XML and Objects

© Charles Weir
Paper for OT2000 Working Group
Friday, March 24, 2000

What has XML to do with objects?

XML is a markup language. It allows you to express content for data structures in a machine independent and easily-to-debug way.
The emphasis on data structures maps clearly to OO languages. OO supports data structure as 'objects'. OO languages define the content of data structures using Class Definitions; XML defines them as DTDs.

However OO languages also define and encapsulate operations on the data for each object. How does this fit in with XML?

Objects are fine while the data is used for processing, but for longer term storage or network transfer we need to use another format – a process I'll call 'streaming'. There are hundreds of thousands of such formats, most of them of binary data (e.g. Microsoft Word files, Java streamed objects ?, EPOC C++ streamed objects, CORBA, Sun's NFS binary format).

Streaming is fairly straightforward to implement for a language and environment. We define input and output stream classes, and provide operations to write each native data format to these streams. (E.g. EPOC streams, Java persistence?. But not C++ IOstreams, which really solve a different problem). Class authors provide stream operations which use these basic operations (and those defined by other classes) to save data to the stream. Of course, the problem for authors is always what to do with pointers; some refer to embedded data; others must be encoded using some kind of identity for the streamed object. [Newbie question: How does XML do this?]

In one sense XML provides another such 'streaming' format. [Question. Can you write XML streamers in this simple way, or are there complexities I've missed?]

So why should we prefer XML?

1. XML makes it easy to automate translation between data formats. It's relatively easy to produce a XSL grammar to, for example, manipulate fields. It's straightforward to produce a translator (DOM?) to convert XML to and from into a binary format. And of course it's easy to produce a XSL grammar to convert the data for simple viewing by HTML browsers.

2. XML allows us, potentially, to standardise base formats. Thus, for example, all word processors can use the same basic DTD. Each WP can add its own extensions and new data formats; other WPs will ignore the extensions they don't understand. That ends the current ghastly problems of, for example, authors of email viewers; rather than having to support a plethora of translators from various formats (Word, WordPerfect, RTF, FrameMaker etc.), they can just provide a basic viewer to whatever standards they need.

XML to set up Configuration Data

The capacity for translation opens up many opportunities for XML even with existing applications that use binary data. There are many situations where we need to define initial values data in the application's streamed format. Two examples are Help Files, resource files and configuration files. In the EPOC tool chain alone we there are four different such formats, each with its own translator and syntax.

Hitherto, the normal solution has been to invent a bespoke data format for the users and provide bespoke translators to the application's binary data format. This is awkward, labour-intensive, difficult to maintain, and often provides inadequate error checking.

An XML-based solution is an attractive alternative. Browsers with DTDs, or text files, can provide data entry; XSL grammars provide the translation to a text version of the final format and some error checking, and a final generic tool for the environment (DOM or whatever) translates the result into the underlying binary format.

So we can use XML to provide translation and data definition for existing stream-based OO data files. The strength of XML – the wide variety of interfaces, tools and translators supporting it, will significantly reduce the programmer effort to support such data files.

XML for Data Transfer

XML is also an interesting choice for network-based interactions transferring objects between different systems. Some of the benefits are:

- It is human-readable, making problems easy to debug
- It is extensible, as discussed above, making it tolerant of different software on the different systems.

Contrast XML, which transfers the content of an object, with CORBA, which allows messages between objects.

The disadvantage, often cited, is that it adds bulk. However modern compression techniques can more than offset the redundancy of the markup, since they compress the data sent as well. For example, the HTTP protocol supports (though it's rarely used) GZIP compression on its messages; GZIP will give good compression on this kind of data – certainly giving smaller results than uncompressed un-marked-up versions of the same data; other compression schemes tailored to different DTDs will be able to

XML for Replication

This is one of the most exciting possibilities. Replication is about sending only changes to objects[1] (identified by global identifiers) rather than repeating entire objects. When an object is an entire database (such as a Lotus Notes database), the benefits of this – particularly for small handheld machines – are clear.

[Q: How will this work?]

Conclusion.

XML seems great. When will the tools be complete? What will it take to make it acceptable to organisations?

[1] XML Fragments specification, says Andy Longshaw, SyncML says Symbian.