Service Modeling Language, Version 1.1

W3C Recommendation 12 May 2009

This version:

http://www.w3.org/TR/2009/REC-sml-20090512/

Latest version:

http://www.w3.org/TR/sml/

Previous version:

http://www.w3.org/TR/2009/PR-sml-20090212/

Authors and Contributors:

Bhalchandra Pandit (Microsoft Corporation)
Valentina Popescu (IBM Corporation)
Virginia Smith (HP)

Abstract

This specification defines the Service Modeling Language, Version 1.1 (SML) used to model complex services and systems, including their structure, constraints, policies, and best practices. SML uses XML Schema and Schematron.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is the 12 May 2009 W3C Recommendation of the Service Modeling Language, Version 1.1 specification. This document has been developed by the Service Modeling Language (SML) Working Group, which is a part of the Extensible Markup Language (XML) Activity.

Comments on this document are welcome via the Working Group's public mailing list (public archive). An implementation report is available.

The design of SML has been widely reviewed and satisfies the Working Group's technical requirements. Only minor editorial changes have been made since the 12 February 2009 Proposed Recommendation.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

Appendices

This page is intentionally left blank.

1. Introduction (Non-Normative)

The Service Modeling Language (SML) provides a rich set of constructs for creating models of complex services and systems. Depending on the application domain, these models may include information such as 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 function properly.
2. 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.
3. 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.
4. Models drive modularity, reuse, and standardization. Most real-world complex services and systems are composed of sufficiently complex parts. Reuse and standardization of services/systems and their parts is a key factor in reducing overall production and operation cost and in increasing reliability.
5. Models enable increased automation of management tasks. Automation facilities exposed by the majority of services/systems today could be driven by software not people both 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 a service, as well as the constraints that each part must satisfy for the 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 XML Schema [[XML Schema Structures], [XML Schema Datatypes]] as the schema language. In addition SML defines a set of extensions to XML Schema to support references that may cross document boundaries.
2. Rules are Boolean expressions that constrain the structure and content of documents in a model. SML uses Schematron [[ISO/IEC 19757-3], [Introduction to Schematron], [Improving Validation with Schematron]] and XPath [[XPath]] for rules.

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 extensions to XML Schema for references that cross document boundaries, Schematron usage in SML, as well as the process of model validation. It is assumed that the reader is familiar with XML Schema and Schematron.

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:

1. SML references – XML documents introduce boundaries across content that needs to be treated as a unit. XML Schema does not have any support for references that cross documents, although it does support references to elements in the same document through `xs:ID`, `xs:IDREF`, `xs:key` and `xs:keyref`. References between elements defined in separate SML documents are fundamental to

the SML specification. SML extends XML Schema to support references that may cross document boundaries, and a set of constraints on those references that apply regardless of whether they cross document boundaries or not.

2. Rules – XML Schema does not support a language for defining arbitrary constraints 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.

2. Notations and Terminology

2.1. Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [[IETF RFC 2119]].

This specification uses the Augmented Backus-Naur Form (ABNF) notation [[RFC 2234]].

This specification follows the same conventions for schema components as those used in the XML schema specification [[XML Schema Structures]]. That is, references to properties of schema components, such as {example property}, are links to the relevant definition, set off with curly braces. References to properties of information items as defined in [[XML Information Set]], such as [children], are notated as links to the relevant section thereof, set off with square brackets.

This specification refers to terms such as XML document, element, attribute, etc. for the sake of brevity. The alternative would be to use terms like "XML document or a Synthetic Infoset", "element information item", "attribute information item", etc. at each place. This would make the specification excessively verbose without adding to or changing the meaning of the existing text. The use of the concise terms is not intended to exclude other XML representations. The concepts defined in this specification apply to all forms of XML representations.

The content of this specification is normative except for sections or texts that are explicitly marked as non-normative. If a section is marked as non-normative, then all contained sub-sections are non-normative, even if they are not explicitly marked as such. All notes are non-normative unless otherwise specified.

2.2. Terminology

The following terms are used in this specification. They are listed here in alphabetical order.

Document

A well-formed XML *document*, as defined in [[XML]].

Implementation-Defined

An *implementation-defined* feature or behavior may vary among ; the precise behavior is not specified by this specification but MUST be specified by the implementor of each .

Implementation-Dependent

An *implementation-dependent* feature or behavior may vary among ; the precise behavior is not specified by this or any other W3C specification and is not required to be specified by the implementor for any particular implementation.

Model

A set of inter-related that describe a service or system. Each *model* consists of two disjoint subsets of documents – and .

Model Definition Documents

The subset of documents in a that describes the schemas and rules that govern the structure and content of the model's documents. This specification defines two types of *model definition* document, and , but permits implementations to define other types of model definition documents. Such other types of model definition documents do not play any role in .

Model Instance Documents

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

Model Processor

A *model processor* is an embodiment that processes a conforming SML model using, in whole or in part, semantics defined by this specification.

Model Validation

Model validation is the process of determining whether or not a is both conforming and valid. [§ 8 – Conformance Criteria on page 26]

Model Validator

A *model validator* is a capable of performing

Rule

A *rule* is a boolean expression that constrains the structure and content of a set of documents in a .

Rule Bindings

A *rule binding* is an association of a set of one or more with a set of zero or more model documents. The documents associated with a given rule document are said to be "bound" to it. For a model to be valid, every and in the must conform to the constraints defined by every rule document it is bound to. It is permissible for a rule document to have no bindings associated with it, and for a model document to be bound to zero rule documents.

Rule Document

A *rule document* is a consisting of .

Schema document

A *schema document* is a that conforms to the XML Schema specification [XML Schema Structures] definition of a schema document.

Schematron Constraint

The information contained within a single `sch:schema` element.

SML Reference

An *SML Reference* is an element with an sml:ref attribute whose value is "true".

Conceptually, an SML reference is used to signal a link from one element in an SML model to another element in the same model.

SML Reference Scheme

An *SML Reference Scheme* is a set of rules defining the syntax used to create an instance of the reference scheme in the context of an , plus a set of rules for resolving an instance of the reference scheme to its . Whenever "reference scheme" occurs in this specification, it should be assumed to mean "SML reference scheme" unless otherwise noted. Despite similar names, the term SML reference scheme is unrelated to XPointer schemes and URI schemes.

Target

An element in a model to which an resolves is called the *target* of that SML reference.

Target-complete Identifier

A *target-complete identifier* is a URI or IRI that contains all the information required to locate the of an . It is a consequence of this definition that a target-complete identifier cannot be a relative URI/IRI.

2.3. XML Namespaces

XML Namespaces used in this specification. Prefix XML Namespace Specification(s) sml http://www.w3.org/ns/sml This specification smlfn http://www.w3.org/ns/sml-function This specification xs http://www.w3.org/2001/XMLSchema [,] sch http://purl.oclc.org/dsdl/schematron [] lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

XML Namespaces used in this specification.

Prefix	XML Namespace	Specification(s)
`sml`	`http://www.w3.org/ns/sml`	This specification
`smlfn`	`http://www.w3.org/ns/sml-function`	This specification
`xs`	`http://www.w3.org/2001/XMLSchema`	[[XML Schema Structures], [XML Schema Datatypes]]
`sch`	`http://purl.oclc.org/dsdl/schematron`	[[ISO/IEC 19757-3]]

3. Dependencies on Other Specifications

Other specifications on which this one depends are listed in [].

4. SML References

Support for in an SML includes:

1. The ability to use multiple SML reference schemes for an SML reference.
2. An extensibility mechanism allowing new SML reference schemes to be defined.
3. Constraints on the type of a referenced element.

4. The ability to define key, unique, and key reference constraints across SML references.

MUST support as defined by this specification.

Appendix Appendix C – SML References Sample on page 43 has an example that shows how SML references are defined and used.

4.1. SML Reference Definitions

4.1.1. SML Reference

An element information item in an SML model instance document is as an if and only if it has an attribute information item for which all of the following is true:

1. Its [local name] is `ref`
2. Its [namespace name] is `http://www.w3.org/ns/sml`
3. Its [normalized value], after whitespace normalization using `collapse` following schema rules, is either `"true"` or `"1"`.

This mechanism enables schema-less identification of SML references; i.e., SML references can be identified without relying on the Post Schema Validation Infoset (PSVI). [[XML Schema Structures]]

It is implementation-defined whether that are not also use the XML Infoset [[XML Information Set]] or the Post Schema Validation Infoset (PSVI) [[XML Schema Structures]] for SML reference identification.

SML model validators must use PSVI to identify SML references. See § 8 – Conformance Criteria on page 26.

An SML reference is considered to be an instance of a specific SML if it can be identified as such according to that SML reference scheme's rules. See § 4.3 – SML Reference Schemes on page 9. An SML reference MAY be an instance of multiple SML reference schemes.

Although its normative definition allows several syntaxes to be used to identify an SML reference, for the sake of brevity and consistency, the rest of this specification uses `sml:ref="true"` to denote an SML reference in examples and text.

The following example shows an SML reference that is an instance of the SML URI Reference scheme.

```
<RefElement sml:ref="true">
    <sml:uri>targetDocument.xml</sml:uri>
</RefElement>
```

4.1.2. Null SML Reference

An SML reference is null if and only if it has an attribute information item for which all of the following is true

1. Its [local name] is `nilref`
2. Its [namespace name] is `http://www.w3.org/ns/sml`
3. Its [normalized value] after whitespace normalization using `collapse` following schema rules, is either `"true"` or `"1"`.

It is a consequence of the preceding that this specification assigns no meaning to the `sml:nilref` attribute when it is used on an element that is not an SML reference. Model validators MAY choose to warn their invokers should they detect this condition in a document.

The following example shows a null SML reference.

```
<RefElement sml:ref="true" sml:nilref="true">
    <sml:uri>targetDocument.xml</sml:uri>
</RefElement>
```

☞ `sml:nilref` may be useful in the case where the schema author defines a complex type specifying `sml:ref="true"` with a fixed value of "true", but the instance author wants to signal the absence of a target.

It is implementation-defined whether that are not also use the XML Infoset [[XML Information Set]] or the Post Schema Validation Infoset (PSVI) [[XML Schema Structures]] to identify null SML references.

☞ SML model validators must use PSVI to identify null SML references. See § 8 – Conformance Criteria on page 26.

4.1.3. Unresolved SML Reference

An SML reference is unresolved if and only if all of the following is true:

1. It is a non-null SML reference.
2. None of the , which the SML reference is recognized as using, resolves to an element in the model.

☞ The notion of unresolved reference is context-dependent. That is, different , based on the set of SML reference schemes they understand and which are used in the model they process, may consider different SML references to be unresolved.

The following example shows an unresolved SML reference (assuming that the document `dummy.xml` does not exist in the model).

```
<RefElement sml:ref="true">
    <sml:uri>dummy.xml</sml:uri>
</RefElement>
```

4.1.4. SML Reference Target

The element node that a non-null SML reference resolves to is its target. The target of an SML reference MUST be part of the same SML model as the SML reference. Null SML references have no target.

The method of determining which documents are part of an SML model is implementation-defined.

☞ For example, an SML model may consist of documents listed in a configuration file or an SML model could be construed as the transitive closure of documents referred to by any SML references starting from a set of documents known to be in the model.

The following example shows an SML reference that targets the second `Course` child element of the root element of the document `target.xml`.

```
<RefElement sml:ref="true" xmlns:e="urn:example">
    <sml:uri>target.xml#smlxpath1(e:Course[2])</sml:uri>
```

```
</RefElement>

document 'target.xml':
----------------------
<Courses xmlns="urn:example">
    <Course>
        <Name>PHY101</Name>
        <Grade>A</Grade>
    </Course>
    <Course>
        <Name>MAT101</Name>
        <Grade>A</Grade>
    </Course>
</Courses>
```

4.2. SML Reference Semantics

MUST attempt to resolve an SML reference using all the reference schemes of which the SML reference is recognized as an instance.

4.2.1. At Most One Target

Every non-null SML reference MUST target at most one element in a . This means that each recognized reference scheme used in the SML reference MUST NOT resolve to more than one target.

The following example shows an SML reference that violates the at-most-one-target rule.

```
<RefElement sml:ref="true" xmlns:e="urn:example">
    <sml:uri>target.xml#smlxpath1(e:Course)</sml:uri>
</RefElement>

document 'target.xml':
----------------------
<Courses xmlns="urn:example">
    <Course>
        <Name>PHY101</Name>
        <Grade>A</Grade>
    </Course>
    <Course>
        <Name>MAT101</Name>
        <Grade>A</Grade>
    </Course>
</Courses>
```

4.2.2. Consistent References

If a non-null SML reference is an instance of multiple reference schemes, all recognized reference schemes MUST resolve to the same target or they all MUST be unresolved.

4.2.3. Identical Targets

To determine if two targets are the same or different, MUST obey the following rules.

1. If both of the following are true, then a model validator MUST consider both targets to be the same.
   A. The definition of the reference scheme(s) specifies how URIs are transformed to .
   B. The two target-complete identifiers are identical using a case-sensitive, codepoint-by-codepoint comparison.
2. Otherwise, a model validator MUST consider both targets to be different when there is something available in the element information items for the targets that tells them apart. For example, if there is an infoset property for which the 2 targets have different values, they are different. This applies recursively for complex-valued properties.
3. For all other cases, it is implementation-defined whether to treat the targets as the same or not.

4.2.4. Multiple References

An element in a document MAY be targeted by multiple SML references.

4.2.5. Null SML References

A null SML reference is an explicit declaration of intent by the document author that the of the SML reference does not exist. If an SML reference is recognized as null, then MUST NOT attempt to recognize any reference schemes used in it.

4.2.6. Deterministic Evaluation of SML Constraints

Each non-null SML reference MUST satisfy all of the following conditions in order to be able to deterministically evaluate SML constraints and rules associated with it.

1. The reference must have at most one target. [§ 4.2.1 – At Most One Target on page 7]
2. The reference MUST be consistent. [§ 4.2.2 – Consistent References on page 7]

4.2.7. smlfn:deref() XPath Extension Function

The `deref()` function takes a node set of elements and returns a node set consisting of element nodes corresponding to the elements referenced by the input node set. In particular, for each SML reference R in the input node set the output node set contains at most one element node.

Let I = input node set; that is, the set of nodes passed to the `deref()` function.

Let O = output node set; that is, the set of nodes returned by the `deref()` function.

The behavior of `deref()` function MUST satisfy the following constraints:

1. For each SML reference R in the input node set I:
   A. If the implementation recognizes no SML reference scheme used in the SML reference R, then no element is added to O.
   B. If the implementation recognizes R as an instance of N supported reference schemes, then `deref()` is not required to attempt to resolve all N schemes. Its behavior in this case is implementation-defined and the set of reference schemes that are actually attempted may be any subset of the recognized schemes. This is subject to the following constraints:

i. If deref() doesn't attempt to resolve any reference scheme or if none of the attempted reference schemes resolves, then no element is added to O.

ii. If at least one of the attempted reference schemes resolves to more than one target element, then 0 or 1 of the targets is added to O.

iii. If one attempted reference scheme resolves to a target different from the target resolved by another attempted reference scheme, then 0 or 1 of the targets is added to O.

iv. If one attempted reference scheme resolves and another doesn't, then 0 or 1 of the targets is added to O.

v. If none of the above is true (that is, all attempted reference schemes resolve to the same one and only one target element, call it T), then one target element (namely, T) is added to O, if it does not already exist in O.

The above describes the behavior required for a general XPath 1.0 deref() library function, and as such exhibits several significant differences from the behavior required to validate SML references during model validation. First, it can be used to successfully process instance documents whose SML model validity is unknown or invalid, although the results in this case may not be interoperable. Second, since XPath 1.0 defines no way for a function to signal erroneous input to its caller, the behavior here is specified to return results for SML references that do not obey all of the validity rules, e.g. a reference whose XPath expression evaluates to more than one node. As described in this section, such a function would be insufficient to check the validity of SML references.

Model validators MUST provide an implementation of the deref() XPath extension function. In addition to the above requirements for general deref() function implementations, for each SML reference using recognized schemes, deref() in model validators MUST attempt to resolve at least one of the recognized schemes.

4.3. SML Reference Schemes

An SML reference MAY be an instance of a variety of reference schemes. SML does not mandate the use of any specific reference schemes. An SML reference scheme MAY use child elements, attributes, both, or neither to capture the information necessary to identify the reference target. It is OPTIONAL that all elements in an SML model be reachable via an SML reference. This will depend on the support defined by the chosen reference scheme.

Although SML does not require the use of any specific scheme, it does specify how a reference MUST be represented when using SML-defined reference schemes. This specification defines the § 4.3.1 – SML URI Reference Scheme on page 10 for use in SML references.

An SML reference scheme definition MUST specify all of the following:

1. The set of rules that, when satisfied, identify an SML reference as an instance of the scheme. An SML reference scheme definition MAY impose additional validity requirements on SML references recognized as instances of that scheme. MUST NOT apply such requirements to SML references that are not instances of the corresponding reference scheme.
2. The set of rules that, when evaluated, resolve the SML reference to its target element node.
3. An assertion that states whether instances of the reference scheme are transformed to . If they are transformed to , the reference scheme definition MUST describe the transformation process.

An SML reference scheme definition MUST specify all of the preceding items as they apply to valid instances of the SML reference scheme, and MAY specify them for other (invalid) instances.

4.3.1. SML URI Reference Scheme

The SML URI Reference Scheme is defined as follows:

1. An SML reference is identified as an instance of the SML URI Reference Scheme if and only if exactly one element information item [[XML Information Set]] whose [local name] is `uri` and whose [namespace name] is `http://www.w3.org/ns/sml` is present as a child of that reference element.

   An instance of the SML reference scheme is valid if it meets all of the following requirements.

   A. The content of the `uri` element MUST be of type `xs:anyURI` as defined in the XML schema specification [[XML Schema Datatypes]].

   B. The fragment identifier (if present) MUST follow the syntax of one of the following.

      i. § 4.3.1.1 – `smlxpath1()` scheme on page 11

      ii. Shorthand Pointer

2. An SML reference that is an instance of the SML URI Reference Scheme is resolved using the following steps:

   A. An XML document *D* is obtained as follows:

      i. If the URI reference is a same-document reference as defined in the applicable URI RFC, then *D* is the document containing the SML reference.

      ii. Otherwise, *D* is determined as follows:

         a) If the URI reference is a relative reference, then let *U* be the result of resolving the reference using the [base URI] property [[XML Information Set]] of the `<sml:uri>` element as the base URI. Otherwise, *U* is the URI reference itself. The computation of the [base URI] property is implementation-defined.

         b) Dereference *U* as defined in the applicable specifications. If the document targeted by *U* is in the current SML model, then *D* is that document. Otherwise, if the document is not in the current SML model, then the SML URI Reference Scheme instance is unresolved (and *D* has no value).

         

         As a result of the above definition, if the retrieved object is not of XML media type or if it is not well-formed XML then, by definition, that object is not a document as defined by this specification. In this case, the SML reference scheme instance is unresolved.

   B. If no fragment component is present in the URI reference, the SML URI Reference Scheme instance resolves to the root element of *D*.

   C. If a fragment component is present in the URI reference, then the appropriate case among the following applies:

      i. If the fragment component complies with the `smlxpath1()` XPointer scheme syntax, then the reference target is obtained by applying the fragment component to *D*, as defined in section § 4.3.1.1 – `smlxpath1()` scheme on page 11 .

      ii. If the fragment component complies with the Shorthand Pointer syntax, then the appropriate case among the following applies:

         a) If a target *T* can be identified in *D* based on the XML-Schema-determined ID, then the reference target is *T*.

b) If a target in *D* cannot be identified based on the XML-Schema-determined ID, then it is implementation-defined whether the reference target in *D* is identified based on other criteria allowed for Shorthand Pointers.

3. Instances of the SML URI Reference Scheme are transformed to through standard URI processing, as described in the applicable URI RFC.

The following example shows an SML reference that is an instance of the SML URI Reference scheme. The reference targets the element with ID `targetId` in document `target.xml`.

```
<RefElement sml:ref="true">
    <sml:uri>target.xml#targetId</sml:uri>
</RefElement>
```

4.3.1.1. `smlxpath1()` scheme

The `smlxpath1()` scheme is intended to be used with the XPointer Framework [[XPointer]] to allow addressing of elements. The § 4.3.1 – SML URI Reference Scheme on page 10 uses it to encode fragment identifiers.

This section describes the syntax and semantics of the `smlxpath1()` scheme and the behavior of XPointer processors with respect to this scheme.

1. Scheme name: `smlxpath1`
2. Scheme syntax using ABNF [[RFC 2234]]:

   ```
   SMLXPath1_Fragment_ID ::= 'smlxpath1' '(' SMLXPath1_SchemeData ')'

   SMLXPath1_SchemeData ::= XPath1.0_LocationPath
   ```

   where,

   `XPath1.0_LocationPath` is the LocationPath production defined in the XPath 1.0 specification [[XPath]].

3. The `deref()` XPath extension function MUST NOT be present in the expression evaluation context function library when processing the location path in `SMLXPath1_SchemeData`.
4. Namespace Binding Context: The `smlxpath1()` scheme inherits the set of namespace bindings available to the parent `sml:uri` element.
5. For a given document *D*, the element targeted by a scheme instance is obtained by applying the location path in `SMLXPath1_SchemeData` to the root element of *D*. The result MUST either be 1 element node or be empty. Otherwise, the XPointer result is an error.

> ☞ In the case of instances of the SML URI Reference scheme, *D* is the document resolved to by the non-fragment part of the URI reference, as defined in item 2.a in section § 4.3.1 – SML URI Reference Scheme on page 10.

The following example shows an SML reference that is an instance of the SML URI Reference scheme. The reference targets the root element of the document `target.xml`.

```
<RefElement sml:ref="true">
    <sml:uri>target.xml#smlxpath1(/*)</sml:uri>
</RefElement>
```

5. SML Constraints

5.1. Constraints on SML References

SML supports the following attributes for expressing constraints on SML references.

Attributes

Name	Description
`sml:acyclic`	Used to specify whether cycles are prohibited for an SML reference.
`sml:targetRequired`	Used to specify that an SML reference's target element is required to be present in the model.
`sml:targetElement`	Used to constrain the name of the SML reference's target.
`sml:targetType`	Used to constrain the type of the SML reference's target.

SML defines a new property for every Complex Type Definition schema component:

{acyclic}

An `xs:boolean` value. Required.

The value of for `xs:anyType` is `false`.

SML defines three new properties for every Element Declaration component:

{target required}

An `xs:boolean` value. Required.

{target element}

An Element Declaration component. Optional.

{target type}

A Type Definition component. Optional.

5.1.1. sml:acyclic

`sml:acyclic` is used to specify whether or not a cycle is allowed on instances of a complex type. MUST support the `sml:acyclic` attribute on any `<xs:complexType>` element in a schema document. This attribute is of type `xs:boolean` and its actual value can be either `true` or `false`.

5.1.1.1. SML Constraint Construction

The property value of a complex type definition is as specified by the appropriate case among the following:

1. If `sml:acyclic` is present, then has the actual value of this attribute.
2. Otherwise, if its {base type definition} is a complex type definition, then has the same value of as its {base type definition}.
3. Otherwise ({base type definition} is a simple type definition), is false.

5.1.1.2. Schema Component Rules

If a complex type definition *CT*'s {base type definition} is also a complex type definition and has true, then *CT* MUST have true.

5.1.1.3. Instance Validity Rules

If *CT* is a complex type definition with true, then instances of *CT* MUST NOT create cycles in the model. More precisely, the directed graph constructed in the following way MUST be acyclic:

1. The nodes in the graph are all the elements resolved to by SML references of type *CT* or types derived from *CT*.
2. If a node *N* in the graph is or contains an SML reference *R* of type *CT* or a type derived from *CT*, and *R* resolves to *T* (which must also be a node in the graph), then an arc is drawn from *N* to *T*.

5.1.2. Constraints on SML Reference Targets

SML defines three attributes: `sml:targetRequired`, `sml:targetElement`, and `sml:target-Type`, for constraining the target of an SML reference. These three attributes are collectively called `sml:target*` attributes. MUST support these attributes on all `xs:element` elements with a name attribute. The `sml:target*` constraints are attached to the element declaration schema component.

5.1.2.1. SML Constraint Construction

1. is as specified by the appropriate case among the following:
   A. If `sml:targetRequired` is present, then is the actual value of this attribute.
   B. Otherwise if the element declaration has a {substitution group affiliation}, then is the same as that of the {substitution group affiliation}.
   C. Otherwise is false.
2. is as specified by the appropriate case among the following:
   A. If `sml:targetElement` is present, then its actual value MUST resolve to a global element declaration *G*, and is *G*.
   B. Otherwise if {substitution group affiliation} is present, then is the same as that of the {substitution group affiliation}.
   C. Otherwise is absent.
3. is as specified by the appropriate case among the following:
   A. If `sml:targetType` is present, then its actual value MUST resolve to a global type definition *T*, and is *T*.
   B. Otherwise if {substitution group affiliation} is present, then is the same as that of the {substitution group affiliation}.
   C. Otherwise is absent.

5.1.2.2. Schema Component Rules

MUST enforce the following:

1. If a global element declaration *E* has a {substitution group affiliation} *G*, then the value of *E*'s SML target constraint property P (one of , or) MUST be a valid restriction of the corresponding property of *G* as defined in section § 5.3 – Valid Restriction of SML Constraint Values on page 16.

2. If two element declarations *E1* and *E2* have the same {namespace name} and {name} and they are both contained (directly, indirectly, or implicitly) in a content model of a complex type, then *E1* and *E2* have the same , , and .

☞ The above condition #2 on the use of `sml:target*` attributes has been defined to reduce the implementation burden on . Please refer to section § 5.4.1 – Overview of SML Constraint Processing and Complex Type Derivation on page 17 for more information.

5.1.2.3. Instance Validity Rules

If an element declaration *E* has `true`, then each element instance of *E* that is also an SML reference MUST target some element in the model. That is, no instance of *E* can be a null or unresolved SML reference.

If an element declaration *E* has *TE*, then each element instance of *E* that is also a resolved SML reference MUST target an element that is an instance of *TE* or an instance of some global element declaration in the substitution group of *TE*.

If an element declaration *E* has *TT*, then each element instance of *E* that is also a resolved SML reference MUST target an element whose [type definition] is *TT* or a type derived from *TT*.

5.1.3. SML Reference Constraints Summary (Non-Normative)

The effect of the above instance validation rules is summarized in the following table.

Target Constraints and SML Reference Categories.

Reference Category	`Acyclic`	`targetRequired`	`targetElement`	`targetType`
Non-reference	Satisfied	Satisfied	Satisfied	Satisfied
Null	Satisfied	Violated	Satisfied	Satisfied
Unresolved	Satisfied	Violated	Satisfied	Satisfied
Resolved	Check	Satisfied	Check	Check

"Check" in the table above means that the appropriate constraint must be evaluated.

The constraints described above can be useful even on element declarations whose instances are not necessarily SML references, because the decision about whether to include a constraint and the decision about whether to make the element an SML reference can be made independently - some choices made by the schema author, other choices made by the instance document author.

5.2. SML Identity Constraints

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

MUST support the following elements for defining SML identity constraints across SML references, as child elements of `xs:element/xs:annotation/xs:appinfo` where the `xs:element` has a name attribute.

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

Appendix Appendix B – Model Definition Document Sample on page 38 and Appendix Appendix E – SML Identity Constraints Sample on page 47 have examples that show how SML identity constraints are defined.

SML identity constraints are attached to the element declaration schema component. SML defines a new property for every element declaration schema component:

{SML identity-constraints definitions}

> A set of SML identity constraint definitions components, which have the same set of properties as XML Schema identity constraint definitions.

5.2.1. Syntax and Semantics

Names of all SML identity constraint definitions exist in a single symbol space, which is disjoint from any symbol space of XML Schema components.

5.2.1.1. SML Constraint Construction

For each `sml:key`, `sml:unique`, or `sml:keyref` element without the `ref` attribute specified, contains a component corresponding to this element, as specified in section 3.11 Identity-constraint Definitions of the XML Schema specification [XML Schema Structures]), where `sml:selector` and `sml:field` elements are used in place of `xs:selector` and `xs:field`.

For each `sml:key`, `sml:unique`, or `sml:keyref` element with the `ref` attribute specified, contains the component resolved to by the actual value of the `ref` attribute, with the following conditions:

1. The name attribute MUST NOT be specified.
2. The `sml:selector` and `sml:field` child elements MUST NOT be specified.
3. If the element is `sml:key`, then the value of `ref` attribute MUST resolve to an SML key constraint.
4. If the element is `sml:unique`, then the value of the `ref` attribute MUST resolve to an SML unique constraint.
5. If element is `sml:keyref`, then the value of the `ref` attribute MUST resolve to an SML keyref constraint, and the `refer` attribute MUST NOT be specified.

In addition to SML identity constraints obtained from the above explicit definitions or references, if an element declaration *S* has a {substitution group affiliation} *G*, then its also contains members of of *G*.

5.2.1.2. Schema Component Rules

1. `sml:selector` XPath expression has the same syntax as that defined in the XML Schema identity constraint selector XPath syntax with one exception. The `sml:selector` XPath MAY use `smlfn:deref()` functions, with function calls nested to any depth, at the beginning of the expression.

The XML Schema identity constraint selector Path production is amended to support this requirement as defined below.

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

2. `sml:field` XPath expression has the same syntax as that defined in the XML Schema identity constraint field XPath syntax with one exception. The `sml:field` XPath MAY use `smlfn:deref()` functions, with function calls nested to any depth, at the beginning of the expression. The XML Schema identity constraint field Path production is amended to support this requirement as defined below.

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

3. The of an element declaration MUST NOT contain two identity constraints with the same name.

> ☞ This could happen if the ref attribute resolves to an identity constraint already contained in the same element declaration's .

4. If a global element declaration *E* has a {substitution group affiliation} *G*, then the value of *E*'s property MUST be a valid restriction of the value of the corresponding property of *G* as defined in section § 5.3 – Valid Restriction of SML Constraint Values on page 16.

5. If two element declarations *E1* and *E2* have the same {namespace name} and {name} and they are both contained (directly, indirectly, or implicitly) in a content model of a complex type, then *E1* and *E2* MUST have the same set of .

> ☞ This rule is defined to reduce the implementation burden for model validators. Please refer to section § 5.4.1 – Overview of SML Constraint Processing and Complex Type Derivation on page 17 for more information.

5.2.1.3. Instance Validity Rules

Validation rules for SML identity constraints are the same as specified in section 3.11 Identity-constraint Definitions of the XML Schema specification [XML Schema Structures]), with the addition of support for the `smlfn:deref()` function.

5.3. Valid Restriction of SML Constraint Values

Let BV = value of SML constraint property P (one of , , or).

Let RV = value that restricts BV.

For RV to be a valid restriction of BV, the appropriate case among the following MUST be true.

1. For , the appropriate case among the following applies.
   A. If BV is true, RV is true.
   B. If BV is false, RV is either true or false.

2. For , one of the following applies.
   A. RV is same as BV.
   B. RV is in the substitution group of BV.
3. For , one of the following applies.
   A. RV is same as BV.
   B. RV is a type derived from BV.
4. For , one of the following applies.
   A. RV is same as BV. That is, all of the following is true.
      i. The number of entries in RV is same as the number of entries in BV.
      ii. For each entry in BV, there exists an entry in RV with the same qualified name ({name} + {target namespace}).
   B. RV is a superset of BV. That is, RV has all of the entries from BV as defined in the previous item and it has one or more additional entries.

5.4. SML Constraints and Complex Type Derivation

5.4.1. Overview of SML Constraint Processing and Complex Type Derivation

This section is non-normative.

For a complex type D derived from its {base type definition} B, if an element declaration ED is included in D and an element declaration EB is included in B, and ED and EB satisfy the "NameAndTypeOK" constraint, then the SML constraints (target* and SML identity constraints) applicable to ED must be

1. the same as those on EB in case of derivation by extension, and
2. the same or more restrictive compared to those on EB in case of derivation by restriction.

SML defines this behavior to ensure that one cannot get rid of SML constraints on elements in a complex type by simply deriving another type from that type.

Enforcing this condition across derivation by restriction would require an implementation to match a restricting particle to the corresponding restricted particle in order to evaluate condition 2 above. This level of support is not provided by most XML Schema frameworks; thus most SML validators would otherwise need to duplicate large parts of XML Schema's compilation logic to verify consistent usage of SML constraints across derivation by restriction. In order to reduce this implementation burden on model validators, SML requires that all element declarations with a given name that are included in a complex type definition must have the same SML constraint value. This allows model validators to find the restricted particle for a restricting particle using a simple name match.

This also means that the value of a given SML constraint applicable to all element declarations of a given name in complex type definition can be logically viewed as available at a single place, for example in a property attached to that complex type, rather than being scattered across element declarations in that type. The next section uses this logical view because it makes it easier to understand and formally define SML constraint behavior across complex type derivation.

5.4.2. Formal Definition

5.4.2.1. Properties

SML defines four properties for every complex type definition schema component CT.

{target required constraint list}

A list of (qname, value) pairs, where,

1. qname is a qualified name ({namespace name} + {name}).
2. value is the value of a property.

{target element constraint list}

A list of (qname, value) pairs, where,

1. qname is a qualified name ({namespace name} + {name}).
2. value is the value of a property.

{target type constraint list}

A list of (qname, value) pairs, where,

1. qname is a qualified name ({namespace name} + {name}).
2. value is the value of a property.

{identity constraint list}

A list of (qname, value) pairs, where,

1. qname is a qualified name ({namespace name} + {name}).
2. value is the value of a property.

The value of the above 4 properties for `xs:anyType` is empty.

5.4.2.2. SML Constraint Construction

Let

- CT = A complex type definition.
- C = SML constraint (one of targetRequired, targetElement, targetType, SML identity constraint).
- P = A property of CT corresponding to constraint C (one of , , ,).
- V = The value of P, a list of (qname, value) pairs.
- ED = An element declaration contained in CT.
- PED = A property of ED corresponding to constraint C (one of , , ,).

Property P is assigned value V as defined below:

1. For each element declaration ED, with qualified name qn, contained in CT:
   A. If ED does not have constraint C, that is, the value of PED is absent (or false in case of targetRequired), then skip ED.
   B. Otherwise, if there is already an entry in V for qn, then skip ED.

If the value of the existing entry is different from the value of PED then it is treated as a schema validation error as defined in section § 5.1.2.2 – Schema Component Rules on page 13 and section § 5.2.1.2 – Schema Component Rules on page 15.

C. Otherwise, the entry (qn, value of PED) is added to the list V.

2. The appropriate case among the following applies:
   A. If CT is derived by extension from a simple type definition then value V is empty.
   B. If CT is derived by extension from a complex type definition BT:

      The initial value of V is computed as defined in list item 1 above and then,

      For each entry (qn, v_b) in the value of P in BT:

      i. If V has an entry (qn, v_c) present, then ensure that v_c is same as v_b. If it is not same, then it is treated as a schema validation error.

      ii. If V does not have any entry (qn, v_c) present, then copy (qn, v_b) into V.

   C. If CT is derived by restriction from a complex type definition BT:

      The initial value of V is computed as defined in list item 1 above and then,

      For each entry (qn, v_b) in the value of P in BT:

      i. If V has an entry (qn, v_c) present, then ensure that v_c is a valid restriction of v_b as defined in section § 5.3 – Valid Restriction of SML Constraint Values on page 16. If it is not, then it is treated as a schema validation error.

      ii. If V does not have any entry (qn, v_c) present, then copy (qn, v_b) into V.

5.4.2.3. Instance Validity Rules

Let,

- CT = the complex type of element declaration ED.
- E = an instance of ED.
- C = a child element of E.

If C matches an element declaration contained in CT and if one or more of CT's constraint properties, defined in § 5.4.2.1 – Properties on page 18, contain an entry matching C's qualified name ({namespace name} + {name}) then the value of each of those entries is used for evaluating the corresponding constraint on C, as defined in section § 5.1.2.3 – Instance Validity Rules on page 14 and section § 5.2.1.3 – Instance Validity Rules on page 16, as if the matching element declaration has the corresponding constraint with that value.

One way for constraints to be embedded in element declarations or type definitions in a schema is for constraint element to be included in a schema document, embedded at the appropriate locations within the `xs:element` or `xs:complexType` elements which describe the element declaration or type definition.

Element declarations and type definitions created by other means can, however, also have constraints embedded within the {application information} of their {annotation} properties. How such embedding is accomplished is outside the scope of this specification and is likely to vary among .

6. 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 [[ISO/IEC 19757-3]] is an ISO/IEC standard for defining assertions concerning a set of XML documents. SML uses Schematron to add support for additional model constraints not supported in XML Schema.

6.1. Informal Description (Non-Normative)

This section assumes that the reader is familiar with Schematron concepts; the Schematron standard is documented in [[ISO/IEC 19757-3]] and [[Introduction to Schematron], [Improving Validation with Schematron]] are good tutorials on an older version of Schematron.

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:

1. An IPv4 address must have four bytes
2. An IPv6 address must have sixteen bytes

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:x-example:IPAddress">
    <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:x-example: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>
</xs:schema>
```

A 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` (which is a part of the containing Schematron constraint) is applicable for all elements whose type is `IPAddress` or a derived type of `IPAddress`. A `pattern` element contains one or more `sch:rule` elements and a single `sch:rule` element contains one or more `assert` and/or `report` elements. Each `sch:rule` element specifies its context using the `context` attribute. This context expression is evaluated in the context of each applicable element and results in an element node set for which the assert and report `test` expressions contained in the `sch:rule` element are evaluated. The `context` expression is defined as an XSLT Pattern. This means that the `smlfn:deref` function may not be used in the location path of a `context` expression.

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. 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:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:x-example:IPAddress">
    <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:x-example: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>
</xs:schema>
```

If a `sch:assert` or `sch:report` is violated, the violation is reported together with the specified message. The message can include substitution strings based on XPath expressions. These can be specified using the `sch:value-of` element. The following example uses the `sch:value-of` element 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 declaration. 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="http://www.university.example.org/ns" />
                <sch:ns prefix="smlfn"
                        uri="http://www.w3.org/ns/sml-function"/>
                <sch:pattern id="StudentPattern">
                    <sch:rule context="u:Students/u:Student">
                        <sch:assert test="smlfn:deref(.)[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:Courses/u:Course)>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 `sch:rule` elements contained in `StudentPattern` are applicable to all element instances of the `StrictUniversity` global element declaration. 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. Thus, these two asserts verify the following conditions for each instance of `StrictUniversity`.

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

Schematron patterns can be authored in separate rule documents that are then bound to a set of documents in the model.

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="http://www.university.example.org/ns" />
    <sch:ns prefix="smlfn" uri="http://www.w3.org/ns/sml-function"/>
    <sch:pattern id="StudentPattern">
        <sch:rule context="u:StrictUniversity/u:Students/u:Student">
            <sch:assert test="smlfn:deref(.)[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:Courses/u:Course)>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 rule document containing the `StudentPattern` pattern to documents that may contain instances of `StrictUniversity` element is implementation-defined.

6.2. Rule Support

are REQUIRED to support and evaluate XPath expressions augmented with the `smlfn:deref()` function in the body of Schematron constraints.

If the `queryBinding` attribute is not specified, then its value is assumed to be set to `"xslt"`. MUST support the `"xslt"` query binding. MAY additionally support query bindings other than `"xslt"`.

6.3. Rules Associated with Schema Components

SML defines a new property for every complex type definition schema component and every element declaration schema component.

{rules}

A set of .

6.3.1. SML Rule Construction

The property contains all of the Schematron constraints applicable to instances of the given type definition or element declaration. Its value is derived in part from `sch:schema` elements embedded within the component, and sometimes in part from the properties of other components.

`sch:schema` elements MAY appear as items in the {application information} of the {annotation} of a global element declaration or a global complex type definition. This specification assigns no meaning to `sch:schema` elements if they appear as items in any other location.

Let the *local-rules* of a given global element declaration or global complex type definition be the set of Schematron constraints embedded in the {application information} of that schema component's {annotation} property. For other schema components, *local-rules* is empty.

The value of the property of a schema component is computed as follows:

1. The value of for `xs:anyType` is the empty set.
2. If the schema component is a global element declaration, then the value of its is the union of its *local-rules* and the appropriate case from the following:
   A. If the element declaration has a {substitution group affiliation}, then the value of of the {substitution group affiliation}.
   B. Otherwise (the element declaration has no {substitution group affiliation}), the empty set.
3. If the schema component is a complex type definition, then the value of its property is the union of its *local-rules* and the appropriate case from the following:
   A. If the component's {base type definition} is a complex type definition, then the of the {base type definition}. This is true for derivation by extension as well as for derivation by restriction.
   B. Otherwise (i.e., when {base type definition} is a simple type definition), the empty set.
4. Otherwise, the value of the property is not affected by this specification.

6.3.2. Schema Component Rules

MUST enforce the following rules.

1. If a complex type *D* is derived by restriction or extension from {base type definition} *B* and if *B* has defined on it then they MUST be automatically copied to *D* and unioned with the defined on *D*.
2. If a complex type *D* is derived by restriction from {base type definition} *B*, then a global element declaration with non-empty contained in *B* MUST NOT be restricted to a local element declaration in *D*.

It is an error if all of the following are true.

A. An element declaration *ED* is contained (directly, indirectly, or implicitly) in *D* and an element declaration *EB* is contained (directly, indirectly, or implicitly) in *B*.

B. *ED* and *EB* satisfy the "NameAndTypeOK" constraint (for XML Schema's definition of valid restrictions, see Schema Component Constraint: Particle Valid (Restriction), Constraints on Particle Schema Components in [XML Schema Structures]).

C. *EB* is a reference to a global element declaration with a Schematron constraint on it.

D. *ED* is a local element declaration with the same name as *EB*.

6.3.3. Instance Validity Rules

MUST behave as follows:

1. Each in of a complex-type definition CT MUST be evaluated for all element instances of type CT in a model during the model's validation.
2. Each in of a global element declaration G MUST be evaluated for all element instances of G in a model during the model's validation.
3. All of the assertion tests in fired rules MUST succeed.

6.4. Rules Authored in Rule Documents

6.4.1. Rule Binding

MUST provide a mechanism to support the binding of Schematron patterns, authored in separate , to a set of documents in a model. Rule documents MAY be bound to model instance documents as well as model definition documents. The mechanism for binding rule documents to a set of documents in a model is implementation-defined.

7. Localization of Natural Language Messages

SML defines the `sml:locid` attribute in support of localization of natural-language texts, for example, `smlif:description` or Schematron messages. MAY support `sml:locid` attribute on the following elements:

1. `sch:assert` and `sch:report` in a .
2. `sch:assert` and `sch:report` in a Schematron `pattern` embedded in the {application information} of the {annotation} property of a complex type definition or an element declaration.
3. Elements in instance documents with textual content.

Model validators that support the `sml:locid` attribute MUST use the `sml:locid` attribute value to access the location of the translated text.

The mechanism for using the `QName` value of the `sml:locid` attribute to locate the translated text is implementation-dependent. For example, the namespace name can be used to identify the resource containing the text and the local name can be used to identify the text within such a resource. Refer to Appendix Appendix F – Localization and Variable Substitution Samples on page 51 for a concrete sample of how the `sml:locid` attribute can be used to support text localization.

7.1. Variable Substitution

It is often the case that a `sch:assert` or `sch:report` message can be reused in different situations. To be able to reuse a message, the rule author must be able to substitute variable content based on the context in which the message is being used.

Although this specification does not mandate the use of variable substitution in Schematron messages, it suggests the use of `xsl:variable` when variable substitution is desired. Refer to Appendix Appendix F – Localization and Variable Substitution Samples on page 51 section for a concrete sample of how the `xsl:variable` can be used in support of reusing localized messages.

8. Conformance Criteria

A program is a *conforming SML model processor* if and only if it satisfies all the constraints imposed on processors elsewhere in this specification.

A conforming SML model processor is a *conforming SML model validator* if and only if it satisfies the following conditions:

1. The validator MUST perform as defined in this specification.
2. The validator MUST support XML 1.0 [[XML]], XML Schema 1.0 [[XML Schema Structures], [XML Schema Datatypes]], and XPath 1.0 [[XPath]] but MAY also additionally support any future versions of these specifications.
3. The validator MUST support Schematron [[ISO/IEC 19757-3]].
4. The validator MUST perform Schematron rule evaluation on the #ALL phase.
5. The validator MUST support the `deref()` XPath extension function.
6. The validator MUST identify all SML references in the model using the Post Schema Validation Infoset. [[XML Schema Structures]]
7. The validator MUST use the Post Schema Validation Infoset to determine if an SML reference in the model is a null SML reference. [[XML Schema Structures]]

The conformance of a model and the validity of a model can be assessed if and only if all documents in the model are available to the model validator. A model validator MUST document its behavior when a model document is found to be unavailable (i.e. the behavior is implementation-defined). It MAY respond to this condition in ways that include but are not limited to: assessing the model as invalid, or treating this as a warning. The intent of the latitude granted to model validators in this case is to provide some implementation flexibility by not prescribing a limited set of choices, however it is be read narrowly rather than as a broad license to take unrelated actions like failing to enforce SML constraints on unrelated documents.

A model is a *conforming SML* if and only if it satisfies the following conditions:

1. Each in the model MUST be a well-formed XML document [[XML]]
2. For each XML Schema document in the model's definition documents, the [validity] property of the root element MUST be "valid" when schema validity is assessed with respect to a schema constructed from the [XML Schema for Schemas] and Appendix A – Normative SML Schema on page 34 schema documents.

3. All schemas assembled from the XML Schema documents in the model's definition documents MUST satisfy the conditions expressed in Errors in Schema Construction and Structure (§5.1). [[XML Schema Structures]]
4. All schemas assembled from the XML Schema documents in the model's definition documents MUST satisfy the conditions expressed in sections § 5.1.1.1 – SML Constraint Construction on page 12, § 5.1.1.2 – Schema Component Rules on page 12, § 5.1.2.1 – SML Constraint Construction on page 13, § 5.1.2.2 – Schema Component Rules on page 13, § 5.2.1.1 – SML Constraint Construction on page 15, § 5.2.1.2 – Schema Component Rules on page 15, § 6.3.1 – SML Rule Construction on page 24 and § 6.3.2 – Schema Component Rules on page 24.
5. Each Schematron document in the model's definition documents MUST be a valid Schematron document [[ISO/IEC 19757-3]]

This specification does not define how schemas are assembled and which schema documents contribute to assembling the schemas.

A *conforming SML model is valid* if and only if it satisfies all of the following conditions:

1. In each instance document in the model, which is bound to a schema, the [validity] property of the root element MUST be "valid", and the [validity] property of all the other elements and all the attributes MUST NOT be "invalid", when schema validity is assessed with respect to any schema that is bound to this instance document. The schema validity assessment starts with no stipulated declaration or definition at the root element. [[XML Schema Structures]]

   The schema-validity of instance documents not bound to any schema does not contribute to the validity or invalidity of the model.

   

   How schemas are bound to instance documents is not defined by this specification. Multiple schemas may be bound to the same instance document.

   SML validity entails NOT being Schema-Invalid on the root or any descendant. SML validity can be non-vacuously checked only after assessment of Schema validity, and only on the portions of the subtree for which PSVI is available.

   Because the depth of PSVI is implementation-dependent, there is variability in the visibility of SML constraints available to the SML validator, and consequently in SML validity results.
2. Each document in the model MUST satisfy all applicable Schematron constraints when validated in the #ALL phase.
3. Each document in the model MUST satisfy all normative statements in this specification that pertain to model documents.

   

   This means, for example, that each document must satisfy all applicable sml:acyclic, sml:target*, and SML identity constraints.

9. SML Extensions Reference (Non-Normative)

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

9.1. Attributes

9.1.1. sml:acyclic

Used to specify that instances of an SML reference of a given type and its derived types do not create any cycles in a model

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

If this attribute is set to true for a complex type CT, then instances of CT (including any derived types of CT) that are SML references cannot create any cycles in a model. In the following example, HostedOnRefType is a complex type declaration whose instances cannot create a cycle:

```
<xs:complexType name="HostedOnRefType" sml:acyclic="true">
...
</xs:complexType>
```

If the `sml:acyclic` attribute is not specified or set to false for a complex type declaration, then instances of this type that are SML references may create cycles in a model.

9.1.2. sml:ref

This global attribute is used to identify SML references.

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

Any element that has sml:ref="true" will be treated as an SML reference.

9.1.3. sml:nilref

This global attribute is used to identify null SML references.

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

Any SML reference that has `sml:nilref="true"` or `sml:nilref="1"` will be treated as a null SML reference.

9.1.4. 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 any element declaration. 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 an instance of `win:Windows`

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

<xs:complexType name="HostOSRefType">
    <xs:sequence>
```

```
            <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

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

9.1.5. sml:targetRequired

Used to specify that instances of an SML reference must target elements in the model; i.e., an instance of the SML reference can not be null or contain an unresolved reference. Therefore it is an error if `sml:targetRequired="true"` is specified on an element declaration where the corresponding SML reference element R has `sml:nilref="true"`.

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

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

```
<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 name="HostOSRef" type="tns:HostOSRefType"
sml:targetRequired="true"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="HostOSRefType">
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

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

9.1.6. 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 any element declarations. 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="tns:OperatingSystemRefType"
            sml:targetType="ibm:LinuxType" minOccurs="0"/>

<xs:complexType name="OperatingSystemRefType">
    <xs:sequence>
        <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

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

9.1.7. sml:locid

This attribute can be defined on the `sch:assert`, `sch:report` and on any element with textual content. The `sml:locid` attribute is used to define the translation location for the text content of the containing element.

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

The mechanism for using the `QName` value of the `sml:locid` attribute to locate a translated text is implementation specific and hence outside the scope of this specification.

9.2. Elements

9.2.1. sml:key

This element is used to specify a key constraint in some scope. The semantics are the same as that for `xs:key` except that `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`.

9.2.2. 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="optional"/>
            </xs:extension>
        </xs:complexContent>
```

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

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

9.2.3. sml:unique

This element is used to specify a uniqueness constraint in some scope. The semantics are the same as that for `xs:unique` except that `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`.

9.2.4. sml:uri

Specifies an SML reference that is an instance of the SML URI reference scheme.

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

This element must be used to specify SML references that use the SML URI Reference Scheme.

9.3. XPath functions

9.3.1. smlfn:deref

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

This function takes a node set and attempts to resolve the SML references. The resulting node set is the set of elements that are obtained by successfully resolving (or de-referencing) the SML references. For example,

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

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

10. References

10.1. Normative

SML-IF 1.1

> *Service Modeling Language Interchange Format Version 1.1*, Bhalchandra Pandit, Valentina Popescu, Virginia Smith, Editors. World Wide Web Consortium, 12 May 2009. This version of the Service Modeling Language Interchange Format specification is available at http://www.w3.org/TR/2009/REC-sml-if-20090512/. The latest version of the Service Modeling Language Interchange Format Version 1.1 specification is available at http://www.w3.org/TR/sml-if/. Available at `http://www.w3.org/TR/2009/REC-sml-if-20090512/`.

IETF RFC 2119

Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, Author. Internet Engineering Task Force, June 1999. Available at http://www.ietf.org/rfc/rfc2119.txt. Available at `http://www.ietf.org/rfc/rfc2119.txt`.

IETF RFC 3986

Uniform Resource Identifier (URI): Generic Syntax , T. Berners-Lee, R. Fielding, L. Masinter, Authors. Internet Engineering Task Force, January 2005. Available at http://www.ietf.org/rfc/rfc3986.txt. Available at `http://www.ietf.org/rfc/rfc3986.txt`.

IETF RFC 3987

Internationalized Resource Identifiers (IRIs) , M. Duerst, M. Suignard, Authors. Internet Engineering Task Force, January 2005. Available at http://www.ietf.org/rfc/rfc3987.txt. Available at `http://www.ietf.org/rfc/rfc3987.txt`.

RFC 2234

Augmented BNF for Syntax Specifications: ABNF, Internet Mail Consortium, November 1997. Available at http://www.rfc-editor.org/rfc/rfc2234.txt. Available at `http://www.rfc-editor.org/rfc/rfc2234.txt`.

ISO/IEC 19757-3

Information technology Document Schema Definition Languages (DSDL) Part 3: Rule-based validation Schematron. International Organization for Standardization and International Electrotechnical Commission, 1 January 2006. Available at http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip Available at `http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip`.

XML-NS

Namespaces in XML 1.0 (Second Edition), Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin, Editors. World Wide Web Consortium, 16 August 2006. This version of the Namespaces in XML Recommendation is http://www.w3.org/TR/2006/REC-xml-names-20060816/. The latest version of Namespaces in XML is available at http://www.w3.org/TR/xml-names/. Available at `http://www.w3.org/TR/2006/REC-xml-names-20060816/`.

XML

Extensible Markup Language (XML) 1.0 (Fourth Edition), T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, Editors. World Wide Web Consortium, 10 February 1998, revised 16 August 2006. The edition cited (http://www.w3.org/TR/2006/REC-xml-20060816) was the one current at the date of publication of this specification as a Candidate Recommendation. The latest version of XML 1.0 is available at http://www.w3.org/TR/xml/. Implementations may follow the edition cited and/or any later edition(s); it is implementation-defined which editions are supported by an implementation. Available at `http://www.w3.org/TR/2006/REC-xml-20060816/`.

XML Information Set

XML Information Set (Second Edition), John Cowan, Richard Tobin, Editors. World Wide Web Consortium, 4 February 2004. This version of the XML Information Set Recommendation is http://www.w3.org/TR/2004/REC-xml-infoset-20040204/. The latest version of XML Information Set is available at http://www.w3.org/TR/xml-infoset/. Available at `http://www.w3.org/TR/2004/REC-xml-infoset-20040204/`.

XML Schema Structures

XML Schema Part 1: Structures Second Edition, H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, Editors. World Wide Web Consortium, 2 May 2001, revised 28 October 2004. This version of the XML Schema Part 1 Recommendation is http://www.w3.org/TR/2004/REC-xmlschema-1-20041028. The latest version of XML Schema 1.0 Part 1 is available at http://www.w3.org/TR/xmlschema-1. Available at `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/`.

XML Schema Datatypes

XML Schema Part 2: Datatypes Second Edition, P. Byron and A. Malhotra, Editors. World Wide Web Consortium, 2 May 2001, revised 28 October 2004. This version of the XML Schema Part 2 Recommendation is http://www.w3.org/TR/2004/REC-xmlschema-2-20041028. The latest version of XML Schema 1.0 Part 2 is available at http://www.w3.org/TR/xmlschema-2. Available at `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/`.

XML Schema for Schemas

XML Schema for XML Schemas. World Wide Web Consortium, 2 May 2001, revised 28 October 2004. Available at `http://www.w3.org/2001/XMLSchema.xsd`.

XPath

XML Path Language (XPath) Version 1.0, J. Clark and S. DeRose, Editors. World Wide Web Consortium, 16 November 1999. This version of XML Path Language (XPath) Version 1.0 is http://www.w3.org/TR/1999/REC-xpath-19991116. The latest version of XML Path Language (XPath) Version 1.0 is available at http://www.w3.org/TR/xpath. Available at `http://www.w3.org/TR/1999/REC-xpath-19991116`.

XPointer

XPointer Framework, P. Grosso, E. Maler, J. Marsh, and N. Walsh, Editors. World Wide Web Consortium, 25 March 2003. This version of the XPointer Framework Recommendation is http://www.w3.org/TR/2003/REC-xptr-framework-20030325/ The latest version of XPointer Framework is available at http://www.w3.org/TR/xptr-framework/. Available at `http://www.w3.org/TR/2003/REC-xptr-framework-20030325/`.

xmlns() Scheme

XPointer xmlns() Scheme, S. DeRose, R. Daniel Jr., E. Maler, and J. Marsh, Editors. World Wide Web Consortium, 25 March 2003. This version of the XPointer xmlns() Scheme Recommendation is http://www.w3.org/TR/2003/REC-xptr-framework-20030325/ The latest version of XPointer xmlns() Scheme is available at http://www.w3.org/TR/xptr-xmlns/. Available at `http://www.w3.org/TR/2003/REC-xptr-xmlns-20030325/`.

10.2. Non-Normative

Introduction to Schematron

> *An Introduction to Schematron*, Eddie Robertsson, Author. O'Reilly Media, Inc., 12 November 2003. Available at http://www.xml.com/pub/a/2003/11/12/schematron.html Available at `http://www.xml.com/pub/a/2003/11/12/schematron.html`.

Improving Validation with Schematron

> *Improving XML Document Validation with Schematron*, Dare Obasanjo, Author. Microsoft Corporation, September 2004. Available at http://msdn.microsoft.com/en-us/library/aa468554.aspx Available at `http://msdn.microsoft.com/en-us/library/aa468554.aspx`.

XML Schema Primer

> *XML Schema Part 0: Primer Second Edition*, D. Fallside and P. Walmsley, Editors. World Wide Web Consortium, 2 May 2001, revised 28 October 2004. This version of the XML Schema Part 0 Recommendation is http://www.w3.org/TR/2004/REC-xmlschema-0-20041028. The latest version of XML Schema Part 0 is available at http://www.w3.org/TR/xmlschema-0. Available at `http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/`.

Appendix A. Normative SML Schema

```
<!--
/*
 * Copyright © ns World Wide Web Consortium,
 *
 * (Massachusetts Institute of Technology, European Research Consortium for
 * Informatics and Mathematics, Keio University). All Rights Reserved. This
 * work is distributed under the W3C® Document License [1] in the hope that
 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 *
 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231
 */
--><xs:schema
      xmlns:sml="http://www.w3.org/ns/sml"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.w3.org/ns/sml"
      elementFormDefault="qualified"
      blockDefault="#all"
      version="1.0"
      xml:lang="en"
      finalDefault=""
      attributeFormDefault="unqualified">
   <!--
        References
        ==========
```

```
 -->

<!-- CONTEXT: To be used in any <xs:element> -->
<xs:attribute name="ref" type="xs:boolean">
   <xs:annotation>
      <xs:documentation>
             Specifies if the element contains a reference
         </xs:documentation>
   </xs:annotation>
</xs:attribute>
<!-- CONTEXT: To be used in any <xs:element> -->
<xs:attribute name="nilref" type="xs:boolean">
   <xs:annotation>
      <xs:documentation>
             Specifies that the reference element denotes a “null” reference.
             To be used only on elements for which sml:ref="true".
         </xs:documentation>
   </xs:annotation>
</xs:attribute>
<!-- CONTEXT: To be used in any <xs:element> -->
<xs:attribute name="targetElement" type="xs:QName">
   <xs:annotation>
      <xs:documentation>
             A qualified name of a global element in the referenced document.
         </xs:documentation>
   </xs:annotation>
</xs:attribute>
<!-- CONTEXT: To be used in any <xs:element>-->
<xs:attribute name="targetRequired" type="xs:boolean">
   <xs:annotation>
      <xs:documentation>
             If true, requires the target element of the reference to
             exist in the model.
         </xs:documentation>
   </xs:annotation>
</xs:attribute>
<!-- CONTEXT: To be used in any <xs:element>-->
<xs:attribute name="targetType" type="xs:QName">
   <xs:annotation>
      <xs:documentation>
             A qualified name of the type of the element in the
             referenced document.
         </xs:documentation>
   </xs:annotation>
</xs:attribute>
```

```
   <!-- CONTEXT: To be used in any <xs:complexType>-->
   <xs:attribute name="acyclic" type="xs:boolean">
      <xs:annotation>
         <xs:documentation>
                If this attribute is set to true for a type D
                then instances of D should not create any
                cycles in a model. See Section 5.1.1.3 titled "Instance Validity
Rules".
            </xs:documentation>
      </xs:annotation>
   </xs:attribute>
   <!-- CONTEXT: To be used in <sch:assert>, <sch:report>
         and elements with textual content.
         This attribute is used to support string localization.
         It is used to define the translation location for
         the text content of the containing element.-->
   <xs:attribute name="locid" type="xs:QName"/>
   <!-- CONTEXT: Represents a reference using the URI scheme. To be
         used as a child  element of elements for which
         sml:ref="true". -->
   <xs:element name="uri" type="xs:anyURI">
      <xs:annotation>
         <xs:documentation>
                References in URI scheme must be representend by this
                element.
            </xs:documentation>
      </xs:annotation>
   </xs:element>
   <!--
        Uniqueness and Key constraints
        ==============================
    -->
   <xs:complexType name="keybase" mixed="false">
      <xs:sequence minOccurs="0">
         <xs:element
               name="selector"
               type="sml:selectorXPathType"/>
         <xs:element
               name="field"
               type="sml:fieldXPathType"
               maxOccurs="unbounded"/>
         <xs:any
               namespace="##other"
               minOccurs="0"
               maxOccurs="unbounded"
```

```
            processContents="lax"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:NCName"/>
   <xs:attribute name="ref" type="xs:QName"/>
   <xs:anyAttribute
         namespace="##other"
         processContents="lax"/>
</xs:complexType>
<xs:element name="key" type="sml:keybase"/>
<xs:element name="unique" type="sml:keybase"/>
<xs:element name="keyref">
   <xs:complexType mixed="false">
      <xs:complexContent>
         <xs:extension base="sml:keybase">
            <xs:attribute
                  name="refer"
                  type="xs:QName"
                  use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:element>
<!--
     Other Complex Types
     ===================
 -->
<xs:complexType name="selectorXPathType" mixed="false">
   <xs:sequence>
      <xs:any
            namespace="##other"
            minOccurs="0"
            maxOccurs="unbounded"
            processContents="lax"/>
   </xs:sequence>
   <xs:attribute name="xpath" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:string">
<!-- The value MUST conform to the selector BNF grammar defined in
                    section '4.4 Identity Constraints' in the SML specification.

                -->
</xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:anyAttribute
```

```
            namespace="##other"
            processContents="lax"/>
   </xs:complexType>
   <xs:complexType name="fieldXPathType" mixed="false">
      <xs:sequence>
         <xs:any
               namespace="##other"
               minOccurs="0"
               maxOccurs="unbounded"
               processContents="lax"/>
      </xs:sequence>
      <xs:attribute name="xpath" use="required">
         <xs:simpleType>
            <xs:restriction base="xs:string">
   <!-- The value MUST conform to the field BNF grammar defined in
                        section '4.4 Identity Constraints' in the SML specification.

                    -->
   </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
      <xs:anyAttribute
            namespace="##other"
            processContents="lax"/>
   </xs:complexType>
</xs:schema>
```

Appendix B. Model Definition Document Sample (Non-Normative)

This sample illustrates the use of the following SML extensions:

1.
2. `key` and `keyref` constraints
3. Schematron constraints

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

<!--
/*
 * Copyright ©  World Wide Web Consortium,
 *
 * (Massachusetts Institute of Technology, European Research Consortium for
 * Informatics and Mathematics, Keio University). All Rights Reserved. This
```

```
 * work is distributed under the W3C® Document License [1] in the hope that
 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 *
 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231
 */
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://example.org/SampleModel" xmlns:sml="http://www.w3.org/ns/sml"
xmlns:smlfn="http://www.w3.org/ns/sml-function"
xmlns:sch="http://purl.oclc.org/dsdl/schematron"
targetNamespace="http://example.org/SampleModel" elementFormDefault="qualified"
finalDefault="" blockDefault="" attributeFormDefault="unqualified">

  <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" mixed="false">
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </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" mixed="false">
    <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>

  <xs:complexType name="GuestAppRefType" sml:acyclic="false" mixed="false">
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>

  <xs:element name="GuestAppRef" type="tns:GuestAppRefType"
sml:targetType="tns:ApplicationType"/>

  <xs:complexType name="OperatingSystemType" mixed="false">
    <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 mixed="false">
   <xs:sequence>
     <xs:element ref="tns:GuestAppRef" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

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

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

  <xs:complexType name="WorkstationType" mixed="false">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element ref="tns:OSRef"/>
      <xs:element name="Applications" minOccurs="0">
 <xs:complexType mixed="false">
   <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://www.w3.org/ns/sml-function"/>
   <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="tns: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>


  <!-- 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="tns: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://www.w3.org/ns/sml-function"/>
   <sch:pattern id="SecureApplication">
     <sch:rule context="sm:Applications/sm:Application">
       <sch:report test="smlfn:deref(.)[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="smlfn:deref(.)[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 C. SML References Sample (Non-Normative)

The following example illustrates the use of SML references. Consider the following schema fragment:

```
<xs:element name="EnrolledCourse">
    <xs:complexType>
        <xs:sequence>
          <xs:element name="Name" type="xs:string"/>
          <xs:element name="Grade" type="xs:string"/>
          <xs:any namespace="##any" minOccurs="0"
```

```
                maxOccurs="unbounded" processContents="lax"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>
</xs:element>

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

```
<Student xmlns="http://www.university.example.org/ns"
         xmlns:sml="http://www.w3.org/ns/sml"
         xmlns:u="http://www.university.example.org/ns">
    <ID>1000</ID>
    <Name>John Doe</Name>
    <EnrolledCourses>
        <EnrolledCourse sml:ref="true">
            <Name>PHY101</Name>
            <Grade>A</Grade>
            <sml:uri>
                http://www.university.example.org/Universities/MIT/Courses.xml
                #smlxpath1(/u:Courses/u:Course[u:Name='PHY101'])
            </sml:uri>
        </EnrolledCourse>
        <EnrolledCourse sml:ref="false">
            <Name>MAT100</Name>
            <Grade>B</Grade>
            <sml:uri>
                http://www.university.example.org/Universities/MIT/Courses.xml
                #smlxpath1(/u:Courses/u:Course[u:Name='MAT100'])
            </sml:uri>
        </EnrolledCourse>
```

```
        <EnrolledCourse>
            <Name>SocialSkills</Name>
            <Grade>F</Grade>
        </EnrolledCourse>
    </EnrolledCourses>
</Student>
```

The first `EnrolledCourse` element in the above example is an SML reference since it specifies `sml:ref="true"`. It uses the SML URI Reference Scheme to target the element for course PHY101. The second and third `EnrolledCourse` elements are not SML references; the second element specifies `sml:ref="false"` and the third element does not specify the `sml:ref` attribute. Note that the second `EnrolledCourse` element contains an `sml:uri` element which satisfies the syntax of the SML URI Reference Scheme (referring to course MAT100) but this will be ignored since `sml:ref="false"` for this `EnrolledCourse` element.

Note that, there are no SML constraints defined on the `EnrolledCourse` element or on the type of that element in the schema. Therefore, even if the first `EnrolledCourse` element instance is marked as an SML reference, no SML constraints are evaluated for that element during model validation. However, checks such as the ones defined in section § 4.2.1 – At Most One Target on page 7 and section § 4.2.2 – Consistent References on page 7 are still performed on that SML reference during model validation.

An `EnrolledCourse` SML reference can be a marked as a null reference if it specifies the `sml:nil-ref="true"` attribute as shown in the following example (the first `EnrolledCourse` element is a null SML reference):

```
<Student xmlns="http://www.university.example.org/ns"
         xmlns:sml="http://www.w3.org/ns/sml"
         xmlns:u="http://www.university.example.org/ns">
    <ID>1000</ID>
    <Name>John Doe</Name>
    <EnrolledCourses>
        <EnrolledCourse sml:ref="true" sml:nilref="true">
            <Name>PHY101</Name>
            <Grade>A</Grade>
        </EnrolledCourse>
        <EnrolledCourse sml:ref="false">
            <Name>MAT100</Name>
            <Grade>B</Grade>
            <sml:uri>
                http://www.university.example.org/Universities/MIT/Courses.xml
                #smlxpath1(/u:Courses/u:Course[u:Name='MAT100'])
            </sml:uri>
        </EnrolledCourse>
        <EnrolledCourse>
            <Name>SocialSkills</Name>
            <Grade>F</Grade>
        </EnrolledCourse>
```

```
    </EnrolledCourses>
</Student>
```

In the above example, the first SML reference, `EnrolledCourse`, defines the `sml:nilref="true"` attribute which marks this as a null SML reference. By specifying a null reference, the document author makes an explicit declaration that this `Student` element does not refer to any target element. Specifying a null reference does not have any SML-defined effect on the interpretation of element in non-SML contexts. In particular, in this case, SML says nothing about the interpretation of the `Grade` and `Name` elements. Any such interpretation is left to the application, its usage context, other specifications, etc.

Appendix D. SML URI Reference Scheme Sample (Non-Normative)

The following example illustrates the use of the SML URI Reference Scheme [§ 4.3.1 – SML URI Reference Scheme on page 10]. Consider the case where all courses offered by MIT are stored in a single XML document – `Courses.xml` – whose URI is `http://www.university.example.org/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="http://www.university.example.org/ns">
    <ID>1000</ID>
    <Name>John Doe</Name>
    <EnrolledCourses>
       <EnrolledCourse sml:ref="true" xmlns:u="http://www.university.example.org/ns">
            <sml:uri>
                http://www.university.example.org/Universities/MIT/Courses.xml
                #smlxpath1(/u:Courses/u:Course[u:Name='PHY101'])
            </sml:uri>
        </EnrolledCourse>
    </EnrolledCourses>
</Student>
```

An SML reference can also reference an element in its own document. To see this consider the following instance document

```
<University xmlns="http://www.university.example.org/ns">
    <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:ref="true"
xmlns:u="http://www.university.example.org/ns">
                    <sml:uri>
                      #smlxpath1(/u:University/u:Courses/u:Course[u:Name='MAT200'])
                    </sml:uri>
                </EnrolledCourse>
            </EnrolledCourses>
        </Student>
    </Students>
</University>
```

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

Appendix E. SML Identity Constraints Sample (Non-Normative)

The following example will be used to illustrate the `sml:key`, `sml:unique`, and `sml:keyref` constraints across SML references. This example consists of three schema documents. `university.xsd` contains the type definitions for a University element, a Student SML reference and a Course SML reference. `students.xsd` contains the type definitions for an EnrolledCourse SML reference and a Student element. `courses.xsd` contains the type definition for a Course element.

```
<!-- from university.xsd -->
<xs:complexType name="StudentRefType">
    <!-- SML reference to a Student -->
    <xs:sequence>
        <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>

<xs:element name="Student" type="StudentRefType"/>

<xs:complexType name="CourseRefType">
    <!-- SML reference to a Course -->
    <xs:sequence>
        <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
```

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

<xs:element name="Course" type="CourseRefType"/>

<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="Student" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
        </xs:element>
        <xs:element name="Courses" minOccurs="0">
        <xs:complexType>
            <xs:sequence>
              <xs:element ref="Course" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<!-- from students.xsd -->
<xs:complexType name="EnrolledCourseRefType">
    <!-- SML reference to a Course -->
    <xs:sequence>
        <xs:element name="Grade" type="xs:string"/>
        <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>

<xs:element name="EnrolledCourse" type="EnrolledCourseRefType"/>

<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="EnrolledCourse" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:element name="Students">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Student" type="StudentType"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<!-- from courses.xsd -->
<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 name="EnrolledStudent" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="StudentID" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:element name="Courses">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Course" type="CourseType"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

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 declaration is the context for the key and unique constraints.

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

```
<xs:element name="PrivateUniversity" type="tns:UniversityType">
    <xs:annotation>
        <xs:appinfo>
            <sml:unique ref="tns:StudentSSNisUnique"/>
        </xs:appinfo>
    </xs:annotation>
</xs:element>
```

In the above example, the `PrivateUniversity` element declaration specifies the `StudentSSNisUnique` unique constraint by referencing its `name` in the `University` element declaration.

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 SML references and across XML documents. The following example uses `sml:keyref` to capture the requirement that students enrolled in a course must be currently enrolled in the 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="tns:StudentIDisKey">
                <sml:selector
xpath="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 a course is a subset of the set of IDs of students currently enrolled in the university. In particular, the `selector` and `field` elements in `StudentIDisKey` key constraint identify the set of IDs of students currently enrolled in the university, and the `selector` and `field` elements in `CourseStudents` key reference constraint identify the set of IDs of students enrolled in courses.

Appendix F. Localization and Variable Substitution Samples (Non-Normative)

The following examples demonstrate how localization can be applied to a message text resulting from Schematron rule evaluation, allowing a to support multiple locales without changes to either the model processor or the Schematron rules. Summarized below are the benefits resulting from using the `sml:locid` localization support:

1. The Schematron rule message text is locale-independent in the sense that the author does not have to be concerned with the rule evaluator's runtime locale. The Schematron rule is defined generically, consumable for any evaluator (including a model processor) for which a translation file is made available at the location defined by the `sml:locid` value's namespace name.
2. There is a clear separation between the message text translation, Schematron rule authoring, and Schematron rule evaluator code. The Schematron rules require no changes when translations for other languages are made available. The same Schematron rule can be used by multiple evaluators, each

supporting a distinct set of languages. To support a new language, all that needs to be done is to add a new translation file under the location identified by the `sml:locid` value's namespace name.

Building on the preceding university example, the following example of a Schematron rule uses the `sml:locid` attribute to locate translation information for the Schematron `sch:assert` error message:

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron"
            xmlns:lang="http://www.university.example.org/translation/">

    <sch:ns prefix="u" uri="http://www.university.example.org/ns" />
    <sch:ns prefix="smlfn" uri="http://www.w3.org/ns/sml-function"/>
    <sch:pattern id="StudentPattern”>
        <sch:rule context="u:Students/u:Student">
            <sch:assert test=".[starts-with(u:ID,'99')]"
                        sml:locid="lang:StudentIDErrorMsg">
              The specified ID <sch:value-of select="string(u:ID)"/> does not begin
 with 99.
            </sch:assert>
        </sch:rule>
    </sch:pattern>
</sch:schema>
```

In general, the namespace name can point to a file containing the translated message, a folder containing a set of translated files or any other type of resource that can help locate the translated message. It is implementation-dependent how the model validator makes use of this information for finding the actual resource containing the translated message.

In this concrete example, the namespace name of the `sml:locid` attribute value is used to define the location of the resource containing the translated text:

```
xmlns:lang="http://www.university.example.org/translation/"
```

Also, in this concrete example, `http://www.university.example.org/translation/` names a folder containing a set of translation resources, and there is one set of translation files located under `http://www.university.example.org/translation/`. Each of these translation files corresponds to a language into which the messages have been translated. The translations to French and German are available in the following files:

1. File `http://www.university.example.org/translation/lang_fr.txt` contains the French translation of the `sch:assert` message.
2. File `http://www.university.example.org/translation/lang_de.txt` contains the German translation of the `sch:assert` message.

The {local part} of the `sml:locid` attribute value (`StudentIDErrorMsg`) is used to define the identity of the message being translated. This identity is used to locate the translated text within the translation resource.

The French translation of the `sch:assert` message is found in the following entry:

```
StudentIDErrorMsg = L'identifiant specifié <sch:value-of select="string(u:ID)"/> ne
 commence pas par 99.
```

The German translation for the `sch:assert` message is found in the following entry:

```
StudentIDErrorMsg = Der angegebene Wert des Attributs ID (<sch:value-of
select="string(u:ID)"/>) beginnt nicht mit "99".
```

Translatable messages, especially strings containing XML tags (such as `<sch:value-of select="string(u:ID)"/>` in the example below), may be best stored in XML containers. This allows more flexibility to manipulate and translate the data. For example, the XML document could utilize ITS (see Internationalization Tag Set (ITS) specification) to add localization-related information.

```
<?xml version="1.0" encoding="UTF-8"/>
<messages xml:lang="en"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron"
  xmlns:its="http://www.w3.org/2005/11/its">
    <msg xml:id='StudentIDErrorMsg'
          its:locNote="This message should not be longer than 128 characters">
         The specified ID <sch:value-of select="string(u:ID)"/> does not begin with
 99.
    </msg>
</messages>
```

Best Practice Recommendation for Variable Substitution

It is often the case that translated text can be reused, for example, a Schematron message can be reused in different `sch:assert` or `sch:report` rules and patterns. In the example above, the author of the Schematron rule may want to use this error message in other contexts:

```
The specified ID <sch:value-of select="string(u:ID)"/> does not begin with 99.
```

Reuse opportunities for this message are limited since the translated message contains implicit assumptions about the rule context. To be able to reuse this message more widely, the message author should substitute a context-independent expression, such as a variable, in place of the `u:ID` in `<sch:value-of select="string(u:ID)"/>`.

Thus, the original message translations

```
StudentIDErrorMsg = L'identifiant specifié <sch:value-of select="string(u:ID)"/> ne
 commence pas par 99.
StudentIDErrorMsg = Der angegebene Wert des Attributs ID (<sch:value-of
select="string(u:ID)"/>) beginnt nicht mit "99".
```

should instead be coded to use a variable's value, as shown below.

```
StudentIDErrorMsg = L'identifiant specifié <sch:value-of select="$studentID"/> ne
commence pas par 99.
StudentIDErrorMsg = Der angegebene Wert des Attributs ID (<sch:value-of
select="$studentID"/>) beginnt nicht mit "99".
```

The error message in `sch:assert`, identified by the `lang:StudentIDErrorMsg` value, can now be reused in Schematron rule contexts other than the one described by the sample above. In cases where variable substitution is used in this way, the responsibility for setting the variable in the message's various usage contexts rests with the rule author(s). Specifically, the Schematron rule used to start the Appendix F portion of this running example would need to set the value of the `studentID` variable. The sample below shows how the revised translation resources would be referenced from a Schematron rule.

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron"
            xmlns:lang="http://www.university.example.org/translation/">
    <sch:ns prefix="u" uri="http://www.university.example.org/ns" />
    <sch:ns prefix="smlfn" uri="http://www.w3.org/ns/sml-function"/>
    <sch:pattern id="StudentPattern”>
        <sch:rule context="u:Students/u:Student">
        <sch:let name="studentID" value="u:ID"/>
            <sch:assert test="starts-with(u:ID,'99')"
                        sml:locid="lang:StudentIDErrorMsg">
                The specified ID <sch:value-of select="$studentID"/> does not begin
 with 99.
            </sch:assert>
        </sch:rule>
    </sch:pattern>
</sch:schema>
```

The error message in `sch:assert` and the localization identifier `lang:StudentIDErrorMsg` can now be reused in contexts other than `u:Students/u:Student`.

If the translated string named by `StudentIDErrorMsg` is intended for use only in the context of Schematron elements, this mechanism is sufficient. If the message is intended for use in additional contexts, the preceding syntax is not necessarily sufficient. In order to allow reuse of translated strings exploiting variable substitution in other contexts, a syntax understood both inside and outside of Schematron elements is required. `xsl:variable` and `sch:let` are possible choices, made more practical by SML's requirement that validators support the “xslt" query binding for Schematron. However, there is currently insufficient practical experience to label this a best practice.

Appendix G. Acknowledgements (Non-Normative)

The editors acknowledge the members of the Service Modeling Language Working Group, the members of other W3C Working Groups, and industry experts in other forums who have contributed directly or indirectly to the process or content of creating this document.

At the time this specification was published, the members of the Service Modeling Language Working Group were:

John Arwe (IBM Corporation), Len Charest (Microsoft Corporation), Sandy Gao (IBM Corporation), Paul Lipton (CA), James Lynn (HP), Kumar Pandit (Microsoft Corporation), Valentina Popescu (IBM Corporation), Virginia Smith (HP), Henry Thompson (W3C/ERCIM), David Whiteman (IBM Corporation), Kirk Wilson (CA).

The Service Modeling Language Working Group has benefited in its work from the participation and contributions of a number of people not currently members of the Working Group, including in particular those named below.

Dave Ehnebuske (IBM), Jon Hass (Dell), Steve Jerman (Cisco), Heather Kreger (IBM), Vincent Kowalski (BMC), Milan Milenkovic (Intel), Bryan Murray (HP), Phil Prasek (HP), Junaid Saiyed (EMC), Harm Sluiman (IBM), C. Michael Sperberg-McQueen (W3C/MIT), Bassam Tabbara (Microsoft), Vijay Tewari (Intel), William Vambenepe (HP), Marv Waschke (CA), Andrea Westerinen (Microsoft), Pratul Dublish (Microsoft), Julia McCarthy (IBM).

Affiliations given above are those current at the time of their work with the working group.

This page is intentionally left blank.