

1 **Service Modeling Language**

2 **Draft Specification**

3 **Version 1.0, 28 February 2007**

4 **Authors**

5 John Arwe, IBM

6 Jordan Boucher, Sun

7 Pratul Dublish, Microsoft

8 Zulah Eckert, BEA

9 Dave Ehnebuske, IBM

10 Jon Hass, Dell

11 Steve Jerman, Cisco

12 Heather Kreger, IBM

13 Vincent Kowalski, BMC

14 Milan Milenkovic, Intel

15 Bryan Murray, HP

16 Phil Prasek, HP

17 Junaid Saiyed, EMC

18 Harm Sluiman, IBM

19 Bassam Tabbara, Microsoft

20 Vijay Tewari, Intel

21 William Vambenepe, HP

22 Marv Waschke, CA

23 Andrea Westerinen, Microsoft

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

29
30 1. A link or URL to the Service Modeling Language Specification at this location:

31 <http://www.serviceml.org/SML-200702.pdf>

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

38
39 THE SERVICE MODELING LANGUAGE SPECIFICATION IS PROVIDED "AS IS," AND THE
40 AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED,
41 INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
42 PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE
43 SERVICE MODELING LANGUAGE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE;
44 NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD
45 PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

46 THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL
47 OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR
48 DISTRIBUTION OF THE SERVICE MODELING LANGUAGE SPECIFICATION.
49

50 The name and trademarks of the Authors may NOT be used in any manner, including advertising
51 or publicity pertaining to the Service Modeling Language Specification or its contents without
52 specific, written prior permission. Title to copyright in the Service Modeling Language
53 Specification will at all times remain with the Authors.
54

55 No other rights are granted by implication, estoppel or otherwise.
56

57 **Abstract**

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

61 **Status**

62 This specification is the first draft of a work in progress. It is being published to
63 solicit feedback. A feedback agreement is required before the working group can
64 accept feedback. Please contact sml-feedback@external.cisco.com for details.

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

68

69	Table of Contents	
70	Service Modeling Language	1
71	Abstract	3
72	Status.....	3
73	Table of Contents	4
74	1. Introduction	6
75	2. Notations and Terminology	7
76	2.1 Notational Conventions.....	7
77	2.2 Terminology	7
78	2.3 XML Namespaces	7
79	3. Schemas.....	8
80	3.1 XML Schema Profile.....	8
81	3.1.1 <xs:redefine>	8
82	3.1.2 Unqualified Local Elements	9
83	3.1.3 targetNamespace on <xs:schema>	9
84	3.2 References.....	9
85	3.2.1 Reference Semantics	13
86	3.2.1.1 At Most One Target	13
87	3.2.1.2 Multiple References	13
88	3.2.1.3 Empty or Null References.....	13
89	3.2.1.4 deref() XPath Extension Function	14
90	3.3 Reference Schemes.....	14
91	3.3.1 URI Scheme	14
92	3.3.1.1 Fragment Identifier	15
93	3.3.2 EPR Scheme	17
94	3.4 Constraints on References.....	17
95	3.4.1 sml:acyclic	18
96	3.4.2 Constraints on Targets	18
97	3.4.2.1 sml:targetElement	19
98	3.4.2.2 sml:targetRequired	20
99	3.4.2.3 sml:targetType	20
100	3.5 Identity Constraints	21
101	3.5.1 University Example.....	22
102	3.5.2 sml:key and sml:unique.....	24
103	3.5.3 sml:keyref.....	25
104	4. Rules	25
105	4.1 Localization of Error Messages	30
106	4.2 Schematron Profile.....	30
107	4.2.1 Limited Support	30
108	5. Structured XML output from Schematron Rules.....	30
109	5.1 smlerr:output	31
110	5.1.1 smlerr:outputids	31

111	5.1.2 smlerr:attributeNode	31
112	5.1.3 smlerr:errorData	32
113	5.1.4 Semantics	32
114	5.1.5 Examples	32
115	6. Model Validation.....	35
116	6.1 Schematron Phase	35
117	7. SML Extension Reference	35
118	7.1 Types	35
119	7.1.1 sml:refType	35
120	7.2 Attributes	36
121	7.2.1 sml:acyclic	36
122	7.2.2 sml:ref.....	36
123	7.2.3 sml:targetElement.....	36
124	7.2.4 sml:targetRequired.....	37
125	7.2.5 sml:targetType	37
126	7.3 Elements	37
127	7.3.1 sml:key.....	37
128	7.3.2 sml:keyref.....	38
129	7.3.3 sml:unique	38
130	7.3.4 sml:uri.....	38
131	7.4 XPath functions	38
132	7.4.1 smlfn:deref.....	38
133	8. Open Issues	38
134	9. Acknowledgements	39
135	10. References	39
136	Appendix I – Normative SML Schema.....	41
137	Appendix II – Normative SML Error Schema	45
138	Appendix III – Sample Model.....	47
139		
140		

141

142 **1. Introduction**

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

- 148 1. Models focus on capturing all **invariant aspects** of a service/system that
149 must be maintained for the service/system to be functional.

- 150 2. Models are units of **communication and collaboration** between designers,
151 implementers, operators, and users; and can easily be shared, tracked, and
152 revision controlled. This is important because complex services are often built
153 and maintained by a variety of people playing different roles.

- 154 3. Models drive **modularity, re-use, and standardization**. Most real-world
155 complex services and systems are composed of sufficiently complex parts.
156 Re-use and standardization of services/systems and their parts is a key factor
157 in reducing overall production and operation cost and in increasing reliability.

- 158 4. Models represent a powerful mechanism for **validating changes** *before*
159 applying the changes to a service/system. Also, when changes happen in a
160 running service/system, they can be validated against the intended state
161 described in the model. The actual service/system and its model together
162 enable a *self-healing service/system* – the ultimate objective. *Models of a*
163 *service/system must necessarily stay decoupled from the live service/system*
164 *to create the control loop*

- 165 5. Models enable increased **automation** of management tasks. Automation
166 facilities exposed by the majority of IT services/systems today could be
167 driven by software – not people – for reliable initial realization of a
168 service/system as well as for ongoing lifecycle management.

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

- 174 1. **Schemas** – these are constraints on the structure and content of the
175 documents in a model. SML uses a profile of XML Schema 1.0 [2,3] as the
176 schema language. SML also defines a set of extensions to XML Schema to
177 support inter-document references.

- 178 2. **Rules** – are Boolean expressions that constrain the structure and content of
179 documents in a model. SML uses a profile of Schematron [4,5,6] and XPath
180 1.0 [9] for rules.

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

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

187 **2. Notations and Terminology**

188 **2.1 Notational Conventions**

189 In this document, the keywords "MUST", "MUST NOT", "REQUIRED", "SHALL",
190 "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and
191 "OPTIONAL" are to be interpreted as described in RFC 2119 [13].

192 **2.2 Terminology**

193 Document

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

195 Model

196 A set of inter-related documents that describe an IT service or system. Each
197 model consists of two disjoint subsets of documents –definition documents
198 and instance documents.

199 Rule

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

202 Model Definition

203 The subset of documents in a model that describes the schemas and rules
204 that govern the structure and content of the model's documents. This
205 specification defines two types of model-definition documents - XML Schema
206 documents that conform to SML's profile of XML Schema and rule documents
207 that conform to SML's profile of Schematron – but permits implementations to
208 define other types of model definition documents. Such other types of model
209 definition documents do not play any role in SML model validation.

210 Model Instance

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

213 Model Validation

214 The process of verifying that all documents in a model are valid with respect
215 to the model's definition documents.

216 Model Validator

217 An embodiment capable of performing model validation

218 **2.3 XML Namespaces**

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

221

222

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

Prefix	XML Namespace	Specification(s)
sml	http://schemas.serviceml.org/sml/2007/02	This specification
smlerr	http://schemas.serviceml.org/smlerr/2007/02	This specification
smlfn	http://schemas.serviceml.org/sml/function/2006/07	This specification
wsa	http://www.w3.org/2005/08/addressing	[WS Addressing Core]
xs	http://www.w3.org/2001/XMLSchema	[XML Schema]
sch	http://purl.oclc.org/dsdl/schematron	[Schematron]
xsi	http://www.w3.org/2001/XMLSchema-instance	[Xml Schema Instance]

224 3. Schemas

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

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

- 229 • **References** – XML Schema does not have any support for *inter-document*
230 references, although it does support *intra-document* references through
231 `xs:ID`, `xs:IDREF`, `xs:key` and `xs:keyref`. Inter-document references are
232 fundamental to SML since a document is a unit of versioning. SML extends
233 XML Schema to support inter-document references and a set of constraints on
234 inter-document references.
- 235 • **Rules** – XML Schema does not support a language for defining arbitrary rules
236 on the structure and content of XML documents. SML uses Schematron to
237 express assertions on the structure and content of XML documents.

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

240 3.1 XML Schema Profile

241 SML supports a strict subset of XML Schema 1.0. This section describes the XML
242 Schema features that are not supported or have limited support in SML. A
243 justification is provided for each feature. A model validator MUST reject a model if
244 the model’s definition documents contain one/more XML Schema documents with
245 any of these features.

246 3.1.1 <xs:redefine>

247 `xs:redefine` is not supported in SML and MUST NOT be used in any schema
248 document that is a part of a model’s definition documents.

249 `xs:redefine` is a feature for schema evolution and versioning in XML Schema. This
250 feature enables schema authors to define a new version of a schema component,
251 and completely replace the original schema component with the new version. XML
252 Schema does not guarantee that the new version of the component is compatible

253 with the original component. Thus, it is possible to break existing schema
254 components that depend on the original component.

255 **3.1.2 Unqualified Local Elements**

256 Unqualified local elements are not supported in SML and MUST NOT be used in any
257 schema document that is a part of a model's definition documents.

258 Local element declarations MUST describe elements with qualified names. This can
259 be done, for example, by specifying `elementFormDefault="qualified"` on
260 `<xs:schema>` or by specifying `form="qualified"` on local `<xs:element>`.

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

263 **3.1.3 targetNamespace on <xs:schema>**

264 `targetNamespace` on `xs:schema` MUST always be specified in all schema documents
265 that are a part of a model's definition documents.

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

268 **3.2 References**

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

273 Support for inter-document references includes:

- 274 • A new data type that represents references to elements in other documents.
- 275 • Multiple addressing schemes for representing references.
- 276 • Constraints on the type of a referenced element.
- 277 • The ability to define key, unique, and key reference constraints across inter-
278 document references.

279 An SML reference is a link from one element to another. It can be represented by
280 using a variety of schemes, such as Uniform Resource Identifiers (URIs) [7] and
281 Endpoint References (EPRs) [8]. SML does not mandate the use of any specific
282 scheme for representing references; implementations are free to choose suitable
283 schemes for representing references. References MUST be supported by model
284 validators that conform to this specification.

285 References MUST be identified by `sml:ref="true"` where `sml:ref` is a global
286 attribute whose definition is as follows:

```
287     <xs:attribute name="ref" type="xs:boolean"/>  
288
```

289 An element that has `sml:ref="true"` MUST be treated as a reference element, i.e.,
290 its child elements MAY contain a reference represented in one or more schemes. This
291 mechanism enables schema-less identification of reference elements, i.e., reference
292 elements can be identified without relying on PSVI.

293
294 The following example illustrates the use of `sml:ref`. Consider the following schema
295 fragment:
296

```

297
298 <xs:element name="EnrolledCourse">
299   <xs:complexType>
300     <xs:sequence>
301       <xs:element name="Name" type="xs:string"/>
302       <xs:element name="Grade" type="xs:string"/>
303       <xs:any namespace="##any" minOccurs="0"
304         maxOccurs="unbounded" processContents="lax"/>
305     </xs:sequence>
306     <xs:anyAttribute namespace="##any" processContents="lax"/>
307   </xs:complexType>
308 </xs:element>
309
310 <xs:complexType name="StudentType">
311   <xs:sequence>
312     <xs:element name="ID" type="xs:string"/>
313     <xs:element name="Name" type="xs:string"/>
314     <xs:element name="EnrolledCourses" minOccurs="0">
315       <xs:complexType>
316         <xs:sequence>
317           <xs:element ref="tns:EnrolledCourse"
318             maxOccurs="unbounded"/>
319         </xs:sequence>
320       </xs:complexType>
321     </xs:element>
322   </xs:sequence>
323 </xs:complexType>
324

```

325 The schema definition in the above example is SML agnostic and does not make use
326 of any SML attributes, elements, or types. The `EnrolledCourse` element, however,
327 has an open content model and this can be used to mark instances of
328 `EnrolledCourse` as reference elements as shown below:
329

```

330
331 <Student xmlns="urn:university"
332         xmlns:sml="http://schemas.serviceml.org/sml/2007/02"
333         xmlns:wsa="http://www.w3.org/2005/08/addressing">
334   <ID>1000</ID>
335   <Name>John Doe</Name>
336   <EnrolledCourses>
337     <EnrolledCourse sml:ref="true">
338       <Name>PHY101</Name>
339       <Grade>A</Grade>
340       <sml:uri>
341         /Universities/MIT/Courses.xml#xmlns(u=urn:university)
342         xpointer(/u:Courses/u:Course[u:Name='PHY101'])
343       </sml:uri>
344       <wsa:EndpointReference>
345         <wsa:Address>http://www.university.example</wsa:Address>
346         <wsa:ReferenceParameters>
347           <University>
348             <Name>MIT</Name>
349           </University>
350           <Course>
351             <Name>PHY101</Name>
352           </Course>
353         </wsa:ReferenceParameters>
354       </wsa:EndpointReference>
355     </EnrolledCourse>
356     <EnrolledCourse sml:ref="false">
357       <Name>MAT100</Name>
358       <Grade>B</Grade>
359       <sml:uri>
360         /Universities/MIT/Courses.xml#xmlns(u=urn:university)
361         xpointer(/u:Courses/u:Course[u:Name='MAT100'])
362       </sml:uri>
363     </EnrolledCourse>
364     <EnrolledCourse>
365       <Name>SocialSkills</Name>
366       <Grade>F</Grade>
367     </EnrolledCourse>
368   </EnrolledCourses>
369 </Student>
370

```

371 The first `EnrolledCourse` element in the above example is a reference element since
372 it specifies `sml:ref="true"`. Assuming that references are represented in URI and
373 EPR schemes, it has two representations of the reference to the element for course
374 `PHY101`. The second and third `EnrolledCourse` elements are not reference elements;
375 the second element specifies `sml:ref="false"` and the third element does not
376 specify the `sml:ref` attribute. Note that the second element has a child element that
377 contains a reference to course `MAT100`, but this reference will be ignored since
378 `sml:ref="false"` for the second element.

379
380 A reference element MAY be empty or have a null value provided that this is allowed
381 by the element's schema. For example, consider the following variation of the
382 `EnrolledCourse` element definition:
383

```

384
385 <xs:element name="EnrolledCourse" nillable="true">
386   <xs:complexType>
387     <xs:sequence>
388       <xs:element name="Name" type="xs:string"/>
389       <xs:element name="Grade" type="xs:string"/>
390       <xs:any namespace="##any" minOccurs="0"
391         maxOccurs="unbounded" processContents="lax"/>
392     </xs:sequence>
393     <xs:anyAttribute namespace="##any" processContents="lax"/>
394   </xs:complexType>
395 </xs:element>

```

The above definition allows null values for instances of `EnrolledCourse`. Thus, an `EnrolledCourse` reference element can have null value as shown in the following example (the first `EnrolledCourse` element has null value):

```

399
400 <Student xmlns="urn:university"
401   xmlns:sml="http://schemas.serviceml.org/sml/2007/02"
402   xmlns:wsa="http://www.w3.org/2005/08/addressing">
403   <ID>1000</ID>
404   <Name>John Doe</Name>
405   <EnrolledCourses>
406     <EnrolledCourse sml:ref="true" xsi:nil="true"/>
407     <EnrolledCourse sml:ref="false">
408       <Name>MAT100</Name>
409       <Grade>B</Grade>
410       <sml:uri>
411         /Universities/MIT/Courses.xml#xmlns (u=urn:university)
412         xpointer (/u:Courses/u:Course [u:Name='MAT100' ])
413       </sml:uri>
414     </EnrolledCourse>
415     <EnrolledCourse>
416       <Name>SocialSkills</Name>
417       <Grade>F</Grade>
418     </EnrolledCourse>
419   </EnrolledCourses>
420 </Student>
421
422

```

SML also supports several schema-based constraints on references. The `sml:refType` type has been defined to allow model authors to make use of these schema-based constraints in their model's schema. The definition of `sml:refType` fixes the value of `sml:ref` to `true`, and hence all elements of type `sml:refType` are reference elements. The `sml:refType` is defined as follows:

```

428
429 <xs:complexType name="refType" sml:acyclic="false">
430   <xs:sequence>
431     <xs:any namespace="##any" minOccurs="0"
432       maxOccurs="unbounded"
433       processContents="lax"/>
434   </xs:sequence>
435   <xs:attribute ref="sml:ref" use="required"
436     fixed="true" />
437   <xs:anyAttribute namespace="##any"
438     processContents="lax"/>
439 </xs:complexType>

```

440 Note that the above definition allows elements and attributes from any namespace to
441 occur in an element whose type is `sml:refType`. Thus, a scheme for references can
442 be implemented by defining an XML namespace for the scheme, and references can
443 be represented in this scheme by nesting element and attribute instances from this
444 namespace as attributes and children of `sml:refType` elements.

445 The following example illustrates the use of `sml:refType` :

```
446 <xs:element name="EnrolledCourse" type="sml:refType"  
447           sml:targetType="tns:CourseType"/>  
448  
449 <xs:complexType name="StudentType">  
450   <xs:sequence>  
451     <xs:element name="ID" type="xs:string"/>  
452     <xs:element name="Name" type="xs:string"/>  
453     <xs:element name="EnrolledCourses" minOccurs="0">  
454       <xs:complexType>  
455         <xs:sequence>  
456           <xs:element ref="tns:EnrolledCourse"  
457                       maxOccurs="unbounded"/>  
458         </xs:sequence>  
459       </xs:complexType>  
460     </xs:element>  
461   </xs:sequence>  
462 </xs:complexType>
```

463 The `EnrolledCourse` element declaration is of type `sml:refType` which marks it as
464 a document reference, and this element declaration is used in `StudentType` to
465 reference the elements corresponding to the courses in which a student is enrolled.

466 Examples of the use of `sml:refType` for `EnrolledCourse` are found in the section,
467 [Reference Schemes](#). This section demonstrates the use of the URI and EPR schemes
468 to define the reference.

469 3.2.1 Reference Semantics

470 3.2.1.1 At Most One Target

471 Every reference MUST target (or resolve to) at most one element in a model.
472 Dangling references are allowed in SML; therefore it is possible that the target of a
473 reference does not exist in a model. It is an error if a reference targets more than
474 one element in a model.

475 If a single reference is represented by multiple schemes, every representation MUST
476 target the same element. Validators MAY check this condition if they understand
477 more than one scheme used to represent the same reference.

478 3.2.1.2 Multiple References

479 An element in a document MAY be targeted by multiple different references. These
480 references may use different schemes and/or be expressed in different ways.

481 3.2.1.3 Empty or Null References

482 A reference element, i.e., an element with `sml:ref="true"`, can have
483 `xsi:nil="true"` or no content, provided that this is allowed by the element's
484 schema definition. A model validator MUST treat such an element as if the reference
485 were not present.

486 3.2.1.4 *deref()* XPath Extension Function

487 Each model validator MUST provide an implementation of the `deref()` XPath
488 extension function that is capable of resolving references expressed in the model
489 validator's chosen scheme(s). This function takes a node-set of elements and returns
490 a node-set consisting of element nodes corresponding to the elements referenced by
491 the input node set. In particular, for each node **R** in the input node set the output
492 node set contains at most one element node.

- 493 • The output node set contains one element node for **R** provided that all of the
494 following conditions are true
 - 495 ○ `sml:ref="true"` for **R**
 - 496 ○ **R** contains at least one reference scheme that is understood by the
497 implementation
 - 498 ○ The reference targets a single element in some document in the model
- 499 • The output node set contains no element node corresponding to **R** if any of
500 the following conditions is true
 - 501 ○ the target of **R** is not in the model
 - 502 ○ **R** is an empty or null reference
 - 503 ○ **R** does not contain any reference scheme that is understood by the
504 implementation
 - 505 ○ `sml:ref` is not specified for **R**
 - 506 ○ `sml:ref="false"` is specified for **R**

507 3.3 Reference Schemes

508 A reference MAY be represented by using a variety of schemes, and SML does not
509 mandate the use of any specific schemes. Uniform Resource Identifiers (URIs) [7]
510 and endpoint references (EPRs) [8] are two common schemes for referencing
511 resources. Although SML does not require the use of either scheme, it does define
512 how a reference MUST be represented using the URI scheme and the EPR scheme.

513 3.3.1 URI Scheme

514 References that are represented using the URI scheme MUST be implemented by
515 using the `sml:uri` global element as a child of reference elements, i.e., elements for
516 which `sml:ref="true"`. More precisely, if a model validator chooses to represent
517 references using the URI scheme,

- 518 • It MUST represent each reference using an instance of the `sml:uri` global
519 element declaration as a child of the reference element.
- 520 • It MUST treat each instance of the `sml:uri` global element declaration,
521 whose parent element is a reference element, as a reference represented in
522 the URI scheme, and MUST attempt to resolve such references.

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

```
525 <EnrolledCourse xmlns="urn:university" sml:ref="true">  
526   <sml:uri>SomeValidUri</sml:uri>  
527 </EnrolledCourse>
```

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

529 Suppose that a model has the following documents, and each document has an
530 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

531

532 The following is a sample instance document for Student 1000 where the references
533 are represented in the URI scheme:

```
534 <Student xmlns="urn:university">  
535   <ID>1000</ID>  
536   <Name>John Doe</Name>  
537   <EnrolledCourses>  
538     <EnrolledCourse sml:ref="true">  
539       <sml:uri>/Universities/MIT/Courses/PHY101.xml</sml:uri>  
540     </EnrolledCourse>  
541     <EnrolledCourse sml:ref="true">  
542       <sml:uri>/Universities/MIT/Courses/MAT200.xml</sml:uri>  
543     </EnrolledCourse>  
544   </EnrolledCourses>  
545 </Student>
```

546 3.3.1.1 Fragment Identifier

547 Fragment identifiers in references that are represented using the URI scheme MUST
548 use the following XPointer [10] profile: Only two schemes – `xmlns()` and `xpointer()` –
549 are supported.

- 550 • The expression specified for the `xpointer` scheme MUST be a restricted XPath
551 1.0 [9] expression that MUST resolve to at most one element node. In
552 particular, this expression MUST NOT contain
 - 553 ○ the union ("|") operator defined for XPath 1.0
 - 554 ○ `point()` and `range()` node tests defined for the `xpointer()` scheme
- 555 • This expression can only use the functions defined in the XPath 1.0 core
556 function library (see [9] for details). It MUST NOT use the `smlfn:deref`
557 function and/or the following functions defined for `xpointer()` scheme (see
558 [11] for details):
 - 559 ○ `range-to`
 - 560 ○ `string-range`
 - 561 ○ `range`
 - 562 ○ `range-inside`
 - 563 ○ `start-point`

- 564 o end-point
- 565 o here
- 566 o origin

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

```
572 <Student xmlns="urn:university">
573   <ID>1000</ID>
574   <Name>John Doe</Name>
575   <EnrolledCourses>
576     <EnrolledCourse sml:ref="true">
577       <sml:uri>
578         /Universities/MIT/Courses.xml#xmlns(u=urn:university)
579         xpointer(/u:Courses/u:Course[u:Name='PHY101'])
580       </sml:uri>
581     </EnrolledCourse>
582   </EnrolledCourses>
583 </Student>
```

584 A reference element can also be used to reference an element in its own document. To
 585 see this consider the following instance document

```
586 <University xmlns="urn:university">
587   <Name>MIT</Name>
588   <Courses>
589     <Course>
590       <Name>PHY101</Name>
591     </Course>
592     <Course>
593       <Name>MAT200</Name>
594     </Course>
595   </Courses>
596   <Students>
597     <Student>
598       <ID>123</ID>
599       <Name>Jane Doe</Name>
600       <EnrolledCourses>
601         <EnrolledCourse sml:ref="true">
602           <sml:uri>
603             #xmlns(u=urn:university)
604             xpointer(/u:University/u:Courses/u:Course[u:Name='MAT200'])
605           </sml:uri>
606         </EnrolledCourse>
607       </EnrolledCourses>
608     </Student>
609   </Students>
610 </University>
```

612 Here, the EnrolledCourse element for the student Jane Doe references the Course
 613 element for MAT200 in the same document.

614 **3.3.2 EPR Scheme**

615 References that are represented using the EPR scheme MUST be implemented by
616 using instances of `wsa:EndpointReference` global element declaration [8] as child
617 elements of reference elements. The following example illustrates how the
618 `EnrolledCourse` reference that references course PHY101 in MIT university can be
619 represented using the EPR scheme:

620

```
621 <EnrolledCourse xmlns="urn:university" sml:ref="true">
622   <wsa:EndpointReference
623     xmlns:u="http://www.university.example/schema">
624     <wsa:Address>http://www.university.example</wsa:Address>
625     <wsa:ReferenceParameters>
626       <u:University>
627         <u:Name>MIT</u:Name>
628       </u:University>
629       <u:Course>
630         <u:Name>PHY101</u:Name>
631       </u:Course>
632     </wsa:ReferenceParameters>
633   </wsa:EndpointReference>
634 </EnrolledCourse>
```

635 **3.4 Constraints on References**

636 SML supports several attributes for expressing constraints on references. All of
637 these attributes (with the sole exception of `sml:acyclic`) can only be specified for
638 element declarations of type `sml:refType` or a derived type of `sml:refType`. The
639 `sml:acyclic` attribute can only be specified on derived types of `sml:refType`
640 (`sml:acyclic="false"` is specified for `sml:refType`).

641 The following table lists the various attributes and elements for constraining
642 references:

643 **Attributes**

Name	Description
<code>sml:acyclic</code>	Supported on <code>sml:refType</code> and its derived types.. If this attribute is set to true for a derived type <code>D</code> of <code>sml:refType</code> , then the acyclic constraint is violated if instances of <code>D</code> (including any derived types of <code>D</code>) create a cycle in a model.
<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:targetRequired</code>	Used to specify that a reference's target element is required to be present in the model. This constraint is violated if a reference is empty, null, or dangling.

644

sml:targetType	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.
----------------	--

646 3.4.1 sml:acyclic

647 Model validators that conform to this specification MUST support the `sml:acyclic`
648 attribute on derived types of `sml:refType`. This is a boolean attribute and its value
649 can be either `true` or `false`. Let **R** be a derived type of `sml:refType`. If
650 `sml:acyclic="true"` is specified for **R**, then **R** is an acyclic reference type, i.e.,
651 instances of **R** MUST NOT create cycles in any model. More precisely, the directed
652 graph whose nodes are documents that contain the source or target elements for
653 instances of **R**, and whose edges are instances of **R** (an edge is directed from the
654 document containing the source element to the document containing the target
655 element), must be acyclic. If `sml:acyclic="false"` is specified for **R**, then **R** is a
656 cyclic reference type, and its instances may create cycles in models. Note that
657 `sml:refType` is a cyclic reference type since `sml:acyclic="false"` is specified for
658 `sml:refType`.

659 A cyclic reference type can be used to derive cyclic or acyclic reference types, but all
660 derived types of an acyclic reference type are acyclic. Model validators that conform
661 to this specification MUST enforce the following:

- 662 • If **CR** is a cyclic reference type and **D_{CR}** is a derived type of **CR**, then **D_{CR}** is an
663 acyclic reference if `sml:acyclic="true"` is specified for **D_{CR}**. Otherwise, **D_{CR}**
664 is a cyclic reference
- 665 • If **AR** is an acyclic reference type and **D_{AR}** is a derived type of **AR**, then
666 `sml:acyclic="true"` holds for **D_{AR}** even if the `sml:acyclic` attribute is not
667 explicitly specified for **D_{AR}**. It is an error for **D_{AR}** to specify
668 `sml:acyclic="false"`

669 3.4.2 Constraints on Targets

670 SML supports three attributes: `sml:targetElement`, `sml:targetRequired`, and
671 `sml:targetType`, for constraining the target of a reference. These three attributes
672 are collectively called `sml:target*` attributes and they MUST be supported on global
673 and local element declarations. Model validators that conform to this specification
674 MUST enforce the following:

675 If one/more of `sml:target*` attributes are specified (either explicitly or by default)
676 for a [particle](#) **P** in a complex-type definition **CT**, then all particles in **CT** that have the
677 same [name](#) as **P** must specify the same set of `sml:target*` attributes as **P** and
678 these attributes must have the same values as those specified for **P**.

679 In particular, all of the following must be enforced:

- 680 • If `sml:targetElement="ns:GTE"` for **P** then `sml:targetElement="ns:GTE"`
681 for all particles in **CT** that have the same name as **P**
- 682 • If `sml:targetRequired="true"` for **P** then `sml:targetRequired="true"` for
683 all particles in **CT** that have the same name as **P**
- 684 • If `sml:targetRequired="false"` for **P** then `sml:targetRequired="false"`
685 for all particles in **CT** that have the same name as **P**

- 686 • If `sml:targetType="ns:T"` for **P** then `sml:targetType="ns:T"` for all
687 particles in **CT** that have the same name as **P**

688 The above conditions on the use of `sml:target*` attributes have been defined to
689 reduce the implementation burden on model validators for verifying that the use of
690 `sml:target*` attributes is consistent across derivation by restriction. These
691 conditions enable model validators to find the restricted particle for a restricting
692 particle using a simple name match when `sml:target*` attributes are specified for
693 these particles. In the absence of the above conditions, it is extremely difficult for
694 SML validators to verify consistent use of `sml:target*` attributes across a base type
695 and its restricted derived type. In order to verify consistent use of an `sml:target*`
696 attribute on a restricted particle in the base type and its restricting particle in a
697 restricted derived type, it is necessary to connect the particles in the derived type
698 with those from the restricted base type. However, this level of support is not
699 provided by most XML Schema frameworks; thus most SML validators would
700 otherwise need to duplicate large parts of XML Schema's compilation logic to verify
701 consistent usage of `sml:target*` attributes across derivation by restriction.

702 3.4.2.1 `sml:targetElement`

703 Model validators that conform to this specification MUST support the
704 `sml:targetElement` attribute on element declarations whose type is `sml:refType` or
705 a derived type of `sml:refType`. The value of this attribute MUST be the qualified
706 name of some global element declaration. Let `sml:targetElement="ns:GTE"` for
707 some element declaration **E**. Then each element instance of **E** MUST reference an
708 element that is an instance of **ns:GTE** or an instance of some global element
709 declaration in the substitution group hierarchy whose head is **ns:GTE**. If a target
710 element constraint is specified for a global element declaration **G** then it continues to
711 apply to all global element declarations in the substitution group hierarchy whose
712 head is **G**. However, a global element declaration in **G**'s substitution group can
713 specify a target element constraint that refines the constraint defined for **G**. In
714 particular, model validators that conform to this specification MUST enforce the
715 following:

- 716 • If `sml:targetElement="ns:GTE"` is specified for **G**, and **S_G** is a global element
717 declaration that specifies **G** as the value of its `xs:substitutionGroup`
718 attribute, then
 - 719 • if `sml:targetElement` is specified for **S_G** then its value MUST be **ns:GTE**
720 or the name of a global element declaration in the substitution group
721 whose head is **ns:GTE**
 - 722 • if `sml:targetElement` is not specified for **S_G**, then
723 `sml:targetElement="ns:GTE"` holds for **S_G** by default.

724 If a target element constraint is specified for a particle **P** in some type **B**, then it
725 continues to apply to each particle **P_R** that is a valid restriction of **P** where **P_R**
726 is defined in some restricted derived type of **B** (see [2]
727 <http://www.w3.org/TR/xmlschema-1/#cos-particle-restrict> for XML Schema's
728 definition of valid restrictions). However, **P_R** can specify a target element constraint
729 that refines the constraint defined for **P**. In particular, model validators that conform
730 to this specification MUST enforce the following:

- 731 • If `sml:targetElement="ns:GTE"` is specified for **P** and `sml:targetElement` is
732 specified for **P_R**, then the value of `sml:targetElement` for **P_R** must be **ns:GTE**
733 or the name of a global element declaration in the substitution group

734 hierarchy whose head is **ns:GTE**. If `sml:targetElement` is not specified for
735 **P_R**, then `sml:targetElement="ns:GTE"` holds for **P_R** by default.

736 3.4.2.2 *sml:targetRequired*

737 Model validators that conform to this specification MUST support the
738 `sml:targetRequired` attribute on element declarations whose type is `sml:refType`
739 or a derived type of `sml:refType`. If `sml:targetRequired="true"` for an element
740 declaration **E**, then each element instance of **E** MUST target some element in the
741 model, i.e., no instance of **E** can be null, empty, or contain a dangling reference.
742 Otherwise, instances of **E** can be empty, null, or contain dangling references. If this
743 attribute is not specified, then its value is assumed to be "false".

744 Model validators that conform to this specification MUST enforce the following:

- 745 • If the `sml:targetRequired` attribute is specified for a global element
746 declaration **G** then the specified value applies by default to each global
747 element declaration **S_G** in the substitution group hierarchy whose head is **G**
748 unless the `sml:targetRequired` attribute is specified for **S_G**.
- 749 • If `sml:targetRequired="true"` is specified for a global element declaration **G**
750 then `sml:targetRequired="false"` MUST NOT be specified for any element
751 declaration in the substitution group hierarchy whose head is **G**.
- 752 • If `sml:targetRequired` attribute is specified for a particle **P** in some type **B**,
753 then the specified value applies by default to each particle **P_R** that is a
754 valid restrictions of **P** unless the `sml:targetRequired` attribute is specified
755 for **P_R** (see [2] <http://www.w3.org/TR/xmlschema-1/#cos-particle-restrict> for
756 XML Schema's definition of valid restrictions).
- 757 • If `sml:targetRequired="true"` for a particle **P** then
758 `sml:targetRequired="false"` MUST NOT be specified for any particle **P_R**
759 that is a valid restriction of **P**.

760 3.4.2.3 *sml:targetType*

761 The `sml:targetType` attribute MUST be supported on element declarations whose
762 type is `sml:refType` or a derived type of `sml:refType`. The value of this attribute
763 MUST be the qualified name of some type declaration. Let `sml:targetType="ns:T"`
764 for some element declaration **E**. Then each element instance of **E** MUST reference an
765 element whose type is **ns:T** or a derived type of **ns:T**.

766 If a target type constraint is specified for a global element declaration **G** then it
767 continues to apply to all global element declarations in the substitution group
768 hierarchy whose head is **G**. However, a global element declaration in **G**'s substitution
769 group can specify a target type constraint that refines the constraint defined for **G**.
770 In particular, model validators that conform to this specification MUST enforce the
771 following:

- 772 • If `sml:targetType="ns:T"` is specified for **G**, and **S_G** is a global element
773 declaration that specifies **G** as the value of its `xs:substitutionGroup`
774 attribute, then
 - 775 ○ if the `sml:targetType` attribute is specified for **S_G** the its value
776 MUST be either **ns:T** or the name of some derived type of **ns:T**
 - 777 ○ if `sml:targetType` is not specified for **S_G**, then
778 `sml:targetType="ns:T"` holds for **S_G** by default

779 If the target type constraint is specified for a particle **P** in some type **B**, then it
780 continues to apply to each particle **P_R** that is a valid restriction of **P** where **P_R** is

781 defined in some restricted derived type of **B**. However, **P_R** can specify a target type
 782 constraint that refines the constraint defined for **P**. In particular, model validators
 783 that conform to this specification MUST enforce the following:

- 784 • If `sml:targetType="ns:T"` is specified for **P** and `sml:targetType` is specified
 785 for **P_R** then the value of the `sml:targetType` for **P_R** must be **ns:T** or the
 786 name of some derived type of **ns:T**. If `sml:targetType` is not specified for **P_R**,
 787 then `sml:targetType="ns:T"` holds for **P_R** by default

788 3.5 Identity Constraints

789 XML Schema supports the definition of key, unique, and key reference constraints
 790 through `xs:key`, `xs:unique`, and `xs:keyref` elements. However, the scope of these
 791 constraints is restricted to a single document. SML defines analogs for these
 792 constraints, whose scope extends to multiple documents by allowing them to
 793 traverse inter-document references.

794 Model validators that conform to this specification MUST support the following
 795 elements for defining identity constraints across 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

796 The syntax and semantics of the above elements are the same as that for the
 797 corresponding elements in XML Schema, except for the following:

- 798 • If an SML identity constraint needs to be specified for an element declaration
 799 **E**, then it MUST be defined in the `xs:annotation/xs:appinfo` descendant
 800 element for the `xs:element` element for **E**
- 801 • An SML identity constraint that is specified for an element declaration **E** can
 802 reuse the definition of an SML identity constraint **ID'** specified for some other
 803 element declaration **E'** by specifying the name of **E'** as the value of its `ref`
 804 attribute. In particular,
 - 805 ○ If the `ref` attribute is specified for an SML identity constraint element
 806 that is specified for an element declaration **E**, then the value of `ref`
 807 attribute MUST NOT be name of any other SML identity constraint
 808 element specified for **E**.
 - 809 ○ If the `ref` attribute is specified for an `sml:key` element, then the value
 810 of `ref` attribute MUST be name of another SML key constraint
 - 811 ○ If the `ref` attribute is specified for an `sml:unique` element then the
 812 value of the `ref` attribute MUST be name of another SML unique
 813 constraint
 - 814 ○ If the `ref` attribute is specified for an `sml:keyref` element then the
 815 value of the `ref` attribute MUST be name of another SML keyref
 816 constraint
 - 817 ○ If the `ref` attribute is specified for an SML identity constraint, then the
 818 `name` attribute MUST NOT be specified

819 o If the `ref` attribute is specified for an SML identity constraint, then the
820 selector and field child elements MUST NOT be specified

821

- 822 • If an SML identity constraint is specified for an element declaration **E**, then
823 this constraint is applicable to all instances of **E** in a model, i.e., the identity
824 constraint MUST be satisfied for each instance of **E** in a valid model
- 825 • The `sml:selector` XPath expression MUST conform to the following extended
826 BNF

827

```
Selector ::= Path ( '|' Path)*
```

828

```
Path ::= ('./')? Step ( '/' Step)* | DerefExpr
```

829

```
DerefExpr ::= 'deref(' Step (/Step)* ')' ( '/' Step)* |
```

830

```
          'deref(' DerefExpr ')' (/Step)*
```

831

```
Step ::= '.' | NameTest
```

832

```
NameTest ::= QName | '*' | NCName ':' '*'
```

833

- 834 • The `sml:field` XPath expression MUST conform to the BNF given above for
835 the selector XPath expression with the following modification

836

```
Path ::= ('./')? ( Step '/' )* ( Step | @NameTest ) |
```

837

```
DerefExpr ( '/' @NameTest)?
```

838

- 839 • Each SML identity constraint that is specified for a global-element declaration
840 **G** MUST be treated as if it is specified by default for all global-element
841 declarations **S_G** that are in the substitution group hierarchy whose head is **G**
- 842 • Each SML identity constraint that is specified for a particle **P** in a complex-
843 type definition **CT** MUST be treated as if it is specified by default for all
844 particles **P_R** in restricted derived types of **CT** that are a valid restriction of **P**
- 845 • If one/more SML identity constraints are specified (either explicitly or by
846 default) for a particle **P** in a complex-type definition **CT**, then all particles in
847 **CT** that have the same [name](#) as **P** MUST specify the same set of identity
848 constraints as **P**. This rule is defined to reduce the implementation burden for
849 model validators. It facilitates the matching of restricting and restricted
particles using their names, and avoids the replication of large parts of XML
Schema's compilation logic for this purpose.

850

3.5.1 University Example

851

The following example will be used to illustrate the `sml:key`, `sml:unique`, and

852

`sml:keyref` constraints across references.

853

```

854
855 <xs:element name="Student"
856         type="sml:refType"
857         sml:targetType="tns:StudentType"/>
858
859 <xs:element name="Course"
860         type="sml:refType"
861         sml:targetType="tns:CourseType"/>
862
863 <xs:complexType name="UniversityType">
864     <xs:sequence>
865         <xs:element name="Name" type="xs:string"/>
866         <xs:element name="Students" minOccurs="0">
867             <xs:complexType>
868                 <xs:sequence>
869                     <xs:element ref="tns:Student" maxOccurs="unbounded"/>
870                 </xs:sequence>
871             </xs:complexType>
872         </xs:element>
873         <xs:element name="Courses" minOccurs="0">
874             <xs:complexType>
875                 <xs:sequence>
876                     <xs:element ref="tns:Course" maxOccurs="unbounded"/>
877                 </xs:sequence>
878             </xs:complexType>
879         </xs:element>
880     </xs:sequence>
881 </xs:complexType>
882
883 <xs:element name="EnrolledStudent"
884         type="sml:refType"
885         sml:targetType="tns:StudentType"/>
886
887 <xs:element name="EnrolledCourse"
888         type="sml:refType"
889         sml:targetType="tns:CourseType"/>
890
891 <xs:complexType name="StudentType">
892     <xs:sequence>
893         <xs:element name="ID" type="xs:string"/>
894         <xs:element name="SSN" type="xs:string" minOccurs="0"/>
895         <xs:element name="Name" type="xs:string"/>
896         <xs:element name="EnrolledCourses" minOccurs="0">
897             <xs:complexType>
898                 <xs:sequence>
899                     <xs:element ref="tns:EnrolledCourse"
900                         maxOccurs="unbounded"/>
901                 </xs:sequence>
902             </xs:complexType>
903         </xs:element>
904     </xs:sequence>

```

```

905     </xs:complexType> <xs:complexType name="CourseType">
906         <xs:sequence>
907             <xs:element name="Name" type="xs:string"/>
908             <xs:element name="EnrolledStudents" minOccurs="0">
909                 <xs:complexType>
910                     <xs:sequence>
911                         <xs:element ref="tns:EnrolledStudent"
912                             maxOccurs="unbounded"/>
913                     </xs:sequence>
914                 </xs:complexType>
915             </xs:element>
916         </xs:sequence>
917     </xs:complexType>

```

918 3.5.2 sml:key and sml:unique

919 XML Schema supports key and uniqueness constraints through `xs:key` and
920 `xs:unique`, but these constraints can only be specified within a single XML
921 document. The `sml:key` and `sml:unique` elements support the specification of key
922 and uniqueness constraints across documents. We'll use the [UniversityType](#)
923 definition to illustrate this concept. It is reasonable to expect that each student in a
924 university must have a unique identity, and this identity must be specified. This can
925 be expressed as follows:

```

926     <xs:element name="University" type="tns:UniversityType">
927         <xs:annotation>
928             <xs:appinfo>
929                 <sml:key name="StudentIDisKey">
930                     <sml:selector xpath="smlfn:deref(tns:Students/tns:Student)/tns:ID"/>
931                     <sml:field xpath="."/>
932                 </sml:key>
933             </xs:appinfo>
934         </xs:annotation>
935     </xs:element>

```

936 The `sml:key` and `sml:unique` constraints are similar but not the same. `sml:key`
937 requires that the specified fields must be present in instance documents and have
938 unique values, whereas `sml:unique` simply requires the specified fields to have
939 unique values but does not require them to be present in instance documents. Thus
940 keys imply uniqueness, but uniqueness does not imply keys. For example, students
941 in a university must have a unique social security numbers, but the university may
942 have foreign students who do not possess this number. This constraint can be
943 specified as follows:

```

944     <xs:element name="University" type="tns:UniversityType">
945         <xs:annotation>
946             <xs:appinfo>
947                 <sml:unique name="StudentSSNisUnique">
948                     <sml:selector xpath="smlfn:deref(tns:Students/tns:Student)"/>
949                     <sml:field xpath="tns:SSN"/>
950                 </sml:unique>
951             </xs:appinfo>
952         </xs:annotation>
953     </xs:element>
954

```

955 The `sml:key` and `sml:unique` constraint are always specified in the context of a
956 scoping element. In the above example, the `University` element declaration is the
957 context for the key and unique constraints.

958 The following example illustrates the use of the `ref` attribute in an SML identity
959 constraint:

```
960     <xs:element name="PrivateUniversity" type="tns:UniversityType">
961       <xs:annotation>
962         <xs:appinfo>
963           <sml:unique ref="tns:StudentSSNisUnique"/>
964         </xs:appinfo>
965       </xs:annotation>
966     </xs:element>
```

967 In the above example, the `PrivateUniversity` element declaration specifies the
968 `StudentSSNisUnique` unique constraint by referencing its definition in the
969 `University` element declaration.

970 3.5.3 `sml:keyref`

971 XML Schema supports key references through `xs:keyref` to ensure that one set of
972 values is a subset of another set of values within an XML document. Such constraints
973 are similar to foreign keys in relational databases. Key references in XML Schema are
974 only supported within a single XML document. The `sml:keyref` element allows key
975 references to be specified across XML documents, and can be used to scope
976 references to point to elements within a valid range. The following example uses
977 `sml:keyref` to capture the requirement that courses in a university can only enroll
978 students from the same university:

```
979     <xs:element name="University" type="tns:UniversityType">
980       <xs:annotation>
981         <xs:appinfo>
982           <sml:key name="StudentIDisKey">
983             <sml:selector xpath="smlfn:deref(tns:Students/tns:Student)"/>
984             <sml:field xpath="tns:ID"/>
985           </sml:key>
986           <sml:keyref name="CourseStudents" refer="tns:StudentIDisKey">
987             <sml:selector xpath="smlfn:deref(
988               smlfn:deref(tns:Courses/tns:Course)/
989               tns:EnrolledStudents/tns:EnrolledStudent)"/>
990             <sml:field xpath="tns:ID"/>
991           </sml:keyref>
992         </xs:appinfo>
993       </xs:annotation>
994     </xs:element>
```

995 The above constraint specifies that for a university, the set of IDs of students
996 enrolled in courses is a subset of the set of IDs of students in a university. In
997 particular, the `selector` and `field` elements in `StudentIDisKey` key constraint
998 identify the set of IDs of students in a university, and the `selector` and `field`
999 elements in `CourseStudents` key reference constraint identify the set of IDs of
1000 students enrolled in courses.

1001 4. Rules

1002 XML Schema supports a number of built-in grammar-based constraints but it does
1003 not support a language for defining arbitrary rules for constraining the structure and
1004 content of documents. Schematron [4] is an ISO standard for defining assertions

1005 concerning a set of XML documents. SML uses a profile of the Schematron schema to
1006 add support for user-defined constraints. SML uses XPath1.0, augmented with the
1007 `smlfn:deref()` extension function, as its constraint language. Model validators that
1008 conform to this specification are REQUIRED to support and evaluate XPath 1.0
1009 expressions augmented with the `smlfn:deref()` function in the body of Schematron
1010 constraints. This section assumes that the reader is familiar with Schematron
1011 concepts; the Schematron standard is documented in [4] and [5,6] are good
1012 tutorials on an older version of Schematron.

1013 User-defined constraints can be specified using the `sch:assert` and `sch:report`
1014 elements from Schematron. The following example uses `sch:assert` elements to
1015 specify two constraints:

- 1016 • An IPv4 address must have four bytes
- 1017 • An IPv6 address must have sixteen bytes

1018
1019

```

1020 <xs:simpleType name="IPAddressVersionType">
1021   <xs:restriction base="xs:string" >
1022     <xs:enumeration value="V4" />
1023     <xs:enumeration value="V6" />
1024   </xs:restriction>
1025 </xs:simpleType>
1026 <xs:complexType name="IPAddress">
1027   <xs:annotation>
1028     <xs:appinfo>
1029       <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
1030         <sch:ns prefix="tns" uri="urn:IPAddress" />
1031         <sch:pattern id="Length">
1032           <sch:rule context=".">
1033             <sch:assert test="tns:version != 'V4' or count(tns:address) = 4">
1034               A v4 IP address must have 4 bytes.
1035             </sch:assert>
1036             <sch:assert test="tns:version != 'V6' or count(tns:address) = 16">
1037               A v6 IP address must have 16 bytes.
1038             </sch:assert>
1039           </sch:rule>
1040         </sch:pattern>
1041       </sch:schema>
1042     </xs:appinfo>
1043   </xs:annotation>
1044   <xs:sequence>
1045     <xs:element name="version" type="tns:IPAddressVersionType" />
1046     <xs:element name="address" type="xs:byte" minOccurs="4" maxOccurs="16" />
1047   </xs:sequence>
1048 </xs:complexType>

```

1049 A Schematron pattern embedded in the `xs:annotation/xs:appinfo` element for a
1050 complex type definition or an element declaration is applicable to all instances of the
1051 complex type or element. In the above example, the pattern `Length` is applicable for
1052 all elements whose type is `IPAddress` or a derived type of `IPAddress`. A pattern can
1053 have one or more rules, and each rule specifies a context expression using the
1054 `context` attribute. The value of the `context` attribute is an XPath expression that is
1055 evaluated in the context of each applicable element, and results in an element node
1056 set for which the `assert` and `report` test expressions defined in the rule are evaluated.
1057 In the above example `context="."`, therefore the two `assert` expressions are
1058 evaluated in the context of each applicable element, i.e., each element of type
1059 `IPAddress`. The test expression for an `assert` is a boolean expression, and the
1060 `assert` is violated (or fires) if its test expression evaluates to false. For example,
1061 the following XML document violates the `assert` that requires an IPv6 address to
1062 have sixteen address bytes

```

1063 <myIPAddress xmlns="urn:IPAddress">
1064   <version>v6</version>
1065   <address>100</address>
1066   <address>200</address>
1067   <address>10</address>
1068   <address>1</address>
1069   <address>10</address>
1070   <address>1</address>
1071 </myIPAddress>

```

1072 In general, a `rule` element can include multiple `assert` and `report` elements. A
1073 `report` also specifies a test expression, just like an `assert`. However, a `report` is
1074 violated (or fires) if its test expression evaluates to true. Thus, an `assert` can be
1075 converted to a `report` by simply negating its test expression. The following example

1076 uses report elements to represent the IP address constraints of the previous
1077 example:

```
1078 <xs:simpleType name="IPAddressVersionType">
1079   <xs:restriction base="xs:string">
1080     <xs:enumeration value="V4"/>
1081     <xs:enumeration value="V6"/>
1082   </xs:restriction>
1083 </xs:simpleType>
1084 <xs:complexType name="IPAddress">
1085   <xs:annotation>
1086     <xs:appinfo>
1087       <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
1088         <sch:ns prefix="tns" uri="urn:IPAddress" />
1089         <sch:pattern id="Length">
1090           <sch:rule context=".">
1091             <sch:report test="tns:version = 'V4' and count(tns:address) != 4"
1092             >
1093               A v4 IP address must have 4 bytes.
1094             </sch:report>
1095             <sch:report test="tns:version = 'V6' and count(tns:address) != 16"
1096             >
1097               A v6 IP address must have 16 bytes.
1098             </sch:report>
1099           </sch:rule>
1100         </sch:pattern>
1101       </sch:schema>
1102     </xs:appinfo>
1103   </xs:annotation>
1104   <xs:sequence>
1105     <xs:element name="version" type="tns:IPAddressVersionType" />
1106     <xs:element name="address" type="xs:byte" minOccurs="4" maxOccurs="16" />
1107   </xs:sequence>
1108 </xs:complexType>
```

1109 If an assert or report is violated, then the violation must be reported during model
1110 validation together with the specified message. Model validation must evaluate each
1111 Schematron pattern for all of its applicable elements contained in the model.

1112 The message can include substitution strings based on XPath expressions. These can
1113 be specified using the sch:value-of element. The following example uses the
1114 sch:value-of element to include the number of specified address bytes in the
1115 message:

```
1116 <sch:assert test="tns:version != 'v4' or count(tns:address) = 4">
1117   A v4 IP address must have 4 bytes instead of the specified
1118   <sch:value-of select="string(count(tns:address))"/> bytes.
1119 </sch:assert>
```

1120 In addition to being embedded in complex type definitions, constraints can also be
1121 embedded in global-element declarations. Such constraints are evaluated for each
1122 instance element corresponding to the global-element definition. Consider the
1123 following example:

1124

```

1125
1126 <xs:element name="StrictUniversity" type="tns:UniversityType">
1127   <xs:annotation>
1128     <xs:appinfo>
1129       <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
1130         <sch:ns prefix="u" uri="urn:university" />
1131         <sch:ns prefix="smlfn"
1132           uri=" http://schemas.serviceml.org/smlfn/query/2006/07"/>
1133         <sch:pattern id="StudentPattern">
1134           <sch:rule context="smlfn:deref(u:Students/u:Student)">
1135             <sch:assert test="starts-with(u:ID,'99')">
1136               The specified ID <sch:value-of select="string(u:ID)"/>
1137               does not begin with 99
1138             </sch:assert>
1139             <sch:assert test="count(u:Courses/u:Course)>0">
1140               The student <sch:value-of select="string(u:ID)"/> must be enrolled
1141               in at least one course
1142             </sch:assert>
1143           </sch:rule>
1144         </sch:pattern>
1145       </sch:schema>
1146     </xs:appinfo>
1147   </xs:annotation>
1148 </xs:element>

```

1149 The constraints defined in `StudentPattern` are applicable to all element instances of
1150 the `StrictUniversity` global element definition. For each `StrictUniversity`
1151 element, the XPath expression specified as the value of the `context` attribute is
1152 evaluated to return a node set, and the test expressions for the two asserts are
1153 evaluated for each node in this node set. The context expression for the rule returns
1154 a node set consisting of all `Student` elements referenced by an instance of
1155 `StrictUniversity`, and the test expressions for the two asserts are evaluated for
1156 each element node in this node set. Thus, these two asserts verify the following
1157 conditions for each instance of `StrictUniversity`

- 1158 • The ID of each student must begin with '99'
- 1159 • Each student must be enrolled in at least one course

1160 Model validators that conform to this specification MUST behave as follows:

- 1161 • Each Schematron pattern that is embedded in the
1162 `xs:annotation/xs:appinfo` element for a global complex-type definition **CT**
1163 MUST be evaluated for all element instances of type **CT** in a model during the
1164 model's validation
- 1165 • Each Schematron pattern that is embedded in the
1166 `xs:annotation/xs:appinfo` element for a global-element declaration **G**
1167 MUST be evaluated for all element instances of **G** in a model during the
1168 model's validation
- 1169 • A pattern MUST be evaluated for an instance element by evaluating the `rule`
1170 elements of the pattern in the order of their definition. The context expression
1171 for a rule MUST be evaluated in the context of the instance element, and all
1172 asserts and reports contained in the first rule whose context expression
1173 evaluates to a non-empty node set MUST be evaluated for each node in this
1174 node set.

1175 Model validators that conform to this specification MUST provide a mechanism to
1176 support binding of Schematron patterns that are authored in separate documents,
1177 i.e., not embedded in schema definition, to a set of documents in a model. The
1178 mechanism for binding such Schematron patterns to a set of documents in a model
1179 are implementation dependent and hence outside the scope of this specification. The

1180 following example shows the constraints for `StrictUniversity` expressed in a
1181 separate document:

```
1182 <?xml version="1.0" encoding="utf-8" ?>
1183 <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
1184   <sch:ns prefix="u" uri="urn:university" />
1185   <sch:ns prefix="smlfn" uri="http://schemas.serviceml.org/smlfn/query/2006/07"
1186   />
1187   <sch:pattern id="StudentPattern">
1188     <sch:rule context="smlfn:deref(u:Students/u:Student)">
1189       <sch:assert test="starts-with(u:ID,'99')">
1190         The specified ID <sch:value-of select="string(u:ID)"/>
1191         does not begin with 99
1192       </sch:assert>
1193       <sch:assert test="count(u:Course/u:Courses)>0">
1194         The student <sch:value-of select="string(u:ID)"/> must be enrolled
1195         in at least one course
1196       </sch:assert>
1197     </sch:rule>
1198   </sch:pattern>
1199 </sch:schema>
```

1200 The binding of the `StudentPattern` pattern to instances of `StrictUniversity`
1201 element is implementation dependent and hence outside the scope of this
1202 specification.

1203 4.1 Localization of Error Messages

1204 Localization of the natural-language error messages, that provide details about
1205 asserts and reports, MAY be supported by model validators that conform to this
1206 specification. Such model validators MAY support the use of `smlerr:localizationid`
1207 attribute on `sch:report` and `sch:assert` to specify the identity of the resource
1208 containing the localized versions of the natural-language error message. Model
1209 validators that conform to this specification but do not support
1210 `smlerr:localizationid` attribute MUST ignore all `smlerr:output` attributes in a
1211 model; they MUST NOT treat the model as invalid just because it contains
1212 `smlerr:localizationid` attributes. The mechanisms for mapping values of
1213 `smlerr:localizationid` to the corresponding localization resources are
1214 implementation dependent and hence outside the scope of this specification.

1215 4.2 Schematron Profile

1216 SML supports a conforming profile of Schematron. All elements and attributes are
1217 supported.

1218 4.2.1 Limited Support

1219 If the `queryBinding` attribute is specified, then its value MUST be set to "xpath1.0"

1220 5. Structured XML Output from Schematron Rules

1221 Schematron has rich support for natural-language error and diagnostic messages
1222 that provide details about failed assertions. As per the Schematron specification the
1223 content of the `sch:assert`, `sch:report`, and the optional `sch:diagnostic` elements
1224 should be natural language assertions or messages. To facilitate machine
1225 processable output from the evaluation of Schematron rules, this specification extends
1226 Schematron by adding support for structured XML output that provides details about
1227 failed assertions. This structured XML data can be consumed by an application to
1228 perform some application-specific tasks required to handle a failed assertion. This is
1229 an OPTIONAL feature and model validators that conform to this specification are not
1230 REQUIRED to support it. Model validators that conform to this specification but do
1231 not support `smlerr:output` element MUST ignore all `smlerr:output` elements in a

1232 model; they MUST NOT treat the model as invalid just because it contains
1233 `smlerr:output` elements.

1234 **5.1 smlerr:output**

1235 This element is used to specify the structured XML output for one/more failed
1236 assertions. It is supported as a child of the `sch:rule` element. An `sch:rule` element
1237 can contain multiple `smlerr:output` elements. The schema definition for
1238 `smlerr:output` is as follows:

```
1239 <xs:element name="output" type="smlerr:outputType"/>
1240
1241 <xs:complexType name="outputType">
1242     <xs:attribute name="id" type="xs:ID"
1243         use="required"/>
1244
1245     <xs:attribute name="applicationUri" type="xs:anyURI"
1246         use="optional"/>
1247     <xs:attribute name="expression" type="xs:string" use="required"/>
1248 </xs:complexType>
1249
```

1250 `id` = a required attribute that defines the identity of an `smlerr:output` element.
1251 This identity is used by an `assert` and/or `report` element to specify that the
1252 expression specified in the `expression` attribute of the `smlerr:output` element must
1253 be evaluated to generate structured XML when the `assert/report` fires.

1254 `applicationUri` = an optional attribute that specifies the identity of the application
1255 for which the output is generated

1256 `expression` = an XPath 1.0 expression that evaluates to a node set containing
1257 element and attribute nodes only. If the node set contains namespace, processing
1258 instructions, comments, or text nodes, then no output is generated. The expression
1259 is evaluated in the context of the node selected by the `context` attribute in the
1260 parent `sch:rule` element. This XPath expression can use the `deref()` extension
1261 function.

1262 **5.1.1 smlerr:outputids**

1263 This global attribute is used in an `assert` or `report` to specify the identities of the
1264 `smlerr:output` elements whose expressions must be evaluated to generate XML
1265 output when the `assert/report` fires.

```
1266 <xs:attribute name="outputids" type="xs:IDREFS"/>
```

1267 **5.1.2 smlerr:attributeNode**

1268 This element is used for serialization of each attribute node in the node set resulting
1269 from the evaluation of the expression in an `smlerr:output` element.

1270

```

1271
1272 <xs:element name="attributeNode" type="smlerr:attributeNodeType"/>
1273
1274 <xs:complexType name="attributeNodeType">
1275   <xs:simpleContent>
1276     <xs:extension base="xs:string">
1277       <xs:attribute name="name" type="xs:QName"/>
1278     </xs:extension>
1279   </xs:simpleContent>
1280 </xs:complexType>

```

1281

1282 name: The value of this attribute is the qualified name of the attribute whose value is
 1283 being serialized.

1284 5.1.3 smlerr:errorData

1285 This element is used for enclosing the structured XML generated by an
 1286 smlerr:output element.

```

1287 <xs:element name="errorData" type="smlerr:errorDataType"/>
1288 <xs:complexType name="errorDataType">
1289   <xs:sequence>
1290     <xs:any namespace="##any" processContents="skip"/>
1291   </xs:sequence>
1292 </xs:complexType>

```

1293 5.1.4 Semantics

1294 When a report/assert fires, then all smlerr:output elements that list the ID of this
 1295 report/assert are evaluated. For each such smlerr:output, the expression specified
 1296 in its expression attribute is evaluated, and the resulting node set serialized into
 1297 XML by concatenating each node and enclosing the serialized XML fragment in the
 1298 smlerr:errorData element to create a well-formed XML document. The resulting
 1299 document is returned to the application that initiated the model validation. The
 1300 serialization is only performed if the node set contains attribute and/or element
 1301 nodes. Otherwise, no structured XML is serialized and an empty smlerr:errorData
 1302 element is returned.

1303 The nodes in the node set may be serialized in any order. Element nodes are
 1304 serialized directly into their XML document order representation, and attribute nodes
 1305 are serialized by using the smlerr:attributeNode element.

1306 All namespace bindings defined (through the sch:ns element) for the parent
 1307 sch:rule, sch:pattern, or sch:schema elements remain valid and can be used in
 1308 the expression specified in the expression attribute.

1309 5.1.5 Examples

1310 The following example illustrates the use of smlerr:output

1311

```

1312
1313     <xs:simpleType name="IPAddressVersionType">
1314         <xs:restriction base="xs:string">
1315             <xs:enumeration value="V4"/>
1316             <xs:enumeration value="V6"/>
1317         </xs:restriction>
1318     </xs:simpleType>
1319     <xs:complexType name="IPAddressType">
1320         <xs:annotation>
1321             <xs:appinfo>
1322                 <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
1323                     <sch:ns prefix="tns" uri="urn:IPAddress" />
1324                     <sch:pattern id="Length">
1325                         <sch:rule context=".">
1326                             <sch:report id="v4" test="tns:version = 'V4'
1327                                     and count(tns:address) != 4"
1328                                     smlerr:outputids="IPXML">
1329
1330                                 A v4 IP address must have 4 bytes.
1331                             </sch:report>
1332                             <sch:report id="v6" test="tns:version = 'V6'
1333                                     and count(tns:address) != 16"
1334                                     smlerr:outputids="IPXML">
1335                                 A v6 IP address must have 16 bytes.
1336                             </sch:report>
1337                             <sml:output applicationUri="someApplicationUri"
1338                                     id="IPXML"
1339                                     expression=".">
1340
1341                                 </sml:output>
1342                             </sch:rule>
1343                         </sch:pattern>
1344                     </sch:schema>
1345                 </xs:appinfo>
1346             </xs:annotation>
1347         <xs:sequence>
1348             <xs:element name="version"
1349                     type="tns:IPAddressVersionType" />
1350             <xs:element name="address" type="xs:byte"
1351                     minOccurs="4" maxOccurs="16" />
1352         </xs:sequence>
1353     </xs:complexType>

```

1354 If the report with id="v4" fires for an element `ipaddress` of type `IPAddressType`,
1355 then the output may look like

1356

```

1357
1358 <smlerr:errorData
1359     xmlns:sml="http://schemas.serviceml.org/smlerr/2007/02">
1360   <ipaddress xmlns="urn:IPAddress">
1361     <version>v4</version>
1362     <address>10</address>
1363     <address>10</address>
1364     <address>20</address>
1365     <address>0</address>
1366     <address>0</address> </ipaddress>
1367 </smlerr:errorData>

```

1368

1369 The following example illustrates an `smlerr:output` element whose expression
1370 results in attribute nodes

```

1371 <xs:complexType name="universityType">
1372   <xs:annotation>
1373     <xs:appinfo>
1374       <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
1375         <sch:ns prefix="u" uri="urn:university" />
1376         <sch:pattern id="Count">
1377           <sch:rule context=".">
1378             <sch:assert id="StudentCount"
1379               test="count(u:student) > 20"
1380               smlerr:outputids="StudentXml">
1381               A university must have more than 20 students
1382             </sch:assert>
1383             <smlerr:output id="StudentXml"
1384               expression="@name|@isPublic">
1385
1386               </smlerr:output>
1387             </sch:rule>
1388           </sch:pattern>
1389         </sch:schema>
1390       </xs:appinfo>
1391     </xs:annotation>
1392   <xs:sequence>
1393     <xs:element name="student" type="sml:refType"
1394       minOccurs="0" maxOccurs="unbounded"/>
1395   </xs:sequence>
1396   <xs:attribute name="name" type="xs:string"/>
1397   <xs:attribute name="isPublic" type="xs:boolean"/>
1398 </xs:complexType>

```

1399

1400 If the assert fires for an element of type `universityType` then the output may look
1401 like

```

1402
1403 <smlerr:errorData
1404     xmlns:sml="http://schemas.serviceml.org/smlerr/2007/02">
1405     <smlerr:attributeNode xmlns:u="urn:university"
1406         name="u:name">MIT</smlerr:attributeNode>
1407     <smlerr:attributeNode xmlns:u="urn:university"
1408         name="u:isPublic">>false</smlerr:attributeNode>
1409 </smlerr:errorData>

```

1410 6. Model Validation

1411 Model validation is the process of examining each document in a model and verifying
1412 that this document is valid with respect to the model's definition documents, i.e.,
1413 each document satisfies the schemas and rules defined in the model's definition
1414 documents. All of the following MUST be true for a valid model:

- 1415 • Each document in the model MUST be a well-formed XML document [12]
- 1416 • Each XML Schema document in the model's definition documents MUST be a
1417 valid XML Schema document [2]
- 1418 • Each Schematron document in the model's definition documents MUST be a
1419 valid Schematron document [4]
- 1420 • Each document in the model MUST be XML Schema valid with respect to the
1421 XML Schema documents in the model's definition documents
- 1422 • Each document in the model MUST satisfy all applicable `sml:target*` and
1423 Schematron constraints
- 1424 • The model MUST NOT contain a cycle whose edges are references of type **R** if
1425 **R** is an acyclic reference type

1426 6.1 Schematron Phase

1427 A phase in schematron can be used to define a collection of patterns. A schematron
1428 processor can optionally evaluate only rules within a specific phase. For model
1429 validation, rule evaluation happens on the #ALL phase, implying that every rule in
1430 every pattern is evaluated.

1431 7. SML Extension Reference

1432 This section is a non-normative reference of the SML extensions to XML Schema and
1433 XPath 1.0.

1434 7.1 Types

1435 7.1.1 sml:refType

1436 A complex type representing a reference to an element.

```

1437
1438 <xs:complexType name="refType" sml:acyclic="false">
1439     <xs:sequence>
1440         <xs:any namespace="##any" minOccurs="0"
1441             maxOccurs="unbounded"
1442             processContents="lax"/>
1443     </xs:sequence>
1444     <xs:attribute ref="sml:ref" use="required"
1445         fixed="true" />

```

```
1446     <xs:anyAttribute namespace="##any"
1447                   processContents="lax"/>
1448 </xs:complexType>
```

1449 No specific scheme is mandated for representing references, and a model validator is
1450 free to choose any suitable scheme. However, each reference value must resolve to
1451 a single element. `sml:refType` can only be used with element declarations; it is not
1452 supported on attribute declarations.

1453 7.2 Attributes

1454 7.2.1 `sml:acyclic`

1455 Used to specify that a derived type of `sml:refType` is acyclic, i.e., its instances do
1456 not create any cycles in a model.

```
1457     <xs:attribute name="acyclic" type="xs:boolean"/>
```

1458 If this attribute is set to true for a derived type `D` of `sml:refType`, then instances of
1459 `D` (including any derived types of `D`) can not create any cycles in a model. More
1460 precisely, the directed graph whose nodes are documents that contain the source or
1461 target elements for instances of `D`, and whose edges are instances of `D` (an edge is
1462 directed from the document containing the source element to the document
1463 containing the target element), must be acyclic. A model is invalid if its documents
1464 result in a cyclic graph using instances of `D`. In the following example, `Hostref` is a
1465 restricted derived type of `sml:refType` and its instances can not create any cycles:

```
1466     <xs:complexType name="Hostref" sml:acyclic="true">
1467       <xs:complexContent>
1468         <xs:restriction base="sml:refType"/>
1469       </xs:complexContent>
1470     </xs:complexType>
```

1472 If the `sml:acyclic` attribute is not specified or set to false for a derived type of
1473 `sml:refType`, then instances of this reference type may create cycles in a model.
1474 Note that `sml:acyclic` is specified as "false" for `sml:refType`; hence its instances
1475 are allowed to create cycles in a model.

1477 7.2.2 `sml:ref`

1478 This global attribute is used to identify reference elements.

```
1479     <xs:attribute name="ref" type="xs:boolean"/>
```

1480 Any element that has `sml:ref="true"` will be treated as a reference element. Note
1481 that `sml:ref="true"` for all elements whose type is `sml:refType` or a derived type
1482 `sml:refType`.

1483

1484 7.2.3 `sml:targetElement`

1485 A `QName` representing the name of a referenced element

```
1486     <xs:attribute name="targetElement" type="xs:QName"/>
```

1487 `sml:targetElement` is supported as an attribute for element declarations whose
1488 type is `sml:refType` or a type derived by restriction from `sml:refType`. The value
1489 of this attribute must be the name of some global element declaration. Let
1490 `sml:targetElement="ns:GTE"` for some element declaration **E**. Then each element
1491 instance of **E** must target an element that is an instance of **ns:GTE** or an instance of

1492 some global element declaration in the substitution group hierarchy whose head is
1493 **ns:GTE**.

1494 In the following example, the element referenced by instances of `HostOS` must be
1495 instances of `win:Windows`

```
1496     <xs:element name="HostOS" type="sml:refType"  
1497               sml:targetElement="win:Windows"  
1498               minOccurs="0"/>
```

1499 A model is invalid if its documents violate one/more `sml:targetElement` constraints.

1500 7.2.4 `sml:targetRequired`

1501 Used to specify that instances of a reference element must target elements in the
1502 model, i.e., an instance of the reference element can not be empty or null, or contain
1503 a dangling reference which does not target any element in the model.

```
1504     <xs:attribute name="targetRequired" type="xs:boolean"/>
```

1505 In the following example, the `targetRequired` attribute is used to specify that
1506 application instances must have a host operating system.

```
1507     <xs:complexType name="ApplicationType">  
1508       <xs:sequence>  
1509         <xs:element name="Name" type="xs:string"/>  
1510         <xs:element name="Vendor" type="xs:string"/>  
1511         <xs:element name="Version" type="xs:string"/>  
1512         <xs:element name="HostOSRef" type="sml:refType"  
1513                   sml:targetRequired="true"/>  
1514       </xs:sequence>  
1515     </xs:complexType>
```

1516 A model is invalid if its documents violate one/more `sml:targetRequired`
1517 constraints.

1518 7.2.5 `sml:targetType`

1519 A `QName` representing the type of a referenced element

```
1520     <xs:attribute name="targetType" type="xs:QName">  
1521
```

1522 `sml:targetType` is supported as an attribute for element declarations whose type is
1523 `sml:refType` or a type derived by restriction from `sml:refType`. If the value of this
1524 attribute is specified as `T`, then the type of the referenced element must either be `T`
1525 or a derived type of `T`. In the following example, the type of the element referenced
1526 by the `OperatingSystem` element must be `"ibm:LinuxType"` or its derived type

```
1527     <xs:element name="OperatingSystem" type="sml:refType"  
1528               sml:targetType="ibm:LinuxType"  
1529               minOccurs="0"/>
```

1530 A model is invalid if its documents violate one/more `sml:targetType` constraints.

1531 7.3 Elements

1532 7.3.1 `sml:key`

1533 This element is used to specify a key constraint in some scope. The semantics are
1534 essentially the same as that for `xs:key` but `sml:key` can also be used to specify key
1535 constraints on other documents, i.e., the `sml:selector` child element of `sml:key`
1536 can contain `deref` functions to resolve elements in another document.

```
1537     <xs:element name="key" type="sml:keybase"/>
```

1538 `sml:key` is supported in the `appinfo` of an `xs:element`.

1539 **7.3.2 sml:keyref**

1540 Applies a constraint in the context of the containing `xs:element` that scopes the range
1541 of a nested document reference.

```
1542     <xs:element name="keyref">
1543       <xs:complexType>
1544         <xs:complexContent>
1545           <xs:extension base="sml:keybase">
1546             <xs:attribute name="refer" type="xs:QName" use="required"/>
1547           </xs:extension>
1548         </xs:complexContent>
1549       </xs:complexType>
1550     </xs:element>>
```

1551 `sml:keyref` is supported in the `appinfo` of an `xs:element`.

1552 **7.3.3 sml:unique**

1553 This element is used to specify a uniqueness constraint in some scope. The
1554 semantics are essentially the same as that for `xs:unique` but `sml:unique` can also
1555 be used to specify uniqueness constraints on other documents, i.e., the
1556 `sml:selector` child element of `sml:unique` can contain `deref` functions to resolve
1557 elements in another document.

```
1558     <xs:element name="unique" type="sml:keybase"/>
```

1559 `sml:unique` is supported in the `appinfo` of an `xs:element`.

1560 **7.3.4 sml:uri**

1561 Specifies a reference in URI scheme.

```
1562     <xs:element name="uri" type="xs:anyURI"/>
```

1563 This element must be used to specify references that use the URI scheme.

1564 **7.4 XPath functions**

1565 **7.4.1 smlfn:deref**

1566 `node-set deref(node-set)`

1567 This function takes a `node-set` of elements and attempts to resolve the references
1568 contained in the elements that have `sml:ref="true"`. The resulting `node-set` is
1569 the set of elements that are obtained by successfully resolving (or de-referencing)
1570 the reference contained in each element in the input `node-set` for which
1571 `sml:ref="true"`. For example,

```
1572     deref(/u:Universities/u:Students/u:Student)
```

1573 will resolve the reference in element `Student`. The target of the reference must always be
1574 an element.

1575 **8. Open Issues**

- 1576 • What should be the semantics of the `smlfn:deref()` function if a reference
1577 element has multiple child elements that represent the reference using the
1578 same scheme? E.g.,

```
1579     <EnrolledCourse xmlns="urn:university" sml:ref="true">
```

```
1580     <sml:uri>SomeValidUri</sml:uri>
1581     <sml:uri>AnotherValidUri</sml:uri>
1582 </EnrolledCourse>
```

- 1583 • Do we need to support an `sml:phase` attribute (similar to the `phase` attribute
1584 in Schematron) that can be used for selective validation of SML constraints?
1585 Should this be extended to apply to XML Schema constraints?
- 1586 • Implementation of references that target non-root elements is challenging for
1587 persistent SML stores built on top of relational database systems. The
1588 standards body to which SML gets submitted should investigate this and, if
1589 needed, explore options to ease the implementation burden for persistent
1590 SML stores using relational databases. Possible areas of investigation are
 - 1591 ○ Restricting the XPath expression in XPointer fragment (similar to XPath
1592 expressions allowed in `xs:selector`)
 - 1593 ○ Defining a new scheme for URIs that contain XPointer fragment
- 1594 • Implementation of SML identity constraints that use the `smlfn:deref()`
1595 function in `sml:field/@xpath` expressions is challenging for persistent SML
1596 stores built on top of relational database systems. The standards body to
1597 which SML gets submitted should investigate this and, if needed, explore
1598 options to ease the implementation burden for persistent SML stores using
1599 relational databases.

1600 **9. Acknowledgements**

1601 Thanks to the following individuals for providing valuable feedback on this
1602 specification:

1603 Don Box, Ray McCollum, Ted Miller and Jeff Parham (Microsoft)

1604 Chris Ferris and Sandy Gao (IBM)

1605 Matt Newman and Virginia Smith (HP)

1606 Johan Van De Groenendaal (Intel)

1607 Gene Golovinsky (formerly at BMC)

1608 John Tollefsrud (Sun)

1609 Drue Reeves (formerly at Dell)

1610 **10. References**

1611 **[1]** XML Schema Part 0: Primer (<http://www.w3.org/TR/xmlschema-0>)

1612 **[2]** XML Schema Part 1: Structures Second Edition

1613 (<http://www.w3.org/TR/xmlschema-1>)

1614 **[3]** XML Schema Part 2: Datatypes Second Edition

1615 (<http://www.w3.org/TR/xmlschema-2>)

1616 **[4]** Document Schema Definition Language (DSDL) – Part 3: Rule-based validation –
1617 Schematron

1618 ([http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip))

1620 **[5]** An Introduction to Schematron

1621 (<http://www.xml.com/pub/a/2003/11/12/schematron.html>)

1622 **[6]** Improving XML Document Validation with Schematron

1623 ([http://msdn.microsoft.com/library/default.asp?url=/library/en-](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxml/html/schematron.asp)
1624 [us/dnxml/html/schematron.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxml/html/schematron.asp))
1625 **[7]** Uniform Resource Identifier (URI): Generic Syntax
1626 (<http://www.ietf.org/rfc/rfc3986.txt>)
1627 **[8]** Web Services Addressing (<http://www.w3.org/TR/ws-addr-core>)
1628 **[9]** XML Path Language (XPath) Version 1.0 (<http://www.w3.org/TR/xpath>)
1629 **[10]** XPointer (<http://www.w3.org/TR/xptr/>)
1630 **[11]** XPointer xpointer() Scheme (<http://www.w3.org/TR/xptr-xpointer/>)
1631 **[12]** Extensible Markup Language (XML) 1.0 (<http://www.w3.org/TR/REC-xml/>)
1632 [13] Key Words for Use in RFCs to Indicate Requirement Levels
1633 (<http://www.ietf.org/rfc/rfc2119.txt>)
1634

1635 **Appendix I – Normative SML Schema**

1636

```
1637 <?xml version="1.0" encoding="utf-8"?>
```

```
1638 <!-- Copyright (c) 2006 by BEA, BMC, CA, Cisco, Dell, EMC, HP, IBM,
1639 Intel, Microsoft, and Sun. All rights reserved. -->
```

```
1640 <!-- Permission to copy, display, and distribute this Service Modeling
1641 Language (SML) Schema Document, in any medium without fee or royalty is
1642 hereby granted, provided that you include the following on ALL copies
1643 of the SML Schema Document, or portions thereof, that you make:
```

1644

```
1645 1. A link or URL to the SML Schema Document at this location:
```

```
1646 http://schemas.serviceml.org/sml/2007/02/sml.xsd
```

1647

```
1648 2. The copyright notice as shown in the SML Schema Document.
```

1649

```
1650 BEA, BMC, CA, Cisco, Dell, EMC, HP, IBM, Intel, Microsoft, and Sun
1651 (collectively, the "Authors") each agree to grant you a royalty-free
1652 license, under reasonable, non-discriminatory terms and conditions to
1653 their respective patents that they deem necessary to implement the
1654 Service Modeling Language Schema Document.
```

1655

```
1656 THE SML SCHEMA DOCUMENT IS PROVIDED "AS IS," AND THE AUTHORS MAKE NO
1657 REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT
1658 LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
1659 PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE SML
1660 SCHEMA DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE
1661 IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY
1662 PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
```

```
1663 THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL,
1664 INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY
1665 USE OR DISTRIBUTION OF THE SML SCHEMA DOCUMENT.
```

1666

```
1667 The name and trademarks of the Authors may NOT be used in any manner,
1668 including advertising or publicity pertaining to the SML Schema
1669 Document or its contents without specific, written prior permission.
1670 Title to copyright in the SML Schema Document will at all times remain
1671 with the Authors. -->
```

1672

```
1673 <xs:schema xmlns:sml="http://schemas.serviceml.org/sml/2007/02"
1674           xmlns:xs="http://www.w3.org/2001/XMLSchema"
1675           targetNamespace="http://schemas.serviceml.org/sml/2007/02"
1676           elementFormDefault="qualified"
1677           blockDefault="#all"
1678           version="1.0"
1679           xml:lang="EN">
```

1680

```
1681 <!--
1682   References
1683   =====
```

1684

```
1685 -->
```

```
1684 <xs:complexType name="refType" sml:acyclic="false">
```

1685

```
1685 <xs:annotation>
```

1686

```
1686 <xs:documentation>
```

1687

```
1687 A complex type representing a reference to an element
1688 in the same or a different document. No specific scheme is
1689 mandated for references; an implementation is free to
```

```

1690         choose an appropriate scheme such as URI, EPR, etc.
1691         The target of the reference must unambiguously identify a
1692         single element.
1693     </xs:documentation>
1694 </xs:annotation>
1695 <xs:sequence>
1696     <xs:any namespace="##any" processContents="lax" minOccurs="0"
1697         maxOccurs="unbounded"/>
1698 </xs:sequence>
1699 <xs:attribute ref="sml:ref" use="required" fixed="true"/>
1700 <xs:anyAttribute namespace="##any" processContents="lax"/>
1701 </xs:complexType>
1702 <!-- CONTEXT: To be used in <xs:element> -->
1703 <xs:attribute name="ref" type="xs:boolean">
1704     <xs:annotation>
1705         <xs:documentation>
1706             Specifies if the element contains a reference
1707         </xs:documentation>
1708     </xs:annotation>
1709 </xs:attribute>
1710 <!-- CONTEXT: To be used in <xs:element> where type="sml:refType"
1711 -->
1712 <xs:attribute name="targetElement" type="xs:QName">
1713     <xs:annotation>
1714         <xs:documentation>
1715             A qualified name of an element in the
1716             referenced document.
1717         </xs:documentation>
1718     </xs:annotation>
1719 </xs:attribute>
1720 <!-- CONTEXT: To be used in <xs:element> where type="sml:refType"
1721 -->
1722 <xs:attribute name="targetRequired" type="xs:boolean">
1723     <xs:annotation>
1724         <xs:documentation>
1725             If true, requires the target element of the reference to
1726             exist in the model.
1727         </xs:documentation>
1728     </xs:annotation>
1729 </xs:attribute>
1730 <!-- CONTEXT: To be used in <xs:element> where type="sml:refType"
1731 -->
1732 <xs:attribute name="targetType" type="xs:QName">
1733     <xs:annotation>
1734         <xs:documentation>
1735             A qualified name of the type of the element in the
1736             referenced document.
1737         </xs:documentation>
1738     </xs:annotation>
1739 </xs:attribute>
1740 <!-- CONTEXT: To be used in sml:refType and its derived types-->
1741 <xs:attribute name="acyclic" type="xs:boolean">
1742     <xs:annotation>
1743         <xs:documentation>
1744             If this attribute is set to true for a derived type D of
1745             sml:refType, then instances of D should not create any
1746             cycles in a model. More precisely, the directed graph whose

```

```

1747         edges represent instances of D, and whose nodes represent
1748         documents that contain the source or target elements for
1749         instances of D, must be acyclic.
1750     </xs:documentation>
1751 </xs:annotation>
1752 </xs:attribute>
1753 <!-- CONTEXT: Represents a reference using the URI scheme. To be
1754         used as a child element of elements for which
1755         sml:ref="true". -->
1756 <xs:element name="uri" type="xs:anyURI">
1757     <xs:annotation>
1758         <xs:documentation>
1759             References in URI scheme must be representend by this
1760             element.
1761         </xs:documentation>
1762     </xs:annotation>
1763 </xs:element>
1764 <!--
1765     Uniqueness and Key constraints
1766     =====
1767 -->
1768 <xs:complexType name="keybase">
1769     <xs:sequence minOccurs="0">
1770         <xs:element name="selector" type="sml:selectorXPathType"/>
1771         <xs:element name="field" type="sml:fieldXPathType"
1772             minOccurs="0" maxOccurs="unbounded"/>
1773         <xs:any namespace="##other" minOccurs="0"
1774             maxOccurs="unbounded" processContents="lax"/>
1775     </xs:sequence>
1776     <xs:attribute name="name" type="xs:NCName"/>
1777     <xs:attribute name="ref" type="xs:QName"/>
1778     <xs:anyAttribute namespace="##other" processContents="lax"/>
1779 </xs:complexType>
1780 <xs:element name="key" type="sml:keybase"/>
1781 <xs:element name="unique" type="sml:keybase"/>
1782 <xs:element name="keyref">
1783     <xs:complexType>
1784         <xs:complexContent>
1785             <xs:extension base="sml:keybase">
1786                 <xs:attribute name="refer" type="xs:QName"
1787                     use="required"/>
1788             </xs:extension>
1789         </xs:complexContent>
1790     </xs:complexType>
1791 </xs:element>
1792 <!--
1793     Other Complex Types
1794     =====
1795 -->
1796
1797 <xs:complexType name="selectorXPathType">
1798     <xs:sequence>
1799         <xs:any namespace="##other" minOccurs="0"
1800             maxOccurs="unbounded" processContents="lax"/>
1801     </xs:sequence>
1802     <xs:attribute name="xpath" use="required">
1803         <xs:simpleType>

```

```
1804         <xs:restriction base="xs:string">
1805             <!-- TODO: add a pattern facet for selector xpath -->
1806         </xs:restriction>
1807     </xs:simpleType>
1808 </xs:attribute>
1809     <xs:anyAttribute namespace="##other" processContents="lax"/>
1810 </xs:complexType>
1811
1812 <xs:complexType name="fieldXPathType">
1813     <xs:sequence>
1814         <xs:any namespace="##other" minOccurs="0"
1815             maxOccurs="unbounded" processContents="lax"/>
1816     </xs:sequence>
1817     <xs:attribute name="xpath" use="required">
1818         <xs:simpleType>
1819             <xs:restriction base="xs:string">
1820                 <!-- TODO: add a pattern facet for field xpath -->
1821             </xs:restriction>
1822         </xs:simpleType>
1823     </xs:attribute>
1824     <xs:anyAttribute namespace="##other" processContents="lax"/>
1825 </xs:complexType>
1826 </xs:schema>
1827
```

1828 **Appendix II – Normative SML Error Schema**

1829

1830 `<?xml version="1.0" encoding="utf-8"?>`

1831 `<!-- Copyright (c) 2006 by BEA, BMC, CA, Cisco, Dell, EMC, HP, IBM,`
1832 `Intel, Microsoft, and Sun. All rights reserved. -->`

1833 `<!-- Permission to copy, display, and distribute this Service Modeling`
1834 `Language (SML) Error Schema Document, in any medium`
1835 `without fee or royalty is hereby granted, provided that you include the`
1836 `following on ALL copies of the SML Error Schema`
1837 `Document, or portions thereof, that you make:`

1838

1839 `1. A link or URL to the SML Error Schema Document at this location:`

1840 `http://schemas.serviceml.org/smlerr/2007/02/smlerr.xsd`

1841

1842 `2. The copyright notice as shown in the SML Schema Document.`

1843

1844 `BEA, BMC, CA, Cisco, Dell, EMC, HP, IBM, Intel, Microsoft, and Sun`
1845 `(collectively, the "Authors") each agree to grant you`
1846 `a royalty-free license, under reasonable, non-discriminatory terms and`
1847 `conditions to their respective patents that they`
1848 `deem necessary to implement the Service Modeling Language Error Schema`
1849 `Document.`

1850

1851 `THE SML ERROR SCHEMA DOCUMENT IS PROVIDED "AS IS," AND THE AUTHORS MAKE`
1852 `NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED,`
1853 `INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS`
1854 `FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE;`
1855 `THAT THE CONTENTS OF THE SML ERROR SCHEMA DOCUMENT ARE SUITABLE FOR ANY`
1856 `PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS`
1857 `WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR`
1858 `OTHER RIGHTS.`

1859 `THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL,`
1860 `INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING`
1861 `TO ANY USE OR DISTRIBUTION OF THE SML ERROR SCHEMA DOCUMENT.`

1862

1863 `The name and trademarks of the Authors may NOT be used in any manner,`
1864 `including advertising or publicity pertaining to the SML`
1865 `Error Schema Document or its contents without specific, written prior`
1866 `permission. Title to copyright in the SML Error Schema`
1867 `Document will at all times remain with the Authors. -->`

1868

1869

1870 `<xs:schema`

1871 `xmlns:smlerr="http://schemas.serviceml.org/smlerr/2007/02"`

1872 `xmlns:xs="http://www.w3.org/2001/XMLSchema"`

1873 `targetNamespace="http://schemas.serviceml.org/smlerr/2007/02"`

1874 `elementFormDefault="qualified"`

1875 `blockDefault="#all"`

1876 `version="1.0"`

1877 `xml:lang="EN">`

1878

1879 `<xs:element name="errorData" type="smlerr:errorDataType"/>`

1880 `<xs:element name="output" type="smlerr:outputType"/>`

1881 `<xs:element name="attributeNode" type="smlerr:attributeNodeType"/>`

1882

```
1883 <xs:attribute name="outputids" type="xs:IDREFS"/>
1884 <xs:attribute name="localizationid" type="xs:anyURI"/>
1885
1886 <xs:complexType name="outputType">
1887   <xs:attribute name="id" type="xs:ID" use="required"/>
1888   <xs:attribute name="applicationUri" type="xs:anyURI"/>
1889   <xs:attribute name="expression" type="xs:string" use="required"/>
1890 </xs:complexType>
1891
1892 <xs:complexType name="attributeNodeType">
1893   <xs:simpleContent>
1894     <xs:extension base="xs:string">
1895       <xs:attribute name="name" type="xs:QName"/>
1896     </xs:extension>
1897   </xs:simpleContent>
1898 </xs:complexType>
1899
1900 <xs:complexType name="errorDataType">
1901   <xs:sequence>
1902     <xs:any namespace="##any" processContents="skip"/>
1903   </xs:sequence>
1904 </xs:complexType>
1905 </xs:schema>
```

1906

1907 **Appendix III – Sample Model**

1908 This sample model illustrates the use of the following SML extensions:

- 1909 • Inter-document references
- 1910 • `key` and `keyref` constraints
- 1911 • User-defined constraints

```
1912 <?xml version="1.0" encoding="utf-8" ?>
1913 <xs:schema targetNamespace="SampleModel"
1914           elementFormDefault="qualified"
1915           xmlns:tns="SampleModel"
1916           xmlns:sml="http://schemas.serviceml.org/sml/2007/02"
1917           xmlns:smlfn=
1918             "http://schemas.serviceml.org/sml/function/2006/07"
1919           xmlns:sch="http://purl.oclc.org/dsdl/schematron"
1920           xmlns:xs="http://www.w3.org/2001/XMLSchema">
1921
1922   <xs:import namespace="http://schemas.serviceml.org/sml/2007/02"/>
1923
1924   <xs:simpleType name="SecurityLevel">
1925     <xs:restriction base="xs:string">
1926       <xs:enumeration value="Low"/>
1927       <xs:enumeration value="Medium"/>
1928       <xs:enumeration value="High"/>
1929     </xs:restriction>
1930   </xs:simpleType>
1931
1932   <xs:complexType name="Hostref" sml:acyclic="true">
1933     <xs:complexContent>
1934       <xs:restriction base="sml:refType"/>
1935     </xs:complexContent>
1936   </xs:complexType>
1937
1938   <!-- This element represents the host operating system for
1939        an application. Note that the type of the referenced
1940        element must be OperatingSystemType or a derived type
1941        of OperatingSystemType -->
1942   <xs:element name="HostOSRef" type="tns:Hostref"
1943             sml:targetType="tns:OperatingSystemType"/>
1944
1945   <xs:complexType name="ApplicationType">
1946     <xs:sequence>
1947       <xs:element name="Name" type="xs:string"/>
1948       <xs:element name="Vendor" type="xs:string"/>
1949       <xs:element name="Version" type="xs:string"/>
1950       <xs:element ref="tns:HostOSRef" minOccurs="0"/>
1951     </xs:sequence>
1952   </xs:complexType>
1953
```

```

1954
1955 <xs:simpleType name="ProtocolType">
1956   <xs:list>
1957     <xs:simpleType>
1958       <xs:restriction base="xs:string">
1959         <xs:enumeration value="TCP"/>
1960         <xs:enumeration value="UDP"/>
1961         <xs:enumeration value="SMTP"/>
1962         <xs:enumeration value="SNMP"/>
1963       </xs:restriction>
1964     </xs:simpleType>
1965   </xs:list>
1966 </xs:simpleType>
1967
1968 <xs:element name="GuestAppRef" type="sml:refType"
1969           sml:targetType="tns:ApplicationType"/>
1970
1971 <xs:complexType name="OperatingSystemType">
1972   <xs:sequence>
1973     <xs:element name="Name" type="xs:string"/>
1974     <xs:element name="FirewallEnabled" type="xs:boolean"/>
1975     <xs:element name="Protocol" type="tns:ProtocolType"/>
1976     <!-- The following element represents the applications hosted by
1977           operating system -->
1978     <xs:element name="Applications" minOccurs="0">
1979       <xs:complexType>
1980         <xs:sequence>
1981           <xs:element ref="tns:GuestAppRef" maxOccurs="unbounded"/>
1982         </xs:sequence>
1983       </xs:complexType>
1984     </xs:element>
1985   </xs:sequence>
1986 </xs:complexType>
1987
1988 <xs:element name="OSRef" type="sml:refType"
1989           sml:targetType="tns:OperatingSystemType"/>
1990
1991 <xs:complexType name="WorkstationType">
1992   <xs:sequence>
1993     <xs:element name="Name" type="xs:string"/>
1994     <xs:element ref="tns:OSRef"/>
1995     <xs:element name="Applications" minOccurs="0">
1996       <xs:complexType>
1997         <xs:sequence>
1998           <xs:element ref="tns:GuestAppRef" maxOccurs="unbounded"/>
1999         </xs:sequence>
2000       </xs:complexType>
2001     </xs:element>
2002   </xs:sequence>
2003 </xs:complexType>
2004

```

```

2005
2006 <xs:element name="Workstation" type="tns:WorkstationType">
2007   <xs:annotation>
2008     <xs:appinfo>
2009       <sch:schema>
2010         <sch:ns prefix="sm" uri="SampleModel"/>
2011         <sch:ns prefix="smlfn"
2012           uri="http://schemas.serviceml.org/sml/function/2006/07"/>
2013         <sch:pattern id="OneHostOS">
2014           <!-- The constraints in the following rule are evaluated
2015             For all instances of the Workstation global element-->
2016           <sch:rule context=".">
2017             <!-- define a named variable - MyApplications -
2018               for use in test expression-->
2019             <sch:let name="MyApplications"
2020               value="smlfn:deref(sm:Applications/sm:GuestAppRef)"/>
2021             <sch:assert test=
2022               "count($MyApplications)=
2023                 count($MyApplications/sm:HostOSRef)">
2024               Each application in workstation
2025               <sch:value-of select="string(sm:Name)"/>
2026               must be hosted on an operating system
2027             </sch:assert>
2028           </sch:rule>
2029         </sch:pattern>
2030       </sch:schema>
2031
2032       <!-- In a workstation, (Vendor,Name,Version) is the key for
2033         guest applications -->
2034       <sml:key name="GuestApplicationKey">
2035         <sml:selector
2036           xpath="smlfn:deref(tns:Applications/tns:GuestAppRef)"/>
2037         <sml:field xpath="tns:Vendor"/>
2038         <sml:field xpath="tns:Name"/>
2039         <sml:field xpath="tns:Version"/>
2040       </sml:key>
2041
2042       <!-- In a workstation, Name is the key for operating system -->
2043       <sml:key name="OSKey">
2044         <sml:selector xpath="smlfn:deref(tns:OSRef)"/>
2045         <sml:field xpath="tns:Name"/>
2046       </sml:key>
2047
2048       <!-- In a workstation, the applications hosted by the
2049         referenced operatinsystem must be a subset of the
2050         applications in the workstation -->
2051       <sml:keyref name="OSGuestApplication"
2052         refer="tns:GuestApplicationKey">
2053         <sml:selector xpath=
2054           "smlfn:deref(tns:OSRef)/tns:Applications/tns:GuestAppRef"
2055         />
2056         <sml:field xpath="tns:Vendor"/>
2057         <sml:field xpath="tns:Name"/>
2058         <sml:field xpath="tns:Version"/>
2059       </sml:keyref>
2060
2061

```

```

2062     <!-- In a workstation, the host operating system of guest
2063           applications must be a subset of the operating system in
2064           the workstation -->
2065     <sml:keyref name="ApplicationHostOS" refer="tns:OSKey">
2066       <sml:selector xpath=
2067         "smlfn:deref(tns:Applications/tns:GuestAppRef)/tns:HostOSRef"
2068       />
2069
2070       <sml:field xpath="tns:Name"/>
2071     </sml:keyref>
2072
2073   </xs:appinfo>
2074 </xs:annotation>
2075 </xs:element>
2076
2077 <xs:element name="SecureWorkstation" type="tns:WorkstationType">
2078   <xs:annotation>
2079     <xs:appinfo>
2080       <sch:schema>
2081         <sch:ns prefix="sm" uri="SampleModel" />
2082         <sch:ns prefix="smlfn"
2083           uri="http://schemas.serviceml.org/sml/function/2006/07"
2084         />
2085         <sch:pattern id="SecureApplication">
2086           <sch:rule
2087             context="smlfn:deref(sm:Applications/sm:Application)">
2088             <sch:report test="sm:SecurityLevel!='High'">
2089               Application <sch:value-of select="string(sm:Name)"/>
2090               from <sch:value-of select="string(sm:Vendor)"/>
2091               does not have high security level
2092             </sch:report>
2093             <sch:assert test="sm:Vendor='TrustedVendor'">
2094               A secure workstation can only contain
2095               applications from TrustedVendor
2096             </sch:assert>
2097           </sch:rule>
2098         </sch:pattern>
2099       </sch:schema>
2100     </xs:appinfo>
2101   </xs:annotation>
2102 </xs:element>
2103
2104 </xs:schema>
2105
2106

```