Data Relationships in the XML world

Yvonne Wilson 3/00

"An object by itself is intensely uninteresting." - Grady Booch, Object Oriented Design with Applications, 1991.

Throughout the technical evolution of technologies which store and manipulate data, a fundamental requirement has existed to render relationships between entities. In the object-oriented world, there are several types of relationships between objects. Objects participate in inheritance relationships (IS-A), containment relationships (HAS-A) and calling relationships (USES-A). Virtually any object-oriented system is composed of multiple objects which relate to each other in these three ways.

In the relational database world, similar patterns are found. One of the most common relationships is that of a master-detail, which is essentially a containment relationship. The inheritance relationship is cumbersome to represent in relational models, yet out of necessity usually implemented by either putting all the "sub-types" in the same table and having some columns not apply for certain rows, or by putting each sub-type in its own table with a foreign key reference to the "super" table. Lastly, separate data entities in a relational model may "use" other entities where they have data values in common. For example, a TAX_RATE table which lists the sales_tax_rate for specific geographic regions may be "used" by a query accessing the SALES_ORDER table, with a where clause which joins on the geographic region for the sales order and the regions listed in the TAX_RATE table. Therefore, we see relationships in the relational world similar to those used in the object oriented world.

Therefore, it is reasonable to expect that needs will arise for the same types of relationships in the XML world. Furthermore, these relationships may be desirable at two different levels. These relationships may be needed at the DTD or "meta-data" level and/or at the XML or "data" level. This paper examines mechanisms for rendering such relationships within XML and related technologies.

Perhaps the simplest relationship to consider is that of containment. For example, it would be handy to be able to define the DTD for "Date" once, and then include that definition in DTDs for any other document which includes date information. This constitutes "DTD-reuse" and can already be done via Parameter Entity References.
Example: <!ELEMENT ProjectDate (% Date;)>

It would also be useful to create an XML document containing an overview description for a project, and then include that overview description in other project documents such as architecture documents and project status documents. This can certainly be accomplished by a regular hyperlink, but this can be inadequate because such documents are frequently printed and the hyperlinked content doesn't come along
with the printed page. It would be more convenient in some cases for the content to be included "inline". This can be accomplished via External Entity References.
Example: <! ENTITY boilerplate SYSTEM "myurl.xml">
Once the above reference has been made, the specification of "&boilerplate" in a .xml document will result in the referenced URL being included "inline".

Another solution for this need appears to be offered by the recent XInclude proposal.
Example: <xinclude:include steps="1" href="myurl.xml"/>

It is next necessary to review the facilities for achieving inheritance. It would also be useful, for instance, to define a DTD for "PROJECT" with elements such as "owner" and "purpose", and then further refine that for different types of projects such as "LARGE_PROJECT" and

"SMALL_PROJECT". While the DTD model is inadequate to represent OO type inheritance, the XSchema model does allow for both extension and refinement of a "super" schema.

Lastly, an examination is needed of how the XML world enables linkages between XML resources. Looking for a "uses" type relationship within DTDs or XML documents is a little awkward in that this type of relationship implies behavior, whereas DTDs and XML documents merely represent data. On the other hand, it is fair to expect that some behavior-implementing process will want to be able to use relationships between documents without having to maintain the relationships itself. One type of requirement is to merely link related documents in some predetermined ways. The XLINK proposal, currently in the "last call" phase of the proposal process, offers possibilities here. It provides for both simple and external links. Simple links are created within an .xml document to point to another document and function much like regular hyperlinks.
Example: <simplelink xlink:href="myurl.xml"> Link Description </simplelink>

External links are much more powerful and allow for the linkage of documents to be specified external to the document being linked.
Example:

```
<student
  xlink:href="/registry/mickeymouse.xml"
  xlink:title="Mickey Mouse"/>
<class
  xlink:href="/classes/xmlclass.xml"
  xlink:title="Basic XML class" />
```

Another requirement is to dynamically establish relationships between various DTDs or XML docs based on their having elements of common meaning. For example, if there is a DTD for "Student" with an element "CITY" and a DTD for "Class" also with element "CITY", it would be helpful to have a way of specifying that these "CITY" elements mean the same thing so that they could be linked. Ideally, the .XML docs reside in some sort of container, such as a DOM (Document Object Model) or XML-enabled database such as Oracle 8i, which can maintain such linkages and support dynamic searching, joining and linking functions across different documents.

There are therefore, several mechanisms emerging in the XML arena which will enable implementation of many of the types of standard data relationships to which we've grown accustomed.

There are, however, still many things which are difficult to do using solely XML and which are best accomplished through the use of both XML and other technologies which have been XML enabled.

One example is combining content from several XML documents and performing mathematical functions (e.g. summation, average, mean) upon the content. Attempting to do this with say, XSL, would be tedious. This sort of function is better done by an XML-enabled database or perhaps a DOM.

It is also cumbersome to synthesize and interleave information from multiple documents. For example, each member of a development team puts together a project description with high level milestones for their project. The manager would like to be able to combine all those documents and create a list of all milestones for the team, sorted by date. This again is cumbersome to do in XSL. Again, this demonstrates that XML and related technologies are not adequate to solve many types of problems in isolation.

It is therefore necessary to XML-enable many existing technologies, such as databases, spreadsheets, project management software, so as to leverage the strengths of those technologies in areas where XML technologies alone are inappropriate.