

XML Access Control Use Case of XACML

- Fine-grained XML Document Access Control -

July 25, 2002

Author: Michiharu Kudo

This is a proposal for how a fine-grained XML document access control could make use of XACML as its policy language. The fine-grained XML document access control means an element-wise (and an attribute-wise) access control in a XML document. Using fine-grained access decisions, an application can return only readable portions to the requesting subject. The policy specification is based on the latest XACML language proposal document [1].

1. Overview

Figure 1 shows a data-flow diagram for fine-grained XML access control.

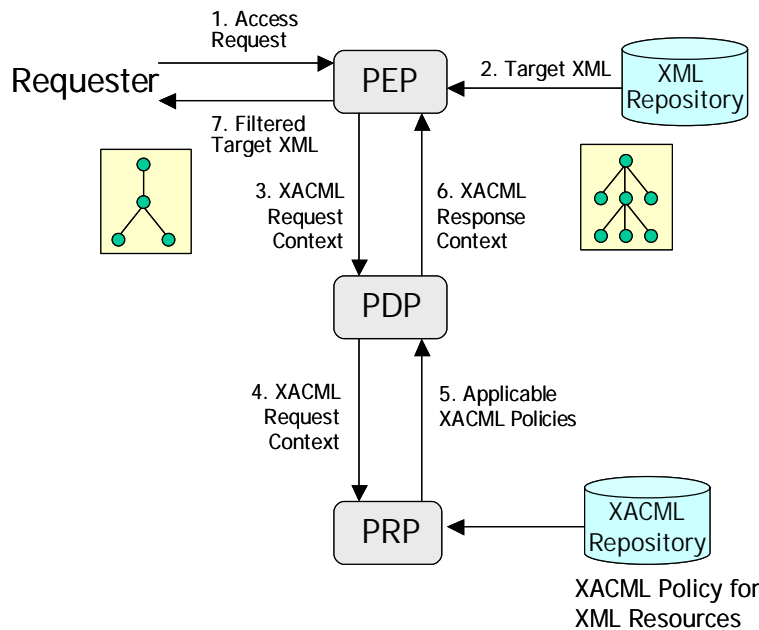


Figure 1 Data-flow diagram

Figure 1 shows an example data flow among XACML components (this diagram just indicates a logical relationship). XML documents are stored in XML repository. A requester submits an access request to PEP with a set of parameters including a target XML identifier (e.g. filename) and a

25 specific portion of the document (specific element, sub-tree, etc.) (step 1). That access request is
26 converted to an XACML Request Context. The PEP also retrieves the target XML document from
27 the repository and embed it in the XACML Request Context (step 2). We assume that this request
28 context carries enough information to the PDP. After receiving the request context (step 3), PDP
29 asks PRP to select a set of applicable XACML access control policies (step 4 and 5). Then the PDP
30 makes decisions whether the requested portion of the document is readable or not. One or more
31 access decisions (permit or deny on each element and attribute) called XACML Response Context
32 are sent back to the PEP (step 6). Using those access decisions, the PEP filters no-readable elements
33 (and attributes) out from the original document. The requester finally receives the XML document
34 that contains only readable portions (step 7). Besides this scenario, it is easy to support another
35 actions such as element-wise update and delete operations.

36 2. Example Document and Policy

37 2.1 Example Document

38 The following document is stored in XML repository as “A00.xml”

```
39 <a:employee xmlns:a="http://myNS">  
40   <a:name>Alice</a:name>  
41   <a:phone>111-1111</a:phone>  
42   <a:salary>10000</a:salary>  
43 </a:employee>
```

45 2.2 Example Policy

46 Employees who belong to a “regular” employee group can read an employee element, a name
47 element, and a phone element but cannot read a salary element. This access control policy is
48 specified in XACML as follows:

```
49 <PolicyStatement PolicyId="PolicyForRegularEmployee" RuleCombiningAlgId="//.../PermitOverrides">  
50   <Description>Policy for regular employee group: employee, name, and phone elements are readable.</Description>  
51   <Target>  
52     <Subjects MatchId="function:string-equal" DataType="xs:boolean">  
53       <AttributeDesignator DataType="xs:string"  
54         Designator="/c:Request/c:Subject/c:SubjectAttribute/c:Attribute[@DataType='Group']/ text()"/>  
55       <Attribute DataType="xs:string">Regular</Attribute>  
56     </Subjects>  
57     <Resources MatchId="function:string-match" DataType="xs:boolean">  
58       <AttributeDesignator DataType="xs:string"  
59         Designator="/c:Request/c:Resource/c:ResourceAttribute/c:Attribute[@DataType='XML']/ text()"/>  
60       <Attribute DataType="xs:string">A0[0-9]*.xml</Attribute>  
61     </Resources>  
62     <Actions MatchId="function:string-equal" DataType="xs:boolean">  
63       <AttributeDesignator DataType="xs:string"  
64         Designator="/c:Request/c:Action[@Namespace="urn:oasis:tc:XACML:action:XMLAC"]/text()"/>  
65       <Attribute DataType="xs:string">read</Attribute>  
66     </Actions>  
67   </Target>  
68   <RuleSet>  
69     <Rule RuleId="rule1" Effect="Permit">  
70       <Target>  
71         <Subjects MatchId="function:true"/>  
72     </Rule>
```

```

73     <Resources MatchId="function:node-equal" DataType="xs:string">
74         <AttributeDesignator
75             Designator="//c:ResourceAttribute/c:Attribute[@DataType='Node']/text()"/>
76         <Attribute DataType="xs:string">//c:ResourceContent/a:employee</Attribute>
77     </Resources>
78     <Action MatchId="function:true"/>
79 </Target>
80 </Rule>
81 <Rule RuleId="rule2" Effect="Permit">
82     <Target>
83         <Subjects MatchId="function:true"/>
84         <Resources MatchId="function:node-equal" DataType="xs:string">
85             <AttributeDesignator
86                 Designator="//c:ResourceAttribute/c:Attribute[@DataType='Node']/text()"/>
87             <Attribute
88                 DataType="xs:string">//c:ResourceContent/a:employee/a:name</Attribute>
89         </Resources>
90         <Action MatchId="function:true"/>
91     </Target>
92 </Rule>
93 <Rule RuleId="rule3" Effect="Permit">
94     <Target>
95         <Subjects MatchId="function:true"/>
96         <Resources MatchId="function:node-equal" DataType="xs:string">
97             <AttributeDesignator
98                 Designator="//c:ResourceAttribute/c:Attribute[@DataType='Node']/text()"/>
99             <Attribute
100                 DataType="xs:string">//c:ResourceContent/a:employee/a:phone</Attribute>
101         </Resources MatchId="function:true"/>
102         <Action MatchId="function:true"/>
103     </Target>
104 </Rule>
105 <Rule RuleId="rule4" Effect="Deny">
106     <Target>
107         <Subjects MatchId="function:true"/>
108         <Resources MatchId="function:node-match" DataType="xs:string">
109             <AttributeDesignator
110                 Designator="//c:ResourceAttribute/c:Attribute[@DataType='Node']/text()"/>
111             <Attribute
112                 DataType="xs:string">//c:ResourceContent/a:employee/a:salary</Attribute>
113         </Resources>
114         <Action MatchId="function:true"/>
115     </Target>
116 </Rule>
117 </RuleSet>
118 </PolicyStatement>
119

```

120 <PolicyStatement> contains a target specification and 4 rules. <Target> specifies applicability of
121 this policy that the requester's group is regular, target XML resource matches "A0[0-9]¥.xml, and
122 the action is read. The first rule compares "//c:ResourceContent/a:employee" element below the
123 request context with the value of the node attribute specified in the request context (some node). If
124 two elements are identical, then this rule returns "Permit". The second and third rules are
125 interpreted similarly. The fourth rule compares "//c:ResourceContent/a:employee/a:salary" element
126 with the value of the node attribute specified in the request context (some node). If two elements are
127 identical, then this rule returns "Deny".

128 2.3 Example Request Context

129 We assume that the following request context is issued by the PEP.

```

130
131 <c:Request xmlns:c="urn:oasis:names:tc:xacml:context" xmlns:a="http://myNS">
132   <c:Subject SubjectCategory="urn:oasis:names:tc:xacml:identifiers:AccessSubject">
133     <c:SubjectId Format="xs:string">Alice</c:SubjectId>
134     <c:SubjectAttribute Name="Group" >
135       <c:Attribute DataType="xs:string">Regular</c:Attribute>
136     </c:SubjectAttribute>
137   </c:Subject>
138   <c:Resource>
139     <c:ResourceSpecifier Format="XPath" Scope="Immediate"
140       ResourceID="//org/A00.xml#xmlns(a=http://myNS)xpointer(/a:employee/a:name)"/>
141     <c:ResourceAttribute Name="Location" >
142       <c:Attribute DataType="xs:string">//org</c:Attribute>
143     </c:ResourceAttribute>
144     <c:ResourceAttribute Name="XML" >
145       <c:Attribute DataType="xs:string">A00.xml</c:Attribute>
146     </c:ResourceAttribute>
147     <c:ResourceAttribute Name="Node" >
148       <c:Attribute
149         DataType="xs:string">//c:ResourceContent/a:employee/a:name</c:Attribute>
150     </c:ResourceAttribute>
151     <c:ResourceContent>
152       <a:employee xmlns:a="http://myNS">
153         <a:name>Alice</a:name>
154         <a:phone>111-1111</a:phone>
155         <a:salary>1000</a:salary>
156       </a:employee>
157     </c:ResourceContent>
158   </c:Resource>
159   <c:Action Namespace="urn:oasis:names:tc:XACML:action:XMLAC">read</c:Action>
160 </c:Request>

```

161 A subject element specifies a subject ID and a group attribute. A resource element specifies a target
162 resource that is encoded using XPath format. This data is given by the requesting application. In
163 the resource ID, three types of data are encoded: an URI to the target XML document (//org), XML
164 document file name (A00.xml), and the target portion of the document is “/a:employee/a:name”,
165 headed by “//c:ResourceContent” that navigates the correct location of the embedded XML
166 document. Each parameter is extracted and three different resource attributes are generated. These
167 parameters are referred from the policy using XPath. (See Section 4.2 if embedding the target XML
168 in the context is not desirable.)

169 2.4 How to Generate Access Decision

170 **Step 1:** When PDP receives the above request context, it passes the context to PRP. The PRP tries to
171 find a set of applicable policies that matches the request context. Since all target conditions
172 specified in the policy statement (PolicyForRegularEmployee) are satisfied, the PRP returns that
173 policy as an applicable policy.

174 **Step 2:** The PDP starts to evaluate the rule portion of the applicable policy. The first rule (rule1)
175 compares two node sets: one that is obtained by applying an XPath expression obtained from the
176 request context (“Node” resource attribute) on the target XML document (embedded in the context)
177 and the other that is obtained by applying an XPath expression described in the rule1
178 (//c:ResourceContent/a:employee) on the embedded target XML document. If both node sets
179 contain an identical node, then the target becomes true.

180 **Step 3:** If two node sets have no intersection, the node-equal function returns false. Then the target
181 of the rule becomes false. In the case of the first rule (rule1), the target becomes false while the
182 target of the second rule (rule2) becomes true. If the target becomes true, the rule combining
183 algorithm (PermitOverrides) returns “permit” or “deny” depending on the value specified in the
184 effect attribute. In the case of the rule2, it returns “permit”.

185 2.5 Example of Response Context

186 The following is an example of XACML Response Context saying that the requesting subject is
187 allowed to read the name element.

```
188  
189 <c:Response xmlns:c="urn:oasis:names:tc:xacml:context">  
190   <c:Result>  
191     <c:Decision>permit</c:Decision>  
192     <c:ResourceId>//org/A00.xml#xmlns(a=http://myNS)xpointer(/a:employee/a:name)</c:ResourceId>  
193   </c:Result>  
194 </c:Response>
```

195 3. Resource Matching

196 From XML resource protection viewpoint, it is important to define which granularity of the XML
197 resource can be protected using fine-grained access control policy. We consider two kinds of
198 reference methods:

- 199 1. Resource matching using XPath expression
- 200 2. Resource matching using regular expression

201

202 If we choose the resource matching using XPath expression, we always need a requested target
203 XML document. On the other hand, the resource matching using regular expression does not need
204 the requested target XML document The difference between two methods is briefly described in
205 Section 3.2.

206 3.1 Resource Matching using XPath Expression

207 A) Access to a specific node of a specific XML file

208 This context fragment shows a case that a requester needs to access a specific node of XML
209 document.

```
210  
211 <c:Resource>  
212   <c:ResourceSpecifier ResourceURI="//org/A00.xml#xpointer(/employee/name)"/>  
213   <c:ResourceAttribute Name="Location" >  
214     <c:Attribute DataType="xs:string">//org</c:Attribute>  
215   </c:ResourceAttribute>  
216   <c:ResourceAttribute Name="XML" >  
217     <c:Attribute DataType="xs:string">A00.xml</c:Attribute>  
218   </c:ResourceAttribute>  
219   <c:ResourceAttribute Name="Node" >
```

```
220     <c:Attribute DataType="xs:string"/>/employee/name</c:Attribute>
221 </c:ResourceAttribute>
222 </c:Resource>
```

223 Target resource reference uses an XPointer syntax [2]¹. The string before “#” character indicates
224 URI of the target XML resource and the string after “#xpointer” indicates XPath expression [3]. In
225 this example, the URI is “//org/A00.xml” and the XPath is “/employee/name”. In this case, this
226 XPath refers to a name element below the employee element. When a namespace identifier is used,
227 the XPointer expression becomes:

```
228
229 <c:Resource>
230   <c:ResourceSpecifier ResourceURI="//org/A00.xml#xmlns(a=http://myNS)xpointer(/a:employee/a:name)"/>
231     <c:ResourceAttribute Name="Location" >
232       <c:Attribute DataType="xs:string"/>/org</c:Attribute>
233     </c:ResourceAttribute>
234     <c:ResourceAttribute Name="XML" >
235       <c:Attribute DataType="xs:string"/>A00.xml</c:Attribute>
236     </c:ResourceAttribute>
237     <c:ResourceAttribute Name="Node" >
238       <c:Attribute DataType="xs:string"/>/a:employee/a:name</c:Attribute>
239     </c:ResourceAttribute>
240   </c:Resource>
```

241 The above specification means that each element in the A00.xml is accompanied with the
242 namespace URI “http://myNS” which namespace prefix is “a”. Note that in the example in Section
243 2, we added a prefix “c:ResourceContext” before “a:employee/a:name” to make sure that the target
244 path refers to the correct location of the request context.

245 The resource URI attribute can be an arbitrary URI syntax such as “file://c:/winnt/report.xml”. In
246 this case, the PDP and the PEP must have the identical meaning about the location of the target
247 resource.

```
248
249 <c:Resource>
250   <c:ResourceSpecifier ResourceURI="file://c:/winnt/report.xml#xpointer(/overview)"/>
251     <c:ResourceAttribute Name="Location" >
252       <c:Attribute DataType="xs:string"/>file://c:/winnt</c:Attribute>
253     </c:ResourceAttribute>
254     <c:ResourceAttribute Name="XML" >
255       <c:Attribute DataType="xs:string"/>report.xml</c:Attribute>
256     </c:ResourceAttribute>
257     <c:ResourceAttribute Name="Node" >
258       <c:Attribute DataType="xs:string"/>/overview</c:Attribute>
259     </c:ResourceAttribute>
260   </c:Resource>
```

261 B) Access to a sub-tree in an XML document

262 There are some cases where the requester needs to access a specific sub-tree below a certain node
263 not just a specific element. The PEP can tell that notion to the PDP by specifying “descendant” at
264 the scope attribute in the resource specifier element. Then PDP returns multiple access decisions in

¹ XACML uses a subset of XPointer specification that allows only XPath expression in “#xpointer()” function. The status of the latest XPointer specification is a candidate recommendation.

265 response to one access request. Note that the PDP must know the hierarchical structure of the target
266 XML document.

```
267 <c:ResourceSpecifier Format="XPointer" Scope="descendant"  
268 ResourceURI="//org/A00.xml#xmlns(a=http://myNS)xpointer(/a:employee)"/>
```

270 The above context fragment is asking access decisions for every node below the employee element.
271 The resultant response context includes four decisions as follows:

```
272 <c:Response xmlns:c="urn:oasis:names:tc:xacml:context">  
273 <c:Result>  
274 <c:Decision>permit</c:Decision>  
275 <c:ResourceURI="//org/A00.xml#xmlns(a=http://myNS)xpointer(/a:employee)"/>  
276 <c:Decision>permit</c:Decision>  
277 <c:ResourceURI="//org/A00.xml#xmlns(a=http://myNS)xpointer(/a:employee/a:name)"/>  
278 <c:Decision>permit</c:Decision>  
279 <c:ResourceURI="//org/A00.xml#xmlns(a=http://myNS)xpointer(/a:employee/a:phone)"/>  
280 <c:Decision>deny</c:Decision>  
281 <c:ResourceURI="//org/A00.xml#xmlns(a=http://myNS)xpointer(/a:employee/a:salary)"/>  
282 </c:Result>  
283 </c:Response>
```

285 Since the salary element is not readable by the regular employee group, “deny” is returned for the
286 salary node. By using this decisions, the PEP can create a requester’s view that does not include a
287 salary element:

```
288 <a:employee xmlns:a="http://myNS">  
289 <a:name>Alice</a:name>  
290 <a:phone>111-1111</a:phone>  
291 </a:employee>
```

293 C) Access to a collection of XML document

294 There are cases where the target XML resource is not one XML document but a collection of the
295 documents.

```
296 <c:Request xmlns:c="urn:oasis:names:tc:xacml:context">  
297 <c:Subject SubjectCategory="urn:oasis:names:tc:xacml:identifiers:AccessSubject">  
298 <c:SubjectId Format="xs:string">Alice</c:SubjectId>  
299 <c:SubjectAttribute Name="Group" >  
300 <c:Attribute DataType="xs:string">Regular</c:Attribute>  
301 </c:SubjectAttribute>  
302 </c:Subject>  
303 <c:Resource>  
304 <c:ResourceSpecifier Format="XPointer" Scope="immediate"  
305 ResourceURI="//org/records#xmlns(a=http://myNS)xpointer(/a:employee)"/>  
306 <c:ResourceAttribute Name="Location" >  
307 <c:Attribute DataType="xs:string">//org</c:Attribute>  
308 </c:ResourceAttribute>  
309 <c:ResourceAttribute Name="XML" >  
310 <c:Attribute DataType="xs:string">A00.xml</c:Attribute>  
311 </c:ResourceAttribute>  
312 <c:ResourceAttribute Name="Node" >  
313 <c:Attribute DataType="xs:string">/a:employee</c:Attribute>  
314 </c:ResourceAttribute>  
315 </c:Resource>  
316 <c:Action Namespace="urn:oasis:names:tc:XACML:action:XMLAC">read</c:Action>  
317 </c:Request>
```

319 Above query assumes that the resource URI “http://org/records” means a collection of XML
 320 documents. If the resource URI means a collection of XML document, the PDP and the PEP must
 321 share the notation for the unique identifier of the XML document. For example, we assume here
 322 that there are two XML documents are meant by that URI, which is identified by emp1 and emp2. A
 323 response context becomes:

```
324
325 <c:Response xmlns:c=" urn:oasis:names:tc:xacml:context">
326   <c:Result>
327     <c:Decision>permit</c:Decision>
328     <c:ResourceId>//org/emp1#xmlns(a=http://myNS)xpointer(/a:employee)</c:ResourceId>
329     <c:Decision>permit</c:Decision>
330     <c:ResourceId>//org/emp2#xmlns(a=http://myNS)xpointer(/a:employee)</c:ResourceId>
331   </c:Result>
332 </c:Response>
```

333 3.2 Resource Matching using Regular Expression

334 Potential problem of using XPath expression in the fine-grained XML document control is that the
 335 XML document is needed to evaluate the XPath expression. In the case that the target XML is huge
 336 or the communication channel is narrow, it is not wise to retrieve a target XML document at the
 337 evaluation time. Another issue is that the PEP does not want to disclose the requested XML
 338 document to the PDP. A simple path expression is useful for such cases. The simple path expression
 339 uses regular expression to match a string representation of the requested path expression and a
 340 string representation of the path expression in the policy.

```
341
342 <PolicyStatement PolicyId="PolicyForRegularEmployee" RuleCombiningAlgId="//.../PermitOverrides">
343   <Description>Policy for regular employee group: employee, name, and phone elements are readable.</Description>
344   <Target>
345     <Subjects MatchId="function:string-equal" DataType="xs:boolean">
346       <AttributeDesignator DataType="xs:string"
347         Designator="/c:Request/c:Subject/c:SubjectAttribute/c:Attribute[@DataType='Group']/ text()"/>
348       <Attribute DataType="xs:string">Regular</Attribute>
349     </Subjects>
350     <Resources MatchId="function:string-match" DataType="xs:boolean">
351       <AttributeDesignator DataType="xs:string"
352         Designator="/c:Request/c:Resource/c:ResourceAttribute/c:Attribute[@DataType='XML']/ text()"/>
353       <Attribute DataType="xs:string">A0[0-9]*.xml</Attribute>
354     </Resources>
355     <Actions MatchId="function:string-equal" DataType="xs:boolean">
356       <AttributeDesignator DataType="xs:string"
357         Designator="/c:Request/c:Action[@Namespace=" urn:oasis:tc:XACML:action:XMLAC"/text()"/>
358       <Attribute DataType="xs:string">read</Attribute>
359     </Actions>
360   </Target>
361   <RuleSet>
362     <Rule RuleId="rule1" Effect="Permit">
363       <Target>
364         <Subjects MatchId="function:true"/>
365         <Resources MatchId="function:string-match" DataType="xs:string">
366           <AttributeDesignator
367             Designator="/c:Request/c:Resource/c:ResourceAttribute/c:Attribute[@DataType='Node']/text()"/>
368           <Attribute DataType="xs:string">/a:employee</Attribute>
369         </Resources>
370         <Action MatchId="function:true"/>
371       </Target>
372     </Rule>
373     <Rule RuleId="rule2" Effect="Permit">
374       <Target>
375         <Subjects matchId="function:true"/>
```



```

376         <Resources MatchId="function:string-match" DataType="xs:string">
377             <AttributeDesignator
378                 Designator="/c:Request/c:Resource/c:ResourceAttribute/c:Attribute[@DataType='Node']/text()"/>
379             <Attribute DataType="xs:string">/a:employee/(a:name|a:phone)</Attribute>
380         </Resources>
381         <Action MatchId="function:true"/>
382     </Target>
383 </Rule>
384 <Rule RuleId="rule3" Effect="Deny">
385     <Target>
386         <Subjects MatchId="function:true"/>
387         <Resources MatchId="function:string-match" DataType="xs:string">
388             <AttributeDesignator
389                 Designator="/c:Request/c:Resource/c:ResourceAttribute/c:Attribute[@DataType='Node']/text()"/>
390             <Attribute DataType="xs:string">/a:employee/a:salary</Attribute>
391         </Resources>
392         <Action MatchId="function:true"/>
393     </Target>
394 </Rule>
395 </RuleSet>
396 </PolicyStatement>

```

397 We assume that the following request context is submitted from the PEP.

```

398 <c:Request xmlns:c="urn:oasis:names:tc:xacml:context">
399     <c:Subject SubjectCategory="urn:oasis:names:tc:xacml:identifiers:AccessSubject">
400         <c:SubjectId Format="xs:string">Alice</c:SubjectId>
401         <c:SubjectAttribute Name="Group" >
402             <c:Attribute DataType="xs:string">Regular</c:Attribute>
403         </c:SubjectAttribute>
404     </c:Subject>
405     <c:Resource>
406         <c:ResourceSpecifier Format="URI" Scope="Immediate"
407             ResourceURI="//org/A00.xml?a:employee/a:name"/>
408         <c:ResourceAttribute Name="Location" >
409             <c:Attribute DataType="xs:string">//org</c:Attribute>
410         </c:ResourceAttribute>
411         <c:ResourceAttribute Name="XML" >
412             <c:Attribute DataType="xs:string">A00.xml</c:Attribute>
413         </c:ResourceAttribute>
414         <c:ResourceAttribute Name="Node" >
415             <c:Attribute DataType="xs:string">/a:employee/a:name</c:Attribute>
416         </c:ResourceAttribute>
417     </c:Resource>
418     <c:Action Namespace="urn:oasis:names:tc:XACML:action:XMLAC">read</c:Action>
419 </c:Request>

```

421 Above query has a simple path expression “/a:employee/a:name”. This path expression matches the
422 path expression written in the second rule “/a:employee/(a:name|a:phone)”.

423 4. Notes

424 4.1 Namespace

425 In this document, we used a namespace prefix in XPath expressions but it assumes the prefix is
426 associated with appropriate namespace URI in the target XML document (XACML Request
427 Context). More strict way of specifying namespaces is to use namespace-URI() function and
428 local-name() function. It makes XPath expression very complicated. For example,

429 /a:employee/a:name becomes /*[namespace-URI() = "http://myNS" and
430 local-name()="employee"]/*[namespace-URI() = "http://myNS" and local-name()="name"].

431 The reason we used the prefix-based syntax is the readability of the policy. If we use the
432 prefix-based syntax, someone who manages the authorization system must be sure about the trust
433 relationship between PEP and PDP so that they can share both the correct namespace URI and the
434 corresponding prefix. Otherwise, the namespace prefix and URI meant by the policy writer may
435 differ from the URI that is handed by the PEP.

436 4.2 Separation of Context from target XML document

437 In this document, a request context is used to embed the target XML document. One of the merits of
438 this scheme is that the access control policy only has to worry about one input document (request
439 context) but not many. Any portion of the target XML document can be referred only using XPath
440 expression. A downside would be that the whole XML document must be copied into the request
441 context structure and the namespace specification makes XPath expression complicated. Alternative
442 is to add some annotation to the attribute designator indicating that the XPath is for the request
443 context or one for the target XML document.

```
444 <Resources MatchId="function:node-equal" DataType="xs:string">  
445 <AttributeDesignator Type="reference" RefApply="context" Apply="document"  
446 Designator="ResourceAttribute/ Attribute[@DataType='Node']/text()"/>  
447 <Attribute DataType="xs:string" Type="value" Apply="document">employee/phone</Attribute>  
448 </Resources MatchId="function:true"/>
```

450 A type attribute indicates that XPath expression is specified as a value or a reference. The reference
451 means that the XPath should be first applied to "RefApply" attribute as a reference. An apply
452 attribute indicates that on which resource should XPath expression be applied. "context" and
453 "document" means the request context and the target XML document, respectively.

454 5. Reference

455 [1] XACML, The OASIS eXtensible Access Control Markup Language (XACML), Committee
456 Draft Version 15 July, 2002, <http://www.oasis-open.org/committees/xacml>

457 [2] XPointer, XML Pointer Language (XPointer) Version 1.0, W3C Candidate Recommendation, 11
458 September 2001, <http://www.w3.org/TR/2001/CR-xptr-20010911>.

459 [3] XPath, XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999,
460 <http://www.w3.org/TR/xpath>.