

XML and Application Servers: A standard too far?

By Ronan Bradley, PolarLake, CEO

This article discusses the view that Application Server vendors have so far failed to extend their products to effectively support XML and Web Services integration. Specifically, the means by which the Java standards bodies and Application Server suppliers have embraced XML means the underlying business value of the XML document is being lost. This gap needs to be filled.

And what is the "gap"? It is the difference between the data and format required by the Application Server and the XML document that it's being integrated with. The "bridging of the gap" includes functions such as message transformation, business rule validation, data enrichment and exception handling.

The emergence of the Java™ platform and then J2EE™ brought portability to application behavior and also a complete set of support services, in theory significantly reducing application development time and cost. The J2EE application server has allowed Java to become the enterprise development language of choice.

It is my view that Application Server vendors have so far failed to extend their products to support XML and Web Services-based integration (for the purposes of this article, it is best to think of Web Services as XML with some bells and whistles added, and when I refer to XML I include Web Services). This failure can be seen in the concerns of Application Server users about performance and complexity when dealing with XML.

XML provides data portability

Before we get into the specific issues around Java support of XML today, a little background.

As Java addressed code portability, XML can be thought of as addressing the *data portability*. It provides a standard and universal data format for representing high-level business information, independent of application or infrastructure. Its popularity is demonstrated when one sees organizations such as ISDA and SWIFT busy defining messages and documents in terms of XML schemas (which are like templates for categories of XML documents).

Java standards miss the point

At first sight, XML is the perfect extension to the J2EE platform, Yin to Java's Yang. But to date this promise has not been delivered; the Java standards process seems to consider XML as simply another way of getting data into a J2EE application server. To put it another way, the goal seems to be to throw away the XML document as quickly as possible and get back to the comfortable world of Java objects.

Reflecting this mindset, most of the Java XML standards are primitive and miss the point. SAX and DOM treat XML as a stream of bits-and-bytes, not business documents. JAXB requires the definition of complex and static maps that translate each type of XML document into a type of Java object. Transformation between formats, the heart of any integration project, is relegated to a scripting activity (using XSLT) with all the resultant complexity and poor performance.

Before I am attacked by fans of Application Servers (and I would count myself one), I am not saying that application servers are not suitable for the new XML world. What I am saying is that the requirements need to be examined and either the application servers significantly extended or other products considered that fill the gap.

XML documents are business transactions

In order to identify the gap, let us think about what a XML document represents. It is a business transaction and the processing requirements are closer to traditional business processes than simply converting XML documents into Java objects or mapping Java function calls into XML.

The XML documents sent to the application will often be complex, and unlikely to be directly mappable to existing Java objects or interfaces.

In fact, the documents are either:

- industry-defined (a good example being FpML for financial services products such as foreign exchange contracts)
- company-wide standards
- defined as part of an inter-company service agreement for B2B integration

This means that the application developer no longer has control of the definition of the incoming data that's to be processed, and that this data will no longer directly match the data structures and objects already in the application.

This creates the problem: "How do I bridge the gap between the format of the incoming XML document and the existing capabilities of the application?"

The four attributes

Real world experience is showing that to effectively integrate XML with Java systems, the documents need to be:

- *validated* against business definitions,
- *enriched* with additional information and
- *transformed* into the correct format before being mapped into the Java domain

In addition:

- *exception handling* must be managed intelligently

Strong validation, transformation and enrichment capabilities are the first three attributes to be checked in any potential solution. This last attribute is probably the least understood by vendors, even though it was reported by the Hurwitz group in July 2001 that nearly 80% of the time spent in building business processes is spent in exception management.

Lets take each of these attributes in turn:

Transformation

The format of an incoming XML document often does not represent the business information in the format required by the application server. There might be more information that must be filtered out, or less information that must be augmented in some way, or the information may be in a different format or order.

This is because the XML schema definitions are often defined independently of the Java data structures and do not directly match. There may not even be a one-to-one relationship between the XML document and Java data structure, and XML elements may map to several objects or may have to be concatenated with other elements before mapping. Therefore, the objective of the transformation is to create an XML document that can be easily mapped into a Java object or call.

Validation

Validity is potentially the most significant problem when dealing with XML-based integration. Accepting an invalid document has potentially disastrous consequences for any application, and unfortunately it is easy to create XML that is technically correct and will pass any schema validation but is nonetheless nonsense from an application perspective.

What is required is *semantic validation*, i.e. ensuring the XML content makes sense, both at the level of individual fields and between fields.

A simple example:

- checking that “start date” is earlier than “end date”

More complex examples can include checking that:

- the sender is allowed to send a particular type of document
- the document has not already been sent
- the size of the order does not exceed the maximum size that the sender is allowed to make, and that the size doesn't exceed the quantity in stock

Enrichment

Enrichment ensures the data is complete, and is likely to involve other applications (e.g. databases, Web Services, etc.) as well as internal calculations (e.g. aggregation, derivation, analysis, etc.).

Some simple examples:

- take a customer ID and look up the associated customer name in a database
- sum the cost of multiple line items to get the total value of the order

In each case, additional processing is required prior to the delivery of the information to the Java application.

Exception Handling

It could be argued that the application should catch any exceptions arising from validity checks, transformation or enrichment, but it may not have needed to do so prior to a particular integration project and it may not be possible or desirable to modify it given the contingent costs and disruption. Also, different validation failures will require different actions, from outright rejection to sending the sales person an email alerting them to the large order.

So what does it all mean?

To quote Mark Twain, ‘to a man with a hammer, everything looks like a nail’. I believe the same attitude exists within the existing Application Server vendor. If XML is just another way of getting data into application servers, why are we spending so much time and money defining XML documents? Instead, we need to recognize the different approaches needed, and select the appropriate tool, not just reach for our trusty hammer. This is true even before we start to think about how we can use XML to bridge between J2EE and .Net