



1

2

eXtensible Access Control Markup Language (XACML) Version 1.0

3

4

OASIS Standard, 18 February 2003

5

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

6

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

7

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

8

Editors:

9

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

10

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

11

Committee members:

12

Anne Anderson, Sun Microsystems, Anne.Anderson@Sun.com

13

Bill Parducci, Overxeer, bill.parducci@overxeer.com

14

Carlisle Adams, Entrust, carlisle.adams@entrust.com

15

Don Flinn, Quadrasis, Don.Flinn@hitachisoftware.com

16

Gerald Brose, Xtradyne, Gerald.Brose@xtradyne.com

17

Hal Lockhart, Entegry, hal.lockhart@entegry.com

18

Konstantin Beznosov, Quadrasis, konstantin.beznosov@quadrasis.com

19

Michiharu Kudo, IBM, kudo@jp.ibm.com

20

Polar Humenn, Self, polar@syr.edu

21

Simon Godik, Overxeer, simon.godik@overxeer.com

22

Steve Andersen, OpenNetwork, sanderson@opennetwork.com

23

Steve Crocker, Pervasive Security Systems, steve.crocker@pervasivesec.com

24

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

25

26

Abstract:

27

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

28

29

30

Status:

31

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.

32

33 If you are on the xacml@lists.oasis-open.org list for committee members, send comments
34 there. If you are not on that list, subscribe to the xacml-comment@lists.oasis-open.org list
35 and send comments there. To subscribe, send an email message to [xacml-comment-
request@lists.oasis-open.org](mailto:xacml-comment-
36 request@lists.oasis-open.org) with the word "subscribe" as the body of the message.

37

38 Copyright (C) OASIS Open 2003. All Rights Reserved.

39	Table of contents	
40	1. Introduction (non-normative)	10
41	1.1. Glossary	10
42	1.1.1 Preferred terms	10
43	1.1.2 Related terms	11
44	1.2. Notation	11
45	1.3. Schema organization and namespaces	12
46	2. Background (non-normative)	12
47	2.1. Requirements	13
48	2.2. Rule and policy combining	14
49	2.3. Combining algorithms	14
50	2.4. Multiple subjects	15
51	2.5. Policies based on subject and resource attributes	15
52	2.6. Multi-valued attributes	15
53	2.7. Policies based on resource contents	16
54	2.8. Operators	16
55	2.9. Policy distribution	16
56	2.10. Policy indexing	17
57	2.11. Abstraction layer	17
58	2.12. Actions performed in conjunction with enforcement	18
59	3. Models (non-normative)	18
60	3.1. Data-flow model	18
61	3.2. XACML context	20
62	3.3. Policy language model	20
63	3.3.1 Rule	21
64	3.3.2 Policy	23
65	3.3.3 Policy set	24
66	4. Examples (non-normative)	25
67	4.1. Example one	25
68	4.1.1 Example policy	25
69	4.1.2 Example request context	27
70	4.1.3 Example response context	28
71	4.2. Example two	28
72	4.2.1 Example medical record instance	29
73	4.2.2 Example request context	30
74	4.2.3 Example plain-language rules	32

75	4.2.4	Example XACML rule instances	32
76	5.	Policy syntax (normative, with the exception of the schema fragments)	46
77	5.1.	Element <PolicySet>	46
78	5.2.	Element <Description>	47
79	5.3.	Element <PolicySetDefaults>	47
80	5.4.	Element <XPathVersion>	48
81	5.5.	Element <Target>	48
82	5.6.	Element <Subjects>	49
83	5.7.	Element <Subject>	49
84	5.8.	Element <AnySubject>	49
85	5.9.	Element <SubjectMatch>	49
86	5.10.	Element <Resources>	50
87	5.11.	Element <Resource>	50
88	5.12.	Element <AnyResource>	51
89	5.13.	Element <ResourceMatch>	51
90	5.14.	Element <Actions>	52
91	5.15.	Element <Action>	52
92	5.16.	Element <AnyAction>	52
93	5.17.	Element <ActionMatch>	52
94	5.18.	Element <PolicySetIdReference>	53
95	5.19.	Element <PolicyIdReference>	53
96	5.20.	Element <Policy>	53
97	5.21.	Element <PolicyDefaults>	55
98	5.22.	Element <Rule>	55
99	5.23.	Simple type EffectType	56
100	5.24.	Element <Condition>	56
101	5.25.	Element <Apply>	56
102	5.26.	Element <Function>	57
103	5.27.	Complex type AttributeDesignatorType	57
104	5.28.	Element SubjectAttributeDesignator	58
105	5.29.	Element <ResourceAttributeDesignator>	59
106	5.30.	Element <ActionAttributeDesignator>	60
107	5.31.	Element <EnvironmentAttributeDesignator>	60
108	5.32.	Element <AttributeSelector>	61
109	5.33.	Element <AttributeValue>	62
110	5.34.	Element <Obligations>	62
111	5.35.	Element <Obligation>	63

112	5.36. Element <AttributeAssignment>	63
113	6. Context syntax (normative with the exception of the schema fragments)	64
114	6.1. Element <Request>	64
115	6.2. Element <Subject>	65
116	6.3. Element <Resource>	65
117	6.4. Element <ResourceContent>	66
118	6.5. Element <Action>	66
119	6.6. Element <Environment>	66
120	6.7. Element <Attribute>	67
121	6.8. Element <AttributeValue>	68
122	6.9. Element <Response>	68
123	6.10. Element <Result>	68
124	6.11. Element <Decision>	69
125	6.12. Element <Status>	70
126	6.13. Element <StatusCode>	70
127	6.14. Element <StatusMessage>	70
128	6.15. Element <StatusDetail>	71
129	7. Functional requirements (normative)	71
130	7.1. Policy enforcement point	71
131	7.2. Base policy	72
132	7.3. Target evaluation	72
133	7.4. Condition evaluation	72
134	7.5. Rule evaluation	72
135	7.6. Policy evaluation	73
136	7.7. Policy Set evaluation	74
137	7.8. Hierarchical resources	75
138	7.9. Attributes	75
139	7.9.1. Attribute Matching	76
140	7.9.2. Attribute Retrieval	76
141	7.9.3. Environment Attributes	76
142	7.10. Authorization decision	76
143	7.11. Obligations	77
144	7.12. Unsupported functionality	77
145	7.13. Syntax and type errors	77
146	8. XACML extensibility points (non-normative)	78
147	8.1. Extensible XML attribute types	78
148	8.2. Structured attributes	78

149	9. Security and privacy considerations (non-normative)	79
150	9.1. Threat model	79
151	9.1.1. Unauthorized disclosure	79
152	9.1.2. Message replay	79
153	9.1.3. Message insertion	80
154	9.1.4. Message deletion	80
155	9.1.5. Message modification	80
156	9.1.6. NotApplicable results	80
157	9.1.7. Negative rules	81
158	9.2. Safeguards	81
159	9.2.1. Authentication	81
160	9.2.2. Policy administration	82
161	9.2.3. Confidentiality	82
162	9.2.4. Policy integrity	82
163	9.2.5. Policy identifiers	83
164	9.2.6. Trust model	83
165	9.2.7. Privacy	83
166	10. Conformance (normative)	84
167	10.1. Introduction	84
168	10.2. Conformance tables	84
169	10.2.1. Schema elements	84
170	10.2.2. Identifier Prefixes	85
171	10.2.3. Algorithms	85
172	10.2.4. Status Codes	86
173	10.2.5. Attributes	86
174	10.2.6. Identifiers	86
175	10.2.7. Data-types	87
176	10.2.8. Functions	87
177	11. References	91
178	Appendix A. Standard data-types, functions and their semantics (normative)	93
179	A.1. Introduction	93
180	A.2. Primitive types	93
181	A.3. Structured types	94
182	A.4. Representations	94
183	A.5. Bags	95
184	A.6. Expressions	95
185	A.7. Element <AttributeValue>	96

186	A.8.	Elements <AttributeDesignator> and <AttributeSelector>	96
187	A.9.	Element <Apply>	96
188	A.10.	Element <Condition>	96
189	A.11.	Element <Function>	97
190	A.12.	Matching elements	97
191	A.13.	Arithmetic evaluation	98
192	A.14.	XACML standard functions	99
193	A14.1	Equality predicates	99
194	A14.2	Arithmetic functions	101
195	A14.3	String conversion functions	102
196	A14.4	Numeric data-type conversion functions	102
197	A14.5	Logical functions	102
198	A14.6	Arithmetic comparison functions	103
199	A14.7	Date and time arithmetic functions	104
200	A14.8	Non-numeric comparison functions	105
201	A14.9	Bag functions	107
202	A14.10	Set functions	108
203	A14.11	Higher-order bag functions	109
204	A14.12	Special match functions	116
205	A14.13	XPath-based functions	117
206	A14.14	Extension functions and primitive types	117
207		Appendix B. XACML identifiers (normative)	118
208	B.1.	XACML namespaces	118
209	B.2.	Access subject categories	118
210	B.3.	XACML functions	118
211	B.4.	Data-types	118
212	B.5.	Subject attributes	119
213	B.6.	Resource attributes	120
214	B.7.	Action attributes	120
215	B.8.	Environment attributes	121
216	B.9.	Status codes	121
217	B.10.	Combining algorithms	121
218		Appendix C. Combining algorithms (normative)	123
219	C.1.	Deny-overrides	123
220	C.2.	Permit-overrides	125
221	C.3.	First-applicable	127
222	C.4.	Only-one-applicable	128

223	Appendix D. Acknowledgments	130
224	Appendix E. Revision history	131
225	Appendix F. Notices	132
226		

227 **Errata**

228 Errata can be found at the following location:

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

230

231 1. Introduction (non-normative)

232 1.1. Glossary

233 1.1.1 Preferred terms

234 **Access** - Performing an *action*

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

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

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

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

241 **Authorization decision** - The result of evaluating *applicable policy*, returned by the *PDP* to the
242 *PEP*. A function that evaluates to "Permit", "Deny", "Indeterminate" or "NotApplicable", and
243 (optionally) a set of *obligations*

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

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

247 **Conjunctive sequence** - a sequence of boolean elements combined using the logical 'AND'
248 operation

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

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

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

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

255 **Disjunctive sequence** - a sequence of boolean elements combined using the logical 'OR'
256 operation

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

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

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

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

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

265 **Policy-combining algorithm** - The procedure for combining the **decision** and **obligations** from
266 multiple **policies**

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

269 **Policy enforcement point (PEP)** - The system entity that performs **access control**, by making
270 **decision requests** and enforcing **authorization decisions**

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

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

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

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

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

277 **Rule-combining algorithm** - The procedure for combining **decisions** from multiple **rules**

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

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

281 1.1.2 Related terms

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

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

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

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

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

289 1.2. Notation

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

292 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",
293 "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be
294 interpreted as described in IETF RFC 2119 [RFC2119]

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

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

301 Listings of XACML schemas appear like this.

302
303 Example code listings appear like this.

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

- 307 • The prefix `xacml`: stands for the XACML policy namespace.
- 308 • The prefix `xacml-context`: stands for the XACML context namespace.
- 309 • The prefix `ds`: stands for the W3C XML Signature namespace [DS].
- 310 • The prefix `xs`: stands for the W3C XML Schema namespace [XS].
- 311 • The prefix `xf`: stands for the XQuery 1.0 and XPath 2.0 Function and Operators
312 specification namespace [XF].

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

316 1.3. Schema organization and namespaces

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

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

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

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

321 The XML Signature [DS] is imported into the XACML schema and is associated with the following
322 XML namespace:

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

324 2. Background (non-normative)

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

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

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

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

351 2.1. Requirements

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

- 353 • To provide a method for combining individual **rules** and **policies** into a single **policy set** that
354 applies to a particular **decision request**.
- 355 • To provide a method for flexible definition of the procedure by which **rules** and **policies** are
356 combined.
- 357 • To provide a method for dealing with multiple **subjects** acting in different capacities.
- 358 • To provide a method for basing an **authorization decision** on **attributes** of the **subject** and
359 **resource**.
- 360 • To provide a method for dealing with multi-valued **attributes**.
- 361 • To provide a method for basing an **authorization decision** on the contents of an information
362 **resource**.
- 363 • To provide a set of logical and mathematical operators on **attributes** of the **subject, resource**
364 and **environment**.
- 365 • To provide a method for handling a distributed set of **policy** components, while abstracting the
366 method for locating, retrieving and authenticating the **policy** components.
- 367 • To provide a method for rapidly identifying the **policy** that applies to a given action, based upon
368 the values of **attributes** of the **subjects, resource** and **action**.
- 369 • To provide an abstraction-layer that insulates the policy-writer from the details of the application
370 environment.

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

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

376 2.2. Rule and policy combining

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

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

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

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

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

396 2.3. Combining algorithms

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

- 404 • Deny-overrides,
- 405 • Permit-overrides,
- 406 • First applicable and
- 407 • Only-one-applicable.

408 In the first case, if a single <Rule> or <Policy> element is encountered that evaluates to "Deny",
409 then, regardless of the evaluation result of the other <Rule> or <Policy> elements in the
410 *applicable policy*, the combined result is "Deny". Likewise, in the second case, if a single "Permit"
411 result is encountered, then the combined result is "Permit". In the case of the "First-applicable"

412 combining algorithm, the combined result is the same as the result of evaluating the first <Rule>,
413 <Policy> or <PolicySet> element in the list of **rules** whose **target** is applicable to the **decision**
414 **request**. The "Only-one-applicable" **policy-combining algorithm** only applies to **policies**. The
415 result of this combining algorithm ensures that one and only one **policy** or **policy set** is applicable
416 by virtue of their **targets**. If no **policy** or **policy set** applies, then the result is "NotApplicable", but if
417 more than one **policy** or **policy set** is applicable, then the result is "Indeterminate". When exactly
418 one **policy** or **policy set** is applicable, the result of the combining algorithm is the result of
419 evaluating the single **applicable policy** or **policy set**.

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

421 **2.4. Multiple subjects**

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

428 **2.5. Policies based on subject and resource attributes**

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

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

445 **2.6. Multi-valued attributes**

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

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

454

2.7. Policies based on resource contents

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

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

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

465

2.8. Operators

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

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

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

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

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

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

493 The XACML method of representing functions borrows from MathML [\[MathML\]](#) and from the
494 XQuery 1.0 and XPath 2.0 Functions and Operators specification [\[XF\]](#).

495

2.9. Policy distribution

496 In a distributed system, individual **policy** statements may be written by several policy writers and
497 enforced at several enforcement points. In addition to facilitating the collection and combination of

498 independent **policy** components, this approach allows **policies** to be updated as required. XACML
499 **policy** statements may be distributed in any one of a number of ways. But, XACML does not
500 describe any normative way to do this. Regardless of the means of distribution, **PDPs** are
501 expected to confirm, by examining the **policy's** <Target> element that the policy is applicable to
502 the **decision request** that it is processing.

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

507 2.10. Policy indexing

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

513 Two approaches are supported:

- 514 1. **Policy** statements may be stored in a database, whose data-model is congruent with that of the
515 <Target> element. The **PDP** should use the contents of the **decision request** that it is
516 processing to form the database read command by which applicable **policy** statements are
517 retrieved. Nevertheless, the **PDP** should still evaluate the <Target> element of the retrieved
518 **policy** or **policy set** statements as defined by the XACML specification.
- 519 2. Alternatively, the **PDP** may evaluate the <Target> element from each of the **policies** or
520 **policy sets** that it has available to it, in the context of a particular **decision request**, in order to
521 identify the **policies** and **policy sets** that are applicable to that request.

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

523 2.11. Abstraction layer

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

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

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

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

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

546 **2.12. Actions performed in conjunction with enforcement**

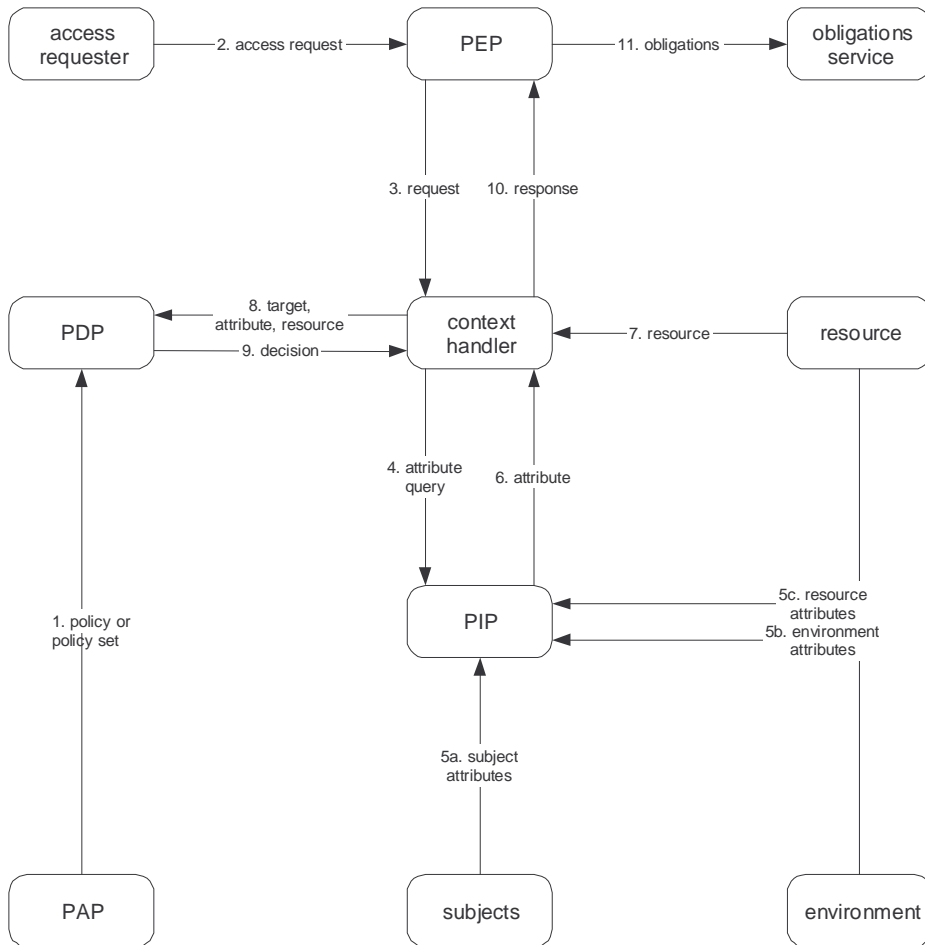
547 In many applications, policies specify actions that **MUST** be performed, either instead of, or in
548 addition to, actions that **MAY** be performed. This idea was described by Sloman [Sloman94].
549 XACML provides facilities to specify actions that **MUST** be performed in conjunction with policy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

556 **3. Models (non-normative)**

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

558 **3.1. Data-flow model**

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



560

561

Figure 1 - Data-flow diagram

562 Note: some of the data-flows shown in the diagram may be facilitated by a repository. For instance,
 563 the communications between the **context handler** and the **PIP** or the communications between the
 564 **PDP** and the **PAP** may be facilitated by a repository. The XACML specification is not intended to
 565 place restrictions on the location of any such repository, or indeed to prescribe a particular
 566 communication protocol for any of the data-flows.

567 The model operates by the following steps.

568 1. **PAPs** write **policies** and **policy sets** and make them available to the **PDP**. These **policies** or
 569 **policy sets** represent the complete policy for a specified **target**.

570 2. The access requester sends a request for access to the **PEP**.

571 3. The **PEP** sends the request for **access** to the **context handler** in its native request format,
 572 optionally including **attributes** of the **subjects**, **resource** and **action**. The **context handler**
 573 constructs an XACML request **context** in accordance with steps 4,5,6 and 7.

574 4. **Subject**, **resource** and **environment attributes** may be requested from a **PIP**.

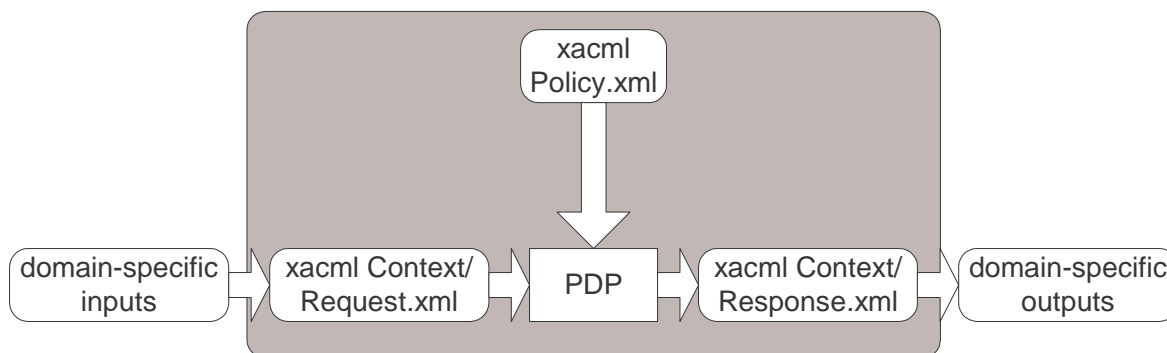
575 5. The **PIP** obtains the requested **attributes**.

576 6. The **PIP** returns the requested **attributes** to the **context handler**.

- 577 7. Optionally, the **context handler** includes the **resource** in the **context**.
- 578 8. The **context handler** sends a **decision request**, including the **target**, to the **PDP**. The **PDP**
 579 identifies the **applicable policy** and retrieves the required **attributes** and (optionally) the
 580 **resource** from the **context handler**. The **PDP** evaluates the **policy**.
- 581 9. The **PDP** returns the response **context** (including the **authorization decision**) to the **context**
 582 **handler**.
- 583 10. The **context handler** translates the response **context** to the native response format of the
 584 **PEP**. The **context handler** returns the response to the **PEP**.
- 585 11. The **PEP** fulfills the **obligations**.
- 586 12. (Not shown) If **access** is permitted, then the **PEP** permits **access** to the **resource**; otherwise, it
 587 denies **access**.

3.2. XACML context

589 XACML is intended to be suitable for a variety of application environments. The core language is
 590 insulated from the application environment by the XACML **context**, as shown in Figure 2, in which
 591 the scope of the XACML specification is indicated by the shaded area. The XACML **context** is
 592 defined in XML schema, describing a canonical representation for the inputs and outputs of the
 593 **PDP**. **Attributes** referenced by an instance of XACML policy may be in the form of XPath
 594 expressions on the **context**, or attribute designators that identify the **attribute** by **subject**,
 595 **resource**, **action** or **environment** and its identifier. Implementations must convert between the
 596 **attribute** representations in the application environment (e.g., SAML, J2SE, CORBA, and so on)
 597 and the **attribute** representations in the XACML **context**. How this is achieved is outside the
 598 scope of the XACML specification. In some cases, such as SAML, this conversion may be
 599 accomplished in an automated way through the use of an XSLT transformation.



600
 601 **Figure 2 - XACML context**

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

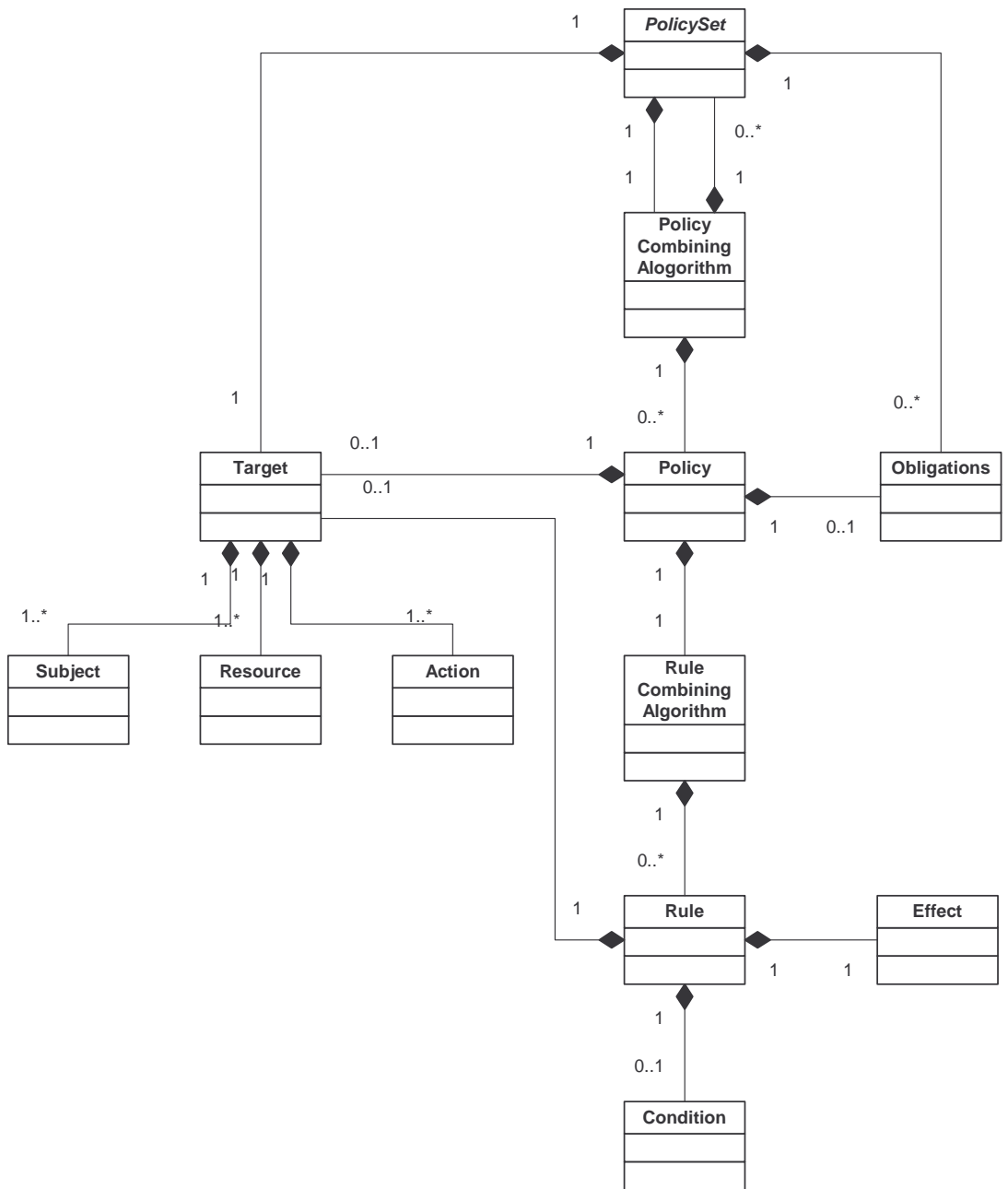
603 See Section 7.9 for a more detailed discussion of the request **context**.

3.3. Policy language model

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

- 606 • **Rule**;
- 607 • **Policy**; and

608 • **Policy set.**
 609 These are described in the following sub-sections.



610

611

Figure 3 - Policy language model

612

3.3.1 Rule

613

A **rule** is the most elementary unit of **policy**. It may exist in isolation only *within* one of the major actors of the XACML domain. In order to exchange **rules** between major actors, they must be encapsulated in a **policy**. A **rule** can be evaluated on the basis of its contents. The main components of a **rule** are:

614

615

616

- 617 • a **target**,
- 618 • an **effect**, and
- 619 • a **condition**.

620 These are discussed in the following sub-sections.

621 3.3.1.1. Rule target

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

- 623 • **resources**;
- 624 • **subjects**; and
- 625 • **actions**

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

633 The <Target> element may be absent from a <Rule>. In this case, the **target** of the <Rule> is
634 the same as that of the parent <Policy> element.

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

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

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

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

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

- 656 1. the contents of the identified node only,
- 657 2. the contents of the identified node and the contents of its immediate child nodes or
- 658 3. the contents of the identified node and all its descendant nodes.

659 All three options are supported in XACML.

660 3.3.1.2. Effect

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

663 3.3.1.3. Condition

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

666 3.3.2 Policy

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

- 669 • a **target**,
- 670 • a **rule-combining algorithm**-identifier;
- 671 • a set of **rules**; and
- 672 • **obligations**.

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

675 3.3.2.1. Policy target

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

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

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

697 3.3.2.2. Rule-combining algorithm

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

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

703 3.3.2.3. Obligations

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

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

708 3.3.3 Policy set

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

- 710 • a **target**,
- 711 • a **policy-combining algorithm**-identifier
- 712 • a set of **policies**; and
- 713 • **obligations**.

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

716 3.3.3.1. Policy-combining algorithm

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

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

722 3.3.3.2. Obligations

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

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

727

728 4. Examples (non-normative)

729 This section contains two examples of the use of XACML for illustrative purposes. The first example
730 is a relatively simple one to illustrate the use of **target**, **context**, matching functions and **subject**
731 **attributes**. The second example additionally illustrates the use of the **rule-combining algorithm**,
732 **conditions** and **obligations**.

733 4.1. Example one

734 4.1.1 Example policy

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

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

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

741 The header for this policy is

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

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

744 [p02] introduces the XACML Policy itself.

745 [p03-p05] are XML namespace declarations.

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

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

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

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

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

```
[p11] <Target>  
[p12] <Subjects>  
[p13] <AnySubject/>  
[p14] </Subjects>  
[p15] <Resources>
```

```
[p16]      <AnyResource/>
[p17]      </Resources>
[p18]      <Actions>
[p19]      <AnyAction/>
[p20]      </Actions>
[p21]      </Target>
```

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

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

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

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

764 [p24] says what **effect** this **rule** has if the **rule** evaluates to “True”. **Rules** can have an **effect** of
 765 either “Permit” or “Deny”. In this case, the rule will evaluate to “Permit”, meaning that, as far as this
 766 one **rule** is concerned, the requested **access** should be permitted. If a **rule** evaluates to “False”,
 767 then it returns a result of “NotApplicable”. If an error occurs when evaluating the **rule**, the **rule**
 768 returns a result of “Indeterminate”. As mentioned above, the **rule-combining algorithm** for the
 769 **policy** tells how various **rule** values are combined into a single **policy** value.

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

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

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

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

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

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

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

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

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

789 4.1.2 Example request context

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

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

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

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

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

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

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

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

803 The <Resource> element contains one or more **attributes** of the **resource** to which
804 the **subject** (or **subjects**) has requested **access**. There can be only one <Resource>

805 per **decision request**. Lines [c13-c16] contain the one **attribute** of the **resource**
806 to which Bart Simpson has requested **access**: the **resource** unix file-system path-
807 name, which is "/medico/record/patient/BartSimpson".

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

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

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

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

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

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

822 4.1.3 Example response context

823 As a result, there is no **rule** in this **policy** that returns a "Permit" result for this request. The **rule-**
824 **combining algorithm** for the **policy** specifies that, in this case, a result of "NotApplicable" should
825 be returned. The response **context** looks as follows:

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

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

```
[r05] <Result>  
[r06]   <Decision>NotApplicable</Decision>  
[r07] </Result>
```

828 The <Result> element in lines [r05-r07] contains the result of evaluating the **decision request**
829 against the **policy**. In this case, the result is "NotApplicable". A **policy** can return "Permit", "Deny",
830 "NotApplicable" or "Indeterminate".

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

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

832 4.2. Example two

833 This section contains an example XML document, an example request **context** and example
834 XACML **rules**. The XML document is a medical record. Four separate **rules** are defined. These
835 illustrate a **rule-combining algorithm**, **conditions** and **obligations**.

4.2.1 Example medical record instance

837 The following is an instance of a medical record to which the example XACML *rules* can be
 838 applied. The <record> schema is defined in the registered namespace administered by
 839 "http://medico.com".

```

840 <?xml version="1.0" encoding="UTF-8"?>
841 <record xmlns="http://www.medico.com/schemas/record.xsd "
842 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
843 <patient>
844 <patientName>
845 <first>Bartholomew</first>
846 <last>Simpson</last>
847 </patientName>
848 <patientContact>
849 <street>27 Shelbyville Road</street>
850 <city>Springfield</city>
851 <state>MA</state>
852 <zip>12345</zip>
853 <phone>555.123.4567</phone>
854 <fax/>
855 <email/>
856 </patientContact>
857 <patientDoB http://www.w3.org/2001/XMLSchema#type="date">1992-03-
858 21</patientDoB>
859 <patientGender
860 http://www.w3.org/2001/XMLSchema#type="string">male</patientGender>
861 <patient-number
862 http://www.w3.org/2001/XMLSchema#type="string">555555</patient-number>
863 </patient>
864 <parentGuardian>
865 <parentGuardianId>HS001</parentGuardianId>
866 <parentGuardianName>
867 <first>Homer</first>
868 <last>Simpson</last>
869 </parentGuardianName>
870 <parentGuardianContact>
871 <street>27 Shelbyville Road</street>
872 <city>Springfield</city>
873 <state>MA</state>
874 <zip>12345</zip>
875 <phone>555.123.4567</phone>
876 <fax/>
877 <email>homers@aol.com</email>
878 </parentGuardianContact>
879 </parentGuardian>
880 <primaryCarePhysician>
881 <physicianName>
882 <first>Julius</first>
883 <last>Hibbert</last>
884 </physicianName>
885 <physicianContact>
886 <street>1 First St</street>
887 <city>Springfield</city>
888 <state>MA</state>
889 <zip>12345</zip>
890 <phone>555.123.9012</phone>
891 <fax>555.123.9013</fax>
892 <email/>
893 </physicianContact>
894 <registrationID>ABC123</registrationID>
895 </primaryCarePhysician>
896 <insurer>

```

```

897     <name>Blue Cross</name>
898     <street>1234 Main St</street>
899     <city>Springfield</city>
900     <state>MA</state>
901     <zip>12345</zip>
902     <phone>555.123.5678</phone>
903     <fax>555.123.5679</fax>
904     <email/>
905 </insurer>
906 <medical>
907     <treatment>
908         <drug>
909             <name>methylphenidate hydrochloride</name>
910             <dailyDosage>30mgs</dailyDosage>
911             <startDate>1999-01-12</startDate>
912         </drug>
913         <comment>patient exhibits side-effects of skin coloration and carpal
914 degeneration</comment>
915     </treatment>
916     <result>
917         <test>blood pressure</test>
918         <value>120/80</value>
919         <date>2001-06-09</date>
920         <performedBy>Nurse Betty</performedBy>
921     </result>
922 </medical>
923 </record>

```

924 4.2.2 Example request context

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

```

928 [01] <?xml version="1.0" encoding="UTF-8"?>
929 [02] <Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
930 [03] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
931 [04] <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-
932 category:access-subject">
933 [05]   <Attribute AttributeId=
934 [06]     "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
935 [07]     DataType=
936 [08]       "urn:oasis:names:tc:xacml:1.0:data-type:x500name"
937 [09]     Issuer="www.medico.com"
938 [10]     IssueInstant="2001-12-17T09:30:47-05:00">
939 [11]       <AttributeValue>CN=Julius Hibbert</AttributeValue>
940 [12]     </Attribute>
941 [13]     <Attribute AttributeId=
942 [14]       "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
943 [15]       DataType="http://www.w3.org/2001/XMLSchema#string"
944 [16]       Issuer="www.medico.com"
945 [17]       IssueInstant="2001-12-17T09:30:47-05:00">
946 [18]         <AttributeValue>physician</AttributeValue>
947 [19]       </Attribute>
948 [20]     <Attribute AttributeId=
949 [21]       "urn:oasis:names:tc:xacml:1.0:example:attribute:physician-id"
950 [22]       DataType="http://www.w3.org/2001/XMLSchema#string"
951 [23]       Issuer="www.medico.com"
952 [24]       IssueInstant="2001-12-17T09:30:47-05:00">
953 [25]         <AttributeValue>jh1234</AttributeValue>
954 [26]       </Attribute>
955 [27]     </Subject>
956 [28] </Resource>

```

```

957 [29] <ResourceContent>
958 [30]   <md:record
959 [31]     xmlns:md="//http:www.medico.com/schemas/record.xsd">
960 [32]       <md:patient>
961 [33]         <md:patientDoB>1992-03-21</md:patientDoB>
962 [34]       </md:patient>
963 [35]       <!-- other fields -->
964 [36]     </md:record>
965 [37]   </ResourceContent>
966 [38] <Attribute AttributeId=
967 [39]   "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
968 [40]   DataType="http://www.w3.org/2001/XMLSchema#string">
969 [41]   <AttributeValue>
970 [42]     //medico.com/records/bart-simpson.xml#
971 [43]     xmlns(md="//http:www.medico.com/schemas/record.xsd)
972 [44]     xpointer(/md:record/md:patient/md:patientDoB)
973 [45]   </AttributeValue>
974 [46] </Attribute>
975 [47] <Attribute AttributeId=
976 [48]   "urn:oasis:names:tc:xacml:1.0:resource:xpath"
977 [49]   DataType="http://www.w3.org/2001/XMLSchema#string">
978 [50]   <AttributeValue>
979 [51]     xmlns(md=http:www.medico.com/schemas/record.xsd)
980 [52]     xpointer(/md:record/md:patient/md:patientDoB)
981 [53]   </AttributeValue>
982 [54] </Attribute>
983 [55] <Attribute AttributeId=
984 [56]   "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
985 [57]   DataType="http://www.w3.org/2001/XMLSchema#string">
986 [58]   <AttributeValue>
987 [59]     http://www.medico.com/schemas/record.xsd
988 [60]   </AttributeValue>
989 [61] </Attribute>
990 [62] </Resource>
991 [63] <Action>
992 [64]   <Attribute AttributeId=
993 [65]     "urn:oasis:names:tc:xacml:1.0:action:action-id"
994 [66]     DataType="http://www.w3.org/2001/XMLSchema#string">
995 [67]     <AttributeValue>read</AttributeValue>
996 [68]   </Attribute>
997 [69] </Action>
998 [70] </Request>

```

999 [02]-[03] Standard namespace declarations.

1000 [04]-[27] **Subject** attributes are placed in the Subject section of the Request. Each **attribute**
1001 consists of the **attribute** meta-data and the **attribute** value.

1002 [04] Each Subject element has SubjectCategory xml attribute. The value of this attribute
1003 describes the role that the **subject** plays in making the **decision request**. The value of "access-
1004 subject" denotes the identity for which the request was issued.

1005 [05]-[12] **Subject** subject-id **attribute**.

1006 [13]-[19] **Subject** role **attribute**.

1007 [20]-[26] **Subject** physician-id **attribute**.

1008 [28]-[62] **Resource** attributes are placed in the Resource section of the Request. Each **attribute**
1009 consists of **attribute** meta-data and an **attribute** value.

1010 [29]-[36] **Resource** content. The XML document that is being requested is placed here.

- 1011 [38]-[46] **Resource** identifier.
- 1012 [47]-[61] The **Resource** is identified with an Xpointer expression that names the URI of the file that
 1013 is accessed, the target namespace of the document, and the XPath location path to the specific
 1014 element.
- 1015 [47]-[54] The XPath location path in the “resource-id” attribute is extracted and placed in the
 1016 xpath attribute.
- 1017 [55]-[61] **Resource** target-namespace **attribute**.
- 1018 [63]-[69] **Action attributes** are placed in the Action section of the Request.
- 1019 [64]-[68] **Action** identifier.

4.2.3 Example plain-language rules

- 1020 The following plain-language rules are to be enforced:
- 1021 Rule 1: A person, identified by his or her patient number, may read any record for which he
 1022 or she is the designated patient.
- 1023 Rule 2: A person may read any record for which he or she is the designated parent or
 1024 guardian, and for which the patient is under 16 years of age.
- 1025 Rule 3: A physician may write to any medical element for which he or she is the designated
 1026 primary care physician, provided an email is sent to the patient.
- 1027 Rule 4: An administrator shall not be permitted to read or write to medical elements of a
 1028 patient record.
- 1029 These **rules** may be written by different **PAPs** operating independently, or by a single **PAP**.

4.2.4 Example XACML rule instances

4.2.4.1. Rule 1

- 1032 Rule 1 illustrates a simple **rule** with a single <Condition> element. The following XACML
 1033 <Rule> instance expresses Rule 1:

```

1035 [01] <?xml version="1.0" encoding="UTF-8"?>
1036 [02] <Rule
1037 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1038 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1039 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1040 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1041 [07]   RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
1042 [08]   Effect="Permit">
1043 [09]   <Description>
1044 [10]     A person may read any medical record in the
1045 [11]     http://www.medico.com/schemas/record.xsd namespace
1046 [12]     for which he or she is a designated patient
1047 [13]   </Description>
1048 [14]   <Target>
1049 [15]     <Subjects>
1050 [16]       <AnySubject/>
1051 [17]     </Subjects>
1052 [18]   <Resources>
1053 [20]     <Resource>

```



```

1054 [21] <!-- match document target namespace -->
1055 [22] <ResourceMatch
1056 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1057 [23] <AttributeValue
1058 DataType="http://www.w3.org/2001/XMLSchema#string">
1059 [24] http://www.medico.com/schemas/record.xsd
1060 [25] </AttributeValue>
1061 [26] <ResourceAttributeDesignator AttributeId=
1062 [27] "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1063 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1064 [28] </ResourceMatch>
1065 [29] <!-- match requested xml element -->
1066 [30] <ResourceMatch
1067 MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1068 [31] <AttributeValue
1069 DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeV
1070 alue>
1071 [32] <ResourceAttributeDesignator AttributeId=
1072 [33] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1073 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1074 [34] </ResourceMatch>
1075 [35] </Resource>
1076 [36] </Resources>
1077 [37] <Actions>
1078 [38] <Action>
1079 [39] <ActionMatch
1080 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1081 [40] <AttributeValue
1082 DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1083 [41] <ActionAttributeDesignator AttributeId=
1084 [42] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1085 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1086 [43] </ActionMatch>
1087 [44] </Action>
1088 [45] </Actions>
1089 [46] </Target>
1090 [47] <!-- compare policy number in the document with
1091 [48] policy-number attribute -->
1092 [49] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1093 equal">
1094 [50] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1095 and-only">
1096 [51] <!-- policy-number attribute -->
1097 [52] <SubjectAttributeDesignator AttributeId=
1098 [53] "urn:oasis:names:tc:xacml:1.0:examples:attribute:policy-number"
1099 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1100 [54] </Apply>
1101 [55] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1102 and-only">
1103 [56] <!-- policy number in the document -->
1104 [57] <AttributeSelector RequestContextPath=
1105 [58] "//md:record/md:patient/md:patient-number/text()"
1106 DataType="http://www.w3.org/2001/XMLSchema#string">
1107 [59] </AttributeSelector>
1108 [60] </Apply>
1109 [61] </Condition>
1110 [62] </Rule>

```

1111 [02]-[06]. XML namespace declarations.

1112 [07] **Rule** identifier.

1113 [08]. When a **rule** evaluates to 'True' it emits the value of the `Effect` attribute. This value is
1114 combined with the `Effect` values of other rules according to the **rule-combining algorithm**.

1115 [09]-[13] Free form description of the **rule**.

1116 [14]-[46]. A **rule target** defines a set of **decision requests** that are applicable to the **rule**. A
1117 **decision request**, such that the value of the
1118 "`urn:oasis:names:tc:xacml:1.0:resource:target-namespace`" **resource attribute** is
1119 equal to "`http://www.medico.com/schema/records.xsd`" and the value of the
1120 "`urn:oasis:names:tc:xacml:1.0:resource:xpath`" **resource attribute** matches the XPath
1121 expression "`/md:record`" and the value of the
1122 "`urn:oasis:names:tc:xacml:1.0:action:action-id`" **action attribute** is equal to "`read`",
1123 matches the **target** of this **rule**.

1124 [15]-[17]. The `Subjects` element may contain either a **disjunctive sequence** of `Subject`
1125 elements or `AnySubject` element.

1126 [16] The `AnySubject` element is a special element that matches any **subject** in the request
1127 **context**.

1128 [18]-[36]. The `Resources` element may contain either a **disjunctive sequence** of `Resource`
1129 elements or `AnyResource` element.

1130 [20]-[35] The `Resource` element encloses the **conjunctive sequence** of `ResourceMatch`
1131 elements.

1132 [22]-[28] The `ResourceMatch` element compares its first and second child elements according to
1133 the matching function. A match is positive if the value of the first argument matches any of the
1134 values selected by the second argument. This match compares the target namespace of the
1135 requested document with the value of "`http://www.medico.com/schema.records.xsd`".

1136 [22] The `MatchId` attribute names the matching function.

1137 [23]-[25] Literal attribute value to match.

1138 [26]-[27] The `ResourceAttributeDesignator` element selects the **resource attribute** values
1139 from the request **context**. The **attribute** name is specified by the `AttributeId`. The selection
1140 result is a **bag** of values.

1141 [30]-[34] The `ResourceMatch`. This match compares the results of two XPath expressions. The
1142 first XPath expression is `/md:record` and the second XPath expression is the location path to the
1143 requested xml element. The "`xpath-node-match`" function evaluates to "True" if the requested XML
1144 element is below the `/md:record` element.

1145 [30] `MatchId` attribute names the matching function.

1146 [31] The literal XPath expression to match. The `md` prefix is resolved using a standard namespace
1147 declaration.

1148 [32]-[33] The `ResourceAttributeDesignator` selects the **bag** of values for the
1149 "`urn:oasis:names:tc:xacml:1.0:xpath`" **resource attribute**. Here, there is just one
1150 element in the **bag**, which is the location path for the requested XML element.

1151 [37]-[45] The `Actions` element may contain either a **disjunctive sequence** of `Action` elements
1152 or an `AnyAction` element.

1153 [38]-[44] The `Action` element contains a **conjunctive sequence** of `ActionMatch` elements.

1154 [39]-[43] The `ActionMatch` element compares its first and second child elements according to the
1155 matching function. Match is positive if the value of the first argument matches any of the values
1156 selected by the second argument. In this case, the value of the `action-id` action attribute in the
1157 request **context** is compared with the value “read”.

1158 [39] The `MatchId` attribute names the matching function.

1159 [40] The **Attribute** value to match. This is an **action** name.

1160 [41]-[42] The `ActionAttributeDesignator` selects **action attribute** values from the request
1161 **context**. The **attribute** name is specified by the `AttributeId`. The selection result is a **bag** of
1162 values. “urn:oasis:names:tc:xacml:1.0:action:action-id” is the predefined name for
1163 the action identifier.

1164 [49]-[61] The `<Condition>` element. A **condition** must evaluate to “True” for the **rule** to be
1165 applicable. This condition evaluates the truth of the statement: the `patient-number` **subject**
1166 **attribute** is equal to the patient-number in the XML document.

1167 [49] The `FunctionId` attribute of the `<Condition>` element names the function to be used for
1168 comparison. In this case, comparison is done with
1169 urn:oasis:names:tc:xacml:1.0:function:string-equal; this function takes two
1170 arguments of the “http://www.w3.org/2001/XMLSchema#string” data-type.

1171 [50] The first argument to the urn:oasis:names:tc:xacml:1.0:function:string-equal
1172 in the `Condition`. Functions can take other functions as arguments. The `Apply` element
1173 encodes the function call with the `FunctionId` attribute naming the function. Since
1174 urn:oasis:names:tc:xacml:1.0:function:string-equal takes arguments of the
1175 “http://www.w3.org/2001/XMLSchema#string” data-type and
1176 `SubjectAttributeDesignator` selects a **bag** of
1177 “http://www.w3.org/2001/XMLSchema#string” values,
1178 “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used. This
1179 function guarantees that its argument evaluates to a **bag** containing one and only one
1180 “http://www.w3.org/2001/XMLSchema#string” element.

1181 [52]-[53] The `SubjectAttributeDesignator` selects a **bag** of values for the `policy-number`
1182 **subject attribute** in the request **context**.

1183 [55] The second argument to the “urn:oasis:names:tc:xacml:1.0:function:string-
1184 equal” in the `Condition`. Functions can take other functions as arguments. The `Apply` element
1185 encodes function call with the `FunctionId` attribute naming the function. Since
1186 “urn:oasis:names:tc:xacml:1.0:function:string-equal” takes arguments of the
1187 “http://www.w3.org/2001/XMLSchema#string” data-type and the `AttributeSelector`
1188 selects a **bag** of “http://www.w3.org/2001/XMLSchema#string” values,
1189 “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used. This
1190 function guarantees that its argument evaluates to a **bag** containing one and only one
1191 “http://www.w3.org/2001/XMLSchema#string” element.

1192 [57] The `AttributeSelector` element selects a **bag** of values from the request **context**. The
1193 `AttributeSelector` is a free-form XPath pointing device into the request **context**. The
1194 `RequestContextPath` attribute specifies an XPath expression over the content of the requested
1195 XML document, selecting the policy number. Note that the namespace prefixes in the XPath
1196 expression are resolved with the standard XML namespace declarations.

4.2.4.2. Rule 2

1198 Rule 2 illustrates the use of a mathematical function, i.e. the <Apply> element with functionId
 1199 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" to calculate date. It also
 1200 illustrates the use of *predicate* expressions, with the functionId
 1201 "urn:oasis:names:tc:xacml:1.0:function:and".

```

1202 [01] <?xml version="1.0" encoding="UTF-8"?>
1203 [02] <Rule
1204 [03] xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1205 [04] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1206 [05] xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1207 [06] xmlns:md="http://www.medico.com/schemas/record.xsd"
1208 [07] RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1209 [08] Effect="Permit">
1210 [09] <Description>
1211 [10]   A person may read any medical record in the
1212 [11]   http://www.medico.com/records.xsd namespace
1213 [12]   for which he or she is the designated parent or guardian,
1214 [13]   and for which the patient is under 16 years of age
1215 [14] </Description>
1216 [15] <Target>
1217 [16]   <Subjects>
1218 [17]     <AnySubject/>
1219 [18]   </Subjects>
1220 [19]   <Resources>
1221 [20]     <Resource>
1222 [21]       <!-- match document target namespace -->
1223 [22]       <ResourceMatch
1224 [23]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1225 [24]           <AttributeValue
1226 [25]             DataType="http://www.w3.org/2001/XMLSchema#string">
1227 [26]               http://www.medico.com/schemas/record.xsd
1228 [27]             </AttributeValue>
1229 [28]             <ResourceAttributeDesignator AttributeId=
1230 [29]               "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1231 [30]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1232 [31]             </ResourceMatch>
1233 [32]           <!-- match requested xml element -->
1234 [33]           <ResourceMatch
1235 [34]             MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1236 [35]               <AttributeValue
1237 [36]                 DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeV
1238 [37]                 alue>
1239 [38]               <ResourceAttributeDesignator AttributeId=
1240 [39]                 "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1241 [40]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
1242 [41]             </ResourceMatch>
1243 [42]           </Resource>
1244 [43]         </Resources>
1245 [44]       <Actions>
1246 [45]         <Action>
1247 [46]           <!-- match 'read' action -->
1248 [47]           <ActionMatch
1249 [48]             MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1250 [49]               <AttributeValue
1251 [50]                 DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1252 [51]               <ActionAttributeDesignator AttributeId=
1253 [52]                 "urn:oasis:names:tc:xacml:1.0:action:action-id"
1254 [53]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
1255 [54]             </ActionMatch>
1256 [55]           </Action>
1257 [56]         </Actions>

```

```

1258 [47] </Target>
1259 [48] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1260 [49]   <!-- compare parent-guardian-id subject attribute with
1261 [50]     the value in the document -->
1262 [51]   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1263 [52]     equal">
1264 [53]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1265 [54]     and-only">
1266 [55]       <!-- parent-guardian-id subject attribute -->
1267 [56]       <SubjectAttributeDesignator AttributeId=
1268 [57]         "urn:oasis:names:tc:xacml:1.0:examples:attribute:
1269 [58]         parent-guardian-id"
1270 [59]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1271 [60]       </Apply>
1272 [61]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1273 [62]     and-only">
1274 [63]       <!-- parent-guardian-id element in the document -->
1275 [64]       <AttributeSelector RequestContextPath=
1276 [65]         "//md:record/md:parentGuardian/md:parentGuardianId/text()"
1277 [66]         DataType="http://www.w3.org/2001/XMLSchema#string">
1278 [67]       </AttributeSelector>
1279 [68]     </Apply>
1280 [69]   </Apply>
1281 [70]   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-
1282 [71]     equal">
1283 [72]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-
1284 [73]     and-only">
1285 [74]       <EnvironmentAttributeDesignator AttributeId=
1286 [75]         "urn:oasis:names:tc:xacml:1.0:environment:current-date"
1287 [76]         DataType="http://www.w3.org/2001/XMLSchema#date"/>
1288 [77]       </Apply>
1289 [78]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-
1290 [79]     yearMonthDuration">
1291 [80]       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-
1292 [81]     one-and-only">
1293 [82]       <!-- patient dob recorded in the document -->
1294 [83]       <AttributeSelector RequestContextPath=
1295 [84]         "//md:record/md:patient/md:patientDoB/text()"
1296 [85]         DataType="http://www.w3.org/2001/XMLSchema#date">
1297 [86]       </AttributeSelector>
1298 [87]     </Apply>
1299 [88]   </Apply>
1300 [89]   <AttributeValue DataType="http://www.w3.org/TR/2002/WD-xquery-
1301 [90]     operators-20020816#yearMonthDuration">
1302 [91]     P16Y
1303 [92]   </AttributeValue>
1304 [93] </Apply>
1305 [94] </Apply>
1306 [95] </Condition>
1307 [96] </Rule>

```

1307 [02]-[47] **Rule** declaration and **rule target**. See Rule 1 in Section 4.2.4.1 for the detailed
1308 explanation of these elements.

1309 [48]-[82] The **Condition** element. **Condition** must evaluate to “True” for the **rule** to be applicable.
1310 This **condition** evaluates the truth of the statement: the requestor is the designated parent or
1311 guardian and the patient is under 16 years of age.

1312 [48] The **Condition** is using the “urn:oasis:names:tc:xacml:1.0:function:and”
1313 function. This is a boolean function that takes one or more boolean arguments (2 in this case) and
1314 performs the logical “AND” operation to compute the truth value of the expression.

1315 [51]-[65] The truth of the first part of the condition is evaluated: The requestor is the designated
1316 parent or guardian. The **Apply** element contains a function invocation. The function name is

1317 contained in the `FunctionId` attribute. The comparison is done with
1318 `"urn:oasis:names:tc:xacml:1.0:function:string-equal"` that takes 2 arguments of
1319 `"http://www.w3.org/2001/XMLSchema#string"` data-type.

1320 [52] Since `"urn:oasis:names:tc:xacml:1.0:function:string-equal"` takes arguments
1321 of the `"http://www.w3.org/2001/XMLSchema#string"` data-type,
1322 `"urn:oasis:names:tc:xacml:1.0:function:string-one-and-only"` is used to ensure
1323 that the **subject attribute** `"urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id"` in
1324 the request **context** contains one and only one value.
1325 `"urn:oasis:names:tc:xacml:1.0:function:string-equal"` takes an argument
1326 expression that evaluates to a **bag** of `"http://www.w3.org/2001/XMLSchema#string"`
1327 values.

1328 [54] Value of the **subject attribute**
1329 `"urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id"` is
1330 selected from the request **context** with the `<SubjectAttributeDesignator>` element. This
1331 expression evaluates to a bag of `"http://www.w3.org/2001/XMLSchema#string"` values.

1332 [58] `"urn:oasis:names:tc:xacml:1.0:function:string-one-and-only"` is used to
1333 ensure that the **bag** of values selected by its argument contains one and only one value of data-
1334 type `"http://www.w3.org/2001/XMLSchema#string"`.

1335 [60] The value of the `md:parentGuardianId` element is selected from the **resource** content with
1336 the `AttributeSelector` element. `AttributeSelector` is a free-form XPath expression,
1337 pointing into the request **context**. The `RequestContextPath` XML attribute contains an XPath
1338 expression over the request **context**. Note that all namespace prefixes in the XPath expression
1339 are resolved with standard namespace declarations. The `AttributeSelector` evaluates to the
1340 **bag** of values of data-type `"http://www.w3.org/2001/XMLSchema#string"`.

1341 [66]-[83] The expression: "the patient is under 16 years of age" is evaluated. The patient is under
1342 16 years of age if the current date is less than the date computed by adding 16 to the patient's date
1343 of birth.

1344 [66] `"urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal"` is used to
1345 compute the difference of two dates.

1346 [67] `"urn:oasis:names:tc:xacml:1.0:function:date-one-and-only"` is used to ensure
1347 that the **bag** of values selected by its argument contains one and only one value of data-type
1348 `"http://www.w3.org/2001/XMLSchema#date"`.

1349 [68]-[69] Current date is evaluated by selecting the
1350 `"urn:oasis:names:tc:xacml:1.0:environment:current-date"` **environment attribute**.

1351 [71] `"urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration"` is
1352 used to compute the date by adding 16 to the patient's date of birth. The first argument is a
1353 `"http://www.w3.org/2001/XMLSchema#date"`, and the second argument is an
1354 `"http://www.w3.org/TR/2002/WD-xquery-operators-
1355 20020816#yearMonthDuration"`.

1356 [73] `"urn:oasis:names:tc:xacml:1.0:function:date-one-and-only"` is used to ensure
1357 that the **bag** of values selected by its argument contains one and only one value of data-type
1358 `"http://www.w3.org/2001/XMLSchema#date"`.

1359 [75]-[76] The `<AttributeSelector>` element selects the patient's date of birth by taking the
1360 XPath expression over the document content.

1361 [79]-[81] Year Month Duration of 16 years.

1362

4.2.4.3. Rule 3

1363

Rule 3 illustrates the use of an **obligation**. The XACML <Rule> element syntax does not include an element suitable for carrying an **obligation**, therefore Rule 3 has to be formatted as a <Policy> element.

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

1404

1405

1406

1407

1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

1418

1419

1420

1421

1422

```
[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Policy
[03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
[04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
[06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
[07]   PolicyId="urn:oasis:names:tc:xacml:examples:policyid:3"
[08]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
[09]     rule-combining-algorithm:deny-overrides">
[10]   <Description>
[11]     Policy for any medical record in the
[12]     http://www.medico.com/schemas/record.xsd namespace
[13]   </Description>
[14]   <Target>
[15]     <Subjects>
[16]       <AnySubject/>
[17]     </Subjects>
[18]     <Resources>
[19]       <Resource>
[20]         <!-- match document target namespace -->
[21]         <ResourceMatch
[22]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[23]           <AttributeValue
[24]             DataType="http://www.w3.org/2001/XMLSchema#string">
[25]             http://www.medico.com/schemas/record.xsd
[26]           </AttributeValue>
[27]           <ResourceAttributeDesignator AttributeId=
[28]             "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
[29]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
[30]           </ResourceMatch>
[31]         </Resource>
[32]       </Resources>
[33]     </Target>
[34]   <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:3"
[35]     Effect="Permit">
[36]     <Description>
[37]       A physician may write any medical element in a record
[38]       for which he or she is the designated primary care
[39]       physician, provided an email is sent to the patient
[40]     </Description>
[41]     <Target>
[42]       <Subjects>
[43]         <Subject>
[44]           <!-- match subject group attribute -->
[45]           <SubjectMatch
[46]             MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[47]             <AttributeValue
[48]               DataType="http://www.w3.org/2001/XMLSchema#string">physician</AttributeVa
[49]             lue>
[50]             <SubjectAttributeDesignator AttributeId=
[51]               "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
[52]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
[53]             </SubjectMatch>
[54]           </Subject>
[55]         </Subjects>
[56]       </Target>
[57]     </Rule>
[58]   </Policy>
```

```

1423 [51] </Subjects>
1424 [52] <Resources>
1425 [53] <Resource>
1426 [54] <!-- match requested xml element -->
1427 [55] <ResourceMatch
1428 MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1429 [56] <AttributeValue
1430 DataType="http://www.w3.org/2001/XMLSchema#string">
1431 [57] /md:record/md:medical
1432 [58] </AttributeValue>
1433 [59] <ResourceAttributeDesignator AttributeId=
1434 [60] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1435 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1436 [61] </ResourceMatch>
1437 [62] </Resource>
1438 [63] </Resources>
1439 [64] <Actions>
1440 [65] <Action>
1441 [66] <!-- match action -->
1442 [67] <ActionMatch
1443 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1444 [68] <AttributeValue
1445 DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
1446 [069] <ActionAttributeDesignator AttributeId=
1447 [070] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1448 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1449 [071] </ActionMatch>
1450 [072] </Action>
1451 [073] </Actions>
1452 [074] </Target>
1453 [075] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1454 equal">
1455 [076] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1456 and-only">
1457 [077] <!-- physician-id subject attribute -->
1458 [078] <SubjectAttributeDesignator AttributeId=
1459 [079] "urn:oasis:names:tc:xacml:1.0:example:
1460 [080] attribute:physician-id"
1461 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1462 [081] </Apply>
1463 [082] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1464 and-only">
1465 [083] <AttributeSelector RequestContextPath=
1466 [084] "//md:record/md:primaryCarePhysician/md:registrationID/text()"
1467 [085] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1468 [086] </Apply>
1469 [087] </Condition>
1470 [089] </Rule>
1471 [090] <Obligations>
1472 [091] <!-- send e-mail message to the document owner -->
1473 [092] <Obligation ObligationId=
1474 [093] "urn:oasis:names:tc:xacml:example:obligation:email"
1475 [094] FulfillOn="Permit">
1476 [095] <AttributeAssignment AttributeId=
1477 [096] "urn:oasis:names:tc:xacml:1.0:example:attribute:mailto"
1478 [097] DataType="http://www.w3.org/2001/XMLSchema#string">
1479 [098] <AttributeSelector RequestContextPath=
1480 [099] "//md:/record/md:patient/md:patientContact/md:email"
1481 [100] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1482 [101] </AttributeAssignment>
1483 [102] <AttributeAssignment AttributeId=
1484 [103] "urn:oasis:names:tc:xacml:1.0:example:attribute:text"
1485 [104] DataType="http://www.w3.org/2001/XMLSchema#string">

```



```

1486 [105] <AttributeValue>
1487 [106]     Your medical record has been accessed by:
1488 [107] </AttributeValue>
1489 [108] </AttributeAssignment>
1490 [109] <AttributeAssignment AttributeId=
1491 [110]     "urn:oasis:names:tc:xacml:example:attribute:text"
1492 [111]     DataType="http://www.w3.org/2001/XMLSchema#string">
1493 [112] <SubjectAttributeDesignator AttributeId=
1494 [113]     "urn:osasis:names:tc:xacml:1.0:subject:subject-id"
1495 [114]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1496 [114] </AttributeAssignment>
1497 [115] </Obligation>
1498 [116] </Obligations>
1499 [117] </Policy>

```

1500 [01]-[09] The `Policy` element includes standard namespace declarations as well as policy specific
1501 parameters, such as `PolicyId` and `RuleCombiningAlgId`.

1502 [07] **Policy** identifier. This parameter is used for the inclusion of the `Policy` in the `PolicySet`
1503 element.

1504 [08]-[09] **Rule combining algorithm** identifier. This parameter is used to compute the combined
1505 outcome of **rule effects** for **rules** that are applicable to the **decision request**.

1506 [10-13] Free-form description of the **policy**.

1507 [14]-[33] **Policy target**. The **policy target** defines a set of applicable decision requests. The
1508 structure of the `Target` element in the `Policy` is identical to the structure of the `Target` element
1509 in the `Rule`. In this case, the **policy target** is a set of all XML documents conforming to the
1510 "http://www.medico.com/schemas/record.xsd" target namespace. For the detailed description of
1511 the `Target` element see Rule 1, Section 4.2.4.1.

1512 [34]-[89] The only `Rule` element included in this `Policy`. Two parameters are specified in the **rule**
1513 header: `RuleId` and `Effect`. For the detailed description of the `Rule` structure see Rule 1,
1514 Section 4.2.4.1.

1515 [41]-[74] A **rule target** narrows down a **policy target**. **Decision requests** with the value of
1516 "urn:oasis:names:tc:xacml:1.0:exampe:attribute:role" **subject attribute** equal to
1517 "physician" [42]-[51], and that access elements of the medical record that "xpath-node-match"
1518 the "/md:record/md:medical" XPath expression [52]-[63], and that have the value of the
1519 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** equal to "read".

1520 [65]-[73] match the **target** of this **rule**. For a detailed description of the rule target see example 1,
1521 Section 4.2.4.1.

1522 [75]-[87] The `Condition` element. For the **rule** to be applicable to the authorization request,
1523 **condition** must evaluate to True. This **rule condition** compares the value of the
1524 "urn:oasis:names:tc:xacml:1.0:examples:attribute:physician-id" **subject**
1525 **attribute** with the value of the `physician id` element in the medical record that is being
1526 accessed. For a detailed explanation of rule condition see Rule 1, Section 4.2.4.1.

1527 [90]-[116] The `Obligations` element. **Obligations** are a set of operations that must be
1528 performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be
1529 associated with a positive or negative **authorization decision**.

1530 [92]-[115] The `Obligation` element consists of the `ObligationId`, the authorization decision
1531 value for which it must fulfill, and a set of attribute assignments.

1532 [92]-[93] `ObligationId` identifies an **obligation**. **Obligation** names are not interpreted by the
1533 **PDP**.

1534 [94] FulfillOn attribute defines an **authorization decision** value for which this **obligation** must
1535 be fulfilled.

1536 [95]-[101] **Obligation** may have one or more parameters. The **obligation** parameter
1537 "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto" is assigned the value
1538 from the content of the xml document.

1539 [95-96] AttributeId declares
1540 "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto" **obligation** parameter.

1541 [97] The **obligation** parameter data-type is defined.

1542 [98]-[100] The **obligation** parameter value is selected from the content of the XML document that is
1543 being accessed with the XPath expression over request **context**.

1544 [102]-[108] The **obligation** parameter
1545 "urn:oasis:names:tc:xacml:1.0:examples:attribute:text" of data-type
1546 "http://www.w3.org/2001/XMLSchema#string" is assigned the literal value "Your
1547 medical record has been accessed by:"

1548 [109]-[114] The **obligation** parameter
1549 "urn:oasis:names:tc:xacml:1.0:examples:attribute:text" of the
1550 "http://www.w3.org/2001/XMLSchema#string" data-type is assigned the value of the
1551 "urn:oasis:names:tc:xacml:1.0:subject:subject-id" **subject attribute**.

1552 4.2.4.4. Rule 4

1553 Rule 4 illustrates the use of the "Deny" Effect value, and a Rule with no Condition element.

```

1554 [01] <?xml version="1.0" encoding="UTF-8"?>
1555 [02] <Rule
1556 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1557 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1558 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1559 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1560 [07]   RuleId="urn:oasis:names:tc:xacml:example:ruleid:4"
1561 [08]   Effect="Deny">
1562 [09]   <Description>
1563 [10]     An Administrator shall not be permitted to read or write
1564 [11]     medical elements of a patient record in the
1565 [12]     http://www.medico.com/records.xsd namespace.
1566 [13]   </Description>
1567 [14]   <Target>
1568 [15]     <Subjects>
1569 [16]       <Subject>
1570 [17]         <!-- match role subject attribute -->
1571 [18]         <SubjectMatch
1572 [19]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1573 [20]           <AttributeValue
1574 [21]             DataType="http://www.w3.org/2001/XMLSchema#string">administra
1575 [22]             r</AttributeValue>
1576 [23]           <SubjectAttributeDesignator AttributeId=
1577 [24]             "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1578 [25]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1579 [26]           </SubjectMatch>
1580 [27]         </Subject>
1581 [28]       </Subjects>
1582 [29]     <Resources>
1583 [30]       <Resource>
1584 [31]         <!-- match document target namespace -->

```