

# Why XML isn't the solution, and why Web Services isn't helping

By Ronan Bradley, PolarLake, CEO

## Abstract

This article discusses the view that adopting XML does not, on its own, do anything for integration. Why? Because XML is not the universal language for representing data, only the universal alphabet for representing data. Different 'languages' (i.e. XML schemas) are being created for different business purposes, and vendors are producing mapping tools that allow the 'transformation gap' to be bridged (i.e. translating between different XML formats, and between XML formats and existing application formats).

## XML - universal alphabet, not language

There is a common and understandable misconception about XML: that in some mysterious way it *automatically* allows systems to integrate. Clearly, to connect System A to System B there needs to be connecting infrastructure such as queuing systems or Web Servers. However, even if this exists, adopting XML does not, on its own, do anything for integration. This is because XML is *not* the universal language for representing data; it is the universal alphabet for representing data.

To use an analogy: It is like the Latin alphabet. More accurately it also includes some extra concepts perhaps analogous to punctuation and grammatical concepts such as verbs and nouns. It can be used for different languages, but just because you recognize Latin characters does not mean you can read English, French and German. However, the effort required to translate between the languages is much less than between English and Mandarin Chinese (which is also reflected in how much easier it is for an English speaker to learn French than Chinese).

Similarly, XML is the alphabet that makes translation between data formats much easier. The necessary next step in using XML as an integration technology is to define languages for different businesses or aspects of the business.

## And now the important part: defining the schema

These languages are built up from sets of schemas, which are templates for XML documents. The XML schemas defined are *business* level documents, and should not include implementation-specific details relating to, for instance, infrastructure choices or programming languages. Schemas are being defined in two ways:

- Across industries, and particularly in financial services, consortia have been quietly working away for a number of years defining schema standards. Prominent standards in finance include RIXML, FpML and XBRL.
- Within companies, specific schema standards are being defined that capture the business documents for that enterprise. While less public, these are probably more widely used today.

The language analogy is a little stretched as the schema sets are better thought of as terminology sets: the way a company thinks of an invoice, a purchase order or a stock trade. Public standards are not being used much inside companies, as they don't capture the uniqueness of each company's business process. This is probably because of the close linkage between the way a company does business and the schema defined. However, I believe they will be used for cross-company transaction, where standardization becomes important.

Agreeing these schema sets is the key to successfully using XML as an integration technology. This is not very prominent in the press, partly because it is going on inside companies and isn't public, but partly because it isn't as sexy as Microsoft and IBM jockeying for position (or Sun or BEA or Oracle etc.)

## What about Web Services?

In 2002, the chief villain (or perhaps victim) in the hype stakes was Web Services. It sometimes feels like there are more proposed Web Services standards than deployed Web Services! The bad news is that most of these standards, around reliability or security or management, are probably several years away from maturity. The good news is that, in my experience, most Web Services and XML-based integration projects won't need them anytime soon. This is because they are inside the firewall and in most cases reliability, security and management facilities are already available. Thus all vendors need to do is integrate with the deployed infrastructure such as IBM MQ-series and HP OpenView, as we do at PolarLake.

A second piece of good news is that the work done on XML schemas transfers automatically to Web Services as the payload of the SOAP document will contain the XML document. This means that the real potential of Web Services inside the enterprise can today be realized

## “Why can't everybody use my XML schemas?”

Having defined the set of XML schemas that we want to use, we need to realize that they aren't going to be the only ones our systems will need to use. You may recall the old joke about how to make non-English speakers understand what you are saying: 'speak loudly enough and slowly enough' and eventually they will understand'. A lot of the first generation solutions for XML and Web Services take a similar approach, with XML support consisting of a set of documents, often closely mapping to existing programming interfaces.

Specifically, support for XML is not the same as support for XML integration. This is because different schema sets are needed for different aspects of the business (Human Resources needs differ from those of Purchasing) and there will never be a single language for all dimensions of the business.

## Bridging the transformation gap

This then exposes a fundamental issue when using XML: bridging the transformation gap. Those who have been looking at XML for a while shouldn't panic. I won't be getting into obscure and overly complex scripting techniques (such as XSLT). This transformation gap is at a higher level.

Imagine the simplest XML or Web Services-based integration problem: an existing order processing system needs to receive orders from another department or another company. The purchase order will have a number of segments, such as the sender's contact details, the list of ordered items, order references and other terms. There are two types of transformation gap:

- **The gap between one format of the information in XML and another:** Department 'A' may represent orders differently from Department 'B', although the business intent maybe the same.
- **The gap between an XML format and an existing application based format:** a relational database representation of the information (in rows and columns) and the XML representation.

Unfortunately, both types are often non-trivial and require a mix of many types of transformation such as:

- Map element A in the input document to element B of the output document (this is the easiest approach)
- Map element A, if it is part of element B, to element C, which is part of element D. Otherwise, if element A is part of element E then map it to element F, and so on.

On top of this, the mapping may need to take external information into account such as retrieving the customer name from a database, based on the customer id, and then put the retrieved name into the output document. (At this end of the spectrum, the line between what is transformation and what is business process can get blurred.)

So how do we bridge the gap? The previously mentioned scripting language XSLT (the transformation standard) is not well suited to many types of real world transformations. It is possible to code your way out of the problem, but this creates hard to maintain “spaghetti code” that must be repeated with each new type of document.

Luckily, some vendors such as PolarLake are addressing this problem with specialist mapping tools that allow the transformation to be created. The good news is that these tools combine the ease of use of some of the XSLT tools with the transformational power of some earlier tools that came as part of large expensive Enterprise Application Integration (EAI) suites.

## So what does it all mean?

As with any new technology standard, you must recognize how much of the problem it solves and figure out how to get the rest done. XML and schemas provide a great basis for defining the business documents which flow around the enterprise. They provide a real benefit in easing the effort required to translate between documents, by abstracting away from the infrastructure and providing a common structure for all the documents. The task of integration still requires effort (albeit greatly reduced), and a new task appears: intelligently transforming between XML documents and other formats.