XML and Objects

Laura Hill

The once popular call for "objects everywhere" has been replaced with "objects everywhere and XML as the go-between", particularly as we head into more global and federated environments. Today's challenges lie not in modeling complex systems, but in aggregating and integrating seas of complex systems and allowing the emergent properties of the result to come forth.

We envision a generic architecture that supports systems composed of collaborating services. Services are clumps of functionality that may have specific quality of service levels and may or may not be contained or controlled within a single organizational boundary. They are the units which are developed, deployed, updated, billed, aggregated, managed and monitored and may be operated on-site, at a service provider, or at a partner site. They are deployed on the Internet, obviating the need for VANs and other expensive infrastructure that keeps the entry bar high for potential players.

It is the infinite potential for aggregation that makes services the appropriate unit for federated system development. Multiple business services can be aggregated into specific business processes by an integration service. A portal service can aggregate multiple presentation, personalization and categorization services. A gateway service might aggregate multiple offerings for distribution within a home. Package integration requirement skyrocket as divisions merge, companies are acquired and new partners knock on the door hourly.

In this world of dynamic aggregations, fluctuating partnerships, decentralized deployment and control, a profusion of firewalls, and a network of variable reliability it is extremely important to keep the elements of a system as independent, coherent and as open to collaboration as possible.

It makes sense to implement individual services using object technology for all the usual reasons – objects support a meaningful way to modularize behavior and present clean interfaces for collaborating intra-service. It makes sense to use XML as the message protocol (transport and interface styles discussed below) for communicating between these services because it supports heterogeneous endpoints, is simple to use and allows each end point to maintain little or no knowledge about the other.

It is clear that objects and XML were destined to collaborate, but what is the model for that collaboration, what do the boundaries between an XML message and an object instance look like and what technologies or tools should be used to facilitate it? The rest of this paper attempts to address these questions.

Models for Object / XML Computing

There appear to be three models for Object / XML computing:

- Event-based
- RPC-style
- Document-based workflow

Event-Based Computing

Event based computing is the most decoupled of the three options – in it, each node has little or no knowledge of the other nodes. Usually based on an asynchronous messaging system with publish-subscribe capabilities, a node subscribes to events that are packaged as XML messages, and handles the event in any way that it chooses. Similarly, if a node wishes to announce an event, it publishes that event without concern for the receiver and eventual processing plans.

Often the XML event message will contain not only the event, but data pertinent to that event as well. For example a new-purchase-order event message, might also contain the details of the actual purchase order. The event itself can be parsed with a SAX parser for identification, and is then

tossed aside. The contained data, if there is any, may be bound to an object or parsed and tossed (see RPC-style section for more details)

Messaging systems are most often used within an organization, but several messaging vendors offer extensions that allow their messaging traffic to be tunneled over HTTP and thus fly through firewalls with little challenge to their progress.

RPC-Style

The RPC-style distributed computing model is well-known and sort of loved. It is a simple model where the programmer has control over what is happening throughout the system. It is easy to debug and relatively straightforward to develop. RPC style computing does imply intimate knowledge of the other nodes in a system, however. When a new node is added, additional interfaces need to be incorporated in the sender.

There are a number of ways that XML and objects can collaborate in RPC-style computing, differentiated by the manner of function dispatch:

- A traditional RPC mechanism such as CORBA, RMI or DCOM, may be used, with an XML message encapsulating the parameter data
- An XML message may be received, parsed and handled by a dispatch table that knows which method to invoke, such as in the EJB 2.0 JMS interface proposal. In this case, dispatch information is contained within the receiving node. This is almost the same as the event-based computing discussed above with the conceptual difference around the nature of the message.
- The XML message can include the dispatch information as well as the parameter data as in the proposed SOAP specification

In any case, there are two primary options for what to do with the data once the node has access to the XML payload.

- Use SAX to get relevant pieces of data out of the message
- Use object binding techniques to bind the message data to a particular object, which may be ephemeral or persistent. With this approach the data has now become associated with the specific behaviors that is part of that object's protocol. This interesting thing here is that different groups of behaviors may be associated with a single XML message depending on the context. The same XML message might be associated with different sets of behaviors as it is embraced by different object nodes

Document-based workflow

In a document-based workflow scenario, the XML message, or document, may be passed from node to node according to a defined workflow. Each node may need to review or modify the document. In this case, it makes little or no sense to instantiate the XML document into an object. The XML document is manipulated by the node using DOM, which maintains the XML document structure, and then sent on its merry way.

Summary

This paper has attempted to classify the different models of Object – XML interactions as they might occur in the types of collaborative, distributed and heterogeneous systems we are building today. A next step might be to identify which of the options are best suited for which environments and problems.