

# 1 **Service Modeling Language**

## 2 **Draft Specification**

3 **Version 0.5, 25 July 2006**

### 4 **Authors**

5 Pratul Dubish, Microsoft  
6 Zulah Eckert, BEA  
7 Dave Ehnebuske, IBM  
8 Eugene Golovinsky, BMC  
9 Steve Jerman, Cisco  
10 Heather Kreger, IBM  
11 Milan Milenkovic, Intel  
12 Bryan Murray, HP  
13 Phil Prasek, HP  
14 Drue Reeves, Dell  
15 Junaid Saiyed, EMC  
16 Harm Sluiman, IBM  
17 Bassam Tabbara, Microsoft  
18 Vijay Tewari, Intel  
19 John Tollefsrud, Sun  
20 William Vambenepe, HP  
21 Andrea Westerinen, Microsoft

22  
23 Permission to copy and display the Service Modeling Language Specification, in any medium  
24 without fee or royalty is hereby granted, provided that you include the following on ALL copies of  
25 the Service Modeling Language Specification, or portions thereof, that you make:

26  
27 1. A link or URL to the Service Modeling Language Specification at this location:

28 <http://go.microsoft.com/fwlink/?LinkId=70293>

29  
30 2. The copyright notice as shown in the Service Modeling Language Specification.

31 BEA, BMC, Cisco, Dell, EMC, HP, IBM, Intel, Microsoft, and Sun (collectively, the "Authors") each  
32 agree to grant you a royalty-free license, under reasonable, non-discriminatory terms and  
33 conditions to their respective patents that they deem necessary to implement the Service  
34 Modeling Language Specification.

35  
36 THE SERVICE MODELING LANGUAGE SPECIFICATION IS PROVIDED "AS IS," AND THE  
37 AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED,  
38 INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A  
39 PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE  
40 MANAGEMENT MODELING LANGUAGE SPECIFICATION ARE SUITABLE FOR ANY  
41 PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE  
42 ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.  
43 THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL  
44 OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR  
45 DISTRIBUTION OF THE MANAGEMENT MODELING LANGUAGE SPECIFICATION.

46

47 The name and trademarks of the Authors may NOT be used in any manner, including advertising  
48 or publicity pertaining to the Service Modeling Language Specification or its contents without  
49 specific, written prior permission. Title to copyright in the Service Modeling Language  
50 Specification will at all times remain with the Authors.

51

52 No other rights are granted by implication, estoppel or otherwise.

53 **Abstract**

54 This specification defines the Service Modeling Language (SML) used to model  
55 complex IT services and systems, including their structure, constraints, policies, and  
56 best practices. SML is based on a profile on XML Schema and Schematron.

57 **Status**

58 This specification is the first draft of a work in progress. It is being published to  
59 solicit feedback. A feedback agreement is required before the working group can  
60 accept feedback. Please contact [sml-feedback@external.cisco.com](mailto:sml-feedback@external.cisco.com) for details.

61 At some future date, the contents may be published under another name or under  
62 several new specifications, as shall be agreed by the authors and their respective  
63 corporations at that time.

64	<b>Table of Contents</b>	
65	<b>Service Modeling Language</b> .....	<b>1</b>
66	<b>Draft Specification</b> .....	<b>1</b>
67	Abstract .....	3
68	Status .....	3
69	Table of Contents .....	4
70	<b>1. Introduction</b> .....	<b>6</b>
71	<b>2. Terminology and Notation</b> .....	<b>7</b>
72	2.1 Terminology.....	7
73	2.2 XML Namespaces .....	7
74	<b>3. Schemas</b> .....	<b>8</b>
75	3.1 XML Schema Profile.....	8
76	3.1.1 <xs:redefine> .....	8
77	3.1.2 Unqualified Local Elements .....	8
78	3.1.3 targetNamespace on <xs:schema>.....	9
79	3.2 References.....	9
80	3.2.1 Reference Semantics .....	10
81	3.3 Reference Schemes.....	11
82	3.3.1 URI Scheme .....	11
83	3.3.2 EPR Scheme .....	13
84	3.4 Constraints on References.....	14
85	3.4.1 sml:acyclic .....	14
86	3.4.2 sml:targetElement.....	15
87	3.4.3 sml:targetType .....	15
88	3.5 Identity Constraints .....	16
89	3.5.1 University Example.....	16
90	3.5.2 sml:key and sml:unique .....	18
91	3.5.3 sml:keyref.....	19
92	<b>4. Rules</b> .....	<b>19</b>
93	4.1 Schematron Profile.....	23
94	4.1.1 Limited Support .....	23
95	<b>5. Model Validation</b> .....	<b>23</b>
96	5.1 Schematron Phase .....	23
97	<b>6. SML Extension Reference</b> .....	<b>23</b>
98	6.1 Types.....	23
99	6.1.1 sml:ref .....	23
100	6.2 Attributes .....	24
101	6.2.1 sml:acyclic .....	24
102	6.2.2 sml:targetElement.....	24
103	6.2.3 sml:targetType .....	24
104	6.2.4 sml:uri.....	25
105	6.3 Elements .....	25

106	6.3.1 sml:key .....	25
107	6.3.2 sml:unique .....	25
108	6.3.3 sml:keyref.....	25
109	6.4 XPath functions .....	26
110	6.4.1 smlfn:deref .....	26
111	<b>7. Acknowledgements .....</b>	<b>26</b>
112	<b>8. References .....</b>	<b>26</b>
113	<b>Appendix I – Sample Model.....</b>	<b>27</b>
114	<b>Appendix II – Complexity of Supporting targetElement and targetType on</b>	
115	<b>Local Element Declarations .....</b>	<b>31</b>
116		

117

## 118 1. Introduction

119 The Service Modeling Language (SML) provides a rich set of constructs for creating  
120 models of complex IT services and systems. These models typically include  
121 information about configuration, deployment, monitoring, policy, health, capacity  
122 planning, target operating range, service level agreements, and so on. Models  
123 provide value in several important ways.

- 124 1. Models focus on capturing all **invariant aspects** of a service/system that  
125 must be maintained for the service/system to be functional. They capture as  
126 much detail as is necessary, and no more.
  
- 127 2. Models are units of **communication and collaboration** between designers,  
128 implementers, operators, and users; and can easily be shared, tracked, and  
129 revision controlled. This is important because complex services are often built  
130 and maintained by a variety of people playing different roles.
  
- 131 3. Models drive **modularity, re-use, and standardization**. Most real-world  
132 complex services and systems are composed of sufficiently complex parts.  
133 Re-use and standardization of services/systems and their parts is a key factor  
134 in reducing overall production and operation cost and in increasing reliability.
  
- 135 4. Models represent a powerful mechanism for **validating changes** *before*  
136 applying the changes to a service/system. Also, when changes happen in a  
137 running service/system, they can be validated against the intended state  
138 described in the model. The actual service/system and its model together  
139 enable a *self-healing service/system* – the ultimate objective. *Models of a*  
140 *service/system must necessarily stay decoupled from the live service/system*  
141 *to create the control loop*
  
- 142 5. Models enable increased **automation** of management tasks. Automation  
143 facilities exposed by the majority of IT services/systems today could be  
144 driven by software – not people – for reliable initial realization of a  
145 service/system as well as for ongoing lifecycle management.

146

147 A model in SML is realized as a set of interrelated XML documents. The XML  
148 documents contain information about the parts of an IT service, as well as the  
149 constraints that each part must satisfy for the IT service to function properly.  
150 Constraints are captured in two ways:

- 151 1. **Schemas** – these are constraints on the structure and content of the  
152 documents in a model. SML uses a profile of XML Schema 1.0 [2,3] as the  
153 schema language. SML also defines a set of extensions to XML Schema to  
154 support inter-document references.
  
- 155 2. **Rules** – are Boolean expressions that constrain the structure and content of  
156 documents in a model. SML uses a profile of Schematron [4,5,6] and XPath  
157 1.0 [9] for rules.

158 Once a model is defined, one of the important operations on the model is to establish  
159 its validity. This involves checking whether all data in a model satisfies the schemas  
160 and rules declared.

161 This specification focuses primarily on defining the profile of XML Schema and  
162 Schematron used by SML, as well as the process of model validation. It is assumed  
163 that the reader is familiar with XML Schema and Schematron.

## 164 **2. Terminology and Notation**

### 165 **2.1 Terminology**

166 Document

167 A well-formed XML 1.0 document (see [12] for a detailed definition)

168 Model

169 A set of inter-related documents that describe an IT service or system. Each  
170 model consists of two disjoint subsets of documents – genic documents and  
171 phenic documents.

172 Rule

173 A Boolean expression that constrains the structure and content of a set of  
174 documents in a model.

175 Genic Documents

176 The subset of documents in a model that describes the schemas and rules  
177 that govern the structure and content of the model's documents. This  
178 specification defines two kinds of genic documents - XML Schema documents  
179 that conform to SML's profile of XML Schema and rule documents that  
180 conform to SML's profile of Schematron.

181 Phenic Documents

182 The subset of documents in a model that describe the structure and content  
183 of the modeled entities.

184 Model Validation

185 The process of verifying that all documents in a model are valid with respect  
186 to the model's genic documents.

187 Model Validator

188 An embodiment capable of performing model validation

### 189 **2.2 XML Namespaces**

190 The XML Namespace URI that must be used in the schema documents of SML models  
191 is:

192 <http://schemas.serviceml.org/sml/2006/07>

193 Table 1 lists XML namespaces that are used in this specification. The choice of any  
194 namespace prefix is arbitrary and not semantically significant.

195

196 **Table 1: XML Namespaces used in this specification.**

Prefix	XML Namespace	Specification(s)
sml	<a href="http://schemas.serviceml.org/sml/2006/07">http://schemas.serviceml.org/sml/2006/07</a>	This specification
smlfn	<a href="http://schemas.serviceml.org/sml/function/2006/07">http://schemas.serviceml.org/sml/function/2006/07</a>	This specification
wsa	<a href="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addressing</a>	[ <a href="#">WS Addressing Core</a> ]
xs	<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>	[ <a href="#">XML Schema</a> ]
sch	<a href="http://purl.oclc.org/dsdl/schematron">http://purl.oclc.org/dsdl/schematron</a>	[ <a href="#">Schematron</a> ]
xsi	<a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a>	[ <a href="#">Xml Schema Instance</a> ]

### 197 **3. Schemas**

198 SML uses a profile of W3C XML Schema 1.0 to define constraints on the structure of  
199 data in a model.

200 SML scenarios require several features that either do not exist or are not fully  
201 supported in XML Schema. These features can be classified as follows:

- 202 • **References** – XML Schema does not have any support for *inter-document*  
203 references, although it does support *intra-document* references through  
204 `xs:ID`, `xs:IDREF`, `xs:key` and `xs:keyref`. Inter-document references are  
205 fundamental to SML since a document is a unit of versioning. SML extends  
206 XML Schema to support inter-document references and a set of constraints on  
207 inter-document references.
- 208 • **Rules** – XML Schema does not support a language for defining arbitrary rules  
209 on the structure and content of XML documents. SML uses Schematron to  
210 express assertions on the structure and content of XML documents.

211 XML Schema supports two forms of extension: “attributes in different namespace”  
212 and “application information elements”; both forms are used by SML extensions.

#### 213 **3.1 XML Schema Profile**

214 SML supports a strict subset of XML Schema 1.0. This section describes the XML  
215 Schema features that are not supported or have limited support in SML. A  
216 justification is provided for each feature. An XML Schema with any of these features  
217 will be rejected by model validators.

##### 218 **3.1.1 <xs:redefine>**

219 `xs:redefine` is not supported in SML.

220 `xs:redefine` is a feature for schema evolution and versioning in XML Schema. This  
221 feature enables schema authors to define a new version of a schema component,  
222 and completely replace the original schema component with the new version. XML  
223 Schema does not guarantee that the new version of the component is compatible  
224 with the original component. Thus, it is possible to break existing schema  
225 components that depend on the original component.

##### 226 **3.1.2 Unqualified Local Elements**

227 Unqualified local elements are not supported in SML.

228 Local element declarations must describe elements with qualified names. This can be  
229 done by specifying `elementFormDefault="qualified"` on `<xs:schema>` or  
230 specifying `form="qualified"` on local `<xs:element>`.

231 This is to avoid element name collisions, and maintain a consistent naming approach  
232 especially when dealing with different schemas.

### 233 3.1.3 targetNamespace on <xs:schema>

234 `targetNamespace` on `xs:schema` is not optional and must always be specified.

235 XML schemas without target namespaces are not supported. They do not work well  
236 with XPath expressions used in constraints within the schema.

## 237 3.2 References

238 XML documents introduce boundaries across content that needs to be treated as a  
239 unit. XML Schema does not have any support for inter-document references. SML  
240 extends XML Schema to support inter-document references and a set of constraints  
241 on inter-document references.

242 Support for inter-document references includes:

- 243 • A new data type that represents references to elements in other documents.
- 244 • Multiple addressing schemes for representing references.
- 245 • Constraints on the type of a referenced element.
- 246 • The ability to define key, unique, and key reference constraints across inter-  
247 document references.

248 An SML reference is a link from one element to another. It can be represented by  
249 using a variety of schemes, such as Uniform Resource Identifiers (URIs) [7] and  
250 Endpoint References (EPRs) [8]. SML does not mandate the use of any specific  
251 scheme for representing references; the `sml:ref` type has been defined to allow  
252 model validators complete flexibility in choosing appropriate schemes.

253 `sml:ref` is a complex type whose definition is as follows:

```
254     <xs:complexType name="ref"  
255         sml:acyclic="false"  
256         final="extension">  
257         <xs:sequence>  
258             <xs:any namespace="##any" minOccurs="0"  
259                 maxOccurs="unbounded"  
260                 processContents="lax"/>  
261         </xs:sequence>  
262         <xs:anyAttribute namespace="##any" processContents="lax"/>  
263     </xs:complexType>
```

264 Note that the above definition allows elements and attributes from any namespace to  
265 occur in an element whose type is `sml:ref`. Thus, a scheme for references can be  
266 implemented by defining an XML namespace for the scheme, and references can be  
267 represented in this scheme by nesting element and attribute instances from this  
268 namespace as attributes and children of `sml:ref` elements.

269 An SML reference is encapsulated in an element of type `sml:ref` or a type derived  
270 from `sml:ref`. This is illustrated in the following example:

271

272

```
<xs:element name="EnrolledCourse" type="sml:ref"
            sml:targetType="tns:CourseType"/>
```

273

274

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

275

```
  <xs:sequence>
```

276

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

277

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

278

```
    <xs:element name="EnrolledCourses" minOccurs="0">
```

279

```
      <xs:complexType>
```

280

```
        <xs:sequence>
```

281

```
          <xs:element ref="tns:EnrolledCourse"
```

282

```
            maxOccurs="unbounded"/>
```

283

```
        </xs:sequence>
```

284

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

285

```
    </xs:element>
```

286

```
  </xs:sequence>
```

287

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

288

289 The `EnrolledCourse` element declaration is of type `sml:ref` which marks it as a  
290 document reference, and this element declaration is used in `StudentType` to  
291 reference the elements corresponding to the courses in which a student is enrolled.

292 Examples of the use of `sml:ref` for `EnrolledCourse` are found in the section,  
293 [Reference Schemes](#). This section demonstrates the use of the URI and EPR schemes  
294 to define the reference.

### 295 3.2.1 Reference Semantics

#### 296 3.2.1.1 At Most One Target

297 Every reference must target (or resolve to) at most one element in a model.  
298 Dangling references are allowed in SML; therefore it is possible that the target of a  
299 reference may not exist in a model. It is an error if a reference targets more than  
300 one element in a model.

#### 301 3.2.1.2 Multiple References

302 An element in a document can be targeted by multiple different references. These  
303 references may use different schemes and/or be expressed in different ways.

#### 304 3.2.1.3 Empty or Null References

305 An element of type `sml:ref` with `xsi:nil="true"` or with no content is valid. A  
306 model validator is required to treat such an element as if the reference were not  
307 present.

#### 308 3.2.1.4 `deref()` XPath Extension Function

309 Each model validator must provide an implementation of the `deref()` XPath  
310 extension function that is capable of resolving references expressed in the model  
311 validator's chosen scheme(s). This function takes a node-set of elements whose type  
312 is `sml:ref` or a type derived by restriction from `sml:ref` and returns a node-set  
313 consisting of element nodes corresponding to the elements referenced by the input  
314 node set. In particular, for each reference node **R** in the input node set, excluding  
315 empty or null references, the output node set contains at most one element node.

- 316 • The output node set contains one element node if **R** targets a single element  
317 in some document in the model
- 318 • The output node set contains no element node if the target of **R** is not in the  
319 model

### 320 3.3 Reference Schemes

321 A reference can be represented by using a variety of schemes, and SML does not  
322 mandate the use of any specific schemes. Uniform Resource Identifiers (URIs) [7]  
323 and endpoint references (EPRs) [8] are two common schemes for referencing  
324 resources. Although SML does not require the use either scheme, it does define how  
325 a reference must be represented using the URI scheme and the EPR scheme.

#### 326 3.3.1 URI Scheme

327 References that are represented using the URI scheme must be implemented by  
328 using the `sml:uri` global attribute on elements whose type is `sml:ref`. More  
329 precisely, if a model validator chooses to represent references using the URI scheme,

- 330 • It must represent the references using the `sml:uri` attribute on elements of  
331 type `sml:ref` or a derived type of `sml:ref`
- 332 • It must treat each instance element of type `sml:ref` with `sml:uri` attribute  
333 as a reference represented using URI scheme, and must be able to resolve  
334 such references

335 For example, if the reference in `EnrolledCourse` element is represented using the  
336 URI scheme, an instance of `EnrolledCourse` will appear as follows:

```
337 <EnrolledCourse xmlns="urn:university" sml:uri="SomeValidUri"/>
```

338 where `SomeValidUri` is a valid URI as defined in [7].

339 Suppose that a model has the following documents, and each document has an  
340 associated URI:

Document	URI
Course PHY101	/Universities/MIT/Courses/PHY101.xml
Course MAT200	/Universities/MIT/Courses/MAT200.xml
Student 1000	/Universities/MIT/Students/1000.xml
Student 1001	/Universities/MIT/Students/1001.xml

341

342 The following is a sample instance document for Student 1000 where the references  
343 are represented in URI scheme by using the `sml:uri` attribute:

```
344 <Student xmlns="urn:university">
345   <ID>1000</ID>
346   <Name>John Doe</Name>
347   <EnrolledCourses>
348     <EnrolledCourse sml:uri="/Universities/MIT/Courses/PHY101.xml"/>
349     <EnrolledCourse sml:uri="/Universities/MIT/Courses/MAT200.xml"/>
350   </EnrolledCourses>
351 </Student>
```

352

### 353 3.3.1.1 Fragment Identifier

354 SML requires the use of the following XPointer [10] profile for representing fragment  
355 identifiers.

- 356 • Only two schemes – xmlns() and xpointer() – are supported.
- 357 • The expression specified for the xpointer scheme must be a restricted  
358 XPath1.0 [9] expression that must resolve to at most one element node. In  
359 particular, this expression must not contain
  - 360 ○ the union (“|”) operator defined for XPath 1.0
  - 361 ○ point() and range() node tests defined for xpointer() scheme
- 362 • This expression can only use the functions defined in the XPath 1.0 core  
363 function library (see [9] for details). It can not use the smlfn:deref function  
364 and/or the following functions defined for xpointer() scheme (see [11] for  
365 details):
  - 366 ○ range-to
  - 367 ○ string-range
  - 368 ○ range
  - 369 ○ range-inside
  - 370 ○ start-point
  - 371 ○ end-point
  - 372 ○ here
  - 373 ○ origin

374 The following example illustrates the use of xpointer fragments. Consider the case  
375 where all courses offered by MIT are stored in a single XML document – Courses.xml  
376 – whose URI is /Universities/MIT/Courses.xml. In this case, the element inside  
377 Courses.xml that corresponds to the course PHY101 can be referenced as follows  
378 (assuming that Courses is the root element in Courses.xml)

```
379 <Student xmlns="urn:university">  
380   <ID>1000</ID>  
381   <Name>John Doe</Name>  
382   <EnrolledCourses>  
383     <EnrolledCourse  
384       sml:uri="/Universities/MIT/Courses.xml#xmlns(u=urn:university)  
385         xpointer(/u:Courses/u:Course[u:Name='PHY101'])"  
386     </EnrolledCourse>  
387   </EnrolledCourses>  
388 </Student>
```

389

390 An element of type `sml:ref` can also be used to reference an element in its own  
391 document. To see this consider the following instance document

```
392     <University xmlns="urn:university">
393         <Name>MIT</Name>
394         <Courses>
395             <Course>
396                 <Name>PHY101</Name>
397             </Course>
398             <Course>
399                 <Name>MAT200</Name>
400             </Course>
401         </Courses>
402         <Students>
403             <Student>
404                 <ID>123</ID>
405                 <Name>Jane Doe</Name>
406                 <EnrolledCourses>
407                     <EnrolledCourse
408                         sml:uri="#xmlns(u=urn:university)
409                         xpointer(/u:University/u:Courses/u:Course[u:Name='MAT200']"
410                     </EnrolledCourse>
411                 </EnrolledCourses>
412             </Student>
413         </Students>
414     </University>
415
```

416 Here, the `EnrolledCourse` element for the student Jane Doe references the  
417 `Course` element for MAT200 in the same document.

### 418 3.3.2 EPR Scheme

419 References that are represented using the EPR scheme must be implemented by  
420 using instances of `wsa:EndpointReference` global element declaration [8] as  
421 children of elements of type `sml:ref`. The following example illustrates how the  
422 `EnrolledCourse` reference that references course PHY101 in MIT university can be  
423 represented using the EPR scheme:

```
424
425 <EnrolledCourse xmlns="urn:university">
426     <wsa:EndpointReference
427         xmlns:u="http://www.university.example/schema">
428         <wsa:Address>http://www.university.example</wsa:Address>
429         <wsa:ReferenceParameters>
430             <u:University>
431                 <u:Name>MIT</u:Name>
432             </u:University>
433             <u:Course>
434                 <u:Name>PHY101</u:Name>
435             </u:Course>
436         </wsa:ReferenceParameters>
437     </wsa:EndpointReference>
438 </EnrolledCourse>
```

439 **3.4 Constraints on References**

440 SML supports several attributes for expressing constraints on references. All of  
 441 these attributes (with the sole exception of `sml:acyclic`) can only be specified for  
 442 element declarations of type `sml:ref` or a derived type of `sml:ref`. The  
 443 `sml:acyclic` attribute can only be specified on derived types of `sml:ref`. The  
 444 following table lists the various attributes and elements for constraining references:

445 **Attributes**

Name	Description
<code>sml:acyclic</code>	Supported on <code>sml:ref</code> and its derived types. Specifies that instances of the type can not result in cycles in a model. If this attribute is set to true for a derived type <code>D</code> of <code>sml:ref</code> , then instances of <code>D</code> (including any derived types of <code>D</code> ) can not create any cycles in a model. More precisely, the directed graph whose nodes are documents that contain the source or target elements for instances of <code>D</code> , and whose edges are instances of <code>D</code> (an edge is directed from the document containing the source element to the document containing the target element), must be acyclic
<code>sml:targetElement</code>	Used to constrain the name of the reference's target element. This constraint is violated if the target element is not an instance of the named global element declaration or an element declaration in the substitution group hierarchy whose head is the named global element declaration.
<code>sml:targetType</code>	Used to constrain the type of the reference's target element. This constraint is violated if the type of the target element is not the same as (or a derived type of) the type whose name is specified as the value of this attribute.

446 **3.4.1 `sml:acyclic`**

447 The `sml:acyclic` attribute is only supported on derived types of `sml:ref`. This is a  
 448 boolean attribute and its value can be either `true` or `false`. Let **R** be a derived type  
 449 of `sml:ref`. If `sml:acyclic="true"` is specified for **R**, then **R** is an acyclic reference  
 450 type, i.e., instances of **R** can not create cycles in any model. If  
 451 `sml:acyclic="false"` is specified for **R**, then **R** is a cyclic reference type, and its  
 452 instances may create cycles in models. Note that `sml:ref` is a cyclic reference type  
 453 since `sml:acyclic="false"` is specified for `sml:ref`.

454 A cyclic reference type can be used to derive cyclic or acyclic reference types, but all  
 455 derived types of an acyclic reference type are acyclic. In particular,

- 456 • If **CR** is a cyclic reference type and **D<sub>CR</sub>** is a derived type of **CR**, then **D<sub>CR</sub>** is an  
 457 acyclic reference if `sml:acyclic="true"` is specified for **D<sub>CR</sub>**. Otherwise, **D<sub>CR</sub>**  
 458 is a cyclic reference
- 459 • If **AR** is an acyclic reference type and **D<sub>AR</sub>** is a derived type of **AR**, then  
 460 `sml:acyclic="true"` holds for **D<sub>AR</sub>** even if the `sml:acyclic` attribute is not  
 461 explicitly specified for **D<sub>AR</sub>**. It is an error for **D<sub>AR</sub>** to specify  
 462 `sml:acyclic="false"`

### 463 3.4.2 sml:targetElement

464 The `sml:targetElement` attribute is supported on element declarations whose type  
465 is `sml:ref` or a derived type of `sml:ref`. The value of this attribute must be the  
466 qualified name of some global element declaration. Let  
467 `sml:targetElement="ns:GTE"` for some element declaration **E**. Then each element  
468 instance of **E** must reference an element that is an instance of **ns:GTE** or an instance  
469 of some global element declaration in the substitution group hierarchy whose head is  
470 **ns:GTE**.

471 If a target element constraint is specified for a global element declaration **G** then it  
472 continues to apply to all global element declarations in the substitution group  
473 hierarchy whose head is **G**. However, a global element declaration in **G**'s substitution  
474 group can specify a target element constraint that refines the constraint defined for  
475 **G**. In particular, if `sml:targetElement="ns:GTE"` is specified for **G**, and **S<sub>G</sub>** is a  
476 global element declaration that specifies **G** as the value of its `xs:substitutionGroup`  
477 attribute, then the value of the `sml:targetElement` for **S<sub>G</sub>** must be **ns:GTE** or the  
478 name of a global element declaration in the substitution group whose head is **ns:GTE**.  
479 If `sml:targetElement` is not specified for **S<sub>G</sub>**, then `sml:targetElement="ns:GTE"`  
480 holds for **S<sub>G</sub>**.

481 If the target element constraint is specified for a local element declaration **L** in some  
482 type **B**, then it continues to apply to each element declaration **L<sub>R</sub>** that is a valid  
483 restriction of **L** where **L<sub>R</sub>** is defined in some restricted derived type of **B** (see [2]  
484 <http://www.w3.org/TR/xmlschema-1/#cos-particle-restrict> for XML Schema's  
485 definition of valid restrictions). However, **L<sub>R</sub>** can specify a target element constraint  
486 that refines the constraint defined for **L**. In particular, if  
487 `sml:targetElement="ns:GTE"` is specified for **L**, then the value of the  
488 `sml:targetElement` for **L<sub>R</sub>** must be **ns:GTE** or the name of a global element  
489 declaration in the substitution group hierarchy whose head is **ns:GTE**. If  
490 `sml:targetElement` is not specified for **L<sub>R</sub>**, then `sml:targetElement="ns:GTE"`  
491 holds for **L<sub>R</sub>**.

### 492 3.4.3 sml:targetType

493 The `sml:targetType` attribute is supported on element declarations whose type is  
494 `sml:ref` or a derived type of `sml:ref`. The value of this attribute must be the  
495 qualified name of some type declaration. Let `sml:targetType="ns:T"` for some  
496 element declaration **E**. Then each element instance of **E** must reference an element  
497 whose type is **ns:T** or a derived type of **ns:T**.

498 If a target type constraint is specified for a global element declaration **G** then it  
499 continues to apply to all global element declarations in the substitution group  
500 hierarchy whose head is **G**. However, a global element declaration in **G**'s substitution  
501 group can specify a target type constraint that refines the constraint defined for **G**.  
502 In particular, if `sml:targetType="ns:T"` is specified for **G**, and **S<sub>G</sub>** is a global  
503 element declaration that specifies **G** as the value of its `xs:substitutionGroup`  
504 attribute, then the value of the `sml:targetType` for **S<sub>G</sub>** must either be **ns:T** or the  
505 name of some derived type of **ns:T**. If `sml:targetType` is not specified for **S<sub>G</sub>**, then  
506 `sml:targetType="ns:T"` holds for **S<sub>G</sub>**.

507 If the target type constraint is specified for a local element declaration **L** in some  
508 type **B**, then it continues to apply to each element declaration **L<sub>R</sub>** that is a valid  
509 restriction of **L** where **L<sub>R</sub>** is defined in some restricted derived type of **B**. However, **L<sub>R</sub>**  
510 can specify a target type constraint that refines the constraint defined for **L**. In  
511 particular, if `sml:targetType="ns:T"` is specified for **L**, then the value of the

512 `sml:targetType` for  $L_R$  must be `ns:T` or the name of some derived type of `ns:T`. If  
 513 `sml:targetType` is not specified for  $L_R$ , then `sml:targetType="ns:T"` holds for  $L_R$ .

### 514 3.5 Identity Constraints

515 XML schema supports the definition of key, unique, and key reference constraints  
 516 through `xs:key`, `xs:unique`, and `xs:keyref` elements. However, the scope of these  
 517 constraints is restricted to a single document. SML extends the scope of these  
 518 constraints to multiple documents by allowing these constraints to traverse inter-  
 519 document references.

520 SML supports the following elements for defining uniqueness constraints across  
 521 references:

Name	Description
<code>sml:key</code>	Similar to <code>xs:key</code> except that the selector and field XPath expression can use <code>smlfn:deref</code> function
<code>sml:unique</code>	Similar to <code>xs:unique</code> except that the selector and field XPath expression can use <code>smlfn:deref</code> function
<code>sml:keyref</code>	Similar to <code>xs:keyref</code> except that the selector and field XPath expression can use <code>smlfn:deref</code> function

522 The syntax and semantics of the above elements are the same as that for the  
 523 corresponding elements in XML schema, except for the following:

- 524 • These three elements are only supported in the `xs:annotation/xs:appinfo`  
 525 element for element declarations (both global and local). They can not be a  
 526 child of an `xs:element` element
- 527 • The value of the `xpath` attribute of the `sml:selector` and `sml:field`  
 528 elements (which are child elements of these three elements) can contain the  
 529 `smlfn:deref` extension function

- 530 • The selector XPath expression must conform to the following extended BNF

```
531 Selector ::= Path ( '|' Path)*
532 Path ::= ( './../')? Step ( '/' Step)* | DerefExpr
533 DerefExpr ::= 'deref(' Step (/Step)* ')' ( '/' Step)* |
534             'deref(' DerefExpr ')' (/Step)*
535 Step ::= '.' | NameTest
536 NameTest ::= QName | '*' | NCName ':' '*'
```

- 537 • The field XPath expression must conform to the BNF given above for the  
 538 selector XPath expression with the following modification

```
539 Selector ::= Path
540 Path ::= ( './../')? ( Step '/' )* ( Step | @NameTest ) |
541         DerefExpr ( '/' @NameTest)?
```

542 A key or uniqueness constraint expressed using `sml:key`, `sml:unique`, or  
 543 `sml:keyref` is applicable to all element instances of its ancestor element declaration,  
 544 i.e., the element that is the parent of the `xs:annotation/xs:appinfo` element which  
 545 holds the `sml:key`, `sml:unique`, or `sml:keyref` element.

#### 546 3.5.1 University Example

547 The following example will be used to illustrate the `sml:key`, `sml:unique`, and  
 548 `sml:keyref` constraints across references.

```

549
550 <xs:element name="Student"
551         type="sml:ref"
552         sml:targetType="tns:StudentType" />
553
554 <xs:element name="Course"
555         type="sml:ref"
556         sml:targetType="tns:CourseType" />
557
558 <xs:complexType name="UniversityType">
559     <xs:sequence>
560         <xs:element name="Name" type="xs:string" />
561         <xs:element name="Students" minOccurs="0">
562             <xs:complexType>
563                 <xs:sequence>
564                     <xs:element ref="tns:Student" maxOccurs="unbounded" />
565                 </xs:sequence>
566             </xs:complexType>
567         </xs:element>
568         <xs:element name="Courses" minOccurs="0">
569             <xs:complexType>
570                 <xs:sequence>
571                     <xs:element ref="tns:Course" maxOccurs="unbounded" />
572                 </xs:sequence>
573             </xs:complexType>
574         </xs:element>
575     </xs:sequence>
576 </xs:complexType>
577
578 <xs:element name="EnrolledStudent"
579         type="sml:ref"
580         sml:targetType="tns:StudentType" />
581
582 <xs:element name="EnrolledCourse"
583         type="sml:ref"
584         sml:targetType="tns:CourseType" />
585
586 <xs:complexType name="StudentType">
587     <xs:sequence>
588         <xs:element name="ID" type="xs:string" />
589         <xs:element name="SSN" type="xs:string" minOccurs="0" />
590         <xs:element name="Name" type="xs:string" />
591         <xs:element name="EnrolledCourses" minOccurs="0">
592             <xs:complexType>
593                 <xs:sequence>
594                     <xs:element ref="tns:EnrolledCourse"
595                         maxOccurs="unbounded" />
596                 </xs:sequence>
597             </xs:complexType>
598         </xs:element>
599     </xs:sequence>
600 </xs:complexType>
601

```

```

602     <xs:complexType name="CourseType">
603         <xs:sequence>
604             <xs:element name="Name" type="xs:string"/>
605             <xs:element name="EnrolledStudents" minOccurs="0">
606                 <xs:complexType>
607                     <xs:sequence>
608                         <xs:element ref="tns:EnrolledStudent"
609                             maxOccurs="unbounded"/>
610                     </xs:sequence>
611                 </xs:complexType>
612             </xs:element>
613         </xs:sequence>
614     </xs:complexType>

```

### 615 3.5.2 sml:key and sml:unique

616 XML schema supports key and uniqueness constraints through `xs:key` and  
617 `xs:unique`, but these constraints can only be specified within a single XML  
618 document. The `sml:key` and `sml:unique` elements support the specification of key  
619 and uniqueness constraints across documents. We'll use the [UniversityType](#)  
620 definition to illustrate this concept. It is reasonable to expect that each student in a  
621 university must have a unique identity, and this identity must be specified. This can  
622 be expressed as follows:

```

623     <xs:element name="University" type="tns:UniversityType">
624         <xs:annotation>
625             <xs:appinfo>
626                 <sml:key name="StudentIDisKey">
627                     <sml:selector xpath="smlfn:deref(tns:Students/tns:Student)/tns:ID"/>
628                     <sml:field xpath="."/>
629                 </sml:key>
630             </xs:appinfo>
631         </xs:annotation>
632     </xs:element>

```

633 The `sml:key` and `sml:unique` constraints are similar but not the same. `sml:key`  
634 requires that the specified fields must be present in instance documents and have  
635 unique values, whereas `sml:unique` simply requires the specified fields to have  
636 unique values but does not require them to be present in instance documents. Thus  
637 keys imply uniqueness, but uniqueness does not imply keys. For example, students  
638 in a university must have a unique social security numbers, but the university may  
639 have foreign students who do not possess this number. This constraint can be  
640 specified as follows:

```

641     <xs:element name="University" type="tns:UniversityType">
642         <xs:annotation>
643             <xs:appinfo>
644                 <sml:unique name="StudentSSNisUnique">
645                     <sml:selector xpath="smlfn:deref(tns:Students/tns:Student)"/>
646                     <sml:field xpath="tns:SSN"/>
647                 </sml:unique>
648             </xs:appinfo>
649         </xs:annotation>
650     </xs:element>

```

651

652 The `sml:key` and `sml:unique` constraint are always specified in the context of a  
653 scoping element. In the above example, the `University` element is the context for  
654 the key and unique constraints.

### 655 3.5.3 `sml:keyref`

656 XML schema supports key references through `xs:keyref` to ensure that one set of  
657 values is a subset of another set of values within an XML document. Such constraints  
658 are similar to foreign keys in relational databases. Key references in XML schema are  
659 only supported within a single XML document. The `sml:keyref` element allows key  
660 references to be specified across XML documents, and can be used to scope  
661 references to point to elements within a valid range. The following example uses  
662 `sml:keyref` to capture the requirement that courses in a university can only enroll  
663 students from the same university:

```
664     <xs:element name="University" type="tns:UniversityType">
665         <xs:annotation>
666             <xs:appinfo>
667                 <sml:key name="StudentIDisKey">
668                     <sml:selector xpath="smlfn:deref(tns:Students/tns:Student)"/>
669                     <sml:field xpath="tns:ID"/>
670                 </sml:key>
671                 <sml:keyref name="CourseStudents" refer="StudentIDisKey">
672                     <sml:selector xpath="smlfn:deref(
673                         smlfn:deref(tns:Courses/tns:Course)/
674                         tns:EnrolledStudents/tns:EnrolledStudent)"/>
675                     <sml:field xpath="tns:ID"/>
676                 </sml:keyref>
677             </xs:appinfo>
678         </xs:annotation>
679     </xs:element>
```

680 The above constraint specifies that for a university, the set of IDs of students  
681 enrolled in courses is a subset of the set of IDs of students in a university. In  
682 particular, the `selector` and `field` elements in `StudentIDisKey` key constraint  
683 identify the set of IDs of students in a university, and the `selector` and `field`  
684 elements in `CourseStudents` key reference constraint identify the set of IDs of  
685 students enrolled in courses.

## 686 4. Rules

687 XML Schema supports a number of built-in grammar-based constraints but it does  
688 not support a language for defining arbitrary rules for constraining the structure and  
689 content of documents. Schematron [4] is a proposed schema for defining assertions  
690 concerning a set of XML documents. Schematron has been submitted to ISO and IEC  
691 for standardization (ISO/IEC FDIS 19757-3), and is currently in Final Draft International  
692 Standard Approval stage (see  
693 <http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40833>  
694 [&scopelist=PROGRAMME](#) for details). SML uses a profile of the Schematron schema  
695 to add support for user-defined constraints. SML uses XPath1.0, augmented with  
696 SML-specific XPath extension functions, as its constraint language. This section  
697 assumes that the reader is familiar with Schematron concepts; the proposed draft  
698 standard for Schematron is documented in [4] and [5,6] are good tutorials on an  
699 older version of Schematron.

700 User-defined constraints can be specified using the `sch:assert` and `sch:report`  
701 elements from Schematron. The following example uses `sch:assert` elements to  
702 specify two constraints:

- 703 • An IPv4 address must have four bytes
- 704 • An IPv6 address must have sixteen bytes

```
705 <xs:simpleType name="IPAddressVersionType">
706   <xs:restriction base="xs:string" >
707     <xs:enumeration value="V4" />
708     <xs:enumeration value="V6" />
709   </xs:restriction>
710 </xs:simpleType>
711 <xs:complexType name="IPAddress">
712   <xs:annotation>
713     <xs:appinfo>
714       <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
715         <sch:ns prefix="tns" uri="urn:IPAddress" />
716         <sch:pattern id="Length">
717           <sch:rule context=".">
718             <sch:assert test="tns:version != 'V4' or count(tns:address) = 4">
719               A v4 IP address must have 4 bytes.
720             </sch:assert>
721             <sch:assert test="tns:version != 'V6' or count(tns:address) = 16">
722               A v6 IP address must have 16 bytes.
723             </sch:assert>
724           </sch:rule>
725         </sch:pattern>
726       </sch:schema>
727     </xs:appinfo>
728   </xs:annotation>
729   <xs:sequence>
730     <xs:element name="version" type="tns:IPAddressVersionType" />
731     <xs:element name="address" type="xs:byte" minOccurs="4" maxOccurs="16" />
732   </xs:sequence>
733 </xs:complexType>
```

734 A Schematron pattern embedded in the `xs:annotation/xs:appinfo` element for a  
735 complex type definition or an element declaration is applicable to all instances of the  
736 complex type or element. In the above example, the pattern `Length` is applicable for  
737 all elements whose type is `IPAddress` or a derived type of `IPAddress`. A pattern can  
738 have one or more rules, and each rule specifies a context expression using the  
739 context attribute. The value of the context attribute is an XPath expression that is  
740 evaluated in the context of each applicable element, and results in an element node  
741 set for which the assert and report test expressions defined in the rule are evaluated.  
742 In the above example `context="."`, therefore the two assert expressions are  
743 evaluated in the context of each applicable element, i.e., each element of type  
744 `IPAddress`. The test expression for an assert is a boolean expression, and the  
745 assert is violated (or fires) if its test expression evaluates to false. For example,  
746 the following XML document violates the `assert` that requires an IPv6 address to  
747 have sixteen address bytes

```
748 <myIPAddress xmlns="urn:IPAddress">
749   <version>v6</version>
750   <address>100</address>
751   <address>200</address>
752   <address>10</address>
753   <address>1</address>
754   <address>10</address>
```

```
755     <address>1</address>
756 </myIPAddress>
```

757 In general, a rule element can include multiple assert and report elements. A  
758 report also specifies a test expression, just like an assert. However, a report is  
759 violated (or fires) if its test expression evaluates to true. Thus, an assert can be  
760 converted to a report by simply negating its test expression. The following example  
761 uses report elements to represent the IP address constraints of the previous  
762 example:

```
763     <xs:simpleType name="IPAddressVersionType">
764         <xs:restriction base="xs:string" sml:numericType="xs:int">
765             <xs:enumeration value="V4" sml:numericValue="0" />
766             <xs:enumeration value="V6" sml:numericValue="1" />
767         </xs:restriction>
768     </xs:simpleType>
769     <xs:complexType name="IPAddress">
770         <xs:annotation>
771             <xs:appinfo>
772                 <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
773                     <sch:ns prefix="tns" uri="urn:IPAddress" />
774                     <sch:pattern id="Length">
775                         <sch:rule context=".">
776                             <sch:report test="tns:version = 'V4' and count(tns:address) != 4"
777                                 >
778                                 A v4 IP address must have 4 bytes.
779                             </sch:report>
780                             <sch:report test="tns:version = 'V6' and count(tns:address) != 16"
781                                 >
782                                 A v6 IP address must have 16 bytes.
783                             </sch:report>
784                         </sch:rule>
785                     </sch:pattern>
786                 </sch:schema>
787             </xs:appinfo>
788         </xs:annotation>
789         <xs:sequence>
790             <xs:element name="version" type="tns:IPAddressVersionType" />
791             <xs:element name="address" type="xs:byte" minOccurs="4" maxOccurs="16" />
792         </xs:sequence>
793     </xs:complexType>
```

794 If an assert or report is violated, then the violation must be reported during model  
795 validation together with the specified message. Model validation must evaluate each  
796 Schematron pattern for all of its applicable elements contained in the model.

797 The message can include substitution strings based on XPath expressions. These can  
798 be specified using the sch:value-of element. The following example uses  
799 sch:value-of to include the number of specified address bytes in the message:

```
800     <sch:assert test="tns:version != 'v4' or count(tns:address) = 4">
801         A v4 IP address must have 4 bytes instead of the specified
802         <sch:value-of select="string(count(tns:address))"/> bytes.
803     </sch:assert>
```

804 In addition to being embedded in complex type definitions, constraints can also be  
805 embedded in global-element declarations. Such constraints are evaluated for each  
806 instance element corresponding to the global-element definition. Consider the  
807 following example:

```

808
809 <xs:element name="StrictUniversity" type="tns:UniversityType">
810   <xs:annotation>
811     <xs:appinfo>
812       <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
813         <sch:ns prefix="u" uri="urn:university" />
814         <sch:ns prefix="smlfn"
815           uri="http://schemas.serviceml.org/smlfn/query/2006/07"/>
816         <sch:pattern id="StudentPattern">
817           <sch:rule context="smlfn:deref(u:Students/u:Student)">
818             <sch:assert test="starts-with(u:ID,'99')">
819               The specified ID <sch:value-of select="string(u:ID)"/>
820               does not begin with 99
821             </sch:assert>
822             <sch:assert test="count(u:Course/u:Courses)>0">
823               The student <sch:value-of select="string(u:ID)"/> must be enrolled
824               in at least one course
825             </sch:assert>
826           </sch:rule>
827         </sch:pattern>
828       </sch:schema>
829     </xs:appinfo>
830   </xs:annotation>
831 </xs:element>

```

832 The constraints defined in StudentPattern are applicable to all element instances of  
833 the StrictUniversity global element definition. For each StrictUniversity  
834 element, the XPath expression specified as the value of the context attribute is  
835 evaluated to return a node set, and the test expressions for the two asserts are  
836 evaluated for each node in this node set. The context expression for the rule returns  
837 a node set consisting of all Student elements referenced by an instance of  
838 StrictUniversity, and the test expressions for the two asserts are evaluated for  
839 each element node in this node set. Thus, these two asserts verify the following  
840 conditions for each instance of StrictUniversity

- 841 • The ID of each student must begin with '99'
- 842 • Each student must be enrolled in at least one course

843 An SML validator is free to provide implementation-specific mechanisms to support  
844 the targeting of constraints that are authored in a separate document, i.e., not  
845 embedded in schema definitions, to a set of instance documents. The following  
846 example shows the constraints for StrictUniversity expressed in a separate  
847 document:

```

848 <?xml version="1.0" encoding="utf-8" ?>
849 <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
850   <sch:ns prefix="u" uri="urn:university" />
851   <sch:ns prefix="smlfn" uri="http://schemas.serviceml.org/smlfn/query/2006/07"
852     />
853   <sch:pattern id="StudentPattern">
854     <sch:rule context="smlfn:deref(u:Students/u:Student)">
855       <sch:assert test="starts-with(u:ID,'99')">
856         The specified ID <sch:value-of select="string(u:ID)"/>
857         does not begin with 99
858       </sch:assert>
859       <sch:assert test="count(u:Course/u:Courses)>0">
860         The student <sch:value-of select="string(u:ID)"/> must be enrolled
861         in at least one course
862       </sch:assert>
863     </sch:rule>
864   </sch:pattern>
865 </sch:schema>

```

866

867 The binding of the `StudentPattern` pattern to instances of `StrictUniversity`  
868 element is implementation dependent and hence outside the scope of this  
869 specification.

## 870 **4.1 Schematron Profile**

871 SML supports a conforming profile of Schematron. All elements and attributes are  
872 supported.

### 873 **4.1.1 Limited Support**

874 If the `queryBinding` attribute is specified, then its value must be set to "xpath1.0"

## 875 **5. Model Validation**

876 Model validation is the process of examining each document in a model and verifying  
877 that this document is valid with respect to the model's genic documents, i.e., each  
878 document satisfies the schemas and rules defined in the model's genic documents.  
879 Validation is required to report all schema and rule violations in a model. In  
880 particular, validation must continue, even if schema or rule violations are found, until  
881 all applicable schemas and rules are evaluated for each document in the model.

### 882 **5.1 Schematron Phase**

883 A phase in schematron can be used to define a collection of patterns. A schematron  
884 processor can optionally evaluate only rules within a specific phase. For model  
885 validation, rule evaluation happens on the `#ALL` phase, implying that every rule in  
886 every pattern is evaluated.

## 887 **6. SML Extension Reference**

888 This section is a reference of the SML extensions to XML Schema and XPath 1.0.

### 889 **6.1 Types**

#### 890 **6.1.1 `sml:ref`**

891 A complex type representing a reference to an element.

892

```
893     <xs:complexType name="ref"  
894                 sml:acyclic="false"  
895                 final="extension">  
896         <xs:sequence>  
897             <xs:any namespace="##any" minOccurs="0"  
898                 maxOccurs="unbounded"  
899                 processContents="lax"/>  
900         </xs:sequence>  
901         <xs:anyAttribute namespace="##any" processContents="lax"/>  
902     </xs:complexType>
```

903

904 No specific scheme is mandated for representing references, and a model validator is  
905 free to choose any suitable scheme. However, each reference value must resolve to  
906 a single element. `sml:ref` can only be used with element declarations; it is not  
907 supported on attribute declarations.

## 908 6.2 Attributes

### 909 6.2.1 sml:acyclic

910 Used to specify that a derived type of `sml:ref` is acyclic, i.e., its instances do not  
911 create any cycles in a model.

```
912     <xs:attribute name="acyclic" type="xs:boolean"/>
```

913 If this attribute is set to true for a derived type  $D$  of `sml:ref`, then instances of  $D$   
914 (including any derived types of  $D$ ) can not create any cycles in a model. More  
915 precisely, the directed graph whose nodes are documents that contain the source or  
916 target elements for instances of  $D$ , and whose edges are instances of  $D$  (an edge is  
917 directed from the document containing the source element to the document  
918 containing the target element), must be acyclic. A model is invalid if its documents  
919 result in a cyclic graph using instances of  $D$ . In the following example, `Hostref` is a  
920 restricted derived type of `sml:ref` and its instances can not create any cycles:

```
921  
922     <xs:complexType name="Hostref" sml:acyclic="true">  
923       <xs:complexContent>  
924         <xs:restriction base="sml:ref"/>  
925       </xs:complexContent>  
926     </xs:complexType>
```

927  
928 If the `sml:acyclic` attribute is not specified or set to false for a derived type of  
929 `sml:ref`, then instances of this reference type may create cycles in a model. Note  
930 that `sml:acyclic` is specified as "false" for `sml:ref`; hence its instances are  
931 allowed to create cycles in a model.

### 932 6.2.2 sml:targetElement

933 A `QName` representing the name of a referenced element

```
934     <xs:attribute name="targetElement" type="xs:QName"/>
```

935 `sml:targetElement` is supported as an attribute for element declarations whose  
936 type is `sml:ref` or a type derived by restriction from `sml:ref`. The value of this  
937 attribute must be the name of some global element declaration. Let  
938 `sml:targetElement="ns:GTE"` for some element declaration  $E$ . Then each element  
939 instance of  $E$  must target an element that is an instance of `ns:GTE` or an instance of  
940 some global element declaration in the substitution group hierarchy whose head is  
941 `ns:GTE`.

942 In the following example, the element referenced by instances of `HostOS` must be  
943 instances of `win:Windows`

```
944     <xs:element name="HostOS" type="sml:ref"  
945       sml:targetElement="win:Windows"  
946       minOccurs="0"/>
```

947 A model is invalid if its documents violate one/more `sml:targetElement` constraints.

### 948 6.2.3 sml:targetType

949 A `QName` representing the type of a referenced element

```
950     <xs:attribute name="targetType" type="xs:QName">
```

951

952 `sml:targetType` is supported as an attribute for element declarations whose type is  
953 `sml:ref` or a type derived by restriction from `sml:ref`. If the value of this attribute  
954 is specified as  $T$ , then the type of the referenced element must either be  $T$  or a

955 derived type of T. In the following example, the type of the element referenced by  
956 the `OperatingSystem` element must be `ibm:LinuxType` or its derived type

```
957     <xs:element name="OperatingSystem" type="sml:ref"  
958             sml:targetType="ibm:LinuxType"  
959             minOccurs="0"/>
```

960 A model is invalid if its documents violate one/more `sml:targetType` constraints.

## 961 6.2.4 `sml:uri`

962 Specifies a reference in URI scheme.

```
963     <xs:attribute name="uri" type="xs:anyURI"/>
```

964 This attribute is only allowed on element declarations whose type is `sml:ref` or a  
965 derived type of `sml:ref`.

966

## 967 6.3 Elements

968

### 969 6.3.1 `sml:key`

970 This element is used to specify a key constraint in some scope. The semantics are  
971 essentially the same as that for `xs:key` but `sml:key` can also be used to specify key  
972 constraints on other documents, i.e., the `sml:selector` child element of `sml:key`  
973 can contain `deref` functions to resolve elements in another document.

```
974     <xs:element name="key" type="sml:keybase"/>
```

975 `sml:key` is supported in the `appinfo` of an `xs:element`.

### 976 6.3.2 `sml:unique`

977 This element is used to specify a uniqueness constraint in some scope. The  
978 semantics are essentially the same as that for `xs:unique` but `sml:unique` can also  
979 be used to specify uniqueness constraints on other documents, i.e., the  
980 `sml:selector` child element of `sml:unique` can contain `deref` functions to resolve  
981 elements in another document.

```
982     <xs:element name="unique" type="sml:keybase"/>
```

983 `sml:unique` is supported in the `appinfo` of an `xs:element`.

### 984 6.3.3 `sml:keyref`

985 Applies a constraint in the context of the containing `xs:element` that scopes the range  
986 of a nested document reference.

987

```
988     <xs:element name="keyref">  
989         <xs:complexType>  
990             <xs:complexContent>  
991                 <xs:extension base="sml:keybase">  
992                     <xs:attribute name="refer" type="xs:QName" use="required"/>  
993                 </xs:extension>  
994             </xs:complexContent>  
995         </xs:complexType>  
996     </xs:element>>
```

997

998

999 `sml:keyref` is supported in the `appinfo` of an `xs:element`.

## 1000 **6.4 XPath functions**

### 1001 **6.4.1 smlfn:deref**

1002 `node-set deref(node-set)`

1003 This function takes a node-set of elements whose type must be `sml:ref` or a type  
1004 derived by restriction from `sml:ref`. The resulting node-set is the set of elements  
1005 that are obtained by resolving (or de-referencing) the input node-set. For example,

1006 `deref(/u:Universities/u:Students/u:Student)`

1007 will resolve the reference in element `Student`. The target of the reference must always be  
1008 an element.

## 1009 **7. Acknowledgements**

1010 Thanks to the following individuals for providing valuable feedback on this  
1011 specification:

1012 Don Box, Ray McCollum, Ted Miller and Jeff Parham (Microsoft)

1013 John Arwe, Chris Ferris, and Sandy Gao (IBM)

1014 Matt Newman and Virginia Smith (HP)

1015 Johan Van De Groenendaal (Intel)

## 1016 **8. References**

1017 **[1]** XML Schema Part 0: Primer (<http://www.w3.org/TR/xmlschema-0>)

1018 **[2]** XML Schema Part 1: Structures Second Edition

1019 (<http://www.w3.org/TR/xmlschema-1>)

1020 **[3]** XML Schema Part 2: Datatypes Second Edition

1021 (<http://www.w3.org/TR/xmlschema-2>)

1022 **[4]** Document Schema Definition Language (DSDL) – Part 3: Rule-based validation –  
1023 Schematron (<http://www.schematron.com/iso/dsdl-3-fdis.pdf>)

1024 **[5]** An Introduction to Schematron

1025 (<http://www.xml.com/pub/a/2003/11/12/schematron.html>)

1026 **[6]** Improving XML Document Validation with Schematron

1027 (<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxml/html/schematron.asp> )

1029 **[7]** Uniform Resource Identifiers (<http://www.ietf.org/rfc/rfc2396.txt>)

1030 **[8]** Web Services Addressing (<http://www.w3.org/TR/ws-addr-core> )

1031 **[9]** XML Path Language (XPath) Version 1.0 (<http://www.w3.org/TR/xpath>)

1032 **[10]** XPointer (<http://www.w3.org/TR/xptr/>)

1033 **[11]** XPointer `xpointer()` Scheme (<http://www.w3.org/TR/xptr-xpointer/>)

1034 **[12]** Extensible Markup Language (XML) 1.0 (<http://www.w3.org/TR/REC-xml/>)

1035

## 1036 Appendix I – Sample Model

1037 This sample model illustrates the use of the following SML extensions:

- 1038 • Inter-document references
- 1039 • key and keyref constraints
- 1040 • User-defined constraints

```
1041 <?xml version="1.0" encoding="utf-8" ?>
1042 <xs:schema targetNamespace="SampleModel"
1043           elementFormDefault="qualified"
1044           xmlns:tns="SampleModel"
1045           xmlns:sml="http://schemas.serviceml.org/sml/2006/07"
1046           xmlns:sch="http://purl.oclc.org/dsdl/schematron"
1047           xmlns:xs="http://www.w3.org/2001/XMLSchema">
1048
1049   <xs:import namespace="http://schemas.serviceml.org/sml/2006/07"/>
1050
1051   <xs:simpleType name="SecurityLevel">
1052     <xs:restriction base="xs:string">
1053       <xs:enumeration value="Low"/>
1054       <xs:enumeration value="Medium"/>
1055       <xs:enumeration value="High"/>
1056     </xs:restriction>
1057   </xs:simpleType>
1058
1059   <xs:complexType name="Hostref" sml:acyclic="true">
1060     <xs:complexContent>
1061       <xs:restriction base="sml:ref"/>
1062     </xs:complexContent>
1063   </xs:complexType>
1064
1065   <!-- This element represents the host operating system for
1066        an application. Note that the type of the referenced
1067        element must be OperatingSystemType or a derived type
1068        of OperatingSystemType -->
1069   <xs:element name="HostOSRef" type="tns:Hostref"
1070             sml:targetType="tns:OperatingSystemType"/>
1071
1072   <xs:complexType name="ApplicationType">
1073     <xs:sequence>
1074       <xs:element name="Name" type="xs:string"/>
1075       <xs:element name="Vendor" type="xs:string"/>
1076       <xs:element name="Version" type="xs:string"/>
1077       <xs:element ref="tns:HostOSRef" minOccurs="0"/>
1078     </xs:sequence>
1079   </xs:complexType>
```

```

1080
1081 <xs:simpleType name="ProtocolType">
1082   <xs:list>
1083     <xs:simpleType>
1084       <xs:restriction base="xs:string">
1085         <xs:enumeration value="TCP"/>
1086         <xs:enumeration value="UDP"/>
1087         <xs:enumeration value="SMTP"/>
1088         <xs:enumeration value="SNMP"/>
1089       </xs:restriction>
1090     </xs:simpleType>
1091   </xs:list>
1092 </xs:simpleType>
1093
1094 <!-- Note that the type of the element referenced by an
1095      GuestAppRef element can not be an extended derived
1096      type of ApplicationType -->
1097 <xs:element name="GuestAppRef" type="sml:ref"
1098           sml:targetType="tns:ApplicationType"/>
1099
1100 <xs:complexType name="OperatingSystemType">
1101   <xs:sequence>
1102     <xs:element name="Name" type="xs:string"/>
1103     <xs:element name="FirewallEnabled" type="xs:boolean"/>
1104     <xs:element name="Protocol" type="tns:ProtocolType"/>
1105     <!-- The following element represents the applications hosted by
1106          operating system -->
1107     <xs:element name="Applications" minOccurs="0">
1108       <xs:complexType>
1109         <xs:sequence>
1110           <xs:element ref="tns:GuestAppRef" maxOccurs="unbounded"/>
1111         </xs:sequence>
1112       </xs:complexType>
1113     </xs:element>
1114   </xs:sequence>
1115 </xs:complexType>
1116
1117 <xs:element name="OSRef" type="sml:ref"
1118           sml:targetType="tns:OperatingSystemType"/>
1119
1120 <xs:complexType name="WorkstationType">
1121   <xs:sequence>
1122     <xs:element name="Name" type="xs:string"/>
1123     <xs:element ref="tns:OSRef"/>
1124     <xs:element name="Applications" minOccurs="0">
1125       <xs:complexType>
1126         <xs:sequence>
1127           <xs:element ref="tns:GuestAppRef" maxOccurs="unbounded"/>
1128         </xs:sequence>
1129       </xs:complexType>
1130     </xs:element>
1131   </xs:sequence>
1132 </xs:complexType>

```

```

1133
1134 <xs:element name="Workstation" type="tns:WorkstationType">
1135   <xs:annotation>
1136     <xs:appinfo>
1137       <sch:schema>
1138         <sch:ns prefix="sm" uri="SampleModel"/>
1139         <sch:ns prefix="smlfn"
1140           uri="http://schemas.serviceml.org/sml/function/2006/07"/>
1141         <sch:pattern id="OneHostOS">
1142           <!-- The constraints in the following rule are evaluated
1143             For all instances of the Workstation global element-->
1144           <sch:rule context=".">
1145             <!-- define a named variable - MyApplications -
1146               for use in test expression-->
1147             <sch:let name="MyApplications"
1148               value="smlfn:deref(sm:Applications/sm:GuestAppRef)"/>
1149             <sch:assert test=
1150               "count($MyApplications)=
1151                 count($MyApplications/sm:HostOSRef)">
1152               Each application in workstation
1153             <sch:value-of select="string(sm:Name)"/>
1154             must be hosted on an operating system
1155             </sch:assert>
1156           </sch:rule>
1157         </sch:pattern>
1158       </sch:schema>
1159
1160       <!-- In a workstation, (Vendor,Name,Version) is the key for
1161         guest applications -->
1162       <sml:key name="GuestApplicationKey">
1163         <sml:selector
1164           xpath="smlfn:deref(tns:Applications/tns:GuestAppRef)"/>
1165         <sml:field xpath="tns:Vendor"/>
1166         <sml:field xpath="tns:Name"/>
1167         <sml:field xpath="tns:Version"/>
1168       </sml:key>
1169
1170       <!-- In a workstation, Name is the key for operating system -->
1171       <sml:key name="OSKey">
1172         <sml:selector xpath="smlfn:deref(tns:OSRef)"/>
1173         <sml:field xpath="tns:Name"/>
1174       </sml:key>
1175
1176       <!-- In a workstation, the applications hosted by the
1177         referenced operatinsystem must be a subset of the
1178         applications in the workstation -->
1179       <sml:keyref name="OSGuestApplication"
1180         refer="GuestApplicationKey">
1181         <sml:selector xpath=
1182           "smlfn:deref(tns:OSRef)/tns:Applications/tns:GuestAppRef"
1183         />
1184         <sml:field xpath="tns:Vendor"/>
1185         <sml:field xpath="tns:Name"/>
1186         <sml:field xpath="tns:Version"/>
1187       </sml:keyref>
1188
1189

```

```

1190     <!-- In a workstation, the host operating system of guest
1191           applications must be a subset of the operating system in
1192           the workstation -->
1193     <sml:keyref name="ApplicationHostOS" refer="OSKey">
1194       <sml:selector xpath=
1195         "smlfn:deref(tns:Applications/tns:GuestAppRef)/tns:HostOSRef"
1196       />
1197
1198       <sml:field xpath="tns:Name"/>
1199     </sml:keyref>
1200
1201   </xs:appinfo>
1202 </xs:annotation>
1203 </xs:element>
1204
1205 <xs:element name="SecureWorkstation" type="tns:WorkstationType">
1206   <xs:annotation>
1207     <xs:appinfo>
1208       <sch:schema>
1209         <sch:ns prefix="sm" uri="SampleModel" />
1210         <sch:ns prefix="smlfn"
1211           uri="http://schemas.serviceml.org/sml/function/2006/07"
1212         />
1213         <sch:pattern id="SecureApplication">
1214           <sch:rule
1215             context="smlfn:deref(sm:Applications/sm:Application)">
1216             <sch:report test="sm:SecurityLevel!='High'">
1217               Application <sch:value-of select="string(sm:Name)"/>
1218               from <sch:value-of select="string(sm:Vendor)"/>
1219               does not have high security level
1220             </sch:report>
1221             <sch:assert test="sm:Vendor='TrustedVendor'">
1222               A secure workstation can only contain
1223               applications from TrustedVendor
1224             </sch:assert>
1225           </sch:rule>
1226         </sch:pattern>
1227       </sch:schema>
1228     </xs:appinfo>
1229   </xs:annotation>
1230 </xs:element>
1231
1232 </xs:schema>
1233

```

1234

1235 **Appendix II – Complexity of Supporting targetElement**  
1236 **and targetType on Local Element Declarations**

1237 This appendix describes the complexity of supporting `sml:targetElement` and  
1238 `sml:targetType` on local elements. The complexity occurs due to derivation by  
1239 restriction, and the necessity to completely (re-)specify the elements in the derived  
1240 type. In order to propagate an `sml:targetElement` or `sml:targetType` constraint, it  
1241 is necessary to connect the elements in the derived type with those from the  
1242 restricted (super-) type. However, this level of support is not provided by most XML  
1243 Schema frameworks. If an XML Schema framework does not provide this support,  
1244 then an SML validator that uses this framework can still support these constraints on  
1245 local elements by duplicating large parts of XML Schema's compilation logic. This  
1246 may substantially increase the effort required to implement an SML validator. An  
1247 SML validator may prefer to support these, constraints on global elements only  
1248 (which requires a simpler analysis across substitution groups) until its underlying  
1249 XML Schema framework provides the support needed to analyze local elements  
1250 across derivation-by-restriction type hierarchies.