Cycles in SML Models

Virginia Smith
James Lynn

1. Terminology

SML reference	An SML reference is a link from one element to another element within the same model (but not necessarily in the same document). An element information item in an SML 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". (This is Sandy's definition from http://lists.w3.org/Archives/Public/public-sml/2007Aug/att-0086/SML_References.html.
Reference source	The reference source is the element that contains the attribute "sml:ref" set to"true" or "1". (From SML 1.1, section 3.1)
Reference target	The reference target is the element referenced within the reference source using a reference scheme such as sml:uri. An SML reference can be an inter-document reference or an intra-document reference.
SML inter-document reference	An SML reference that is resolved to an information item from another document in the same SML model. (SML 1.1 actually states that the information item must be an element – section 3.1.2.4)
SML intra-document reference	An SML reference that is resolved to an information item (or element) from the same document that contains the SML reference.
Document-based cycle	Consider a directed graph whose nodes are the instance documents in an SML model that contain either an SML reference or the target information item for an SML reference and whose

edges are instances of the SML reference. The edge (arc) is directed from the document containing the reference to the document containing the target. A cycle results when a path can be traced along the edges that encounters a document node already encountered previously in the same path.

Element-based cycle Consider a directed graph whose nodes are the SML references in an SML model and the target information items to which the references resolve and whose edges are formed from the reference to its target. The edge (arc) is directed from the SML reference to the target. A cycle results when a path can be traced along the edges that encounters a node already encountered previously in the same path.

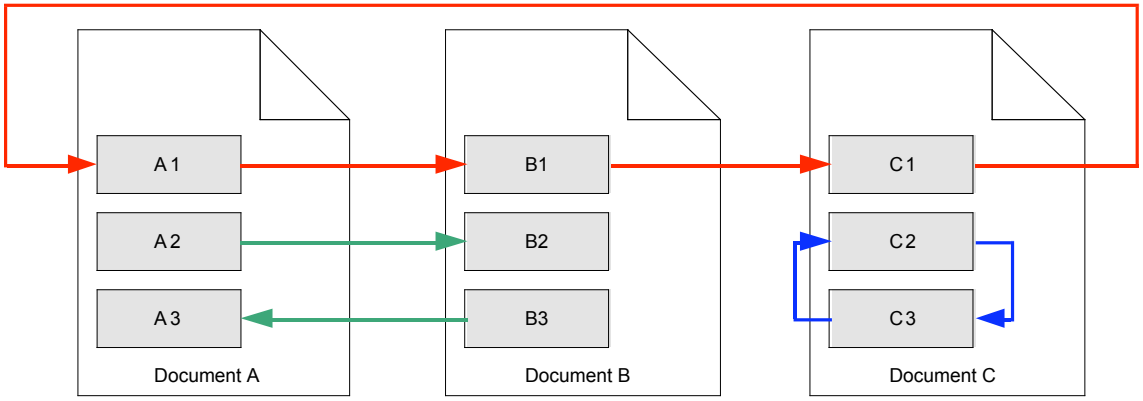
Acyclic An SML reference is defined to be acyclic when its definition forbids cycles that include itself (or its containing document) as a node (sml:acyclic="true").

2. Problem Discussion

In the current SML 1.1 draft specification, there is a mismatch between how SML references are defined and how cycles are defined. References are defined as a 'connection' between elements and cycles are defined based on 'connections' between documents. This section discusses several problems that arise from this mismatch.

The following diagram shows several scenarios for SML references and their targets. The table indicates whether each cycle in the diagram would be considered a valid cycle in an element-based graph or a document-based graph.

Figure 1 General cycle scenarios



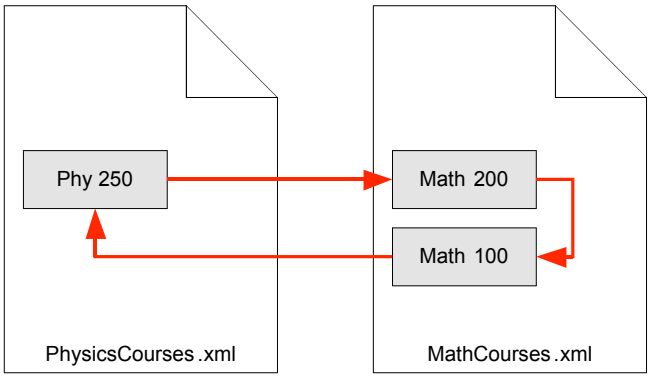
Cycle	Document-based cycle?	Element-based cycle?
Red Cycle (valid cycle)	✓	✓
Green Cycle (unrelated reference cycle)	✓ (false positive)	×
Blue Cycle (intra-document cycle)	× (false negative)	✓

2.1. Valid Cycles

The red cycle in Figure 1 illustrates the most obvious example of a cycle. This would be considered a cycle, whether its representative directed graph is element-based or document-based.

The diagram below shows an example of a valid cycle where the references refer to prerequisite courses. This example depicts an undesirable condition where a student cannot register for one of these courses because it is impossible to meet the prerequisite requirement.

Figure 2 Valid cycle example



The following SML fragments illustrate the use of reference nodes of type "PrerequisiteType" to create a cycle as illustrated in the above diagram:

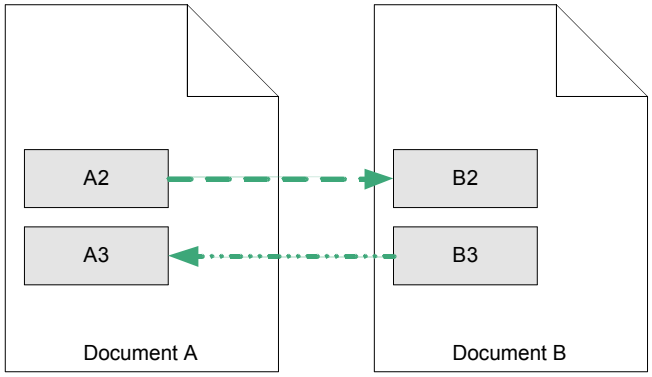
```
<Course>
  <Name>Math200</Name>
  <Prerequisite sml:ref="true">
    <sml:uri>
    http://www.university.example.org/Universities/MIT/MathCourses.xml#xmlns(u=http://
    www.university.example.org/ns) (/u:Courses/u:Course[u:Name='Math100'])
    </sml:uri>
  </Prerequisite>
</Course>
 . . .
<Course>
  <Name>Math100</Name>
  <Prerequisite sml:ref="true">
    <sml:uri>
    http://www.university.example.org/Universities/MIT/PhysicsCourses.xml#xmlns(u=http
    ://www.university.example.org/ns) (/u:Courses/u:Course[u:Name='Phy250'])
    </sml:uri>
  </Prerequisite>
</Course>
 . . .
<Course>
  <Name>Phy250</Name>
  <Prerequisite sml:ref="true">
    <sml:uri>
    http://www.university.example.org/Universities/MIT/MathCourses.xml#xmlns(u=http://
    www.university.example.org/ns) (/u:Courses/u:Course[u:Name='Math200'])
    </sml:uri>
  </Prerequisite>
</Course>
```

2.2. Unrelated References

The green cycle in Figure 1 illustrates the unrelated reference problem. Here the reference from A2 to B2 and the reference from B3 to A3 are completely unrelated. This is not a valid cycle (of element references) but will appear as a cycle in a document-based graph.

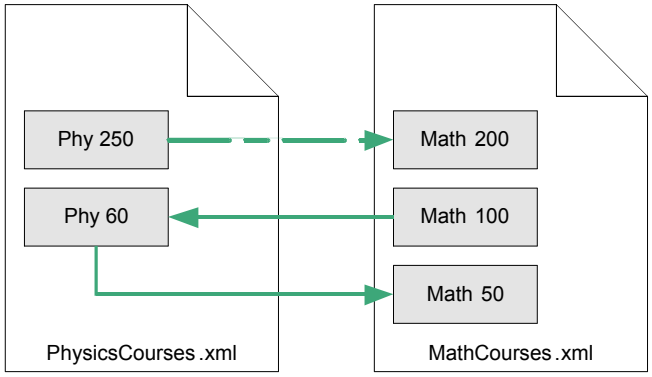
This false positive can be eliminated if the references are "typed", as in the following diagram, based on the elements (or element types) that they link. The graph node representing Document A will have 2 different edges emanating from it. Since the edges are of different types, no cycle is detected.

Figure 3 Unrelated reference cycle



The following diagram illustrates a condition where Math and Physics courses exchange prerequisites but, in fact, no cycle is created. This requires that the edges representing the links from the source to the target are somehow identified based on the elements linked rather than the document. In other words, the edges would be typed based on the elements linked.

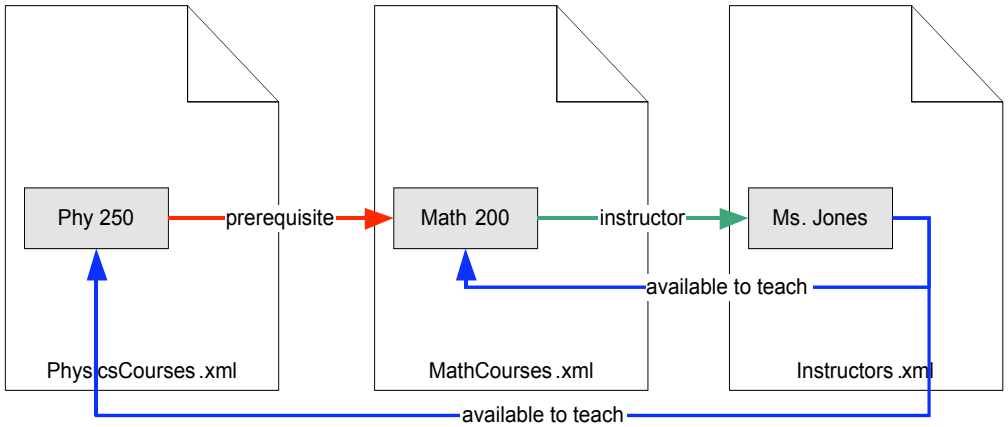
Figure 4 Unrelated reference cycle example #1



The question remains: how will we differentiate these links? In the example above, using the element's type (e.g., ClassType) will not be sufficient.

And, in case you don't have a headache yet, consider the following example of unrelated references that may look like a cycle but do not create a valid cycle. It is crucial that the "prerequisite", "instructor", and "available to teach" references be identified as separate edge types.

Figure 5 Unrelated references example #2

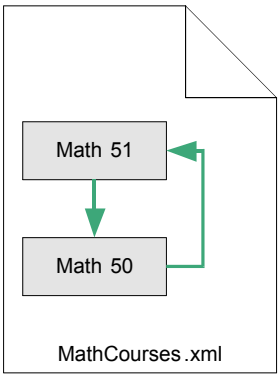


2.3. Intra-document References

The blue cycle in Figure 1 illustrates the intra-document reference problem. Here the reference is from C2 to C3 and from C3 back to C2. The elements C2 and C3 are contained in the same document. In this case, a cycle exists between references completely within a single document and does not appear as a cycle in a document-based graph.

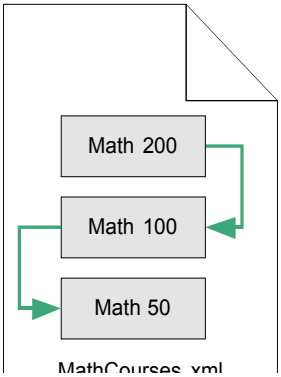
This scenario is illustrated in the following diagram where 2 Math courses have each other assigned as prerequisites. This anomaly will not be detected in a document-based graph where the edges only represent links between documents. This is a valid cycle and will be detected only if there is element-based cycle detection inside documents.

Figure 6 Intra-document references creating a valid cycle



The above scenario could be represented in a document-based graph by having an edge that points from the document node back to itself. However, we would still need to differentiate the above valid intra-document cycle from the scenario below where the edges would point back to the document node itself but is not actually a valid cycle. We cannot handle these scenarios correctly unless we use an element-based graph.

Figure 7 Intra-document references with no cycle



3. Requirements

3.1. Ability to detect SML reference cycles

Cycles must be detectable so that untenable situations such as described in section 2.1 can be flagged as error conditions.

?? what is a good scenario for allowing cycles?

4. Constraints

4.1. Reasonable implementation effort

If the implementation of an SML validator is not reasonably achievable, the adoption of SML will be negatively impacted.

Two areas in the implementation are potential problems:

1. Detecting cycles in general. Based on comments from HP and IBM implementation engineers, we do not believe this presents a challenge to the implementer.
2. Detecting cycles in a persistent data store. The concern was raised by Microsoft that their specific implementation of storing SML models in a database could make it difficult to detect cycles defined at the element level. One of the members of the WG (Kumar) is researching this.

5. Analysis

An element-based graph appears to be necessary to handle scenarios where the graph path includes intra-document references.

A document-based graph where the edges are "typed" by the reference source elements would be sufficient to identify valid cycles of inter-document references and correctly identify when unrelated references do not create a valid cycle. However, the question of how we type this edge still remains. This "edge typing" requires some knowledge of the reference elements themselves and the ability to distinguish between them. We may not be able to do that based on the complex type of the element. In Figure 4 above, there is no real cycle but the types of the elements will probably be the same so identifying the edge simply based on a complex type is not sufficient.

We may have to place a requirement on the SML model creator to create references between elements with careful consideration (modeler beware!). If cycles are based on the type of the reference which creates the edges in the graph, care must be taken to create the node types and use them in an unambiguous manner. At a minimum, if the reference nodes differ semantically they should differ in type.

6. Conclusion

While an SML validator can, in many cases, detect cycles using document-based graphs, knowledge of the reference elements themselves will be necessary to avoid false positives (as in the unrelated reference example) and to correctly handle intra-document references and cycles.

With this in mind, SML should define cycles as element-based. This would not preclude implementers from using document-based cycles (with typed edges) as an optimization when possible.

We could also consider placing explicit restrictions (or at least guidelines) on the use of SML references, such requiring a complex type definition for each semantically unique reference type and/or no intra-document references.

7. Acknowledgements

Yuan Chen (HP) contributed to this document.