PDF-XCHANGE Click to buy NOW! www.docu-track.com

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.

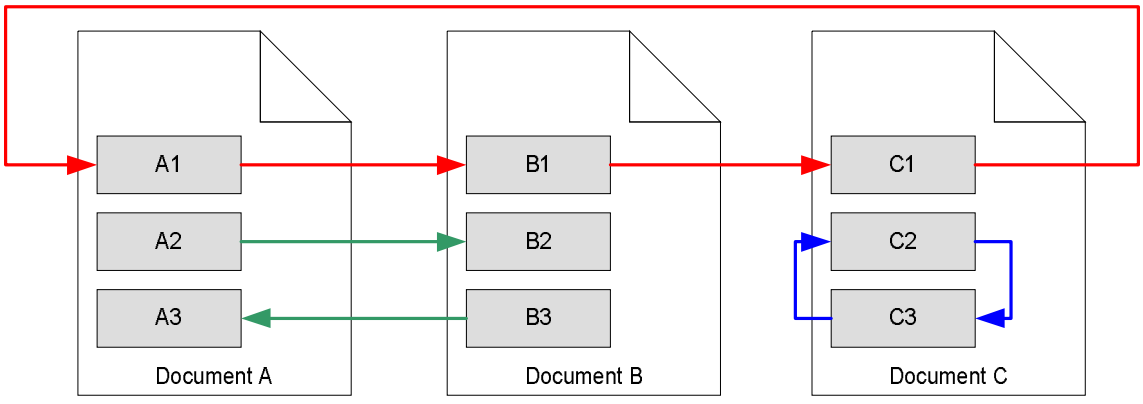
	An SML reference can be an inter-document reference or an intra-document reference.
SML inter-document reference	An SML reference that is resolved to an information item from another document in the same SML model. (SML 1.1 actually states that the information item must be an element – section 3.1.2.4)
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	An element-based cycle is formed for an element E if, when recursively following an SML reference that is a descendant of E to its target, the path leads back to E.

2. Element- vs. Document-Based Cycles

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)	ü	ü

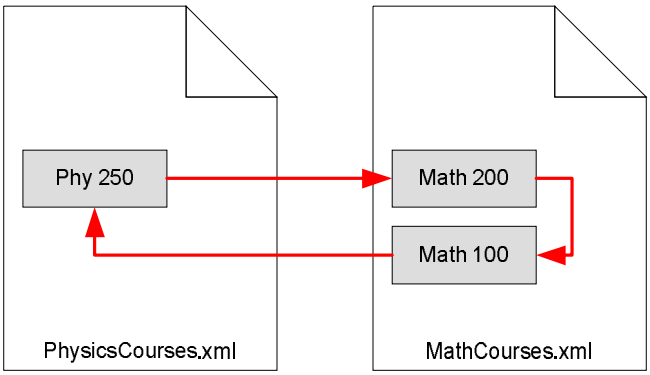
Green Cycle (unrelated reference cycle)	ü (false positive)	û
Blue Cycle (intra-document cycle)	û (false negative)	ü

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:

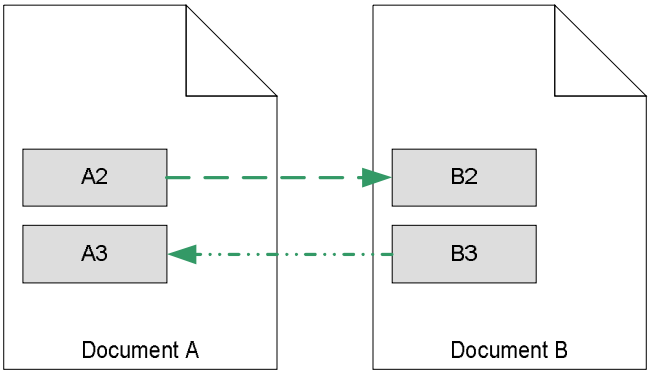
```
<Course>
  <Name>Math200</Name>
  <Prerequisite sml:ref="true">
    <sml:uri>
    http://www.university.example.org/Universities/MIT/MathCourses.xml#xmlns(u=http://
    www.university.example.org/ns) (/u:Courses/u:Course[u:Name='Math100'])
    </sml:uri>
  </Prerequisite>
</Course>
 . . .
<Course>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

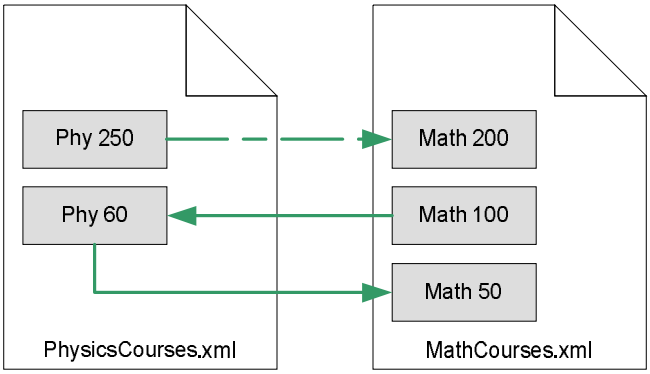
This false positive can be eliminated if the references are "typed", as in the following diagram, based on the elements (or element types) that they link. The graph node representing Document A will have 2 different edges emanating from it. Since the edges are of different types, no cycle is detected.

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.

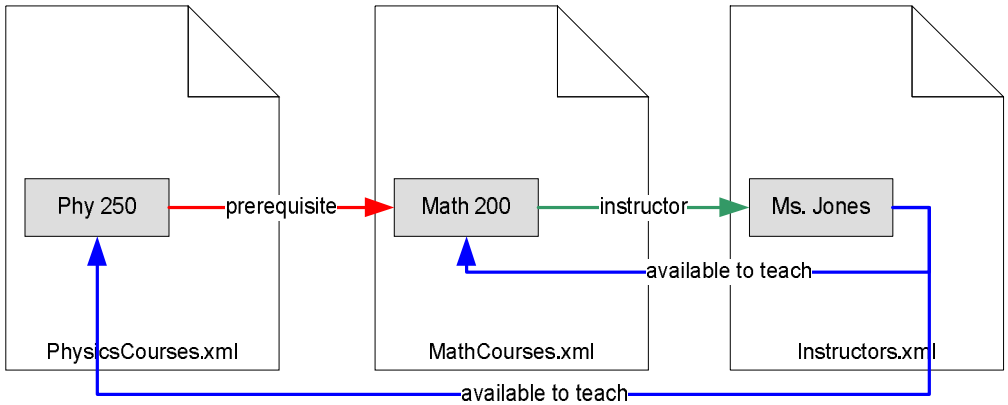
Figure 4 Unrelated reference cycle example #1



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

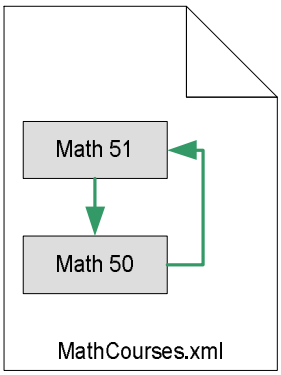


2.3. Intra-document References

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.

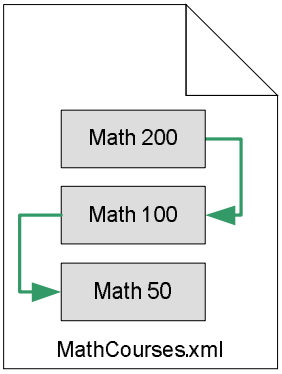
This scenario is illustrated in the following diagram where 2 Math courses have each other assigned as prerequisites. This anomaly will not be detected in a document-based graph where the edges only represent links between documents. This is a valid cycle and will be detected only if there is element-based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



2.4. Requirements

2.4.1. 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?

2.5. Constraints

2.5.1. 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.

2.6. 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.

2.7. 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.

3. Proposal

3.1. Cycle Definition

Cycles will be defined in SML based on elements. Update the SML specification, section 4.3.1, with this new definition.

> A cycle is formed for an element E if, when recursively following an SML reference that is a descendant of E to its target, the path leads back to E.

3.2. Syntax/Semantics

3.2.1. Proposal 1

Replace the text in section 4.3.1 with the following.

> Model validators that conform to this specification MUST support the sml:acyclic element on any <xs:complexType> element in a schema document as shown below.

```
<annotation>
    <appinfo>
        <sml:acyclic ref="restricted xpath expression"/>
    </appinfo>
</annotation>
```

Replace section 4.3.1.3 in the SML specification with the following text.

> For any element information item E of type P or a type derived from P, let R be the node that is referenced by the ref attribute of the sml:acyclic element in the annotation for E.
>
> 1. If R doesn't exist or R is not a reference element or R is a null reference, stop. No cycle is detected.
>
> 2. Otherwise deref() R to its target T.
>
> 2.1 If T is absent, OK.
>
> 2.2 If T is the same as E, then a cycle is detected and this is an error.
>
> 2.3 If T is of type P or a type derived from it, then let R the node that is referenced by the ref attribute of the sml:acyclic element in the annotation for T and repeat from step 1.
>
> 2.4 Otherwise, stop. No cycle is detected. (T is of a different type than E.)

Notes / Issues

- This proposal puts the acyclic restriction in the <annotation> of a complexType rather than as an attribute of the element. Multiple sml:acyclic elements can be attached to a type in the case where the type content contains multiple SML references.
- Cycles are homogeneous.That is, they contain only one node type and one reference link type.
- The complexType defines the cycle node. The sml:acyclic element defines the cycle arc.

3.2.2. Proposal 2

Replace the text in section 4.3.1 with the following.

```
Model validators that conform to this specification MUST support the
sml:acyclic attribute on any complexType as shown below.

<xs:attribute name="sml:ref" type="xs:boolean"/>
<xs:attribute name="sml:acyclic" type="<restricted xpath expression>"/>
```

Example schema fragment:

```
<xs:complexType name="dependencyRefType">
    <xs:sequence>
        <xs:any/>
    </xs:sequence>
    <xs:attribute name="sml:ref" type="xs:boolean" use="required"
                  fixed="true"/>
    <xs:attribute name="sml:acyclic" use="required"/>
</xs:complexType>
<xs:complexType name="nodeType">
    <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="DependsOn" type="dependencyRefType"/>
        <xs:element name="OtherRef">
             <xs:sequence>
                 <xs:any/>
             </xs:sequence>
             <xs:attribute name="sml:ref" type="xs:boolean"
                           use="required" fixed="true"/>
             <xs:attribute name="sml:acyclic" use="required"
                           fixed="./OtherRef"/>
             </xs:sequence>
         </xs:element>
</xs:complexType>
<xs:element name="Module" type="nodeType"></xs:element>
```

Example instance fragment:

```
<Module>
    <Name>App1</Name>
    <DependsOn sml:ref="true" sml:acyclic="./DependsOn">
        <sml:uri>URI</sml:uri>
    </DependsOn>
    <OtherRef sml:ref="true" sml:acyclic="./OtherRef>
        <sml:uri>URI</sml:uri>
    </OtherRef>
```

```
</Module>
```

Replace section 4.3.1.3 in the SML specification with the following text.

```
For any element information item E (e.g. "Module"), let R be an SML
   reference (e.g. "DependsOn") that is a child of E and contains an
   sml:acyclic attribute.

1. If R doesn't exist or R is not a reference element or R is a null
   reference, stop. No cycle is detected.

2. Otherwise deref() R to its target T.

2.1 If T is absent, OK.

2.2 If T is the same as E, then a cycle is detected and this is an
   error.

2.3 Otherwise, let R be the node that is referenced by the xpath
   expression (evaluated in the context of T) in the sml:acyclic
   attribute of R and repeat from step 1.
```

Notes / Issues

- A global relationship can be defined independent of its use in the model as in the example above.
- Instance document creator must know what the value of sml:acyclic should be (usually "./<same name>". This could be avoided by using a global or local element instead of a complexType to define the relationship link – see "OtherRef" in the example above. Should we require that sml:acyclic be supported on elements rather than complexTypes?
- Cycles can be defined by incorporating different elements or types. The next node in the cycle is found by following an SML reference. The next SML reference to follow is defined by the sml:acyclic attribute content (of the previous SML reference).
- Cycles can also be defined using different links (SML reference elements) since the next SML reference is discovered by evaluating the sml:acyclic xpath expression.
- "sml:ref" is required to indicate that that the element is a reference. "sml:acyclic" is only required when the reference cannot have cycles. Can these attributes be present in the instance document only (as is currently in the spec) or must they be part of the schema definition and therefore controlled by the schema document?

3.2.3. Proposal 3

Replace the text in section 4.3.1 with the following.

```
Model validators that conform to this specification MUST support the
sml:acyclic element on any <xs:complexType> element in a schema
document as shown below.

<annotation>
  <appinfo>
    <sml:acyclic name="<string>" ref="<restricted xpath expression>"/>
  </appinfo>
</annotation>
```

Example schema fragment:

```
<xs:complexType name="PersonType">
    <xs:annotation>
        <xs:appinfo>
```

```
                    <sml:acyclic name="x1" ref="./Child"/>
                </xs:appinfo>
            </xs:annotation>
            <xs:sequence>
                <xs:element name="Name"/>
                <xs:element name="Child" sml:ref="true"/>
                <xs:element name="Friend" sml:ref="true"/>
            </xs:sequence>
        </xs:complexType>
        <xs:element name="Person" type="PersonType"></xs:element>
```

Example instance fragment:

```
        <Person>
            <Name>Mary</Name>
            <Child sml:ref="true">
                <sml:uri>URI</sml:uri>
            </Child>
            <Friend sml:ref="true">
                <sml:uri>URI</sml:uri>
            </Friend>
        </Module>
```

Replace section 4.3.1.3 in the SML specification with the following text.

For any element information item E of type P or a type derived from P (e.g. "Person"), let R be the node that is referenced by the xapth expression (*evaluated in the context of E*) in the "ref" attribute of the sml:acyclic element in the annotation for E. Let C be the cycle name that is specified in the name attribute of same sml:acyclic element (e.g. "x1").

1. If R doesn't exist or R is not a reference element or R is a null reference, stop. No cycle is detected.
2. Otherwise deref() R to its target T.

2.1 If T is absent, OK.

2.2 If T is the same as E, then a cycle is detected and this is an error.

2.3 Otherwise, let R be the node that is referenced by the ref attribute of an sml:acyclic element in the annotation for T whose name attribute is equal to C. Then repeat from step 1.

Notes / Issues

- Cycles can be defined incorporating different elements/types using the cycle "name". The schema author must coordinate this cycle name with all necessary elements in the schema. An example of this is a dependency relationships among deployment CIs of different types.
- Cycles can be be defined using different links (SML reference elements) within the cycle since cycles are created by following the cycle "name". (But how often will this be used?)

3.3. Proposal Discussion

3.3.1. Inheritance

- In schema, annotations are not inherited.
- In SML, because we are defining new component properties (acyclic, target*, key/keyref/unique), we can define inheritance rules of these properties, which can be different from schema's rules about annotations.
- For acyclic, we don't need to inherit it, because it applies to instances of the current type and all types derived from it. (Implementations would need to be very smart to avoid checking the same constraint multiple times if we do inherit it.)

3.3.2. Comparison

2 allows for defining a relationship independent of the elements that participate in these relationships. This means the relationship can be reused. 1/3 define the acyclicity as an attribute of the elements (nodes) in the cycle.

Iin addition, since 2 defines the relationship on the SML reference itself rather than on the element containing the reference and so the acyclic-ness travels with the reference.

However, in 2, the sml:acyclic content repeats the name of the element which feels like a hack. It allows for completely different reference to be "next in line" but will that ever be a reasonable thing to do?

1/3 define acyclic-ness in the <annotation> element which seems like a nice thing to avoid if we can.

3.3.3. Schema Author Responsibilities **(old notes – I want to save them here)**

Since sml:ref applies to elements and is no longer controlled by the schema document, it is worth pointing out that there are still requirements on the schema author with regard to SML references and cycles. Specifically, the schema author should:

- Identify any model elements that could or should include content via an SML reference and specify that element as a complexType with an open content model (for both elements and attributes). An alternative is to define the complexType as specifically allowing for sml:ref and the chosen reference scheme(s) as content.
- Identify any possible cycles in the model that would be problematic (for example, prerequisite courses) and add the sml:acyclic element to the <annotation> for this type.

Any open content allowed in a complexType creates a potential for an element of that type to contain an SML reference. This should be OK since the required content for the element must still be present. This also means the required content cannot be "referenced out" to another document on the whim of the model instance creator.

4. Acknowledgements

Yuan Chen (HP) and Sandy Gao contributed to this document.