Why XML?
Chris Ferris

XML stands out as one of the least object-oriented technologies to surface in many years. It was created in an effort to address the shortcomings of HTML with regards to electronic publishing and web applications encoding.

XML is lacking, presently, in all the characteristics that we expect from an OO technology:

No inheritance

No data hiding

No behavior (and hence no encapsulation)

No polymorphism

No types

So, why is XML seen as a complement to an OO technology such as Java? The common slogan that 'Java is portable code, and XML is portable data' misses a key point and may be at the heart of some of the controversy. Java is not JUST portable code, as an OOL it provides portable OBJECTS: data and behavior. So why do we need XML?

We need XML because we need to communicate between applications, across networks. Given the current technology, it's just not feasible to communicate using objects, because we have little or no control over the the systems with which we need to communicate.

Sun's IT organization recognized this over 7 years ago. In an effort to improve our application integration capabilities, we developed something we called SDDL or Self Describing Data Language which had many of the same characteristics of XML/DOM. We developed SDDL to describe and provide the content of event messages to be exchanged between application systems via an internally developed MOM infrastructure. We also developed a DOM-like API to access the message content.

Our objective was to replace all of the one-of interfaces between application systems, typically batch extract/import, with a standard set of object and event definitions which modeled the business processes and which would be shared by all application systems.

We recognized early on that the systems which were the target of our application integration efforts had widely divergent uses for the information they either produced or consumed. In addition, they also have widely divergent schema representation of the

same information, usually tuned to the needs of the system. An ERP system's schema would be tuned to high transaction levels where a data mart, data warehouse or reporting system would be tuned for ad-hoc queries against the information.

We concluded that passing objects on the wire (an encapsulation of data AND behavior) was inappropriate to the task given the fact that:

- many of the application systems with which we needed to integrate are not object oriented but relational and procedural in nature

- our intent was to enable the exchange of information, not behavior

We believe that XML, XML Schema, XSLT and the XML Data Binding JSR will provide us with a standards-based replacement for our SDDL,enabling a richer set of integration possibilities and addressing some of the issues we now face as we extend our application interfaces to the web and to the external systems of our customers, partners and suppliers.

XML Schema will offer us the characteristics required to more expressively describe and constrain the objects and events published to our MOM infrastructure, giving us the same features we currently enjoy.

XSLT will provide us with an improved facility for transforming the event message content to make it more palatable to consuming applications as well as providing us with the ability to filter content to exclude sensitive information from those systems which shouldn't have access.

The XML Data Binding JSR will facilitate the development of interesting Java applications which either produce or consume events. This will also provide for localized behavior to be bound at runtime to the data content of the XML event.

In conclusion, we need XML to support asynchronous distributed processing across a heterogeneous world of systems with widely divergent uses for the data they exchange.