This is an attempt to have a consistent story around SML references. (sml:ref="true"; not including sml:keyrefs.) This covers SML Bugzillas bugs 4658, 4673, 4682, 4683, 4780/4795, 4834, 4865, 4884, 4976.

Some of concepts and suggestions are already implemented in the first public working draft of the specifications. They are included here to make this proposal complete. Resolutions to Bugzilla bugs are marked with the corresponding bug number.

1 What's an SML reference?

An element information item in an SML model instance document is an SML reference if and only if it has an attribute information item whose {name} is "ref" and whose {namespace} is <sml namespace> and whose {normalized value}, after whitespace normalization using "collapse" following schema rules, is either "true" or "1".

Note: this doesn't depend on having a schema, and will not be affected if a schema is used to produce a PSVI.

Rationale: schemas may not always exist. People need to understand SML documents with SML semantics (just like people can process XML documents without a schema; perform XSLT without a schema.)

2 How are SML references handled?

If element R is a non-null SML reference, then R (for reference-handling purposes) is processed as a unit by SML reference processors (who know how to de-reference and/or assess the validity of an SML reference). Note that the above only applies to R in the context of reference-handling. Often, R may also have non-reference-scheme content and/or attributes (collectively: information items) that may be used in other contexts for other purposes where it may or may not be treated as a unit.

Each non-null SML reference matches the definition of zero or more Reference Schemes. Document authors decide which reference scheme(s) to use; in order to be useful, at least one reference scheme used should be understood by each processor of the document. Each processor decides which reference scheme(s) to recognize. If a single processor instantiation resolves SML references for multiple purposes, e.g. both for the purpose of assessing validity and for the purpose of de-referencing, the processor must recognize a set of reference schemes for the purpose of assessing validity that is a superset of the set of reference schemes for other purposes, even if distinct implementations are used. The definition of each reference scheme includes the rules by which an instance of the reference scheme is recognized by a processor given a non-null SML reference.

Since a single reference may match the definition of multiple reference schemes, any given processor may recognize a set of reference schemes that intersects unpredictably with the set of reference schemes used by document authors, and any single reference scheme recognized in a reference by a processor may or may not be resolvable according to the definition of the reference scheme, there are a number of cases to consider. For the purpose of assessing reference validity, a processor must check each reference for all reference schemes it recognizes and ensure that the SML Reference Semantics are satisfied for each reference scheme the processor recognizes – in particular, that all reference schemes recognized in a single reference resolve to the same target element. For the purpose of de-referencing an SML reference, finding at least one resolvable reference scheme may be sufficient in some other contexts (e.g. where the invoker of deref() as out of band knowledge that the reference-containing document has already passed SML validation). The SML-defined deref() Xpath function is one such context; a processor's deref() implementation might choose to examine reference schemes with the same thoroughness specified for assessing reference validity, or it might choose to use a weaker form. Both choices are equally acceptable.

The general form for handling non-null references is shown below. Note: this assumes that all attempts to resolve a reference will always return 0, 1 or many targets. That is, the attempt never fails (but resolution may fail) and all model documents are always reachable.

2.1 Resolving an SML reference to assess its validity

To assess the validity of an SML reference,

V1. If the processor implementation recognizes no scheme used in the reference, then R is unresolved.

V2. Else the processor implementation recognizes R as using N>0 schemes it supports, then (answers bug [4976]) it MUST attempt to resolve R using all N schemes, and

V2.1. If none of the recognized schemes resolves, then R is unresolved.

V2.2. If at least one of the recognized schemes resolves to more than one target element, then the model is invalid.

V2.3. If one scheme resolves to a target that's different from the target resolved by another scheme, then the model is invalid.

V2.4. If one scheme resolves and another doesn't, then the model is invalid.

V2.5. If none of the above is true (that is, all recognized schemes resolve to the same one and only one target element, call it T), then R is resolved to T.

2.2 Resolving an SML reference to dereference it

For deref() function implementation (which should not, and not allowed by XPath 1.0, to report errors),

D1. If the processor recognizes no scheme used in the reference, then deref() returns no target for R.

D2. Assume the processor recognizes R as using N schemes supported by the processor, then (answers bug [4683]) deref() is not required to attempt to resolve all N schemes. Its behavior in this case is implementation-defined, but it has to satisfy the following constraints:

D2.1. It returns 0 or 1 target element.

D2.2. If none of the recognized schemes resolves, then it returns no target for R.

D2.3. If it returns no target for R, then at least the condition of V2.1~V2.4 (for assessing the validity of an SML reference) is true for those schemes that are attempted by deref().

D2.4. If it returns a target element T, then T MUST be among the target(s) resolved by at least one recognized and attempted scheme.

In summary, deref() has to obey the "at most one target" rule, and it can't lie about the result (e.g. it can't return a target while having not attempted any scheme). The above rules allow a deref() implementation to behave in (and not limited to) any of the following ways or some combination of them:

- Only use the first scheme it recognizes, and ignore other schemes.
- Try the recognized schemes one by one until one resolves.
- If a scheme or multiple schemes resolve to more than one target, make R unresolved.
- If a scheme or multiple schemes resolve to more than one target, pick one of them as the target for R.

Note: implementation-defined means a processor MUST clearly *document* its deref() behavior.

Note: D2.3 above indicates deref() implementations are *not* required to attempt to resolve any recognized schemes.

Any attempt to resolve a reference is limited to the *current* model. That is, target MUST be within the current model.

So any given reference falls into one of 3 categories: null, resolved, and unresolved. (More about “null” in section 4”.) Suggest we delete the definition of “dangling reference” and use “unresolved reference” instead, because the former doesn’t carry more information than the latter. For unresolved references, it’s not always possible to tell whether it’s because the reference is wrongly specified or whether the target is outside of the model. (But “dangling” would imply the latter case.)

Rationale: the reference is a unit, because we should not constrain how schemes are defined. e.g. maybe my scheme uses multiple <my:uri> elements, and I specify "the reference resolves to the first <my:uri> that actually resolves". Then even though there could be many <my:uri> sub-elements, they are *not* multiple instances of the same scheme and don't constitute an error if more than one resolves.

There is a question about "when no scheme is recognized, should it be viewed as a null or unresolved reference?" The above gives the answer "unresolved", because this case is distinct from “this reference doesn’t have a value (null)”.

All schemes used by the same reference element are meant to target the same element, and SML processors assessing the validity of a reference are expected to report violation of this expectation. But for reasons like performance, deref() implementations may not want to try to resolve all schemes, hence the looser behavior.

2.3 Constraints on SML processors

If a single SML processor invocation assesses the validity of SML references and processes SML references for other purposes, e.g. it de-references them, then the set of reference schemes recognized during validity assessment MUST be a superset of those recognized during other processing. This ensures that any usage of an SML reference that an implementation might assume is valid, and therefore short-circuit certain processing, is actually assessed for validity. An example of allowable short-circuiting is described in section 2.2 (de-referencing an SML reference), versus that described in section 2.1 (assessing the validity of an SML reference).

3 How are schemes defined?

Because references are handled as units, we don't need to get into details in terms of how new schemes should be defined. In particular:

- Schemes may overlap. That is, the same sub-element/attribute can be used by multiple schemes.

- Schemes can use attributes. (More on this follows.)

The only questions a scheme definition needs to and MUST answer are: (Answers bug [4865].)

- How is a scheme recognized. That is, there is a set of rules that, when satisfied, identify elements with sml:ref="true" as instances of this scheme. (e.g. for the "uri/iri" scheme, the rule can include: there is one and only one sub-element named <sml:uri>, whose value is xs:anyURI.) That is, either a reference element is an instance of (or an example of, or uses) a scheme, or it is not. “How many instances of a reference scheme does an SML reference R use/include” is an ill-formed question.
- How a scheme is de-referenced. That is, if an element R is recognized as an instance of this scheme and is de-referenced using this scheme, an approach is defined to resolve R to a set of target element nodes. This MUST cover all the reference elements recognized as an instance of this scheme.

Because rules are specified in terms of the reference elements, we don't need to deal with "what if there are 2 instances of the same scheme in a single reference R". (Answers bug [4658].) For example, an

sml:uri reference scheme that allowed for multiple <sml:uri> child elements but failed to specify how those child elements are used (e.g. which one would actually be used for de-referencing or in what order they should be used) is not an acceptable scheme definition.

Definers of a reference scheme MAY define it in such a way that it intersects with another reference scheme, e.g. with partially overlapping information items. Definers of a reference scheme SHOULD NOT define it in such a way that it fully contains another reference scheme, e.g. where the information items of one are a superset of the other; SML processor implementations are likely to find it difficult to reliably determine which such scheme(s) the author intended to use and hence which it should recognize.

Rationale: again, we don't want to constrain how people want to define their schemes, especially for documents that pre-date SML. All we can require is that given an SML reference element R (with sml:ref="true"), a scheme must be defined in a way that a) processors can tell whether R uses such a scheme, and b) if so how to resolve R using the scheme to a set of nodes. (The set may contain more than 1 node, which will result in an error because SML references must resolve to a single node, but no one else cares about why there was more than 1.)

4 How are null references identified and handled?

“Null” references are distinct from unresolved references. A null reference is an explicit declaration of intent by the document author that the reference itself does not exist, and a processing directive (NOT a hint) to processors not to search the reference for reference scheme information items.

Propose to introduce another attribute, say, sml:nilref, with a boolean value (following the same rules as sml:ref). This attribute is only allowed when sml:ref="true"; if sml:nilref 's value is "true", then the reference is viewed as "null". This is the only case where a reference element denotes a “null” reference. (Answers bug [4884].) In the SML schema, define a global sml:nilref attribute with xs:boolean type.

When a reference element is recognized as “null”, then processors MUST NOT attempt to resolve it. Any scheme-related content (including attributes) recognized by the processor is ignored for reference-related purposes. The question of whether a null reference is resolved or not is undefined; it is an ill-formed question.

“targetRequired” should also apply to “null” references. That is, it’s an error if targetRequired=true is specified and the corresponding reference element R has sml:nilref=true. (Answers bug [4780/4795].)

Rationale: Null references are different from empty references (e.g. content of the reference element is intentionally empty; this is an explicit declaration of intent from the author). Empty references are just normal references (whose value happens to be empty).

Using the university example, if each course has a “primary teacher” and an “assistant teacher”, wouldn’t it be nice for the processor/application to know “this course doesn’t have an assistant teacher” rather than “I (the processor) don’t understand the way you (the author) specified the assistant teacher, and I don’t know who (s)he is”?

As for recognizing “null” references, xsi:nil doesn't work unless we constrain scheme definitions to not use attributes (i.e., to use element content only). Also when schema validation is used, xsi:nil disallows any sub-element or textual content; but there may be cases where a null reference still has non-reference-related content.

This also answers the question "do we support schemes involving attributes"? Because whether a reference is null or not is controlled by sml:nilref, it can cover attributes well (whereas xsi:nil can't). (Answers bug [4682].)

“Null” should not be a special case to “targetRequired”. If we view “nilref” as similar to “xsi:nil” in schema, then “targetRequired=true” is like the opposite of “nillable”.

5 What's an SML reference type in schema?

There are 5 classes of (complex) types with regard to how they interact with the sml:ref attribute.

1. Those who are derived from the current sml:refType.
2. Those who have an explicit reference to the global attribute declaration with use=required and fixed=true
3. Those who have an explicit reference to the global attribute declaration, with either true or false value
4. Those who allow sml:ref via an attribute wildcard or an explicit reference to the global attribute declaration, with either true or false value
5. Any complex type, including those who don't allow sml:ref with a "true" value

Similarly there are 5 classes of element declarations, which have the corresponding classes of complex types as their declared type definitions.

SML processors or applications may choose to do different things for these different classes (e.g. static analysis, GUI-based tooling, etc.).

There is no need to define "SML reference type". The only thing SML needs to define is, for which class(s) of types and element declarations are sml:acyclic and sml:targetXXX allowed to be specified.

Discussed during 2007-08 F2F. The WG decided to choose class #5.

(Similar to schema disallowing defining new xsi: attributes, see [1]) SML can disallow defining sml: attributes (by user schemas). (This removes the possibility for a schema to define a local sml:ref, which would confuse processors.)

[1] http://www.w3.org/TR/xmlschema-1/#no-xsi

Suggest to remove the special sml:refType (as it has no special meaning any more).

Rationale:

Class 1 makes it impossible to use existing schemas and make existing complex type reference types (because schema doesn't support multiple-inheritance). Classes 2~4 all make it hard for the scenario where there is a base (possibly abstract) type with SML constraints specified, and sml:ref attribute is only allowed by types derived from this base type.

6 How to handle non-references

When an element is not a reference element (i.e. doesn't have sml:ref=true), how should it be handled in reference-related cases? There are 2 scenarios:

1. targetXXX constraints

   Discussed during 2007-08 F2F; WG decided to make them trivially "satisfied" for non-references. That is, targetXXX constraints are enforced when and only when an element is an SML reference.

2. deref() (e.g. in identity constraints)

   Should those non-refs be passed to deref()? Or an error should be issue and deref() is not called?

   During the 2007-08 F2F, WG decided to answer "no error; deref() returns unresolved".

Rationale: to expand a little on item #1. Each element in the instance (whose corresponding element declaration has any one of the above constraints) can be in one of 4 categories:

- non-reference
- null reference
- unresolved reference
- resolved reference

And for the 4 constraints, we have the following 4 x 4 matrix applies:

Reference\Constraint	Acyclic	targetRequired	targetElement	targetType
Non-reference	Satisfied			
Null	Satisfied	Violated	Violated	Violated
Unresolved	Satisfied	Violated	Violated	Violated
Resolved	Check	Satisfied	Check	Check

There are 2 options for the "non-reference" row for the targetXXX constraints:

a) "violated", because targetXXX means "it must be a reference and its target must …"

b) "satisfied", because targetXXX means "if it is a refernce then its target must …".

WG picked answer (b), because for those who want #1 answer, they could have had use=required and fixed=true.

7 Which infoset is used when checking for SML references?

The following discussion only explicitly mentions recognizing SML references (using sml:ref). The same applies to null reference recognition (using sml:nilref).

If the sml:ref=true value cited in Section 1 comes from a Schema default value for sml:ref, it will only be in the schema processor's output infoset (PSVI), and not in the original document (the schema processor's input infoset). Then it's possible that in the input infoset there is no sml:ref, but in PSVI there is sml:ref=true. Should it be viewed as an SML reference?

During the 2007-08 F2F, some members were worried about potential inter-op problems between performing and not performing schema assessment; others are not as worried, because such problem already exists between processors who support different reference schemes. Some other members also see the value in viewing defaulted sml:ref=true as references. This makes it possible to transform existing XML data into SML models, without having to modify all the instances and add sml:ref attributes to them.

This issue was ***not*** resolved during the F2F.

Constraints:

Not all schema processors make their input infoset available to their consumers.

Not all schema processors expose PSVI sufficient to tell whether or not the sml:ref=true value came from a default supplied by schema versus other sources (fixed+required in schema, explicit on the instance, etc.)

Proposal:

Change SML and SML-IF in a coordinated fashion as described below.

SML changes :

Processors are allowed to use either infoset in SML for validity assessment. SML itself is insufficient for specification of complete interoperability, this decision preserves flexibility for dependent specifications to enable (or restrict, or neither) things as necessary for their interoperability requirements.

SML-IF changes :

There are 2 approaches we can take to ensure interoperability.

One possibility is to constrain the consumer. When assessing validity of the SML-IF instance, defaulted sml:ref=true attributes in the PSVI MUST NOT be used. This preserves the existing behavior that psvi is not required to find SML references in the interoperable subset, but if PSVI is used it requires the consumer's schema processor implementation to expose enough information in PSVI to tell whether the sml:ref=true was assigned by a default attribute value.

The other possibility is to constrain the producer. If SML-IF complete=true (reminder, this is not a change: complete=true is the case where we assert that SML-IF specifies sufficient constraints for interoperability), the producer is responsible for ensuring at least one of the following is true. Each of these conditions is sufficient to guarantee that XML schema will not contribute a value of sml:ref=true, so the schema processor's input infoset and its output infoset (PSVI) would be equivalent with respect to sml:ref=true. An SML processor assessing the validity of the SML-IF document MAY check these conditions, and if it discovers they are violated then it SHOULD report an invalid assessment output.

1. All SML references specify sml:ref=true explicitly in the model instance documents

   Note: this is much harder to check than #2; we may want to nuke it. Including it here for completeness

2. No model definition documents used for XML schema validation of the model specify a default or fixed value of sml:ref=true

If SML-IF:complete=false, then

1. psvi MAY be used to assess validity
2. An SML processor assessing the validity of the SML-IF document MAY check conditions 1 and 2 above, and if it discovers they are violated its output is implementation-defined.