

# Service Modeling Language

## Draft Specification

Version 0.5, 25 July 2006

### Authors

Pratul Dubish, Microsoft  
Zulah Eckert, BEA  
Dave Ehnebuske, IBM  
Eugene Golovinsky, BMC  
Steve Jerman, Cisco  
Heather Kreger, IBM  
Milan Milenkovic, Intel  
Bryan Murray, HP  
Phil Prasek, HP  
Drue Reeves, Dell  
Junaid Saiyed, EMC  
Harm Sluiman, IBM  
Bassam Tabbara, Microsoft  
Vijay Tewari, Intel  
John Tollefsrud, Sun  
William Vambenepe, HP  
Andrea Westerinen, Microsoft

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

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

<http://www.intel.com/technology/manage>

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

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

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

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

## **Abstract**

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

## **Status**

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

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

# Table of Contents

<b>Service Modeling Language</b> .....	<b>1</b>
<b>Draft Specification</b> .....	<b>1</b>
Abstract .....	3
Status .....	3
Table of Contents .....	4
<b>1. Introduction</b> .....	<b>6</b>
<b>2. Terminology and Notation</b> .....	<b>7</b>
2.1 Terminology.....	7
2.2 XML Namespaces .....	7
<b>3. Schemas</b> .....	<b>8</b>
3.1 XML Schema Profile.....	8
3.1.1 <xs:redefine> .....	8
3.1.2 Unqualified Local Elements .....	8
3.1.3 targetNamespace on <xs:schema>.....	9
3.2 References.....	9
3.2.1 Reference Semantics .....	10
3.3 Reference Schemes.....	11
3.3.1 URI Scheme .....	11
3.3.2 EPR Scheme .....	13
3.4 Constraints on References.....	14
3.4.1 sml:acyclic .....	14
3.4.2 sml:targetElement.....	15
3.4.3 sml:targetType .....	15
3.5 Identity Constraints .....	16
3.5.1 University Example.....	16
3.5.2 sml:key and sml:unique .....	18
3.5.3 sml:keyref.....	19
<b>4. Rules</b> .....	<b>19</b>
4.1 Schematron Profile.....	23
4.1.1 Limited Support .....	23
<b>5. Model Validation</b> .....	<b>23</b>
5.1 Schematron Phase .....	23
<b>6. SML Extension Reference</b> .....	<b>23</b>
6.1 Types.....	23
6.1.1 sml:ref .....	23
6.2 Attributes .....	24
6.2.1 sml:acyclic .....	24
6.2.2 sml:targetElement.....	24
6.2.3 sml:targetType .....	24
6.2.4 sml:uri.....	25
6.3 Elements .....	25

6.3.1 sml:key .....	25
6.3.2 sml:unique .....	25
6.3.3 sml:keyref.....	25
6.4 XPath functions .....	26
6.4.1 smlfn:deref .....	26
<b>7. Acknowledgements .....</b>	<b>26</b>
<b>8. References .....</b>	<b>26</b>
<b>Appendix I – Sample Model .....</b>	<b>27</b>
<b>Appendix II – Complexity of Supporting targetElement and targetType on Local Element Declarations .....</b>	<b>31</b>

# 1. Introduction

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

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

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

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

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

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

## 2. Terminology and Notation

### 2.1 Terminology

Document

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

Model

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

Rule

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

Genic Documents

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

Phenic Documents

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

Model Validation

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

Model Validator

An embodiment capable of performing model validation

### 2.2 XML Namespaces

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

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

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

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

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

### 3. Schemas

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

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

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

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

#### 3.1 XML Schema Profile

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

##### 3.1.1 <xs:redefine>

`xs:redefine` is not supported in SML.

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

##### 3.1.2 Unqualified Local Elements

Unqualified local elements are not supported in SML.

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

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

### 3.1.3 targetNamespace on <xs:schema>

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

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

## 3.2 References

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

Support for inter-document references includes:

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

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

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

```
<xs:complexType name="ref"
  sml:acyclic="false"
  final="extension">
  <xs:sequence>
    <xs:any namespace="##any" minOccurs="0"
      maxOccurs="unbounded"
      processContents="lax"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

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

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

```

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

<xs:complexType name="StudentType">
  <xs:sequence>
    <xs:element name="ID" type="xs:string"/>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="EnrolledCourses" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:EnrolledCourse"
                      maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

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

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

### 3.2.1 Reference Semantics

#### 3.2.1.1 At Most One Target

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

#### 3.2.1.2 Multiple References

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

#### 3.2.1.3 Empty or Null References

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

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

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

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

### 3.3 Reference Schemes

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

#### 3.3.1 URI Scheme

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

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

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

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

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

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

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

```
<Student xmlns="urn:university">
  <ID>1000</ID>
  <Name>John Doe</Name>
  <EnrolledCourses>
    <EnrolledCourse sml:uri="/Universities/MIT/Courses/PHY101.xml"/>
    <EnrolledCourse sml:uri="/Universities/MIT/Courses/MAT200.xml"/>
  </EnrolledCourses>
</Student>
```

### 3.3.1.1 Fragment Identifier

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

- Only two schemes – `xmlns()` and `xpointer()` – are supported.
- The expression specified for the `xpointer` scheme must be a restricted XPath1.0 [9] expression that must resolve to at most one element node. In particular, this expression must not contain
  - the union (“|”) operator defined for XPath 1.0
  - `point()` and `range()` node tests defined for `xpointer()` scheme
- This expression can only use the functions defined in the XPath 1.0 core function library (see [9] for details). It can not use the `smlfn:deref` function and/or the following functions defined for `xpointer()` scheme (see [11] for details):
  - `range-to`
  - `string-range`
  - `range`
  - `range-inside`
  - `start-point`
  - `end-point`
  - `here`
  - `origin`

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

```
<Student xmlns="urn:university">
  <ID>1000</ID>
  <Name>John Doe</Name>
  <EnrolledCourses>
    <EnrolledCourse
      sml:uri="/Universities/MIT/Courses.xml#xmlns(u=urn:university)
      xpointer(/u:Courses/u:Course[u:Name='PHY101'])"
    </EnrolledCourse>
  </EnrolledCourses>
</Student>
```

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

```
<University xmlns="urn:university">
  <Name>MIT</Name>
  <Courses>
    <Course>
      <Name>PHY101</Name>
    </Course>
    <Course>
      <Name>MAT200</Name>
    </Course>
  </Courses>
  <Students>
    <Student>
      <ID>123</ID>
      <Name>Jane Doe</Name>
      <EnrolledCourses>
        <EnrolledCourse
          sml:uri="#xmlns(u=urn:university)
          xpointer(/u:University/u:Courses/u:Course[u:Name='MAT200']"
        </EnrolledCourse>
      </EnrolledCourses>
    </Student>
  </Students>
</University>
```

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

### 3.3.2 EPR Scheme

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

```
<EnrolledCourse xmlns="urn:university">
  <wsa:EndpointReference
    xmlns:u="http://www.university.example/schema">
    <wsa:Address>http://www.university.example</wsa:Address>
    <wsa:ReferenceParameters>
      <u:University>
        <u:Name>MIT</u:Name>
      </u:University>
      <u:Course>
        <u:Name>PHY101</u:Name>
      </u:Course>
    </wsa:ReferenceParameters>
  </wsa:EndpointReference>
</EnrolledCourse>
```

### 3.4 Constraints on References

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

#### Attributes

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

#### 3.4.1 `sml:acyclic`

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

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

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

### 3.4.2 sml:targetElement

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

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

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

### 3.4.3 sml:targetType

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

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

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

sml:targetType for **L<sub>R</sub>** must be **ns:T** or the name of some derived type of **ns:T**. If sml:targetType is not specified for **L<sub>R</sub>**, then sml:targetType="ns:T" holds for **L<sub>R</sub>**.

### 3.5 Identity Constraints

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

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

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

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

- These three elements are only supported in the xs:annotation/xs:appinfo element for element declarations (both global and local). They can not be a child of an xs:element element
- The value of the xpath attribute of the sml:selector and sml:field elements (which are child elements of these three elements) can contain the smlfn:deref extension function
- The selector XPath expression must conform to the following extended BNF

```
Selector ::= Path ( '|' Path)*
Path ::= ( './../')? Step ( '/' Step)* | DerefExpr
DerefExpr ::= 'deref(' Step (/Step)* ')' ( '/' Step)* |
             'deref(' DerefExpr ')' (/Step)*
Step ::= '.' | NameTest
NameTest ::= QName | '*' | NCName ':' '*'
```

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

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

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

#### 3.5.1 University Example

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

```

<xs:element name="Student"
  type="sml:ref"
  sml:targetType="tns:StudentType" />

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

<xs:complexType name="UniversityType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string" />
    <xs:element name="Students" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:Student" maxOccurs="unbounded" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Courses" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:Course" maxOccurs="unbounded" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:element name="EnrolledStudent"
  type="sml:ref"
  sml:targetType="tns:StudentType" />

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

<xs:complexType name="StudentType">
  <xs:sequence>
    <xs:element name="ID" type="xs:string" />
    <xs:element name="SSN" type="xs:string" minOccurs="0" />
    <xs:element name="Name" type="xs:string" />
    <xs:element name="EnrolledCourses" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:EnrolledCourse"
            maxOccurs="unbounded" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="CourseType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="EnrolledStudents" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:EnrolledStudent"
            maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

### 3.5.2 sml:key and sml:unique

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

```

<xs:element name="University" type="tns:UniversityType">
  <xs:annotation>
    <xs:appinfo>
      <sml:key name="StudentIDisKey">
        <sml:selector xpath="smlfn:deref(tns:Students/tns:Student)/tns:ID"/>
        <sml:field xpath="."/>
      </sml:key>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

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

```

<xs:element name="University" type="tns:UniversityType">
  <xs:annotation>
    <xs:appinfo>
      <sml:unique name="StudentSSNisUnique">
        <sml:selector xpath="smlfn:deref(tns:Students/tns:Student)"/>
        <sml:field xpath="tns:SSN"/>
      </sml:unique>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

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

### 3.5.3 `sml:keyref`

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

```
<xs:element name="University" type="tns:UniversityType">
  <xs:annotation>
    <xs:appinfo>
      <sml:key name="StudentIDisKey">
        <sml:selector xpath="smlfn:deref(tns:Students/tns:Student)"/>
        <sml:field xpath="tns:ID"/>
      </sml:key>
      <sml:keyref name="CourseStudents" refer="StudentIDisKey">
        <sml:selector xpath="smlfn:deref(
          smlfn:deref(tns:Courses/tns:Course)/
          tns:EnrolledStudents/tns:EnrolledStudent)"/>
        <sml:field xpath="tns:ID"/>
      </sml:keyref>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

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

## 4. Rules

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

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

- An IPv4 address must have four bytes
- An IPv6 address must have sixteen bytes

```
<xs:simpleType name="IPAddressVersionType">
  <xs:restriction base="xs:string" >
    <xs:enumeration value="V4" />
    <xs:enumeration value="V6" />
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="IPAddress">
  <xs:annotation>
    <xs:appinfo>
      <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
        <sch:ns prefix="tns" uri="urn:IPAddress" />
        <sch:pattern id="Length">
          <sch:rule context=".">
            <sch:assert test="tns:version != 'V4' or count(tns:address) = 4">
              A v4 IP address must have 4 bytes.
            </sch:assert>
            <sch:assert test="tns:version != 'V6' or count(tns:address) = 16">
              A v6 IP address must have 16 bytes.
            </sch:assert>
          </sch:rule>
        </sch:pattern>
      </sch:schema>
    </xs:appinfo>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="version" type="tns:IPAddressVersionType" />
    <xs:element name="address" type="xs:byte" minOccurs="4" maxOccurs="16" />
  </xs:sequence>
</xs:complexType>
```

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

```
<myIPAddress xmlns="urn:IPAddress">
  <version>v6</version>
  <address>100</address>
  <address>200</address>
  <address>10</address>
  <address>1</address>
  <address>10</address>
```

```

    <address>1</address>
</myIPAddress>

```

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

```

<xs:simpleType name="IPAddressVersionType">
  <xs:restriction base="xs:string" sml:numericType="xs:int">
    <xs:enumeration value="V4" sml:numericValue="0" />
    <xs:enumeration value="V6" sml:numericValue="1" />
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="IPAddress">
  <xs:annotation>
    <xs:appinfo>
      <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
        <sch:ns prefix="tns" uri="urn:IPAddress" />
        <sch:pattern id="Length">
          <sch:rule context=".">
            <sch:report test="tns:version = 'V4' and count(tns:address) != 4"
            >
              A v4 IP address must have 4 bytes.
            </sch:report>
            <sch:report test="tns:version = 'V6' and count(tns:address) != 16"
            >
              A v6 IP address must have 16 bytes.
            </sch:report>
          </sch:rule>
        </sch:pattern>
      </sch:schema>
    </xs:appinfo>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="version" type="tns:IPAddressVersionType" />
    <xs:element name="address" type="xs:byte" minOccurs="4" maxOccurs="16" />
  </xs:sequence>
</xs:complexType>

```

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

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

```

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

```

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

```

<xs:element name="StrictUniversity" type="tns:UniversityType">
  <xs:annotation>
    <xs:appinfo>
      <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
        <sch:ns prefix="u" uri="urn:university" />
        <sch:ns prefix="smlfn"
          uri="http://schemas.serviceml.org/smlfn/query/2006/07" />
        <sch:pattern id="StudentPattern">
          <sch:rule context="smlfn:deref(u:Students/u:Student)">
            <sch:assert test="starts-with(u:ID,'99')">
              The specified ID <sch:value-of select="string(u:ID)"/>
              does not begin with 99
            </sch:assert>
            <sch:assert test="count(u:Course/u:Courses)>0">
              The student <sch:value-of select="string(u:ID)"/> must be enrolled
              in at least one course
            </sch:assert>
          </sch:rule>
        </sch:pattern>
      </sch:schema>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

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

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

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

```

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

```

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

## 4.1 Schematron Profile

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

### 4.1.1 Limited Support

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

## 5. Model Validation

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

### 5.1 Schematron Phase

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

## 6. SML Extension Reference

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

### 6.1 Types

#### 6.1.1 `sml:ref`

A complex type representing a reference to an element.

```
<xs:complexType name="ref"
  sml:acyclic="false"
  final="extension">
  <xs:sequence>
    <xs:any namespace="##any" minOccurs="0"
      maxOccurs="unbounded"
      processContents="lax"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

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

## 6.2 Attributes

### 6.2.1 sml:acyclic

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

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

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

```
<xs:complexType name="Hostref" sml:acyclic="true">
  <xs:complexContent>
    <xs:restriction base="sml:ref"/>
  </xs:complexContent>
</xs:complexType>
```

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

### 6.2.2 sml:targetElement

A `QName` representing the name of a referenced element

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

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

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

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

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

### 6.2.3 sml:targetType

A `QName` representing the type of a referenced element

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

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

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

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

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

#### 6.2.4 `sml:uri`

Specifies a reference in URI scheme.

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

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

### 6.3 Elements

#### 6.3.1 `sml:key`

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

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

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

#### 6.3.2 `sml:unique`

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

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

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

#### 6.3.3 `sml:keyref`

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

```
<xs:element name="keyref">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="sml:keybase">
        <xs:attribute name="refer" type="xs:QName" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>>
```

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

## 6.4 XPath functions

### 6.4.1 `smlfn:deref`

`node-set deref(node-set)`

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

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

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

## 7. Acknowledgements

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

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

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

Matt Newman and Virginia Smith (HP)

Johan Van De Groenendaal (Intel)

## 8. References

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

[2] XML Schema Part 1: Structures Second Edition

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

[3] XML Schema Part 2: Datatypes Second Edition

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

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

[5] An Introduction to Schematron

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

[6] Improving XML Document Validation with Schematron

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

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

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

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

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

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

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

## Appendix I – Sample Model

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

- Inter-document references
- key and keyref constraints
- User-defined constraints

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="SampleModel"
  elementFormDefault="qualified"
  xmlns:tns="SampleModel"
  xmlns:sml="http://schemas.serviceml.org/sml/2006/07"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:import namespace="http://schemas.serviceml.org/sml/2006/07"/>

  <xs:simpleType name="SecurityLevel">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Low"/>
      <xs:enumeration value="Medium"/>
      <xs:enumeration value="High"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="Hostref" sml:acyclic="true">
    <xs:complexContent>
      <xs:restriction base="sml:ref"/>
    </xs:complexContent>
  </xs:complexType>

  <!-- This element represents the host operating system for
  an application. Note that the type of the referenced
  element must be OperatingSystemType or a derived type
  of OperatingSystemType -->
  <xs:element name="HostOSRef" type="tns:Hostref"
    sml:targetType="tns:OperatingSystemType"/>

  <xs:complexType name="ApplicationType">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Vendor" type="xs:string"/>
      <xs:element name="Version" type="xs:string"/>
      <xs:element ref="tns:HostOSRef" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
```

```

<xs:simpleType name="ProtocolType">
  <xs:list>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="TCP"/>
        <xs:enumeration value="UDP"/>
        <xs:enumeration value="SMTP"/>
        <xs:enumeration value="SNMP"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:list>
</xs:simpleType>

<!-- Note that the type of the element referenced by an
      GuestAppRef element can not be an extended derived
      type of ApplicationType -->
<xs:element name="GuestAppRef" type="sml:ref"
            sml:targetType="tns:ApplicationType"/>

<xs:complexType name="OperatingSystemType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="FirewallEnabled" type="xs:boolean"/>
    <xs:element name="Protocol" type="tns:ProtocolType"/>
    <!-- The following element represents the applications hosted by
          operating system -->
    <xs:element name="Applications" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:GuestAppRef" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:element name="OSRef" type="sml:ref"
            sml:targetType="tns:OperatingSystemType"/>

<xs:complexType name="WorkstationType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element ref="tns:OSRef"/>
    <xs:element name="Applications" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:GuestAppRef" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```

<xs:element name="Workstation" type="tns:WorkstationType">
  <xs:annotation>
    <xs:appinfo>
      <sch:schema>
        <sch:ns prefix="sm" uri="SampleModel"/>
        <sch:ns prefix="smlfn"
          uri="http://schemas.serviceml.org/sml/function/2006/07"/>
        <sch:pattern id="OneHostOS">
          <!-- The constraints in the following rule are evaluated
            For all instances of the Workstation global element-->
          <sch:rule context=".">
            <!-- define a named variable - MyApplications -
              for use in test expression-->
            <sch:let name="MyApplications"
              value="smlfn:deref(sm:Applications/sm:GuestAppRef)"/>
            <sch:assert test=
              "count($MyApplications)=
                count($MyApplications/sm:HostOSRef)">
              Each application in workstation
              <sch:value-of select="string(sm:Name)"/>
              must be hosted on an operating system
            </sch:assert>
          </sch:rule>
        </sch:pattern>
      </sch:schema>

      <!-- In a workstation, (Vendor,Name,Version) is the key for
        guest applications -->
      <sml:key name="GuestApplicationKey">
        <sml:selector
          xpath="smlfn:deref(tns:Applications/tns:GuestAppRef)"/>
        <sml:field xpath="tns:Vendor"/>
        <sml:field xpath="tns:Name"/>
        <sml:field xpath="tns:Version"/>
      </sml:key>

      <!-- In a workstation, Name is the key for operating system -->
      <sml:key name="OSKey">
        <sml:selector xpath="smlfn:deref(tns:OSRef)"/>
        <sml:field xpath="tns:Name"/>
      </sml:key>

      <!-- In a workstation, the applications hosted by the
        referenced operatinsystem must be a subset of the
        applications in the workstation -->
      <sml:keyref name="OSGuestApplication"
        refer="GuestApplicationKey">
        <sml:selector xpath=
          "smlfn:deref(tns:OSRef)/tns:Applications/tns:GuestAppRef"
        />
        <sml:field xpath="tns:Vendor"/>
        <sml:field xpath="tns:Name"/>
        <sml:field xpath="tns:Version"/>
      </sml:keyref>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

```

<!-- In a workstation, the host operating system of guest
      applications must be a subset of the operating system in
      the workstation -->
<sml:keyref name="ApplicationHostOS" refer="OSKey">
  <sml:selector xpath=
    "smlfn:deref(tns:Applications/tns:GuestAppRef)/tns:HostOSRef"
  />

  <sml:field xpath="tns:Name" />
</sml:keyref>

</xs:appinfo>
</xs:annotation>
</xs:element>

<xs:element name="SecureWorkstation" type="tns:WorkstationType">
  <xs:annotation>
    <xs:appinfo>
      <sch:schema>
        <sch:ns prefix="sm" uri="SampleModel" />
        <sch:ns prefix="smlfn"
          uri="http://schemas.serviceml.org/sml/function/2006/07"
        />
        <sch:pattern id="SecureApplication">
          <sch:rule
            context="smlfn:deref(sm:Applications/sm:Application)">
            <sch:report test="sm:SecurityLevel!='High'">
              Application <sch:value-of select="string(sm:Name)"/>
              from <sch:value-of select="string(sm:Vendor)"/>
              does not have high security level
            </sch:report>
            <sch:assert test="sm:Vendor='TrustedVendor'">
              A secure workstation can only contain
              applications from TrustedVendor
            </sch:assert>
          </sch:rule>
        </sch:pattern>
      </sch:schema>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

</xs:schema>

```

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

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