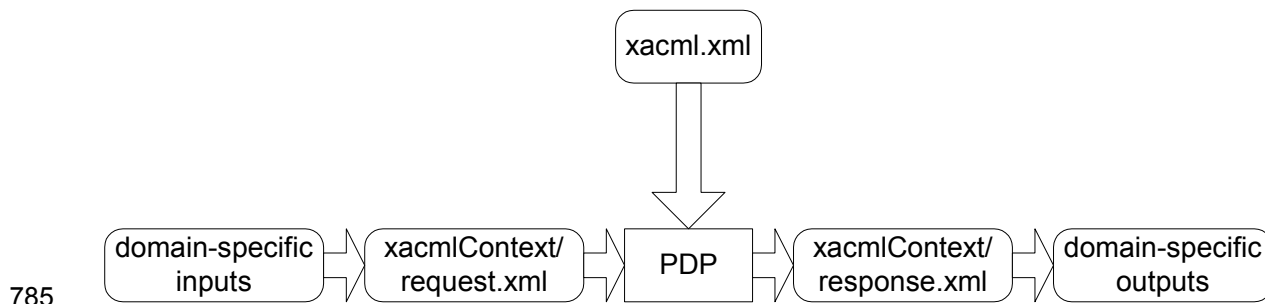
- 738 1. **PAPs** write **policies** and make them available to the **PRP**. From the point of view of an
739 individual **PAP**, its **policies** represent the complete policy for a particular **target**. However, the
740 **PDP** may be aware of other **PAPs** that it considers authoritative for the same **target**. In which
741 case, it is the **PDP's** job to obtain all the **policies** and combine them in accordance with a
742 **policy-combining algorithm**. The result should be a self-consistent **policy set**.
- 743 2. The **PEP** sends request context to the **PDP**, perhaps in the form of a SAML [SAML] request.
744 The request context contains some or all of the **attributes** required by the **PDP** to render an
745 **authorization decision**, in accordance with **applicable policy**. The decision request and all
746 attributes relevant to that request are converted to an XACML input context
747 ("xacmlContext:request") by the **PDP** or by another entity that it trusts to do this conversion.
- 748 3. The **PDP** locates and retrieves the **policy** applicable to the request context from the **PRP**.
- 749 4. The **PRP** returns the **applicable policy** to the **PDP** in the form of an XACML
750 <PolicyStatement> or <PolicySetStatement>. The **PDP** ensures that the **input**
751 **context** is in the scope of the <PolicyStatement> or <PolicySetStatement>.
- 752 5. The **PDP** examines the authorization **input context** and the **policy** to ascertain whether it has
753 all the **attribute** values required to render an **authorization decision**. If it does not, then it
754 requests **attributes** from suitable **PIPs**, perhaps in the form of SAML requests of the attribute
755 query type [SAML].
- 756 6. The **PIP** (which may be a SAML attribute authority) locates and retrieves the requested
757 **attributes** from other systems by a means, and in a form, that is out of scope for this
758 specification.
- 759 7. The **PIP** returns the requested **attributes** to the **PDP**, perhaps in the form of SAML responses
760 containing SAML attribute assertions. The **PDP** (or another trusted entity) incorporates these
761 attribute values into the input context and evaluates the **policy** with respect to this input
762 context. The result of this evaluation is a decision encoded in an output context
763 ("xacmlContext:response") document. The response context is converted to an **authorization**
764 **decision** protocol message by the **PDP** or by another entity trusted to do that conversion.
- 765 8. If the **policy** were to evaluate to TRUE, then the **PDP** returns a response context, perhaps in
766 the form of a SAML response, to the **PEP** containing the "Permit" `saml:Decision` attribute
767 and (optional) **obligations**.
- 768 9. The **PEP** fulfills the **obligations**.

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

775 4.2 XACML Context

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

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



785

786

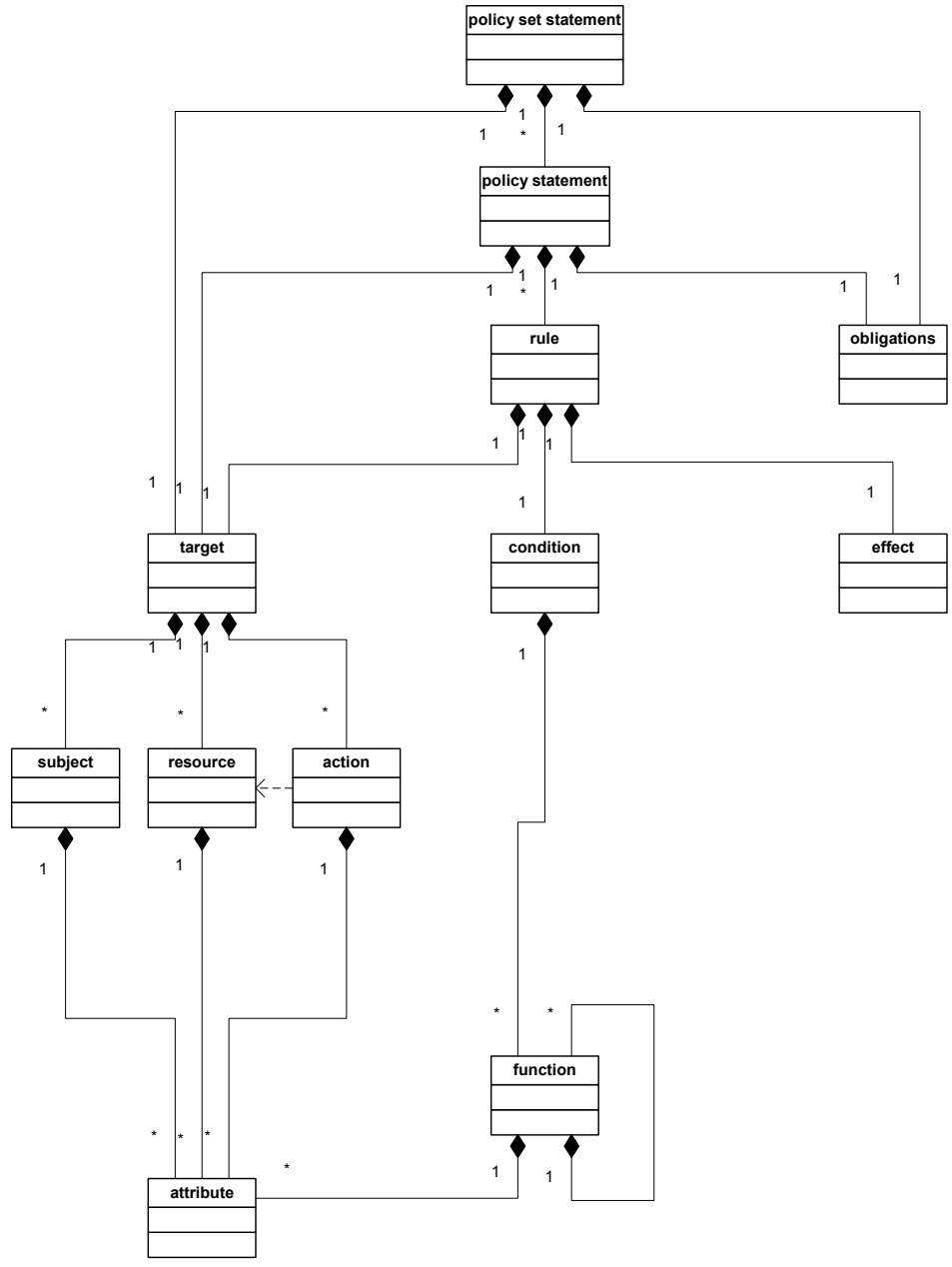
Figure 2 - Context

787 4.3 Policy language model

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

- 789 • **Rule;**
- 790 • **Policy statement;** and
- 791 • **Policy set statement.**

792 These are described in the following sub-sections.



793

794

Figure 3 - Policy language model

795 **4.3.1 Rule**

796 The main components of a *rule* are:

- 797 • a *target*;
- 798 • an *effect*; and
- 799 • a *condition*.

800 These are discussed in the following sub-sections.

801 **4.3.1.1 Target**

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

- 803 • **resources**;
- 804 • **subjects**; and
- 805 • **actions**

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

811 **4.3.1.2 Effect**

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

814 **4.3.1.3 Condition**

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

817 **4.3.1.4 Rule evaluation**

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

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

820 **Table 1 - Rule truth table**

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

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

828 4.3.2 Policy statement

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

- 831 • a **target**;
- 832 • a **rule-combining algorithm**-identifier;
- 833 • a set of **rules**; and
- 834 • **obligations**.

835 4.3.2.1 Target

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

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

- 841 • the **target** of the **policy** may be the union of the **target** definitions for **resource**, **subject** and
842 **action** that are contained in the component **rules**; or
- 843 • the **target** of the **policy** may be the intersection of the **target** definitions for **resource**, **subject**
844 and **action** that are contained in the component **rules**.

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

```
848 <Target>  
849   <Subjects MatchId="function:rfc822Name-equal"  
850   DataType="xs:boolean">  
851     <AttributeDesignator  
852     Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden  
853     tifier:rfc822Name']" DataType="identifier:rfc822Name"/>  
854     <Attribute DataType="identifier:rfc822Name">@</Attribute>  
855   </Subjects>  
856   <Resources MatchId="function:string-match" DataType="xs:boolean">  
857     <AttributeDesignator  
858     Designator="//xacmlContext/Request/Resource/@ResourceURI"  
859     DataType="xs:anyURI"/>  
860     <Attribute  
861     DataType="xs:anyURI">//medico.com/record.*</Attribute>  
862   </Resources>  
863   <Actions MatchId="function:subset" DataType="xs:boolean">  
864     <AttributeDesignator  
865     Designator="//xacmlContext/Action[@Namespace=]"  
866     DataType="xs:string"/>  
867     <Attribute DataType="xs:string">read</Attribute>  
868   </Actions>  
869 </Target>
```

870

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



```

875     <AttributeDesignator
876     Designator="//xacmlContext/Request/Resource/patientName"
877     DataType="xs:string"/>
878 </Condition>

```

879 Following is the merged condition.

```

880 <Condition FunctionId="function:and" DataType="xs:boolean">
881   <Function FunctionId="function:string-match"
882   DataType="xs:boolean">
883     <AttributeDesignator
884     Designator="//xacmlContext/Request/Resource/@ResourceURI"
885     DataType="xs:anyURI"/>
886     <Attribute
887     DataType="xs:anyURI">//medico.com/record.*</Attribute>
888   </Function>
889   <Function FunctionId="function:subset" DataType="xs:boolean">
890     <AttributeDesignator
891     Designator="//xacmlContext/Action[@Namespace=]"
892     DataType="xs:string"/>
893     <Attribute DataType="xs:string">read</Attribute>
894   </Function>
895   <Function FunctionId="function:string-equal"
896   DataType="xs:boolean">
897     <AttributeDesignator
898     Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden
899     tifier:patientName']" DataType="xs:string"/>
900     <AttributeDesignator
901     Designator="//xacmlContext/Request/Resource/patientName"
902     DataType="xs:string"/>
903   </Function>
904 </Condition>

```

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

907 In the case that a **rule target** is present, the **rule** is evaluated according to the truth table of Table
908 1.

909 4.3.2.2 Rule-combining algorithm

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

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

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

916 4.3.2.3 Obligations

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

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

922 4.3.2.4 Example policy statement

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

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

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

```
932 <?xml version="1.0" encoding="UTF-8"?>
933 <saml:Assertion MajorVersion="0" MinorVersion="28"
934 AssertionID="A7F34K90" Issuer="medico.com" IssueInstant="2002-03-
935 22T08:23:47-05:00">
936   <PolicyStatement PolicyId="//medico.com/rules/policy5"
937   RuleCombiningAlgId="urn:oasis:names:tc:XACML:identifier:ruleCombinin
938   gAlgorithms:denyOverrides"
939   xmlns="urn:oasis:names:tc:xacml:0.15i:policy"
940   xmlns:function="urn:oasis:names:tc:xacml:0.15i:function"
941   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
942   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
943   xsi:schemaLocation="urn:oasis:names:tc:xacml:0.15i:policy
944   D:\MYDOCU~1\Standards\XACML\v15\draft-xacml-schema-policy-15i.xsd">
945     <Target>
946       <Subjects MatchId="function:superset" DataType="xs:boolean">
947         <AttributeDesignator
948 Designator="//xacmlContext/Request/Subject/Attribute[@DataType='iden
949 tifier:role']" DataType="xs:string"/>
950         <Attribute DataType="xs:string"></Attribute>
951       </Subjects>
952       <Resources MatchId="function:anyURI-equal"
953 DataType="xs:boolean">
954         <AttributeDesignator
955 Designator="//xacmlContext/Request/Resource/@ResourceURI"
956 DataType="xs:anyURI"/>
957         <Attribute
958 DataType="xs:anyURI">//medico.com/record/medical.*</Attribute>
959       </Resources>
960       <Actions MatchId="function:subset" DataType="xs:boolean">
961         <AttributeDesignator
962 Designator="//xacmlContext/Action[@Namespace=]"
963 DataType="xs:string"/>
964         <Attribute DataType="xs:string">read</Attribute>
965       </Actions>
966     </Target>
967     <RuleSet>
968       <RuleDesignator>
969         <RuleId>//medico.com/rules/rule1</RuleId>
970       </RuleDesignator>
971       <RuleDesignator>
972         <RuleDigest
973 Base64Digest="H7jiE0+jwkn63k/JhB3+D9aI4V3J9z/o0"/>
974       </RuleDesignator>
975       <Rule RuleId="//medico.com/rules/rule3" Effect="Permit">
```

```

976         <Description>A physician may write any medical element for
977 which he or she is the designated primary care
978 physician</Description>
979         <Condition FunctionId="function:and"
980         DataType="xs:boolean">
981             <Function FunctionId="function:string-equal"
982             DataType="xs:boolean">
983                 <AttributeDesignator
984                 Designator="//xacmlContext/Request/Subject/SubjectId"
985                 DataType="xs:string"/>
986                 <AttributeDesignator
987                 Designator="//xacmlContext/Request/Resource/physicianName"
988                 DataType="xs:string"/>
989             </Function>
990             <Function FunctionId="function:present"
991             DataType="xs:boolean">
992                 <AttributeDesignator
993                 Designator="//xacmlContext/Request/Resource/patient/email"
994                 DataType="xs:string"/>
995             </Function>
996         </Condition>
997     </Rule>
998     <RuleDesignator>
999         <RuleId>//medico.com/rules/rule4</RuleId>
1000     </RuleDesignator>
1001 </RuleSet>
1002 <Obligations>
1003     <Obligation ObligationId="//medico.com/emailer"
1004     FulfilOn="Permit">
1005         <AttributeDesignator
1006         Designator="//xacmlContext/Request/Resource/patient/email"
1007         DataType="xs:string"/>
1008         <AttributeAssignment DataType="xs:string"
1009         AttributeId="//medico.com/text">Your medical record has been
1010         accessed by:</AttributeAssignment>
1011         <AttributeDesignator
1012         Designator="//xacmlContext/Request/Subject/SubjectId"
1013         DataType="xs:string"/>
1014     </Obligation>
1015 </Obligations>
1016 </PolicyStatement>
1017 </saml:Assertion>

```

1018 4.3.3 Policy set statement

1019 A *policy set* comprises four main components:

- 1020 • a *target*;
- 1021 • a set of *policy statements*;
- 1022 • *obligations*; and
- 1023 • a *policy-combining algorithm*-identifier.

1024 The *target* and *policy statement* components are to be interpreted as described above.

1025 4.3.3.1 Obligations

1026 The writer of a *policy set statement* MAY add *obligations* to the *policy set*, in addition to those
1027 contained in the component *policies* and *policy sets*.

1028 4.3.3.2 Policy-combining algorithm

1029 The **policy-combining algorithm** is the algorithm by which the results of evaluating the component
1030 **policies** are combined to form the value of the **policy set**. In the case that the **PDP** uses a **policy**
1031 **set** to determine its response to a **decision request**, the `saml:Decision` value is the result of
1032 evaluating the **policy set**.

1033 When a **PDP** evaluates a **policy set** containing **obligations**, it returns certain of those **obligations**
1034 to the **PEP** in its response context. The XACML `<obligation>` elements that are returned to the **PEP**
1035 are those whose `xacml:FulfilOn` attributes have the same value as the result of evaluating the
1036 **policy set**.

1037 As a consequence of this procedure, no **obligations** are returned to the **PEP** if the **policies** from
1038 which they are drawn are not evaluated or their evaluated result is Indeterminate or Not applicable.

1039 See Section 8.8 for an example of a **policy-combining algorithm**.

1040 5 Policy syntax (normative, with the exception of 1041 the schema fragments)

1042 5.1 Element `<PolicySetStatement>`

1043 The `<PolicySetStatement>` element is a top-level element in the XACML schema.

```
1044 <xs:element name="PolicySetStatement"  
1045 type="xacml:PolicySetStatementType"/>
```

1046 5.2 Element `<PolicyStatement>`

1047 The `<PolicyStatement>` element is a top-level element in the XACML schema.

```
1048 <xs:element name="PolicyStatement"  
1049 type="xacml:PolicyStatementType"/>
```

1050 5.3 Element `<Rule>`

1051 The `<Rule>` element is a top-level element in the XACML schema.

```
1052 <xs:element name="Rule" type="xacml:RuleType"/>
```

1053 5.4 Complex type `PolicySetStatementType`

1054 Elements of type `PolicySetStatementType` extend the `saml:StatementAbstractType` so that they
1055 MAY be included in a `<saml:Assertion>` element. The `<saml:Assertion>` element contains
1056 some policy meta-data, such as the identity of the PAP that issued the policy set statement and the
1057 date and time at which it was issued.

1058 The main elements of this type definition are the `<Target>`, `<PolicySet>` and `<Obligations>`
1059 elements and the `policyCombiningAlgId` attribute. The `<PolicySet>` element SHALL contain
1060 references to the set of policies that are to be combined in the policy set. The `<Target>` element
1061 MAY be declared by the creator of elements of this type, or it MAY be computed from the
1062 `<Target>` elements of the referenced `<PolicyStatement>` elements, either as an intersection or

1063 as a union. The <Obligations> element SHALL contain the set of <Obligation> elements
1064 that MUST be discharged by the PEP. The PolicyCombiningAlgId attribute SHALL contain a
1065 identifier of the policy-combining algorithm by which the referenced <PolicyStatement>
1066 elements MUST be combined.

1067 An instance of this type MAY be referenced by its PolicySetId attribute value.

```
1068 <xs:complexType name="PolicySetStatementType">  
1069   <xs:complexContent>  
1070     <xs:extension base="saml:StatementAbstractType">  
1071       <xs:sequence>  
1072         <xs:element name="Description" type="xs:string"  
1073 minOccurs="0"/>  
1074         <xs:element name="Target" type="xacml:TargetType"/>  
1075         <xs:element name="PolicySet" type="xacml:PolicySetType"  
1076 maxOccurs="unbounded"/>  
1077         <xs:element name="Obligations"  
1078 type="xacml:ObligationsType" minOccurs="0"/>  
1079       </xs:sequence>  
1080       <xs:attribute name="PolicySetId" type="xs:anyURI"  
1081 use="required"/>  
1082       <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI"  
1083 use="required"/>  
1084     </xs:extension>  
1085   </xs:complexContent>  
1086 </xs:complexType>
```

1087 5.5 Complex type PolicyStatementType

1088 Elements of type PolicyStatementType extend the saml:StatementAbstractType so that they MAY
1089 be included in a <saml:Assertion> element. The <saml:Assertion> element contains some
1090 policy meta-data, such as the identity of the PAP that issued the policy statement and the date and
1091 time at which it was issued.

1092 The main elements of this type definition are the <Target>, <RuleSet> and <Obligations>
1093 elements and the RuleCombiningAlgId attribute. The <RuleSet> element SHALL contain
1094 references to the <Rule> elements that are to be combined in a policy. The <Target> element
1095 MAY be declared by the creator of elements of this type, or it MAY be computed from the
1096 <Target> elements of the referenced <Rule> elements, either as an intersection or as a union.
1097 The <Obligations> element SHALL contain the set of <Obligation> elements that MUST be
1098 discharged by the PEP. The RuleCombiningAlgId attribute SHALL contain a reference to the
1099 rule-combining algorithm by which the <Rule> elements MUST be combined.

1100 An instance of this type MAY be referenced by its PolicyId attribute value.

```
1101 <xs:complexType name="PolicyStatementType">  
1102   <xs:complexContent>  
1103     <xs:extension base="saml:StatementAbstractType">  
1104       <xs:sequence>  
1105         <xs:element name="Description" type="xs:string"  
1106 minOccurs="0"/>  
1107         <xs:element name="Target" type="xacml:TargetType"/>  
1108         <xs:element name="RuleSet" type="xacml:RuleSetType"  
1109 maxOccurs="unbounded"/>  
1110         <xs:element name="Obligations"  
1111 type="xacml:ObligationsType" minOccurs="0"/>  
1112       </xs:sequence>  
1113       <xs:attribute name="PolicyId" type="xs:anyURI"  
1114 use="required"/>
```

```

1115         <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI"
1116 use="required"/>
1117     </xs:extension>
1118 </xs:complexContent>
1119 </xs:complexType>

```

1120 5.6 Complex type RuleType

1121 The main elements of this type definition are the <Target> and <Condition> elements, and the
1122 Effect attribute.

1123 An instance of this type MAY be referenced by its RuleId attribute value.

```

1124 <xs:complexType name="RuleType">
1125     <xs:sequence>
1126         <xs:element name="Description" type="xs:string"
1127 minOccurs="0"/>
1128         <xs:element name="Target" type="xacml:TargetType"
1129 minOccurs="0"/>
1130         <xs:element name="Condition" type="xacml:ConditionType"
1131 minOccurs="0"/>
1132     </xs:sequence>
1133     <xs:attribute name="RuleId" type="xs:anyURI" use="required"/>
1134     <xs:attribute name="Effect" type="xacml:EffectType"
1135 use="required"/>
1136 </xs:complexType>

```

1137 5.7 Complex type EffectType

1138 This type definition defines the values allowed for the **effect** of a **rule** and the circumstances under
1139 which an **obligation** must be performed.

```

1140 <xs:simpleType name="EffectType">
1141     <xs:restriction base="xacmlContext:DecisionType">
1142         <xs:enumeration value="Permit"/>
1143         <xs:enumeration value="Deny"/>
1144     </xs:restriction>
1145 </xs:simpleType>

```

1146 5.8 Complex type TargetType

1147 Elements of this type identify the set of decision requests of type xacmlContext:RequestType that
1148 the parent element is intended to evaluate. It contains definition for subjects, resources and
1149 actions. If the subject, resource and action identified in the request context match the definitions in
1150 this element, then the policy MAY be used to evaluate the request. All matches MUST be satisfied.
1151 If one or more element in the context satisfies each match, then the match is satisfied.

```

1152 <xs:complexType name="TargetType">
1153     <xs:sequence>
1154         <xs:element name="Subjects" type="xacml:MatchType"
1155 maxOccurs="unbounded"/>
1156         <xs:element name="Resources" type="xacml:MatchType"
1157 maxOccurs="unbounded"/>
1158         <xs:element name="Actions" type="xacml:MatchType"
1159 maxOccurs="unbounded"/>
1160     </xs:sequence>
1161 </xs:complexType>

```

1162 5.9 Complex type MatchType

1163 Elements of type MatchType identify a set of entities by matching values in the context with values
1164 embedded in the policy. The <xacml:AttributeDesignator> element identifies one or more
1165 values in the <xacmlContext:Request> element. It MUST contain a URI that is a legal XPath
1166 expression over the <xacmlContext:Request>. The <xacml:Attribute> MUST contain
1167 a literal value. The types of the two attributes MUST be compatible with the function identified by
1168 the MatchId attribute.

```
1169 <xs:complexType name="MatchType">  
1170 <xs:sequence>  
1171 <xs:element ref="xacml:AttributeDesignator"/>  
1172 <xs:element ref="xacml:Attribute"/>  
1173 </xs:sequence>  
1174 <xs:attribute name="MatchId" type="xacml:MatchIdType"/>  
1175 <xs:attribute name="DataType" type="xs:anyURI"  
1176 fixed="xs:boolean"/>  
1177 </xs:complexType>
```

1178 5.10 Complex type ObligationsType

1179 Elements of type ObligationsType contain a set of <xacml:Obligation> elements.

```
1180 <xs:complexType name="ObligationsType">  
1181 <xs:sequence>  
1182 <xs:element name="Obligation" type="xacml:ObligationType"  
1183 maxOccurs="unbounded"/>  
1184 </xs:sequence>  
1185 </xs:complexType>
```

1186 5.11 Complex type ObligationType

1187 Elements of type ObligationType contain an identifier for the obligation and a set of attributes that
1188 form arguments of the action defined by the obligation. The FulfilOn attribute indicates the
1189 decision value for which this obligation must be fulfilled.

```
1190 <xs:complexType name="ObligationType">  
1191 <xs:choice maxOccurs="unbounded">  
1192 <xs:element ref="xacml:AttributeDesignator"/>  
1193 <xs:element name="AttributeAssignment"  
1194 type="xacml:AttributeAssignmentType"/>  
1195 </xs:choice>  
1196 <xs:attribute name="ObligationId" type="xs:anyURI"  
1197 use="required"/>  
1198 <xs:attribute name="FulfilOn" type="xacml:EffectType"  
1199 use="required"/>  
1200 </xs:complexType>
```

1201 5.12 Element <Function>

1202 <Function> elements reference a function of type FunctionType.

```
1203 <xs:element name="Function" type="xacml:FunctionType"/>
```

1204 5.13 Complex type ConditionType

1205 Elements of type ConditionType are functions whose return type is boolean.

1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220

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

1221 5.14 Complex type FunctionType

1222 Elements of type FunctionType define a function. Xacml-defined functions are described in the
1223 accompanying table. Function definitions may take any combination of <Function>,
1224 <Attribute> and <AttributeDesignator> as arguments. In addition, the function's return
1225 type MUST be included in the DataType attribute.

1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237

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

1238 5.15 Element <Attribute>

1239 <Attribute> elements contain a literal attribute value.

1240

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

1241 5.16 Complex type AttributeType

1242 Elements of type AttributeType contain a literal attribute value. The type of the value MUST be
1243 contained in the DataType attribute. The attribute MAY be of one of the xml:schema embedded
1244 types. Alternatively, it MAY be of a structured type defined in some other namespace.

1245
1246
1247
1248
1249
1250
1251
1252

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


1253 5.17 Element <AttributeDesignator>

1254 <AttributeDesignator> elements reference an attribute value in <xacmlContext:Request>
1255 by means of an XPath expression. The expected type of the attribute MUST be included in
1256 the attributeDesignator.

```
1257 <xs:element name="AttributeDesignator"  
1258 type="xacml:AttributeDesignatorType"/>
```

1259 5.18 Complex type AttributeDesignatorType

1260 Elements of type AttributeDesignatorType reference an attribute value in
1261 <xacmlContext:Request>.

```
1262 <xs:complexType name="AttributeDesignatorType">  
1263 <xs:attribute name="Designator" type="xs:anyURI"/>  
1264 <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
1265 <!-- Designator must be a legal XPath expression over  
1266 xacmlContext:Request -->  
1267 </xs:complexType>
```

1268 5.19 Complex type AttributeAssignmentType

1269 Elements of type AttributeAssignmentType contain attribute contents and an AttributeId. The
1270 AttributeId is part of attribute meta-data, and is used when an attribute cannot be referenced by
1271 its location in <xacmlContext:Request>. This situation may arise in <Obligation>.

```
1272 <xs:complexType name="AttributeAssignmentType">  
1273 <xs:complexContent>  
1274 <xs:extension base="xacml:AttributeType">  
1275 <xs:attribute name="AttributeId" type="xs:anyURI"/>  
1276 </xs:extension>  
1277 </xs:complexContent>  
1278 </xs:complexType>
```

1279 5.20 Complex type PolicySetType

1280 Elements of type PolicySetType identify a set of policies. Members of the set MAY be identified by
1281 any of the following: reference to a <PolicySetStatement> by its PolicySetId attribute,
1282 reference to a <PolicyStatement> by its PolicyId attribute, inclusion of a
1283 <PolicySetStatement>, inclusion of a <PolicyStatement>, inclusion of a
1284 <saml:Assertion> containing a <PolicySetStatement> or inclusion of a
1285 <saml:Assertion> containing a <PolicyStatement>. The referenced policies MUST be
1286 combined as defined by the policy-combining algorithm identified by the PolicyCombiningAlgId
1287 attribute in the parent <PolicySetStatement>.

```
1288 <xs:complexType name="PolicySetType">  
1289 <xs:choice maxOccurs="unbounded">  
1290 <xs:element name="PolicySetId" type="xs:anyURI"/>  
1291 <xs:element name="PolicyId" type="xs:anyURI"/>  
1292 <xs:element ref="xacml:PolicySetStatement"/>  
1293 <xs:element ref="xacml:PolicyStatement"/>  
1294 <xs:element name="PolicySetAssertion"  
1295 type="saml:AssertionType"/>  
1296 <xs:element name="PolicyAssertion"  
1297 type="saml:AssertionType"/>  
1298 </xs:choice>
```

1299

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

1300 5.21 Complex type RuleSetType

1301 Elements of type RuleSetType SHALL contain a set of <Rule> or <RuleDesignator> elements.

1302
1303
1304
1305
1306
1307
1308

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

1309 5.22 Complex type RuleDesignatorType

1310 Elements of type RuleDesignatorType SHALL designate a rule by identifier or by digest.

1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322

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

1323 6 Function names and legal type combinations

1324 6.1 Functions

1325 The table in this section lists the combinations of datatypes for which the various functions. For
1326 each function name, the table indicates the valid combination of datatypes and the datatype of the
1327 result.

Function Name	Function Data Type	First operand Data Type	Remaining operands Data Type	Operation
function:integer-add	xs:integer	xs:integer	xs:integer	A + B
function:decimal-add	xs:decimal	xs:decimal	xs:decimal	A + B
function:add-yearMonthDuration- to-date	xs:date	xs:date	xs:yearMonthD uration	A + B
function:add-yearMonthDuration- to-date	xs:date	xs:date	xs:dayTimeDur ation	A + B
function:add-dayTimeDuration-	xs:time	xs:time	xs:dayTimeDur	A + B

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

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

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

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

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

1328 **7 Context syntax (normative, with the exception of**
1329 **the schema fragments)**

1330 **7.1 Element <Request>**

1331 The <Request> element is a top-level element in the XACML context schema.

1332

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

1333 **7.2 Element <Response>**

1334 The <Response> element is a top-level element in the XACML context schema.

1335

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

1336 **7.3 Complex type RequestType**

1337 Elements of type RequestType contain the data required by the PDP in order to render a decision.
1338 This includes information about the subjects, resource and actions, as well as environmental
1339 information that pertains to none of these.

1340

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

1341 <xs:sequence>
```

```

1342     <xs:element name="Subject" type="xacmlContext:SubjectType"
1343 maxOccurs="unbounded"/>
1344     <xs:element name="Resource"
1345 type="xacmlContext:ResourceType"/>
1346     <xs:element name="Action" type="xacmlContext:ActionType"
1347 maxOccurs="unbounded"/>
1348     <xs:element name="Environment"
1349 type="xacmlContext:EnvironmentType" minOccurs="0"/>
1350   </xs:sequence>
1351 </xs:complexType>

```

1352 7.4 Complex type ResponseType

1353 Elements of type ResponseType contain one or more results of a policy evaluation.

```

1354 <xs:complexType name="ResponseType">
1355   <xs:sequence>
1356     <xs:element name="Result" type="xacmlContext:ResultType"
1357 maxOccurs="unbounded"/>
1358   </xs:sequence>
1359 </xs:complexType>

```

1360 7.5 Complex type ResultType

1361 Elements of type ResultType contain information related to a single decision, including the value of
1362 the decision, the resource to which it relates, and any obligations and advice associated with the
1363 decision.

```

1364 <xs:complexType name="ResultType">
1365   <xs:sequence>
1366     <xs:element name="Decision"
1367 type="xacmlContext:DecisionType"/>
1368     <xs:element name="ResourceId" type="xs:string"
1369 minOccurs="0"/>
1370     <xs:element name="Obligations" type="xacml:ObligationsType"
1371 minOccurs="0"/>
1372     <xs:element name="Advice" type="xacmlContext:AdviceType"
1373 minOccurs="0" maxOccurs="unbounded"/>
1374   </xs:sequence>
1375 </xs:complexType>

```

1376 7.6 Complex type SubjectType

1377 Elements of type SubjectType identify a subject of a request context by means of an identifier or a
1378 key. Optionally, attributes of the subject MAY be provided and information relating to the PEP's
1379 authentication of the subject MAY be supplied.

```

1380 <xs:complexType name="SubjectType">
1381   <xs:sequence>
1382     <xs:choice minOccurs="0">
1383       <xs:element name="SubjectId"
1384 type="xacmlContext:SubjectIdType"/>
1385       <xs:element ref="ds:KeyInfo"/>
1386     </xs:choice>
1387     <xs:element name="SubjectAttribute"
1388 type="xacmlContext:AttributeType" minOccurs="0"
1389 maxOccurs="unbounded"/>

```



```

1390     <xs:element name="AuthenticationInfo"
1391 type="xacmlContext:AuthenticationInfoType" minOccurs="0"
1392 maxOccurs="unbounded"/>
1393     </xs:sequence>
1394     <xs:attribute name="SubjectCategory" type="xs:anyURI"
1395 default="identifier:AccessSubject"/>
1396 </xs:complexType>

```

1397 7.7 Complex type SubjectIdType

1398 Elements of type SubjectIdType contain information that identifies a subject. The identifier itself is a
1399 string. However, Format and Qualifier attributes are included to assist with the interpretation of
1400 the string. The Format attribute indicates the name-form of the identifier and hence the function by
1401 which it MUST be matched. (Note: why not call this "DataType", to be consistent throughout?) The
1402 qualifier indicates the security or administrative domain that qualifies the name of the subject. It
1403 provides a means to federate names from disparate user stores without collision. (Note: Why isn't
1404 this a name, with an accompanying DataType? Why isn't it a list of names?).

```

1405 <xs:complexType name="SubjectIdType">
1406   <xs:simpleContent>
1407     <xs:extension base="xs:string">
1408       <xs:attribute name="Format" type="xs:anyURI"
1409 use="optional"/>
1410       <xs:attribute name="Qualifier" type="xs:string"
1411 use="optional"/>
1412     </xs:extension>
1413   </xs:simpleContent>
1414 </xs:complexType>

```

1415 7.8 Complex type AuthenticationInfoType

1416 Elements of this type contain information related to the PEP's authentication of the subject.

```

1417 <xs:complexType name="AuthenticationInfoType">
1418   <xs:attribute name="Method" type="xs:anyURI" use="optional"/>
1419   <xs:attribute name="Instant" type="xs:dateTime" use="optional"/>
1420 </xs:complexType>

```

1421 7.9 Complex type AttributeType

1422 Elements of this type contain an attribute and attribute meta-data. It extends the xacml definition of
1423 attribute with an AttributeId, and Issuer identity and an IssueInstant.

```

1424 <xs:complexType name="AttributeType">
1425   <xs:complexContent>
1426     <xs:extension base="xacml:AttributeType">
1427       <xs:attribute name="AttributeId" type="xs:anyURI"
1428 use="required"/>
1429       <xs:attribute name="Issuer" type="xs:anyURI"
1430 use="optional"/>
1431       <xs:attribute name="IssueInstant" type="xs:dateTime"
1432 use="optional"/>
1433     </xs:extension>
1434   </xs:complexContent>
1435 </xs:complexType>

```

1436 7.10 Complex type ResourceType

1437 Elements of this type contain information about the resource for which access is being requested.
1438 It MAY contain any combination of <ResourceSpecifier>, <ResourceContent> and
1439 <ResourceAttribute> elements. If present, the <ResourceAttribute> elements contain a
1440 an attribute of the resource.

```
1441 <xs:complexType name="ResourceType">
1442   <xs:sequence>
1443     <xs:element name="ResourceSpecifier"
1444       type="xacmlContext:ResourceSpecifierType" minOccurs="0"/>
1445     <xs:element name="ResourceContent"
1446       type="xacmlContext:ResourceContentType" minOccurs="0"/>
1447     <xs:element name="ResourceAttribute"
1448       type="xacmlContext:AttributeType" minOccurs="0"
1449       maxOccurs="unbounded"/>
1450   </xs:sequence>
1451 </xs:complexType>
```

1452 7.11 Complex type ResourceSpecifierType

1453 Elements of this type SHALL contain a ResourceId. This is in the form of a string. Interpretation of
1454 the string depends upon the value of the Format attribute. (Note: Perhaps the format attribute
1455 should be required). The scope attribute is used in the case where the resource is structured as a
1456 hierarchy. It indicates which part of the resource the decision request applies to.

```
1457 <xs:complexType name="ResourceSpecifierType">
1458   <xs:attribute name="Format" type="xs:anyURI" use="optional"/>
1459   <xs:attribute name="Scope"
1460     type="xacmlContext:SpecifierScopeType" use="optional"/>
1461   <xs:attribute name="ResourceId" type="xs:string"
1462     use="required"/>
1463 </xs:complexType>
```

1464 7.12 Complex type SpecifierScopeType

1465 Elements of this type indicate which part of a resource a decision request applies to. The value
1466 Immediate indicates the request applies just to the node of the resource identified by the
1467 ResourceId in the parent element. The Children value indicates that the request applies to the
1468 node identified in the parent element and its immediate children. The Descendants value indicates
1469 that the request applies to the node identified in the parent element and all its descendants.

```
1470 <xs:simpleType name="SpecifierScopeType">
1471   <xs:restriction base="xs:string">
1472     <xs:enumeration value="Immediate"/>
1473     <xs:enumeration value="Children"/>
1474     <xs:enumeration value="Descendants"/>
1475   </xs:restriction>
1476 </xs:simpleType>
```

1477 7.13 Complex type ResourceContentType

1478 Elements of this type contain the resource to which access is requested.

```
1479 <xs:complexType name="ResourceContentType">
1480   <xs:sequence>
1481     <xs:any namespace="##any" processContents="lax" minOccurs="0"
1482       maxOccurs="unbounded"/>
1483 </xs:complexType>
```

1483
1484
1485

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

1486 7.14 Complex type ActionType

1487 Elements of type ActionType contain a specification of the requested actions (Note: should this be
1488 of type xs:list? It would mean that an individual action could not contain whitespace).

1489
1490
1491
1492
1493
1494
1495

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

1496 7.15 Complex type DecisionType

1497 Elements of type DecisionType contain the result of policy evaluation.

1498
1499
1500
1501
1502
1503
1504

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

1505 7.16 Complex type EnvironmentType

1506 Elements of type EnvironmentType contain a set of attributes of the environment. These attributes
1507 MAY form part of policy evaluation.

1508
1509
1510
1511
1512
1513

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

1514 7.17 Complex type AdviceType

1515 Elements of type AdviceType contain information that MAY be used by the PEP. (Note: if we don't
1516 have a specific use for this, why don't we leave it out in this version? Users of the specification will
1517 still be able to extend the response schema to include advice, if they have a definite need for it).

1518
1519
1520
1521
1522
1523
1524
1525

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

1526 8 XACML identifiers (normative)

1527 This section defines standard identifiers for commonly-used entities. All XACML-defined identifiers
1528 have the common base:

1529 urn:oasis:names:tc:XACML:identifier

1530 8.1 Access Subject

1531 The identifier for the system entity that is requesting access.

1532 urn:oasis:names:tc:xacml:identifier:AccessSubject

1533 8.2 Time of day

1534 8.3 Attributes

1535 XACML-defined attributes are represented by an element of type <saml:AttributeDesignatorType>.
1536 It has two attributes: AttributeNamespace and AttributeName. All XACML-defined attributes have
1537 the following value for AttributeNamespace:

1538 urn:oasis:names:tc:XACML:identifier:attributes/

1539 8.3.1 Role

1540 urn:oasis:names:tc:XACML:identifier:attributes/role

1541 8.3.2 RFC822 Name

1542 RFC822 name attributes have the following value of AttributeName:

1543 urn:oasis:names:tc:XACML:identifier:rfc822Name

1544 8.3.3 X.500 distinguished name

1545 X.500 distinguished name attributes have the following value of AttributeName:

1546 urn:oasis:names:tc:XACML:identifier:x500Name

1547 8.3.4 Unix file-system path

1548 UNIX file-system path attributes have the following value of AttributeName:

1549 urn:oasis:names:tc:XACML:identifier:attribute:UFS

1550 8.3.5 Uniform resource identifier

1551 Uniform resource identifier attributes have the following value of AttributeName:

1552 urn:oasis:names:tc:XACML:identifier:attribute:URI

1553 **8.4 Authentication locality**

1554 **8.5 Deny-overrides rule-combining algorithm**

1555 The deny-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

1556 urn:oasis:names:tc:XACML:identifier:ruleCombiningAlgorithms:denyOverrides

1557 **8.6 Deny-overrides policy-combining algorithm**

1558 The deny-overrides policy-combining algorithm has the following value for policyCombiningAlgId:

1559 urn:oasis:names:tc:XACML:identifier:policyCombiningAlgorithms:denyOverrides

1560 **8.7 Permit-overrides rule-combining algorithm**

1561 The permit-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

1562 urn:oasis:names:tc:XACML:identifier:ruleCombiningAlgorithms:permitOverrides

1563 **8.8 Permit-overrides policy-combining algorithm**

1564 The permit-overrides policy-combining algorithm has the following value for policyCombiningAlgId:

1565 urn:oasis:names:tc:XACML:identifier:policyCombiningAlgorithms:permitOverride
1566 s

1567 **9 Combining algorithms (normative)**

1568 This section contains a description of the rule-combining and policy-combining algorithms specified
1569 by XACML.

1570 **9.1 Deny-overrides**

1571 The following is a specification for the "deny-overrides" rule-combining algorithm. The identifier for
1572 this algorithm is given in Section 8.5.

1573 In the entire set of rules to be evaluated, if any of the rules evaluates to "deny", then the
1574 rule combination is defined to evaluate to "deny" (that is, "deny" takes precedence,
1575 regardless of how many rules evaluate to "permit", and causes the whole combination to
1576 return "deny"). Any rule that evaluates to "indeterminate" (that is, its return status cannot
1577 be determined for any reason) has the same effect as a "deny" in that it causes the
1578 combination to return "deny". Finally, if none of the rules are found to be applicable to the
1579 request, the rule combination returns "notApplicable".

1580 What follows is a pseudocode representation of how the above specification MAY be implemented.
1581 This is provided for illustrative and explanatory purposes.

```
1582 effect policy(rule[]){  
1583     atLeastOnePermit = false;  
1584     for( i=0; i<=noOfRules; i++ ){  
1585         if(rule[i] == deny){  
1586             return(deny);
```

```

1587     }
1588     if(rule[i] == indeterminate){
1589         return(deny);
1590     }
1591     if(rule[i] == permit){
1592         atLeastOnePermit = true;
1593     }
1594 }
1595 if atLeastOnePermit {
1596     return(permit);
1597 }
1598 else{
1599     return(notApplicable);
1600 }
1601 }

```

1602 The following is a specification for the "deny-overrides" policy-combining algorithm. The identifier
1603 for this algorithm is given in Section 7.6.

1604 In the entire set of policies to be evaluated, if any of the policies evaluates to "deny", then
1605 the policy combination is defined to evaluate to "deny" (that is, "deny" takes precedence,
1606 regardless of how many policies evaluate to "permit", and causes the whole combination to
1607 return "deny"). Any policy that evaluates to "indeterminate" (that is, its return status cannot
1608 be determined for any reason) has the same effect as a "deny" in that it causes the
1609 combination to return "deny". Finally, if none of the policies are found to be applicable to
1610 the request, the policy combination returns "notApplicable".

1611 What follows is a pseudocode representation of how the above specification MAY be implemented.
1612 This is provided for illustrative and explanatory purposes.

```

1613 effect policySet(policy[]){
1614     atLeastOnePermit = false;
1615     for( i=0; i<=noOfPolicies; i++ ){
1616         if(policy[i] == deny){
1617             return(deny);
1618         }
1619         if(policy[i] == indeterminate){
1620             return(deny);
1621         }
1622         if(policy[i] == permit){
1623             atLeastOnePermit = true;
1624         }
1625     }
1626     if atLeastOnePermit {
1627         return(permit);
1628     }
1629     else{
1630         return(notApplicable);
1631     }
1632 }

```

1632 Obligations of the individual policies SHALL be combined as described in Section 4.3.2.3.

1633 9.2 Permit-overrides

1634 The following is a specification for the "permit-overrides" rule-combining algorithm. The identifier
1635 for this algorithm is given in Section 8.7.

1636 In the entire set of rules to be evaluated, if any of the rules evaluates to "permit", then the
1637 rule combination is defined to evaluate to "permit" (that is, "permit" takes precedence,
1638 regardless of how many rules evaluate to "deny" or "indeterminate", and causes the whole
1639 combination to return "permit"). If all of the rules found to be applicable to the request

1640 evaluate to "deny" or "indeterminate", then the rule combination is defined to evaluate to
1641 "deny". If none of the rules is found to be applicable to the request, the rule combination
1642 returns "notApplicable".

1643 What follows is a pseudocode representation of how the above specification MAY be implemented.
1644 This is provided for illustrative and explanatory purposes.

```
1645 effect policy(rule[]) {  
1646     atLeastOneDenyOrIndeterminate = false;  
1647     for( i=0; i<=noOfRules; i++) {  
1648         if (rule[i] == permit) {  
1649             return(permit);  
1650         }  
1651         if (rule[i] == indeterminate) {  
1652             atLeastOneDenyOrIndeterminate = true;  
1653         }  
1654         if (rule[i] == deny) {  
1655             atLeastOneDenyOrIndeterminate = true;  
1656         }  
1657     }  
1658     if atLeastOneDenyOrIndeterminate {  
1659         return(deny);  
1660     } else {  
1661         return(notApplicable);  
1662     }  
1663 }
```

1664 The following is a specification for the "permit-overrides" policy-combining algorithm. The identifier
1665 for this algorithm is given in Section 8.8.

1666 In the entire set of policies to be evaluated, if any of the policies evaluates to "permit", then
1667 the policy combination is defined to evaluate to "permit" (that is, "permit" takes precedence,
1668 regardless of how many policies evaluate to "deny" or "indeterminate", and causes the
1669 whole combination to return "permit"). If all of the policies found to be applicable to the
1670 request evaluate to "deny" or "indeterminate", then the policy combination is defined to
1671 evaluate to "deny". If none of the policies is found to be applicable to the request, the
1672 policy combination returns "notApplicable".

1673 What follows is a pseudocode representation of how the above specification MAY be implemented.
1674 This is provided for illustrative and explanatory purposes.

```
1675 effect policySet(policy[]) {  
1676     atLeastOneDenyOrIndeterminate = false;  
1677     for( i=0; i<=noOfPolicies; i++) {  
1678         if (policy[i] == permit) {  
1679             return(permit);  
1680         }  
1681         if (policy[i] == indeterminate) {  
1682             atLeastOneDenyOrIndeterminate = true;  
1683         }  
1684         if (policy[i] == deny) {  
1685             atLeastOneDenyOrIndeterminate = true;  
1686         }  
1687     }  
1688     if atLeastOneDenyOrIndeterminate {  
1689         return(deny);  
1690     } else {  
1691         return(notApplicable);  
1692     }  
1693 }
```

1694 Obligations of the individual policies SHALL be combined as described in Section 4.3.2.3.

1695

10 Profiles (normative but not mandatory to implement)

1696

1697

10.1 XACML

1698

Describes subsets of XACML appropriate to general classes of problem

1699

10.2 SAML

1700

Describes the subset of SAML that is relevant to XACML

1701

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

1702

- **PDP** has no policy for the requested target

1703

- **PDP** cannot retrieve the required attributes

1704

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

1705

1706

1707

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

1708

1709

1710

```
<xsl:stylesheet version = '1.0'
```

1711

```
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
```

1712

```
xmlns:saml='http://www.oasis-
```

1713

```
open.org/committees/security/docs/draft-sstc-schema-assertion-
```

1714

```
28.xsd' xmlns:sampl="http://www.oasis-
```

1715

```
open.org/committees/security/docs/draft-sstc-schema-protocol-
```

1716

```
28.xsd">
```

1717

1718

```
<xsl:template match="sampl:Request">
```

1719

```
  <Request>
```

1720

```
    <xsl:apply-templates
```

1721

```
select="sampl:AuthorizationDecisionQuery/saml:Subject"/>
```

1722

```
    <xsl:apply-templates
```

1723

```
select="sampl:AuthorizationDecisionQuery/saml:Action"/>
```

1724

```
    <xsl:apply-templates select="sampl:AuthorizationDecisionQuery"
```

1725

```
mode="Resource"/>
```

1726

```
  </Request>
```

1727

```
</xsl:template>
```

1728

1729

```
<xsl:template match="saml:NameIdentifier">
```

1730

```
  <xsl:element name="Subject">
```

1731

```
    <xsl:attribute name="SubjectCategory">
```

1732

```
urn:oasis:names:tc:xacml:identifiers:AccessSubject</xsl:te
```

1733

```
xt>
```

1734

```
  </xsl:attribute>
```

1735

```
  <xsl:element name="SubjectId">
```

1736

```
    <xsl:if test="@NameQualifier">
```

1737

```
      <xsl:attribute name="NameQualifier">
```

1738

```
        <xsl:value-of select="@NameQualifier"/>
```

1739

```
      </xsl:attribute>
```

1740

```
    </xsl:if>
```

1741

```
  <xsl:if test="@Format">
```

1742


```

1743         <xsl:attribute name="Format">
1744             <xsl:value-of select="@Format"/>
1745         </xsl:attribute>
1746     </xsl:if>
1747     <xsl:value-of select="."/>
1748 </xsl:element>
1749 <xsl:apply-templates select="//saml:Evidence" mode="Evidence"/>
1750 </xsl:element>
1751 </xsl:template>
1752
1753 <xsl:template match="saml:SubjectConfirmation">
1754     <xsl:element name="SubjectAttribute">
1755         <xsl:element name="SubjectMetaData">
1756             <xsl:attribute name="Name">
1757                 <xsl:text>SubjectConfirmation</xsl:text>
1758             </xsl:attribute>
1759         </xsl:element>
1760         <xsl:element name="AttributeValue">
1761             <xsl:copy-of select="."/>
1762         </xsl:element>
1763     </xsl:element>
1764 </xsl:template>
1765
1766 <xsl:template match="saml:Evidence" mode="Evidence">
1767     <xsl:element name="SubjectAttribute">
1768         <xsl:element name="SubjectMetaData">
1769             <xsl:attribute name="Name">
1770                 <xsl:text>Evidence</xsl:text>
1771             </xsl:attribute>
1772         </xsl:element>
1773         <xsl:element name="AttributeValue">
1774             <xsl:copy-of select="."/>
1775         </xsl:element>
1776     </xsl:element>
1777 </xsl:template>
1778
1779 <xsl:template match="samlp:AuthorizationDecisionQuery"
1780 mode="Resource">
1781     <xsl:element name="Resource">
1782         <xsl:element name="ResourceSpecifier">
1783             <xsl:attribute name="ResourceURI">
1784                 <xsl:value-of select="@Resource"/>
1785             </xsl:attribute>
1786         </xsl:element>
1787     </xsl:element>
1788 </xsl:template>
1789
1790 <xsl:template match="saml:Action">
1791     <Action>
1792         <xsl:if test="@Namespace">
1793             <xsl:value-of select="@Namespace"/>
1794         </xsl:if>
1795     </xsl:if>
1796     <xsl:value-of select="."/>
1797 </Action>
1798 </xsl:template>
1799
1800 </xsl:stylesheet>

```

1801 **10.3 XML Digital Signature**

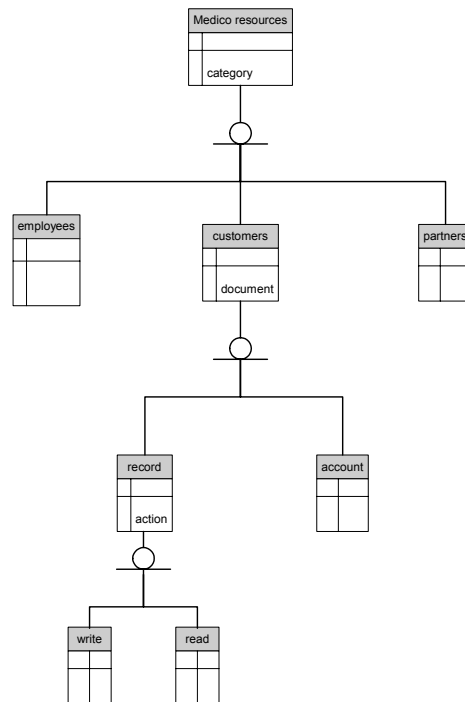
1802 Describes how XACML instances shall be integrity-protected in the case where XML DSig is used.
1803 **PAPs** MAY sign XACML <policyStatement> elements. When a **PAP** combines <policyStatement>
1804 elements, it MAY sign the resulting <policySetStatement> element.

1805 **10.4 LDAP**

1806 The <policyStatement> and <policySetStatement> elements MAY be distributed from the
1807 PAP to the PDP by means of an LDAP repository. In this case, conformant implementations
1808 SHALL behave as described in this section.

1809 **10.4.1 Directory information tree (DIT)**

1810 The <xacml:target> element conforms to a data model. XACML does not specify the **target**
1811 data model, but it MUST be agreed between the **PAP** and the **PDP**. The data model MUST be
1812 semi-hierarchical. That is, it MUST have one or more disjoint trees for **resources** and/or **subjects**.
1813 **Actions** are leaf nodes of the **resource** node to which they apply (see Figure 4). Each level in the
1814 tree is identified with an **attribute** name. A "path" in the tree is a list of attribute-name/value pairs
1815 linking a node to the root. The form of a **target** is a set of paths, one or more for each tree in the
1816 data model.



1817

1818

Figure 4 - Medico Inc "resource" tree

1819 Figure 4 gives an example of a resource tree. One path in this tree is defined by the following list of
1820 attribute-name/value pairs:

1821
1822

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

1823 The Directory Information Tree of the repository SHALL be congruent with that of the **target** data
1824 model. A <policyStatement> element shall be an LDAP attribute of the entry at the lowest

1825 node of every path in the DIT. In our example, the policy for reading a customer record SHALL be
1826 an attribute of the entry defined by the above path. In practice, the **policy statements** may be
1827 referenced from these entries rather than stored at them.

1828 A node MAY have more than one **target** associated with it.

1829 An authorization **decision request** also specifies a set of paths by (directly or indirectly) providing
1830 **resource**, **subject** and **action attribute** values.

1831 A **policy statement** is said to be applicable to a **decision request** if and only if every path in the
1832 policy statement's <target> element is part of a path in the input context's <Request> element.

1833 For instance, a **policy statement** whose **target** is:

```
1834 

---

Root = Medico resources: category = customer

---


```

1835 is attached to that entry in the DIT and is applicable to an input context whose <Request> element
1836 identifies the following resource/action:

```
1837 

---

Root = Medico resources: category = customer: document = record: action =  
1838 read

---


```

1839 10.4.2 Policy combination

1840 When **policy statements** are combined in a **policy set statement**, the **policy set statement**
1841 **target** MUST be computed, and the repository must be updated. **Policy statements** that conform
1842 to different **target** data models MUST NOT be combined.

1843 The **policy set statement target** SHALL be computed by separately combining trees of the same
1844 type from each of the original **policy statement targets**. The combination may be in the form of a
1845 union or an intersection.

1846 A union combination retains all of the original paths. If, as the result, all possible paths containing a
1847 particular DIT node are retained, then the path may be truncated at that node.

1848 An intersection combination retains a path from one **target** if and only if it includes a path from the
1849 other **target**.

1850 The **policy set statement** SHOULD be stored at the lowest node of every retained path.

1851 Some attribute values may (themselves) have an internal tree structure (e.g. DNS names). Sub-
1852 trees of such structures SHALL be represented by a regular expression (ref). When such an
1853 **attribute** defines a level in a **target** tree, the sub-tree defined by each node at that level SHALL be
1854 attached at that node.

1855 10.4.3 Directory schema

1856 This directory schema defines an auxiliary object class (**xacmlPolicyInfo**) for adding XACML policy
1857 data to entries, as well as a directory attribute (**xacmlPolicyData**) to contain the policies or
1858 references to entries containing policies.

1859 Alternatively, XACML policies may be stored in policy-specific entries and referenced from the
1860 resource, action and/or subject entries to which they relate. This schema defines a structural object
1861 class (**xacmlPolicyObject**) for defining such entries, as well as a directory attribute
1862 (**xacmlPolicyRDN**) to contain the string used to name the policy-specific entry in the directory. The
1863 **xacmlPolicyData** directory attribute is also used in these entries to contain the policies themselves.

1864 A **PDP** uses an LDAP Directory User Agent (DUA) to search the resource/subject trees in the
1865 directory to find the resource, action or subject of interest and retrieve the **xacmlPolicyData**
1866 directory attribute from that entry. That attribute may contain the XACML policy or a pointer to

1867 another directory entry that contains the XACML policy. If it contains only a pointer, the PDP must
1868 query the directory again to retrieve the **xacmlPolicyData** directory attribute from the entry related
1869 to the pointer. The content of the pointer is the value of the **xacmlPolicyRDN** directory attribute
1870 that is the final Relative Distinguished Name (RDN) for the policy entry in the directory.

1871 It is the **PDP's** responsibility to confirm that the retrieved policy is applicable to the **decision**
1872 **request** (i.e., the input context) that it is processing.

1873 10.4.4 Object Class Definitions

1874 The following object classes are defined for the LDAP profile for XACML.

1875 10.4.4.1 XACML Policy Info

1876 The **xacmlPolicyInfo** object class is used in defining entries for objects that hold XACML policy
1877 information in addition to other data (e.g., as part of a resource, action, or subject entry).

1878

```
1879 xacmlPolicyInfo OBJECT-CLASS ::= {  
1880     SUBCLASS OF {top}  
1881     KIND      auxiliary  
1882     MAY CONTAIN {xacmlPolicyData}  
1883     ID        id-???-oc-xacmlPolicyInfo }
```

1884 10.4.4.2 XACML Policy Object

1885 The **xacmlPolicyObject** object class is used in defining entries for objects that hold only XACML
1886 policy information.

1887

```
1888 xacmlPolicyObject OBJECT-CLASS ::= {  
1889     SUBCLASS OF {top}  
1890     KIND      structural  
1891     MUST CONTAIN {xacmlPolicyRDN}  
1892     MAY CONTAIN {xacmlPolicyData}  
1893     ID        id-???-oc-xacmlPolicyObject }
```

1894

1895 The **xacmlPolicyRDN** directory attribute is used to name the entry and position it in a policy
1896 subtree.

1897 10.4.5 Attribute Definitions

1898 The following directory attributes are defined for the LDAP profile for XACML.

1899 10.4.5.1 XACML Policy Data

1900 The **xacmlPolicyData** directory attribute is used to store XACML policy information.

1901

```
1902 xacmlPolicyData ATTRIBUTE ::= {  
1903     WITH SYNTAX XacmlPolicySyntax  
1904     ID        id-???-at-xacmlPolicyData }
```

1905

```
1906 XacmlPolicySyntax ::= SEQUENCE {  
1907     policyPointer [0] UTF8String OPTIONAL,
```

1908
1909
1910

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

1911
1912
1913

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

1914
1915

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

1916

10.4.5.2 XACML Policy RDN

1917
1918

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

1919

1920
1921
1922
1923

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

1924

10.4.6 Matching Rule Definitions

1925
1926

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

1927

1928
1929
1930

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

1931
1932
1933

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

1934

11 Operational Model (normative)

1935

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

1936

11.1 Policy Decision Point (PDP)

1937
1938
1939
1940
1941
1942
1943

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

1944

Decision Convergence

1945
1946
1947

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

1948 When the PDP returns an decision of "indeterminate" with an error code of "insufficient
1949 information", a PDP MUST NOT list the names of any attribute of the subject or the resource of the
1950 request for which values were already supplied in the request. Note, this requirement forces the
1951 PDP to eventually return a decision of "permit", "deny", or "indeterminate" with some other reason,
1952 in response to successively-refined requests.

1953 **12 XACML extensibility points (non-normative)**

1954 Describes the points within the XACML model and schema where extensions can be added

1955 **12.1 URIs**

1956 The following XML attributes are URIs.

- 1957 • Function,
- 1958 • ruleCombiningAlgId,
- 1959 • policyCombiningAlgId,
- 1960 • saml:AttributeNameSpace and
- 1961 • saml:AttributeName.

1962 **13 Security and privacy (non-normative)**

1963 This section identifies possible security and privacy vulnerabilities that should be considered when
1964 implementing an XACML-based system. This section is strictly informative. It has been left to the
1965 implementers to assess whether these vulnerabilities apply to their environment and to select the
1966 appropriate safeguards.

1967 **13.1 Authentication**

1968 Authentication here means the ability of one party in a transaction to determine the identity of the
1969 other party in the transaction. Authentication may be in one direction, or it may be bilateral¹.

1970 Given the sensitive nature of access-control systems, it is important for a **PEP** to authenticate the
1971 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that another
1972 process could provide false or invalid **authorization decisions** and compromise security of the
1973 access-control system.

1974 It is equally important for a **PDP** to authenticate the identity of its clients and assess the level of
1975 trust to determine what, if any, sensitive data should be passed. One should keep in mind that
1976 even simple permit or deny responses could be exploited if someone was allowed to make
1977 unlimited requests to a **PDP**.

1978 Many different techniques may be used to provide this authentication, such as co-located code, a
1979 private network, a VPN, or digital signatures. Authentication may also be done as part of the
1980 communication protocol used to exchange the contexts. In this case, the authentication may be
1981 performed at the message level or at the session level.

1982 **13.2 Confidentiality**

1983 Confidentiality means that the contents of a message can be read only by the desired recipients
1984 and not by anyone else who encounters the message while it is in transit². There are two areas in
1985 which confidentiality should be considered: one is confidentiality during transmission; the other is
1986 confidentiality within a `<policyStatement>`.

1987 **13.2.1 Communication Confidentiality**

1988 All data within an access-control system should be treated as confidential. This includes the
1989 `<policyStatement>`, the XACML requests and responses, and any external data that may be
1990 referenced as part of the decision-making process. If someone is able to eavesdrop on the
1991 communication they may be able to understand under what circumstances access will be granted,
1992 which may allow them to impersonate a valid request.

1993 Any security concerns or requirements related to transmitting or exchanging XACML
1994 `<policyStatement>` elements are outside the scope of the XACML standard. While it is often
1995 important to ensure that the integrity and confidentiality of `<policyStatement>` elements is
1996 maintained when they are exchanged between two parties, it is left to the implementers to
1997 determine the appropriate mechanisms for their environment.

1998 **13.2.2 Statement Level Confidentiality**

1999 In some cases, an implementation may want to encrypt only parts of an XACML policy. For
2000 instance, a **PRP** only needs access to the target elements in order to find the appropriate rules.
2001 The other elements could be encrypted while they are stored in a repository.

2002 The XML Encryption Syntax and Processing standard from W3C can be used to encrypt all or parts
2003 of an XML document. This standard is recommend for use with XACML.

2004 It should go without saying that if a repository is used to facilitate the communication of cleartext
2005 (i.e., unencrypted) policy between the **PAP** and the **PRP** or between the **PDP** and the **PIP**, then a
2006 secure repository should be used to store this sensitive data.

2007 **13.3 Policy Integrity**

2008 The XACML policy, used by the **PDP** to evaluate the request contexts, is the heart of the system.
2009 There are two aspects in maintaining the integrity of the policy. One is to ensure that
2010 `<policyStatement>` elements have not been altered since they were originally written or
2011 generated by the **PAP**. The other is to ensure that `<policyStatement>` elements have not been
2012 inserted or deleted from the set of policies.

2013 In the many cases, this can be achieved by ensuring the integrity of the systems and implementing
2014 session-level techniques to secure the communication between parties. The selection of the
2015 appropriate techniques has been left to the implementers.

2016 However, when policy is distributed between organizations to be acted on at a later time, or when
2017 the policy travels with data, it would be useful to have a digital signature of the policy included with
2018 the policy statements. In these cases, the XML Signature Syntax and Processing standard from
2019 W3C is recommended to be used with this standard. See section 8.3 [???] for examples of using
2020 XML digital signatures with XACML.

2021 Digital signatures SHOULD only be used to ensure the integrity of the statements. Digital
2022 signatures SHOULD NOT be use as a method of selecting or evaluating policy. The **PDP** SHOULD

2023 NOT request a rule based on who signed the rule or whether or not it had been signed (as such a
2024 basis for selection would, itself, be a matter of policy).

2025 **13.4 Elements included by reference**

2026 There is a risk that references and extensions contained within a `<policyStatement>` may have
2027 been altered since the policy was originally created, thereby changing the intent of the
2028 `<policyStatement>`. For instance, if a `<policyStatement>` were to include a rule by
2029 reference, then there is no guarantee that the rule has not been changed between the time that the
2030 policy was written and the time that it is being evaluated.

2031 A `<ruleDigest>` element can be used to uniquely identify a rule. The `<ruleDigest>` element
2032 contains a digest of the original rule. If the rule changed, then the rule digest would also change.
2033 Therefore, if the `<policyStatement>` is signed or integrity-protected in some other way (so that
2034 the `<ruleDigest>` cannot be altered without detection), the *PDP* can be certain that the
2035 referenced rules have not changed since the policy was created.

2036 Alternatively, a digital signature of the source item could be included with the reference. **[I don't
2037 see this in the schema. Can we do this?]** This technique will also allow the *PDP* to ensure that a
2038 rule or extension has not been altered (although integrity protection is still required on the policy
2039 itself; otherwise, the included signatures may be removed or replaced).

2040 **13.5 Trust Model**

2041 Discussions of authentication, integrity, and confidentiality mechanisms necessarily assume an
2042 underlying trust model: how can one entity come to believe that a given key is uniquely associated
2043 with a specific, identified entity so that the key can be used to encrypt data for that entity or verify
2044 signatures (or other integrity structures) from that entity? Many different types of trust model exist,
2045 including strict hierarchies, distributed authorities, the Web, the bridge, and so on.

2046 All considerations with respect to choosing and establishing a suitable trust model for a given
2047 environment are outside the scope of XACML. Suffice it to say, however, that a trust model **MUST**
2048 be in place in order for any of the security mechanisms described in this section to be applied.
2049 Secure access control is not possible in any environment until a trust model appropriate for that
2050 environment has been established and implemented.

2051 **13.6 Privacy**

2052 It is important to be aware that any transactions that occur with respect to access control may
2053 reveal private information about the participants. For example, if an XACML policy states that
2054 certain data may only be read by individuals with "Gold Card Member" status, then any transaction
2055 in which an entity is given access to that data leaks information to external observers about that
2056 entity's status. Privacy considerations may therefore lead to encryption or access control policies
2057 surrounding XACML policy instances themselves, confidentiality-protected channels for the
2058 request/response protocol messages, protection of user attributes in storage and in transit, and so
2059 on.

2060 Selection and use of privacy mechanisms appropriate for a given environment are outside the
2061 scope of XACML. The decision regarding whether, how, and when to deploy such mechanisms is
2062 left to the implementers associated with the environment.

2063 Footnotes

2064 1 - Security and Privacy Considerations for the OASIS Security Assertion Markup Language
2065 (SAML) section 4.1

2068 **14 Conformance (normative)**

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

- 2070 1. An implementation of a policy administration points that produces policy statements that
2071 conform with the XACML schema; and
- 2072 2. An implementation of a policy decision point that produces decisions in response to a request
2073 context on the basis of XACML policy statements that conform with the XACML schema.

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

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

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

2081 <http://www.oasis-open.org/> ...

2082 XACML Test Suite

2083 The test suite comprises three directories:

- 2084 • Request context
- 2085 • Policy
- 2086 • Response context

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

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

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

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

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

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

15References

2102

2103
2104

[RFC2119]

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

2105

[RegEx]

2106

[LDAP]

2107
2108

[SAML]

Security Assertion Markup Language available from <http://www.oasis-open.org/committees/security/#documents>

2109
2110

[XMLSig]

D. Eastlake et al., *XML-Signature Syntax and Processing*, <http://www.w3.org/TR/xmlsig-core/>, World Wide Web Consortium.

2111
2112

[XMLSig-XSD]

XML Signature Schema available from <http://www.w3.org/TR/2000/CR-xmlsig-core-20001031/xmlsig-core-schema.xsd>.

2113

2114 **Appendix A. Acknowledgments**

2115 The following individuals were voting members of the committee during the development of this
2116 specification:

2117 Affinitex James MacLean JMaclean@affinitex.com
2118 Self Simon Godik sgodik@crosslogix.com
2119 Crosslogix Ken Yagen kyagen@crosslogix.com
2120 Crosslogix Daniel Engovatov dengovatov@crosslogix.com
2121 Entegriy Hal Lockhart hal.lockhart@entegriy.com
2122 Entrust Carlisle Adams carlisle.adams@entrust.com
2123 Entrust Tim Moses tim.moses@entrust.com
2124 Hitachi Don Flinn Don.Flinn@hitachisoftware.com
2125 Hitachi Konstantin Beznosov konstantin.beznosov@quadrasis.com
2126 IBM Michiharu Kudoh kudo@jp.ibm.com
2127 Self Bill Parducci bill@parducci.net
2128 Self Polar Humenn polar@syr.edu
2129 Sterling Commerce Suresh Damodaran Suresh_Damodaran@stercomm.com
2130 University of Milan Pierangela Samarati samarati@pinky.crema.unimi.it
2131 University of Milan Ernesto Damiani edamiani@crema.unimi.it
2132 Sun Microsystems Sekhar Vajjhala sekhar.vajjhala@sun.com
2133 Sun Microsystems Anne Anderson Anne.Anderson@Sun.com
2134 Xtradyne Gerald Brose Gerald.Brose@xtradyne.com

2135

Appendix B. Revision History

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

2136

2137

Appendix C. Notices

2138 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
2139 that might be claimed to pertain to the implementation or use of the technology described in this
2140 document or the extent to which any license under such rights might or might not be available;
2141 neither does it represent that it has made any effort to identify any such rights. Information on
2142 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
2143 website. Copies of claims of rights made available for publication and any assurances of licenses to
2144 be made available, or the result of an attempt made to obtain a general license or permission for
2145 the use of such proprietary rights by implementors or users of this specification, can be obtained
2146 from the OASIS Executive Director.

2147 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
2148 applications, or other proprietary rights which may cover technology that may be required to
2149 implement this specification. Please address the information to the OASIS Executive Director.

2150 Copyright © The Organization for the Advancement of Structured Information Standards [OASIS]
2151 2001. All Rights Reserved.

2152 This document and translations of it may be copied and furnished to others, and derivative works
2153 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
2154 published and distributed, in whole or in part, without restriction of any kind, provided that the above
2155 copyright notice and this paragraph are included on all such copies and derivative works. However,
2156 this document itself may not be modified in any way, such as by removing the copyright notice or
2157 references to OASIS, except as needed for the purpose of developing OASIS specifications, in
2158 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights
2159 document must be followed, or as required to translate it into languages other than English.

2160 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
2161 successors or assigns.

2162 This document and the information contained herein is provided on an "AS IS" basis and OASIS
2163 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
2164 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY
2165 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
2166 PARTICULAR PURPOSE.