Service Modeling Language Interchange Format Version 1.1

W3C Recommendation 12 May 2009

This version:

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

Latest version:

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

Previous version:

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

Authors and Contributors:

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

Abstract

This specification defines the interchange format for Service Modeling Language, Version 1.1 (SML) models. This format identifies the model being interchanged, distinguishes between model definition documents and model instance documents, and defines the binding of rule documents with other documents in the interchange model.

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 Interchange Format 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

1. Introduction (Non-Normative)

As defined in the Service Modeling Language, Version 1.1 (SML) Specification [[SML 1.1]] an SML model is a collection of XML documents that may be used to describe complex services and systems such as a set of IT resources, services and their interrelations.

In every SML model there are two distinguished subsets of the model's documents, the definition documents and the instance documents. The model's definition documents describe the abstract structure of the model, and provide much of the information a model validator needs to decide whether the model, as a whole, is valid. The model's instance documents describe or support the description of the individual resources that the model portrays.

The SML Specification identifies two categories of model definition documents that participate in SML model validation: Schema documents and rule documents. Schema documents in a model are XML documents that conform to the [[SML 1.1]] defined extensions to XML Schema [[XML Schema Structures], [XML Schema Datatypes]]. Rule documents in a model include XML documents that conform to the [[SML 1.1]] defined extensions of Schematron [[ISO/IEC 19757-3]].

To ensure accurate and convenient interchange of the documents that make up an SML model, it is useful to define both an implementation-neutral interchange format that preserves the content and interrelationships among the documents and a constrained form of SML model validation. For this purpose, this specification defines a standard format called the SML Interchange Format (SML-IF) and a process called interchange model validation.

The specification consists of two parts. The first part is an informal description of SML-IF to set the context. This is followed by SML-IF's normative definition.

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 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 are links to the relevant definition, set off with curly braces, for instance {example property}. References to properties of information items as defined in [[XML Information Set]] are notated as links to the relevant section thereof, set off with square brackets, for example [children].

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. This specification also uses terms defined in the [[SML 1.1]] specification.

Alias

An *alias* is a temporary name assigned to a model document [[SML 1.1]] within the context of an .

Implementation-Defined

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

Implementation-Dependent

An *implementation-dependent* feature or behavior may vary among processors conforming to this specification; 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.

Interchange Model

An *interchange model* is an SML model [[SML 1.1]] being interchanged.

Interchange Model Validation

Interchange model validation is the process of performing SML model validation [[SML 1.1]] on the interchange model while maintaining all assertions and interrelationships among the documents in the interchange model as defined by this specification.

Schema Binding

A *schema binding* is an association of a namespace with a set of schema documents in the and the instance documents [[SML 1.1]] that should be validated against this set of schema documents.

SML-IF Consumer

An *SML-IF consumer* is a program that processes an using, in whole or part, semantics defined by this specification. It may or may not perform .

SML-IF Document

An *SML-IF document* is an XML representation of an . It includes the model's identity, its documents (by value or by reference), metadata about its documents, and a syntactic representation of concepts defined as part of an SML model but lacking an SML-defined sytnax (e.g. rule bindings).

SML-IF Producer

An *SML-IF producer* is a program able to generate an from an SML model.

3. Dependencies on Other Specifications

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

Conforming implementations of this specification MUST support SML 1.1 [[SML 1.1]], XML 1.0 [[XML]] and XML Schema 1.0 [[XML Schema Structures], [XML Schema Datatypes]]. Conforming implementations MAY additionally support later versions of the XML or XML Schema specifications.

Although SML 1.1 and SML-IF allow conforming implementations to support newer versions of dependent specifications, there are interoperability implications to be considered when documents based on those versions are interchanged using SML-IF. When an SML-IF document interchanges data built using newer versions of the SML and SML-IF dependent specifications, consumers of the SML-IF document not supporting these versions may be unable to interpret some of the data exchanged by this document.

4. Informal Description (Non-Normative)

To represent an SML model in a standard way for interchange, the following topics need to be addressed.

Packaging: The collection of XML documents that make up a model to be interchanged need to be gathered together. In doing so, the model definition and model instance documents need to be distinguished from one another since they play distinct roles in the model.

Explicit references: The documents to be interchanged may explicitly refer to one another and to documents that are not packaged with the documents being interchanged. [[SML 1.1]] SML references among SML model instance documents are an obvious example. Less obvious are such references as certain `schemaLocation` attributes in schema documents and `xsi:schemaLocation` attributes in instance documents. Section § 4.4 – Schema Bindings on page 11 defines how `schemaLocation` is processed in these cases.

Rule bindings and schema bindings: [[SML 1.1]] permits models in which rule documents apply to all, none, or subsets of the model's documents. SML-IF specifies how to describe which rule documents apply to which of the model's documents.

Model validation: The process of SML model validation defined in [[SML 1.1]] contains points of variability that, left unconstrained, would make it difficult for SML-IF to ensure interoperability of independent implementations in any practical way. Many of these sources of variability are inherited from other specifications that SML uses, e.g. URI comparison RFC 3986 ([[IETF RFC 3986]]) and the source of Schema components ([[XML Schema Structures]]) used to validate model instance documents. SML-IF constrains these points of variability, with the goal of ensuring interoperability when specific conditions are met and of increasing the likelihood of interoperability in other cases. The enforcement of these additional constraints on SML model validation occurs during the process of .

4.1. Packaging

An SML-IF document packages a collection of SML model documents to be interchanged as a single XML document. All SML-IF documents conform to the XML Schema defined in the normative part of this specification.

Informally, the structure of SML-IF documents, using the pseudo-schema notation from WSDL 2.0 [[WSDL 2.0 Core Language]] is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<model xmlns="http://www.w3.org/ns/sml-if"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
      SMLIFVersion="xs:token Version number of the SML-IF spec used to generate the
 current document"
       schemaComplete="xs:boolean">
    <identity>
        <name>
            xs:anyURI Namespace identifying the model
        </name>
        <version> ?
            xs:token <!-- The version of this model. E.g., 1.2 or 0.3 -->
        </version>
        <displayName sml:locid="xs:anyURI URI identifying the translation
```

```
                                resource for the display name" ?> ?
            xs:string Descriptive name of model intended for display
        </displayName>
        <baseURI>
          xs:anyURI <!-- Base URI for relative references defined in the interchange
 model;
                           must be an absolute reference -->
        </baseURI> ?
        <description sml:locid="xs:anyURI URI identifying the translation
                                resource for the description" ?> ?
            xs:string Textual description of model for human consumption
        </description>
    </identity>
    <ruleBindings> ?
        <ruleBinding> *
            <documentAlias="xs:anyURI"/> ?
            <ruleAlias="xs:anyURI"/>
        </ruleBinding>
    </ruleBindings>
    <schemaBindings> ?
        <defaultSchema> ?
            <namespaceBinding/> *
        </defaultSchema>
        <schemaBinding> *
            <namespaceBinding/> *
            <documentAlias/> *
        </schemaBinding>
        <noSchemaBinding> ?
            <documentAlias/> *
        </noSchemaBinding>
    </schemaBindings>
    <definitions> ?
        <document> *
            <docInfo> ?
             <baseURI> ?
                 xs:anyURI <!-- If a document has a baseURI, then this will be used
 to form the
                                base URI for all relative URIs subject to SML URI
processing
                                contained by that document. -->
             </baseURI>
             <aliases> ?
                 <alias> *
                   xs:anyURI <!-- A URI by which SML references from other documents
 may refer to this document. -->
```

```
                </alias>
            </aliases>
           </docInfo>
               [
               <data>
                   xs:any <!-- At most one definition document goes here -->
               </data>
               |
               <base64Data>
                   xs:any <!-- At most one base64 encoded definition document goes
here -->
               </base64Data>
               |
               <locator>
                   <documentURI/> ?
                     xs:any <!-- A URI or IRI that points to a definition document
goes here -->
               </locator>
               ]
       </document>
   </definitions>
   <instances> ?
       <document> *
           <docInfo> ?
               <baseURI> ?
                    xs:anyURI <!-- If a document has a baseURI, then this will be
used to form the
                              base URI for all relative URIs subject to SML URI
processing
                              contained by that document. -->
               </baseURI>
               <aliases> ?
                  <alias> *
                      xs:anyURI <!-- A URI by which SML references from other
documents may refer to this document. -->
                  </alias>
               </aliases>
           </docInfo>
           [
           <data>
               xs:any<!-- At most one instance document goes here -->
           </data>
           |
           <base64Data>
               xs:any <!-- At most one base64 encoded instance document goes here
```

```
-->
            </base64Data>
            |
            <locator>
                <documentURI/> ?
                    xs:any <!-- A URI or IRI that points to an instance document
goes here -->
            </locator>
            ]
        </document>
    </instances>
</model>
```

A document producer can specify the version of the specification under which the current document was generated, and with which conformance is claimed, in the `SMLIFVerion` attribute. For example, if this version of SML-IF is used as the basis of a document, the value of this attribute would be the value "1.1".

The `identity` element provides information applications can use to identify and describe the set of SML documents being interchanged. The `baseURI` child element is one way to specify a base URI to be used by relative URI references in the . Another way to specify a base URI is to use the `document/docInfo/baseURI` element. [§ 5.3.2 – Base URIs on page 17]

The `SMLIFVersion` attribute is defined on the `model` element and may be useful when diagnosing failures encountered while processing SML-IF documents. For example, if a document asserts conformance with version 1.1 of the SML-IF specification and a human can see that it is not in fact conformant, then it is likely that the problem occurred during the production of the document. If the same document appears to humans to be conformant, then the focus of diagnosis might shift toward the and its invocation parameters.

The `schemaComplete` attribute is defined on the `model` element and is used to indicate that the schemas constructed from the definition documents in the interchange model are complete, in the sense that the validity of the interchanged SML model is fully determined by these schemas. Formally, however, the `schemaComplete` attribute does not express any assertion that the schemas so constructed are in fact complete, or that using these schemas will not result in any errors indicating that some components are missing from the schemas. The only formal effect of `schemaComplete` attribute with a value of `true` or `1` is to specify precisely the schemas with which interchange model validation is to be performed.

The optional `ruleBindings` element is used to contain information that associates rule documents with the documents they apply to. See § 4.3 – Rule Bindings on page 10 for further details.

Every document in the appears as content of a `document` element in either the `definitions` or the `instances` element, depending on whether the document in question is a model definition or a model instance document. There can be at most one embedded document contained by a `document/data` element. Both `definitions` and `instances` are optional. So, for example, if there are no model definition documents being packaged, the `definitions` element must be omitted.

The first child of each `document` is typically a `docInfo` element that contains a `baseURI` element and a list of `alias` elements. The `baseURI` element can be used to specify a base URI for relative references in the document. Defining base URIs is specified in § 5.3.2 – Base URIs on page 17. The content of each `alias` element is a URI with no fragment component (i.e., one with no "#" in it). Each of the `alias` elements serves as a name that other documents can use to refer to this document. Examples of how aliases are used to handle URI references are given in § 4.2 – URI References on page 7.

A document in the interchange model can be represented in either of two ways, by embedding its content, or by providing a reference to it. Which is being used is indicated by the child of the `document` element. A document can be embedded as-is or in a base64 encoded format. In the former case, a `data` element is used to contain the actual content of the document whereas a `base64Data` element is used for the latter. The base64 format is typically used for, but is not restricted to, documents with DTD. If the document is being referenced rather than embedded, a `locator` element is used to contain the reference. The content of a `locator` can be a `documentURI` element defined by SML-IF or anything else understood by the .

Although it is not fully shown in the pseudo-schema above, the SML-IF schema has an "open content model." To provide extensibility, essentially every element in it can contain additional content and/or attributes from other XML namespaces.

4.2. URI References

When processing the SML model packaged inside an SML-IF document, certain URI references (as defined in RFC 3986 [[IETF RFC 3986]]) may need to be processed to find their corresponding target. For example, in order to assess SML validity of the interchanged model, SML references using the SML URI Reference Scheme [[SML 1.1]] need to be resolved. In addition, in order to assemble schemas from multiple schema documents as part of the interchange model validity assessment, the schemaLocation attribute on an `xs:include` element needs to be processed to locate the schema document.

To see how these URI references are handled, consider the following SML-IF document:

```
<?xml version="1.0" encoding="UTF-8"?>
<model xmlns="http://www.w3.org/ns/sml-if" version="1.0">
    <identity>

<name>http://www.university.example.org/sml/models/Sample/InterDocReferences</name>

        <baseURI>http://www.university.example.org/Universities/</baseURI>
    </identity>
    <definitions>
        <document>
            <data>
                <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
                    <xs:include
schemaLocation="http://www.university.example.org/university/enrollmodel.xsd"/>
                </xs:schema>
            </data>
        </document>
    </definitions>

    <instances>
        <document>
            <data>
                <Student xmlns="http://www.university.example.org/ns"
                         xmlns:sml="http://www.w3.org/2007/09/sml"
```

```
                        xmlns:u="http://www.university.example.org/ns">
                    <ID>1000</ID>
                    <Name>John Doe</Name>
                    <EnrolledCourses>
                        <EnrolledCourse sml:ref="true">
                            <!-- SML Reference to a course INside the interchange
model -->
                            <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="true">
                            <!-- SML Reference to a course INside the interchange
model -->
                            <sml:uri>

http://www.university.example.org/Universities/SFU/Courses.xml#smlxpath1(/u:Courses/u:Course[u:Name='MUSIC205'])

                            </sml:uri>
                        </EnrolledCourse>
                        <EnrolledCourse sml:ref="true">
                            <!-- SML Reference to a course OUTside the interchange
model -->
                            <sml:uri>

http://www.university.example.org/Universities/Capella/Courses.xml#smlxpath1(/u:Courses/u:Course[u:Name='LIT103'])

                            </sml:uri>
                        </EnrolledCourse>
                    </EnrolledCourses>
                </Student>
            </data>
        </document>
        <document>
            <!-- The following alias matches the first course referenced above -->
            <docInfo>
                <aliases>

<alias>http://www.university.example.org/Universities/MIT/Courses.xml</alias>
                </aliases>
            </docInfo>
            <data>
                <Courses xmlns="http://www.university.example.org/ns">
```

```
                    <Course>
                        <Name>PHY101</Name>
                    </Course>
                    <Course>
                        <Name>MAT200</Name>
                    </Course>
                </Courses>
            </data>
        </document>
        <document>
            <docInfo>
                <baseURI>SFU/Courses.xml</baseURI>
             <!-- The following alias matches the second course referenced above
(after
             being converted to an absolute URI)  -->
                </aliases>
                    <alias>SFU/Courses.xml</alias>
                </aliases>
            </docInfo>
            <data>
                <Courses xmlns="http://www.university.example.org/ns">
                    <Course>
                        <Name>ENG106</Name>
                    </Course>
                    <Course>
                        <Name>MUSIC205</Name>
                    </Course>
                </Courses>
            </data>
        </document>
    </instances>
</model>
```

When not packaged in an SML-IF document, certain URI references (e.g. values of `sml:uri` elements or certain `schemaLocation` attributes) are dereferenced to find their corresponding document. When these references are packaged in an SML-IF document, consumers of the SML-IF document need to first examine whether the target document or element is packaged in the same SML-IF document. To determine this, the fragment component, if any, is temporarily ignored to form a URI. This URI is then compared against the `alias` URIs of packaged model documents.

If the URI is equal to the URI in an `alias` element (see § 5.3.1 – URI equality on page 16), the will not attempt to look for targets of this URI outside of the SML-IF document, although there may exist a document retrievable at this URI. If the URI is not equal to the URI in any `alias` element, then the SML-IF document does not contain the corresponding target of the original URI reference. The consumer may or may not attempt to look for targets outside of the SML-IF document, depending on the nature of the URI reference. Formal rules about how URI references are processed are defined in section § 5.3.4 – URI Reference Processing on page 19.

Several examples of resolving references can be seen in the example SML-IF document shown above, illustrating the use of both relative and absolute alias URI values. In the first example, a reference with an absolute URI, the following SML reference, must first be separated into its document URI and fragment components:

```
http://www.university.example.org/Universities/MIT/Courses.xml#smlxpath1(/u:Courses/u:Course[u:Name='PHY101'])
```

After removing the fragment, the document portion of the reference is:

```
http://www.university.example.org/Universities/MIT/Courses.xml
```

This document URI is equal to the URI listed in an `alias` accompanying the `Courses` document. So, by applying the fragment in the URI reference to the `Courses` document, we determine that the reference is to the `Course` element whose `Name` element has `"PHY101"` as its content.

The second example reference, using a relative URI, is processed similarly. The full reference is:

```
http://www.university.example.org/Universities/SFU/Courses.xml#smlxpath1(/u:Courses/u:Course[u:Name='MUSIC205'])
```

After removing the fragment, the document portion of the reference is:

```
http://www.university.example.org/Universities/SFU/Courses.xml
```

This URI is equal to an `alias` defined on the last instance document in the interchange model, after the `model/identity/baseURI` content is applied to the relative URI contained by the document's `alias` element. So, by applying the fragment in the reference to the `Courses` document, we determine that the reference is to the `Course` element whose `Name` element has `"MUSIC205"` as its content.

The third example, showing an unresolved reference, is processed similarly. The full reference is:

```
http://www.university.example.org/Universities/Capella/Courses.xml#smlxpath1(/u:Courses/u:Course[u:Name='LIT103'])
```

After removing the fragment, the document portion of the reference is:

```
http://www.university.example.org/Universities/Capella/Courses.xml
```

This document URI is not equal to the URI in any `alias` element. This means that it is an unresolved SML reference.

The URI:

```
http://www.university.example.org/university/enrollmodel.xsd
```

(value of the `schemaLocation` attribute on the `include` element) is not equal to any `alias`. The may or may not attempt to locate a schema document using this URI reference.

4.3. Rule Bindings

[[SML 1.1]] uses Schematron patterns embedded in SML schemas and in separate explicitly bound rule documents to express constraints that cannot be expressed in XML Schemas. Schematron patterns embedded in SML Schema documents all have well defined targets. [[SML 1.1]] permits models in which rule documents apply to all, none, or subsets of the model's documents. SML-IF uses the list of `ruleBinding` elements contained in the optional `ruleBindings` element to associate rule documents with the documents in the interchange model to which they apply. Each `ruleBinding` associates the documents having an alias beginning with the URI prefix given in the `documentAlias` with the rule

documents having an alias beginning with the prefix given in the `ruleAlias`. So, for example, the `ruleBinding`:

```
<ruleBinding>
    <documentAlias="http://www.university.example.org/sml/infrastructure/"/>
    <ruleAlias="http://www.university.example.org/sml/infrastructurerules/"/>
</ruleBinding>
```

Would associate documents that have the aliases such as:

`http://www.university.example.org/sml/infrastructure/server427.xml`

and

`http://www.university.example.org/sml/infrastructure/switch6E.xml`

with rule documents that have aliases such as:

`http://www.university.example.org/sml/infrastructurerules/assetistracked.sch`

and

`http://www.university.example.org/sml/infrastructurerules/managedbycorporate.sch`

SML-IF specifies rule bindings among documents in the interchange model. It does not specify rule bindings that apply to documents not in the interchange model. That said, it is often the case that the intent of transferring an SML-IF document is to relate its contents with other SML documents not in the interchange model. For example, the intent might be to merge the interchange model with an existing SML model. In such cases the context of use may choose to extend the definition of `ruleBinding` to bind documents not in the interchange model. For example, if the interchange model is merged into an existing model, the merge process might choose to extend the definition of `ruleBinding` elements to bind rule documents in the interchange model to documents in the merged model that weren't included in the interchange model.

4.4. Schema Bindings

Schema documents can be connected with other schema documents using composition features provided by XML Schema. This includes `xs:include`, `xs:redefine`, and `xs:import`. A schema document's validity may depend on other schema documents it includes/redefines/imports, or even other schema documents that include/redefine/import it. When performing over the SML model packaged in an SML-IF instance, an SML-IF consumer must draw associations between XML Schema definition documents and instance documents, both to completely validate XML Schema documents themselves and to establish the schema-validity of the instance documents.

The XML Schema specification provides more flexibility in constructing the schema used for assessment than is appropriate for the semantics defined by SML and SML-IF for

1. It allows XML Schema processors latitude in terms of locating schema documents (resolving namespace and schema location attributes) and composing schema documents together to form a single schema.
2. Schema location attributes can be ignored in some cases (`xsi:schemaLocation` in instance documents and `schemaLocation` attribute on `xs:import`) and allowed to "fail to resolve" in others (`schemaLocation` attribute on `xs:include` and `xs:import`).
3. Multiple imports of the same namespace allow all but the first one to be ignored.

As a result, SML-IF cannot guarantee general case interoperability based only on XML Schema and, therefore, needs to specify how to determine such associations. This section describes a method to achieve this goal.

An SML-IF document can be:

1. Schema-complete - All schema documents are included in the SML-IF document, either as an § 5.2.1 – Embedded Documents on page 15 or as § 5.2.2 – Referenced Documents on page 16.
2. Schema-incomplete - Some required schema documents may not be included in the SML-IF document, either as an embedded document or a referenced document.

It is necessary for an SML-IF producer to declaratively distinguish between these two cases because making that distinction is not always possible for an based on the content alone. SML-IF uses the `schemaComplete` attribute on the `model` element to indicate whether this SML-IF document includes all necessary schema definition documents. When this attribute is specified with a value of "true", then the schema validity of the schema definition documents and instance documents depend only on built-in components or components from definition documents included in the SML-IF document. Built-in components include:

1. four xsi: attributes (defined by XML Schema)
2. all schema built-in types (xs:anyType and simple types defined in XML Schema Part 2)
3. sml:ref attribute declaration
4. sml:uri element declaration

An SML model represented by a schema-incomplete SML-IF document is not necessarily invalid. However, SML-IF cannot guarantee interoperability for a schema-incomplete SML-IF document.

SML-IF uses a list of `schemaBinding` elements contained in the optional `schemaBindings` element to associate a namespace with a set of schema documents in the interchange model and the instance documents that should be validated against this set of schema documents. Each `namespaceBinding` child of a `schemaBinding` element associates the namespace specified in its `namespace` attribute with the schema documents whose aliases are specified in its `aliases` attribute. In addition, the instance documents that are to be assessed against this set of schemas are specified in the `documentAlias` child element of the same `schemaBinding` element.

The following example illustrates schema bindings.

```
<schemaBindings>
    <!-- Each "schemaBinding" element corresponds to a schema and model
         instance documents that are assessed against this schema -->
    <schemaBinding>
        <!-- all "namespaceBinding" children together build the schema -->
        <namespaceBinding namespace="ns1" aliases="xsd1-a xsd1-b"/>
        <namespaceBinding namespace="ns2" aliases="xsd2-v1"/>
        <!-- list all applicable instances; same as for rule bindings -->
        <documentAlias>doc1</documentAlias>
        <documentAlias>doc2-v1-a</documentAlias>
        <documentAlias>doc2-v1-b</documentAlias>
    </schemaBinding>
    <schemaBinding>
```

```
        <namespaceBinding namespace="ns1" aliases="xsd1-a xsd1-b"/>
        <namespaceBinding namespace="ns2" aliases="xsd2-v2"/>
        <documentAlias>doc1</documentAlias>
        <documentAlias>doc2-v2</documentAlias>
    </schemaBinding>
</schemaBindings>
<definitions>
    <!-- schema documents for xsd1-a, xsd1-b, xsd2-v1, xsd2-v2 -->
</definitions>
```

There are cases where many instance documents use the same schema. In this case, it is desirable to have a default schema binding rather than specifying a `schemaBinding` that lists all these instance documents. The `defaultSchema` can be used to cover instance documents not included in any other`schemaBinding` as in the following example.

```
<schemaBindings>
    <!-- The "defaultSchema" element corresponds to a schema that governs
         all instance documents *not* included in any "schemaBinding". -->
    <defaultSchema>
        <!-- all "namespaceBinding" children together build the schema -->
        <namespaceBinding namespace="ns1" aliases="ns1.xsd"/>
        <namespaceBinding namespace="ns2" aliases="ns2.xsd"/>
    </defaultSchema>
</schemaBindings>
```

There may be cases where an instance document should not be bound to any schema, including the default schema. The `noSchemaBinding` element can be used in this case to cover such instance documents as in the following example.

```
<schemaBindings>
    <!-- The "noSchemaBinding" element contains the aliases for
         all instance documents *not* bound to any schema. -->
    <noSchemaBinding>
        <documentAlias>doc-a</documentAlias>
        <documentAlias>doc-b</documentAlias>
    </noSchemaBinding>
</schemaBindings>
```

4.5. Interoperability of SML Models

The goal of SML-IF is to enable the exchange of SML models. However, this interoperability goal is affected by several aspects of SML models.

1. Use of the SML URI Reference Scheme as defined in the SML specification is the only guaranteed way of achieving interoperability for all SML references in the model. Use of any other reference scheme requires that the know about its use in the document and understand how to dereference it.
2. SML documents can be included by reference using the `locator` element and, therefore, are not directly embedded in the SML-IF document. This can be very useful, especially when the SML-IF document is large or when the documents are readily accessible to the consumer. However, the

`locator` element may be ignored by the consumer, may not resolve, or may resolve to different resources in different contexts. Because of these uncertainties, interoperability is not guaranteed when documents are included by reference.

3. The SML-IF document may be schema-incomplete [§ 4.4 – Schema Bindings on page 11]. An SML model represented by a schema-incomplete SML-IF document is not necessarily invalid. However, SML-IF cannot guarantee interoperability for a schema-incomplete SML-IF document.
4. The SML-IF document may use reference schemes that do not use target-complete identifiers. In addition to the requirements imposed by SML on reference scheme definitions, SML-IF imposes additional requirements on references schemes that do not use target-complete identifiers in order to make them useful in the context of SML-IF [§ 5.3.4 – URI Reference Processing on page 19].
5. The presence of relative references subject to SML-IF URI processing introduces the necessity to transform them into absolute references [§ 5.3.4 – URI Reference Processing on page 19]. SML-IF provides two alternative mechanisms [§ 5.3.2 – Base URIs on page 17] for doing so, one of which is deprecated. SML-IF producers can construct SML-IF documents that use either only absolute URIs or both base URI mechanisms in order to achieve interoperability with the maximum number of consumers.

5. SML Interchange Format Definition

This section normatively defines the Service Modeling Language Interchange Format (SML-IF). It defines the requirements that SML-IF documents must adhere to and how URI references contained in them are to be interpreted by consumers of SML-IF documents.

5.1. Conformance Criteria

SML-IF defines two levels of conformance for SML-IF Documents:

1. Minimal Conformance: A *minimally conforming SML-IF Document* MUST adhere to all SML-IF document requirements as described in the normative sections of this specification.
2. Reference Conformance: A *referentially conforming SML-IF Document* MUST adhere to all SML-IF document requirements as described in the normative sections of this specification. In addition, each non-null SML reference in the document MUST be an instance of the SML URI Reference Scheme [[SML 1.1]].

A *conforming SML-IF Producer* MUST be able to generate a referentially conforming SML-IF Document from a conforming SML model.

When a producer generates a referentially conforming SML-IF document from a conforming source model, it is expected that the source model and the generated model are equivalent. That is, the source model and the destination model both have the same validity, same number of documents with similar structure and content differing only in places where references are updated to have equivalent SML URI scheme representation. However, this specification does not normatively define the notion of model equivalence.

A *conforming SML-IF Consumer* MUST process a conforming SML-IF Document using, in whole or part, semantics defined by this specification. It is OPTIONAL that a conforming SML-IF Consumer process all elements defined in this specification, but any element that is processed MUST be processed according

to the requirements stated in the normative sections of this specification. In particular, if a conforming SML-IF Consumer performs , then that process MUST be performed as described in this specification.

5.2. SML-IF Documents

The purpose of SML-IF is to package the set of documents that constitute an SML model into a standard format so that it can be exchanged in a standard way.

An SML-IF document MUST be a well-formed XML document [[XML]].

An SML-IF document MUST be valid under the XML Schema given in Appendix A.

The definition and instance documents packaged by an SML-IF document MAY form a valid SML model but it is not required to do so.

Each document in the interchange model MUST be represented in the SML-IF document by a separate `document` element as follows:

1. Each definition document in the interchange model MUST appear as a descendant of a `model/definitions/document` element. The order of the `document` children is not significant.
2. Each instance document in the interchange model MUST appear as a descendant of a `model/instances/document` element. The order of the `document` children is not significant.

Each document in the interchange model MUST be included in the SML-IF document either as an embedded document (where the document to be included is embedded in the SML-IF document) or by including a reference to the document.

5.2.1. Embedded Documents

Documents that are to be embedded in the SML-IF document MUST be embedded as text or in an encoded format as follows:

1. If the document is embedded as text, it MUST be included as the content of a `model/definitions/document/data` element if it is a definition document or a `model/instances/document/data` element if it is an instance document. Each `model/*/document/data` element MUST contain at most one document.
2. If the document is embedded in an encoded format, then the octet stream representing the document MUST be encoded in base64 format. The resultant data stream MUST be embedded as the content of a `model/definitions/document/base64Data` element if it is a definition document or a `model/instances/document/base64Data` element if it is an instance document. Each `model/*/document/base64Data` element MUST contain at most one document. Documents that contain a DTD MUST be embedded in this encoded format.

When extracting an embedded document that is contained in a `base64Data` element, an SML-IF consumer MUST decode the content of the `base64Data` element first and then process the resulting document as an embedded instance document. All embedded instance documents not encoded in base64 MUST be processed as if they contained the same DTD as the one associated with the SML-IF document.

If the `model/*/document/data` element has no child element, then an SML-IF consumer MUST treat the document as if it is not part of the interchange model. If the `model/*/document/base64Data` element has a zero-length sequence of octets as its value, then an SML-IF consumer MUST treat the document as if it is not part of the interchange model.

5.2.2. Referenced Documents

Documents that are to be referenced rather than embedded MUST be included as follows:

1. If the document is a definition document, the location of the document MUST be included as the content of a `model/definitions/document/locator` element.
2. If the document is an instance document, the location of the document MUST be included as the content of a `model/instances/document/locator` element.

SML-IF specifies one way that MAY be used to provide the location of the referenced document, the `documentURI` element. It is a consequence of `documentURI` schema definition that it contains a URI reference, i.e., it may be an absolute URI or relative reference. When it is a relative reference, the [base URI] property SHOULD be used to transform it to an absolute URI, as stated in [§ 5.3.2 – Base URIs on page 17].

An SML-IF consumer MAY choose to locate a referenced document. If an SML-IF consumer chooses not to locate a referenced document or if it attempts to locate the referenced document and this attempt fails, then the SML-IF consumer MUST treat the referenced document as if it is not part of the . If either of these conditions occurs, the SML-IF consumer SHOULD make its invoker aware of this condition.

5.2.3. Schema Completeness

The `smlif:schemaComplete` attribute is defined on the `model` element. The attribute indicates whether or not all the definition documents required for are included in the .

If `schemaComplete` has the value `true` or `1`, then schemas used for interchange model validation MUST contain only schema components declared in built-in components or in model definition documents within the interchange model. If `schemaComplete` has the value `false` or `0`, then this specification does not constrain whether or not definition documents required for interchange model validation are retrieved from outside the interchange model.

5.2.4. SML-IF Document Version

An SML-IF producer MAY specify the version of the SML-IF specification with which conformance is declared by including the version number of the relevant specification as the value of the `SMLIFVersion` attribute in the document's `model` element. This value MUST be `"1.1"` for documents declared by the producer to conform to the SML-IF 1.1 specification. SML-IF Consumers MUST attempt to process an SML-IF document regardless of the value of the `SMLIFVersion` attribute. That is, an SML-IF Consumer MUST NOT reject the document solely because of the value of the `SMLIFVersion` attribute.

Requiring to continue processing in the face of unknown version values makes it easier to deploy documents that support future versions of this specification.

5.3. URI References

5.3.1. URI equality

SML-IF uses URI equality extensively to handle references among documents in the interchange model. To determine whether two URIs are equal, MUST perform case sensitive codepoint-by-codepoint comparison of the corresponding characters in the URI references.

5.3.2. Base URIs

If a document in the interchange model contains a relative reference subject to SML-IF URI processing (see § 5.3.4 – URI Reference Processing on page 19), then the base URI used to transform the relative URI reference into an absolute URI is the value of its [base URI] property according to the rules in section 4.3 of [XML Base]. When a base URI is needed to transform a relative reference, then the information necessary to calculate the [base URI] property MUST be embedded within the SML-IF document's content using at least one of the following mechanisms.

1. The base URI is embedded using the `xml:base` attribute according to [XML Base]. The value of an element's [base URI] property is calculated according to [XML Base].
2. The base URI is embedded using the `smlif:baseURI` element as described in § 5.3.2.1 – smlif:baseURI on page 18. The value of an element's [base URI] property is calculated as described in that section.

Because this specification requires that the base URI information be embedded in the document content, it follows that an element's [base URI] will never be computed from the URI of the document entity or external entity (see section 4.2 of [XML Base]) containing the element.

MUST support at least one of these mechanisms. The selection of which base URI mechanism(s) a consumer's implementation supports is implementation-defined, i.e. it might be done as a fixed coding choice, as a run-time parameter, by scanning the content, or through any other means. MUST support `xml:base` and MAY support `smlif:baseURI`.

If an SML-IF consumer supports both mechanisms and the interchange model document it is consuming contains markup for both mechanisms, then the SML-IF consumer MUST use the [base URI] value calculated using the `xml:base` mechanism.

All of the base URI mechanisms used in each interchange model document MUST be used consistently. In other words, all of the base URI mechanisms whose markup appears in the document MUST result in the s ame [base URI] value being calculated for each relative reference subject to SML-IF URI processing. SML-IF consumers MAY check this consistency.

As a consequence of the granularity of the consistency requirement, a single SML-IF document may use different mechanisms in distinct interchange model documents. In this scenario, it is true that only consumers that support all mechanisms used would be able to process the entire SML-IF document correctly.

Consistency checking of base URI results by SML-IF consumers is made optional to avoid requiring the potential overhead of performing twice as many calculations per relative reference as is minimally required to consume the model. An SML-IF consumer might choose to check base URI mechanism consistency based on input parameters, always, never, or based on any other criteria it chooses. If both base URI mechanisms are used in a given interchange model document contained within a conforming SML-IF document, and a consumer understands both mechanisms, such a consumer must use the xml:base mechanism to compute the [base URI] property. This consumer may choose to ignore the smlif:baseURI information or it may choose to verify that consistent results are obtained from both mechanisms. If both base URI mechanisms are used in a given interchange model document contained within a non-conforming SML-IF document, SML-IF provides no guarantees about the consistency of any [base URI] property computed using both mechanisms.

SML-IF producers have several combinations to consider when defining base URIs in an SML-IF document:

1. When the interchange model contains no relative URI references subject to SML-IF URI processing, neither `xml:base` nor `smlif:baseURI` is necessary.

2. When relative URI references subject to SML-IF URI processing exist in the interchange model and all require the same base URI value, providing an `xml:base` or `smlif:baseURI` value for the model element is sufficient.
3. When relative URI references subject to SML-IF URI processing exist in the interchange model and they require different base URI values, a combination of `xml:base` values or a combination of `smlif:baseURI` values can be used to ensure each document has the correct base URI.
4. When relative URI references subject to SML-IF URI processing exist within the same SML model document and they require different base URI values, `xml:base` can be used within the document to ensure that each relative URI has the correct base URI.

5.3.2.1. smlif:baseURI

This syntax is supported in this version of the SML-IF specification for compatibility with existing SML-IF documents. It is, however, deprecated and may be removed in a future version of this specification.

In the `smlif:baseURI` mechanism, two base URI values values are used to compute the value of an element's [base URI] property, which is then used to resolve relative URI references defined in the interchange model that are subject to SML-IF URI processing (see § 5.3.4 – URI Reference Processing on page 19).

Interchange model base URI

A URI reference that complies with the "absolute-URI" production as defined in RFC 3986 ([[IETF RFC 3986]]). The value of the *interchange model base URI* is the content of the `/model/identity/baseURI` element, if any.

This is roughly equivalent to specifying the same value in an `xml:base` attribute on the /model element.

Document base URI

A URI reference that complies with the "absolute-URI" production as defined in RFC 3986 ([[IETF RFC 3986]]). Each document in the interchange model has a document base URI whose value is a computed value.

For each document in the interchange model, the value of the is computed as follows:

1. If the document has a `docInfo/baseURI` element, let *U* be its value.
   A. If *U* is a relative reference, let *B* be the value of the . Then the value of the document base URI is the result of transforming *U* into an absolute URI, using *B* as the base URI.
   B. Otherwise the value of the document base URI is *U*.
2. Otherwise if the has a value, then the value of the document base URI is the value of the interchange model base URI.
3. Otherwise, the document base URI has no value.

According to the `smlif:baseURI` mechanism, the [base URI] property of an element is calculated as follows:

1. If the element is part of a document in the interchange model (i.e. it has as one of its ancestor elements smlif:locator, smlif:data, smlif:base64Data), its [base URI] value is the .
2. Otherwise, its [base URI] value is the .

5.3.3. Document Aliases

For each document in the , the `document` element contains a set of zero or more `alias` elements that are used to define the of the document.

Conceptually, each document in the interchange model has the following property:

[aliases]

A set of URI references that comply with the “absolute-URI” production as defined in RFC 3986 ([[IETF RFC 3986]]).

The value of the content of is computed as follows:

1. For each `alias` child element under the model document’s `docInfo/aliases`, there is a corresponding member in the . Let *U* be the value of such child element:
   A. If *U* is a relative reference, let *B* be value of the [base URI] property of the containing `alias` element, then contains the result of transforming *U* into an absolute URI, using *B* as the base URI, as defined in section 5 of RFC 3986 ([[IETF RFC 3986]]).
   B. Otherwise contains *U*.

Aliases MUST be unique. That is, there MUST NOT exist two model documents whose properties overlap.

As a consequence of the above property definition’s reliance on the “absolute-URI” production, the `alias` elements MUST NOT contain fragment components.

5.3.4. URI Reference Processing

When processing an SML-IF document, there are 3 categories of URI references that may need to be resolved:

1. `schemaLocation` attributes on `xs:include` and `xs:redefine` in schema documents, when they are model definition documents.
2. URI references specified in instances of SML reference schemes that use target-complete identifiers [[SML 1.1]].
3. URI references specified in instances of SML reference schemes that do not use target-complete identifiers.

It is clear which references fall into category #1. An example of category #2 is URI references used in SML references that use the SML URI Reference Scheme. When new reference schemes that use URI references are defined, whether they fall into category #2 or #3 will be clear from the reference scheme definitions. Resolution of URI references in category #3 is defined in their respective scheme definitions. It is also possible to have reference schemes that do not use URI references. Their resolution is governed by their scheme definitions and is not covered by this section.

To process a URI reference *UR* that is within categories #1 or #2 above, the following steps are performed:

1. Determine the document *D* that possibly contains the target:
   A. If *UR* is a same-document reference [[IETF RFC 3986]], then *D* is the model document that contains *UR*.
   B. Otherwise

i. If *UR* has a fragment component, then let *UR'* be the URI reference formed by removing the fragment component; otherwise let *UR'* be *UR*.

ii. If *UR'* is a relative reference, then transform *UR'* to form an (absolute) URI *U*, using its [base URI] as the base URI, as defined in section 5 of RFC 3986 ([[IETF RFC 3986]]); otherwise let *U* be *UR'*.

iii. If there exists a model document with an alias URI that is equal to *U* (§ 5.3.1 – URI equality on page 16), then *D* is that document; otherwise *D* has no value.

2. If *D* has no value, then

   A. If *UR* is within category #1 (`schemaLocation`), then the SML-IF document does not contain the target schema document. Whether the continues to dereference *UR* or *U* is governed by other sections of this specification.

   B. Otherwise (*UR* is within category #2, used in an SML reference), *UR* has no target.

3. If *D* has a value, then

   A. If *UR* is within category #1 (`schemaLocation`), then *UR* has a target if and only if all of the following are true.

      i. *D* is a schema document that is also a model definition document in the interchange model.

      ii. *UR* does not contain a fragment component.

   B. If *UR* is within category #2, then

      i. If *UR* does not contain a fragment component, then it targets the root element of D.

      ii. Otherwise (*UR* contains a fragment component), the fragment component of *UR* is applied to the root element of *D*, which may result in 0, 1, or many target elements.

To process a URI reference UR that is within category #3 above, a set of steps corresponding to those described above for categories #1 and #2 MUST be defined as part of the reference scheme definition.

5.4. Document Bindings

5.4.1. URI Prefix Matching

To associate SML rule or schema documents with the subset of documents in the model to which they apply, SML-IF uses a combination of the mechanism described above [§ 5.3.3 – Document Aliases on page 19] and URI prefix matching.

Two URIs, one called the *prefix*, and one called the *target* participate in URI prefix matching. The target is said to match the prefix if and only if the target, truncated to the length of the prefix, is equal to the prefix as defined in section § 5.3.1 – URI equality on page 16.

5.4.2. Rule Bindings

A rule binding is an association of a set of one or more rule documents 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 document in the model 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.

The `ruleBinding` element is used in SML-IF to express rule bindings. In any given binding the set of rule documents is that subset of rule documents in the interchange model with an alias that matches the URI prefix given by the content of the `ruleAlias` element. The set of model documents in the binding is that subset of the documents in the interchange model with an alias that matches the URI prefix given by the content of the `documentAlias` element. If the `documentAlias` element is omitted in a `ruleBinding`, the set of model documents in the binding is all documents in the interchange model.

Since the URI prefixes specified as `ruleAlias` and `documentAlias` elements are aliases, they are subject to all of the processing for aliases as described in [§ 5.3.3 – Document Aliases on page 19]. For example, if they are relative references then they would be transformed to absolute URIs before comparison.

SML-IF consumers MAY choose to extend the sets of documents involved in bindings to include documents not contained in the interchange model. For example, if an SML-IF document is used to represent a model fragment that is intended to be merged with some other model, it is entirely possible that some or all of the bindings may involve not just the documents in the interchange model, but documents in the other model.

5.4.3. Schema Bindings

SML-IF consumers MAY choose to ignore the `schemaBindings` element when present in the SML-IF document, in which case the consumer SHOULD make its invoker aware of this situation.

If an SML-IF consumer chooses to process the schemaBindings element, then, as part of the , for every schema binding SB in the model, i.e. every `/model/schemaBindings/schemaBinding` element, the SML-IF consumer MUST perform the following steps for instance document validation.

1. Compose a schema using all documents specified under all SB's `namespaceBinding` children.
2. Whenever an `import` for a namespace N is encountered, perform the following steps.
   A. If there is a `namespaceBinding` child of SB whose `namespace` attribute matches N, then components from schema documents listed in the corresponding `aliases` attribute are used. As with rule bindings, URI prefixing [§ 5.4.1 – URI Prefix Matching on page 20] is used for matching schema document aliases. At most one `namespaceBinding` is allowed per namespace N within a given SB. If more than one namespace binding exists for the namespace as part of a single schema binding, the SML-IF document is in error. If the set of aliases for namespace N is empty, the namespace has no schema documents defining it in the schema binding.
   B. Otherwise, if there are schema documents in the SML-IF document whose targetNamespace is N, then components from all those schema documents are used.
   C. Otherwise, if this is a schema-complete SML-IF document (`/model/@schemaComplete` = "true"), then no component from N (other than built-ins) is included in the schema being composed.
   D. Otherwise, it is implementation-defined whether SML-IF consumer attempts to retrieve components for N from outside the SML-IF document.
3. Whenever an `include` or `redefine` is encountered, the `schemaLocation` is used to match aliases of schema documents, as with base SML-IF.
   A. If there is a schema document in the SML-IF document matching that alias, then that document is used.
   B. Otherwise, if this is a schema-complete SML-IF document, then the `include` or `redefine` is unresolved (which is allowed by XML Schema validity assessment rules).

C. Otherwise, it is implementation-defined whether an SML-IF consumer attempts to resolve `include` or `redefine` to schema documents outside the SML-IF document.

4. The instance documents that are referenced in the `documentAlias` element of SB MUST be validated against the schema constructed in steps 1 through 3. `sml:target*` and SML identity constraints can now be checked. Similar to `documentAlias` under `ruleBinding` elements [§ 5.4.2 – Rule Bindings on page 20], each `documentAlias` can refer to multiple documents via URI prefixing.

Whether or not a `schemaBindings` element is present or is ignored, SML-IF consumers MUST process an `include` or `redefine` element as described in step 3 above.

The common use case where match-all namespace matching is desired can be achieved by omitting `schemaBindings` without introducing any additional complexity into the SML-IF document.

If an SML-IF consumer chooses to process the `schemaBindings` element, then the following rules regarding the default schema must be followed:

1. If the optional `defaultSchema` element is present, then an SML-IF consumer MUST compose a default schema from this element following rules 1 to 3 above, replacing SB in the text with DS (i.e., the `/model/schemaBindings/defaultSchema` element).
2. Otherwise, an SML-IF consumer MUST compose a default schema using all schema documents included in the SML-IF document.
3. An SML-IF consumer MUST use this default schema to validate those SML instance documents whose alias is not matched by any `documentAlias` in a `schemaBinding` element or `noSchemaBinding` element. Note that URI prefixing [§ 5.4.1 – URI Prefix Matching on page 20] is used for matching document aliases.

In all other cases, the SML-IF consumer MUST compose a schema using all schema documents included in the SML-IF document and MUST use this schema to validate all instance documents in the .

Examples of these cases include:

1. An SML-IF consumer chooses not to process the schemaBindings element.
2. No schema documents are found among the SML-IF document's definition documents.

The distinction between schema and schema documents is both intentional and important; the absence of schema documents does not imply the absence of a schema. A schema containing only built-in components will be constructed given zero schema documents as input, and this schema will be used to validate all instance documents in the interchange model. This distinction has an impact on model validation results according to the definition of validity for a conforming SML model [§ 5.1 – Conformance Criteria on page 14].

6. References

6.1. Normative

SML 1.1

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

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 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 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

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

XML Base

XML Base, Jonathan Marsh, Editor. World Wide Web Consortium, 27 June 2001. This version of the XML Base Recommendation is http://www.w3.org/TR/2001/REC-xmlbase-20010627/. The latest version of XML Base is available at http://www.w3.org/TR/xmlbase/. Available at `http://www.w3.org/TR/2001/REC-xmlbase-20010627/`.

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 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`.

6.2. Non-Normative

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`.

Canonical XML

Canonical XML, J. Boyer, Author. World Wide Web Consortium, 15 March 2001. This version of the Canonical XML Recommendation is http://www.w3.org/TR/2001/REC-xml-c14n-20010315. The latest version of Canonical XML is available at http://www.w3.org/TR/xml-c14n. Available at `http://www.w3.org/TR/2001/REC-xml-c14n-20010315`.

XML-Signature

XML-Signature Syntax and Processing, D. Eastlake, J. Reagle, and D. Solo, Editors. Internet Engineering Task Force & World Wide Web Consortium, 12 February 2002. This version of the XML-Signature Syntax and Processing Recommendation is http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/. The latest version of XML-Signature Syntax and Processing is available at http://www.w3.org/TR/xmldsig-core/. Available at `http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/`.

WS-Addressing Core

Web Services Addressing 1.0 - Core, M. Gudgin, M. Hadley, and T. Rogers, Editors. World Wide Web Consortium, 9 May 2006. This version of the WS-Addressing Core Recommendation is http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/. The latest version of WS-Addressing Core is available at http://www.w3.org/TR/ws-addr-core/. Available at `http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/`.

WSDL 2.0 Core Language

> *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, R. Chinnici, M. Gudgin, J-J. Moreau, S. Weerawarana, Editors. World Wide Web Consortium, 23 May 2007. This version of the "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" Specification is available is available at http://www.w3.org/TR/2007/PR-wsdl20-20070523/. The latest version of "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" is available at http://www.w3.org/TR/wsdl20/. Available at `http://www.w3.org/TR/2007/PR-wsdl20-20070523/`.

XLink

> *XML Linking Language (XLink) Version 1.0*, Steve DeRose, Eve Maler, David Orchard, Editors. World Wide Web Consortium, 27 June 2001. This version of the XLink Recommendation is http://www.w3.org/TR/2001/REC-xlink-20010627/ The latest version of XLink is available at http://www.w3.org/TR/xlink/. Available at `http://www.w3.org/TR/2001/REC-xlink-20010627/`.

Appendix A. SML-IF 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:smlif="http://www.w3.org/ns/sml-if"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.w3.org/ns/sml-if"
      elementFormDefault="qualified"
      blockDefault="#all"
      version="1.0"
      xml:lang="EN"
      finalDefault=""
      attributeFormDefault="unqualified">
   <xs:element name="model" type="smlif:modelType"/>
   <xs:complexType name="modelType" mixed="false">
      <xs:sequence>
         <xs:element name="identity" type="smlif:identityType"/>
         <xs:element ref="smlif:ruleBindings" minOccurs="0"/>
```

```
         <xs:element ref="smlif:schemaBindings" minOccurs="0"/>
         <xs:element
               name="definitions"
               type="smlif:documentCollectionType"
               minOccurs="0"/>
         <xs:element
               name="instances"
               type="smlif:documentCollectionType"
               minOccurs="0"/>
         <xs:any
               namespace="##other"
               processContents="lax"
               minOccurs="0"
               maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute
            name="SMLIFVersion"
            type="xs:token"
            use="optional"/>
      <xs:attribute
            name="schemaComplete"
            type="xs:boolean"
            default="false"/>
      <xs:anyAttribute
            namespace="##other"
            processContents="lax"/>
   </xs:complexType>
   <!-- If there is a need for localized string values, e.g. in displayName
         or description, the sml:locid global attribute can be
         used -->
   <xs:complexType name="identityType" mixed="false">
      <xs:sequence>
         <xs:element name="name" type="smlif:uriType"/>
         <xs:element
               name="version"
               type="smlif:tokenType"
               minOccurs="0"/>
         <xs:element
               name="displayName"
               type="smlif:displayType"
               minOccurs="0"/>
         <xs:element
               name="baseURI"
               type="smlif:uriType"
               minOccurs="0"/>
```

```
      <xs:element
            name="description"
            type="smlif:displayType"
            minOccurs="0"/>
      <xs:any
            namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute
         namespace="##other"
         processContents="lax"/>
</xs:complexType>
<xs:complexType name="displayType" mixed="false">
   <xs:simpleContent>
      <xs:extension base="xs:string">
         <xs:anyAttribute
               namespace="##other"
               processContents="lax"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
<xs:complexType name="tokenType" mixed="false">
   <xs:simpleContent>
      <xs:extension base="xs:token">
         <xs:anyAttribute
               namespace="##other"
               processContents="lax"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
<xs:complexType name="uriType" mixed="false">
   <xs:simpleContent>
      <xs:extension base="xs:anyURI">
         <xs:anyAttribute
               namespace="##other"
               processContents="lax"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
<xs:element
      name="ruleBindings"
      type="smlif:ruleBindingCollectionType"/>
<xs:complexType
```

```
        name="ruleBindingCollectionType"
        mixed="false">
     <xs:sequence>
        <xs:element
              ref="smlif:ruleBinding"
              maxOccurs="unbounded"/>
        <xs:any
              namespace="##other"
              processContents="lax"
              minOccurs="0"
              maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:anyAttribute
           namespace="##other"
           processContents="lax"/>
  </xs:complexType>
  <xs:element name="ruleBinding" type="smlif:ruleBindingType"/>
  <xs:complexType name="ruleBindingType" mixed="false">
     <xs:sequence>
        <xs:element
              name="documentAlias"
              type="smlif:uriType"
              minOccurs="0"/>
        <xs:element name="ruleAlias" type="smlif:uriType"/>
        <xs:any
              namespace="##other"
              processContents="lax"
              minOccurs="0"
              maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:anyAttribute
           namespace="##other"
           processContents="lax"/>
  </xs:complexType>
  <xs:element
        name="schemaBindings"
        type="smlif:schemaBindingCollectionType"/>
  <xs:complexType
        name="schemaBindingCollectionType"
        mixed="false">
     <xs:sequence>
        <xs:element ref="smlif:defaultSchema" minOccurs="0"/>
        <xs:element
              ref="smlif:schemaBinding"
              minOccurs="0"
```

```
            maxOccurs="unbounded"/>
      <xs:element
            ref="smlif:noSchemaBinding"
            minOccurs="0"
            maxOccurs="1"/>
      <xs:any
            namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute
         namespace="##other"
         processContents="lax"/>
</xs:complexType>
<xs:element
      name="schemaBinding"
      type="smlif:schemaBindingType"/>
<xs:complexType name="schemaBindingType" mixed="false">
   <xs:sequence>
      <xs:element
            ref="smlif:namespaceBinding"
            minOccurs="0"
            maxOccurs="unbounded"/>
      <xs:element
            name="documentAlias"
            type="smlif:uriType"
            minOccurs="0"
            maxOccurs="unbounded"/>
      <xs:any
            namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute
         namespace="##other"
         processContents="lax"/>
</xs:complexType>
<xs:element
      name="namespaceBinding"
      type="smlif:namespaceBindingType"/>
<!-- The value of the aliases attribute in the complexType below
    is a list of instance document URIs -->
<xs:complexType name="namespaceBindingType" mixed="false">
```

```
      <xs:attribute
            name="namespace"
            type="xs:anyURI"
            use="optional"/>
      <xs:attribute name="aliases" use="required">
         <xs:simpleType>
            <xs:list itemType="xs:anyURI"/>
         </xs:simpleType>
      </xs:attribute>
      <xs:anyAttribute
            namespace="##other"
            processContents="lax"/>
   </xs:complexType>
   <xs:element
         name="noSchemaBinding"
         type="smlif:noSchemaBindingType"/>
   <xs:complexType name="noSchemaBindingType" mixed="false">
      <xs:sequence>
         <xs:element
               name="documentAlias"
               type="smlif:uriType"
               minOccurs="0"
               maxOccurs="unbounded"/>
         <xs:any
               namespace="##other"
               processContents="lax"
               minOccurs="0"
               maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute
            namespace="##other"
            processContents="lax"/>
   </xs:complexType>
   <xs:element
         name="defaultSchema"
         type="smlif:defaultSchemaType"/>
   <xs:complexType name="defaultSchemaType" mixed="false">
      <xs:sequence>
         <xs:element
               ref="smlif:namespaceBinding"
               minOccurs="0"
               maxOccurs="unbounded"/>
         <xs:any
               namespace="##other"
               processContents="lax"
```

```
            minOccurs="0"
            maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute
         namespace="##other"
         processContents="lax"/>
</xs:complexType>
<xs:complexType name="documentCollectionType" mixed="false">
   <xs:sequence>
      <xs:element ref="smlif:document" maxOccurs="unbounded"/>
      <xs:any
            namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute
         namespace="##other"
         processContents="lax"/>
</xs:complexType>
<xs:element name="document" type="smlif:documentType"/>
<xs:complexType name="documentType" mixed="false">
   <xs:sequence>
      <xs:element ref="smlif:docinfo" minOccurs="0"/>
      <xs:choice>
         <xs:element name="data" type="smlif:dataType"/>
         <xs:element
               name="base64Data"
               type="smlif:base64DataType"/>
         <xs:element name="locator" type="smlif:locatorType"/>
      </xs:choice>
      <xs:any
            namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute
         namespace="##other"
         processContents="lax"/>
</xs:complexType>
<xs:element name="docinfo" type="smlif:docinfoType"/>
<xs:complexType name="docinfoType" mixed="false">
   <xs:sequence>
      <xs:element
```

```
               name="baseURI"
               type="smlif:uriType"
               minOccurs="0"/>
         <xs:element ref="smlif:aliases" minOccurs="0"/>
         <xs:any
               namespace="##other"
               processContents="lax"
               minOccurs="0"
               maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute
            namespace="##other"
            processContents="lax"/>
   </xs:complexType>
   <xs:element name="aliases" type="smlif:aliasCollectionType"/>
   <xs:complexType name="aliasCollectionType" mixed="false">
      <xs:sequence>
         <xs:element
               name="alias"
               type="smlif:uriType"
               minOccurs="0"
               maxOccurs="unbounded"/>
         <xs:any
               namespace="##other"
               processContents="lax"
               minOccurs="0"
               maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute
            namespace="##other"
            processContents="lax"/>
   </xs:complexType>
   <xs:complexType name="dataType" mixed="false">
      <xs:annotation>
         <xs:documentation>
               The wildcard with processContents "skip" matches the root element
of the
               model document being packaged. The value of processContents is set
to "skip" so
               that the contained element is not processed for schema validation.
As a result,
               validity of the packaged document will not affect validity of the
IF document
               itself.
            </xs:documentation>
```

```
      </xs:annotation>
      <xs:sequence>
         <xs:any
               processContents="skip"
               minOccurs="0"
               namespace="##any"
               maxOccurs="1"/>
      </xs:sequence>
      <xs:anyAttribute
            namespace="##other"
            processContents="lax"/>
   </xs:complexType>
   <xs:complexType name="base64DataType" mixed="false">
      <xs:simpleContent>
         <xs:extension base="xs:base64Binary">
            <xs:anyAttribute
                  namespace="##other"
                  processContents="lax"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="locatorType" mixed="false">
      <xs:sequence>
         <xs:element
               name="documentURI"
               type="smlif:uriType"
               minOccurs="0"/>
         <xs:any
               namespace="##other"
               processContents="lax"
               minOccurs="0"
               maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute
            namespace="##other"
            processContents="lax"/>
   </xs:complexType>
</xs:schema>
```

Appendix B. Localization of IF Identity Sample (Non-Normative)

The following example shows how the `sml:locid` attribute can be used to define the translation information for the interchange model identity attributes:

```
<model xmlns="http://www.w3.org/ns/sml-if" version="1.0"
       xmlns:sml="http://www.w3.org/ns/sml">
       xmlns:lang="http://www.university.example.org/translation/core">
    <identity>
        <name sml:locid="lang:nameID“>Univerity interchange model</name>
        <description sml:locid="lang:descrID"> This document contains a list of
universities.</description>
    </identity>
</model>
```

In this example, the [namespace name] URI information of the `sml:locid` attribute can be used to define the location for the resource containing the translated text. The `smlif:name` and `smlif:description` elements are using the same URI to identify the resource containing the translated strings:

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

The [local part] information of the `sml:locid` attribute can be used to define the id of the text being translated. This information will be used to locate the translation of the name and description texts within the translation resource.

Appendix C. 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.