



*Is all the effort and disruption worthwhile?*

# XML—Friend or Foe?

**A**t Tradeum Inc. we became early adopters of XML just a few months after the publication of the XML standard in February 1998. Our reasons were compelling: we were creating a dynamic trading engine that implements auctions and true exchanges in a business-to-business environment.

At the heart of the application was a parametric matching engine that matches buyer-and-seller offers based on industry-specific parameters describing the goods to be traded, price and other terms and conditions, and buyer and seller identity. What better data framework for receiving parametric buy-and-sell offers than XML, the framework in which more and more industries would be standardizing data interchanges?

By creating a trading engine that works with any DTD (as opposed to the approach that imposes a DTD such as cXML, which is more suited to cataloged goods), our customers – the market makers and enterprises creating B2B exchanges – can connect directly to buyers and sellers using the standard XML formats in their industry. By embedding the special range and wild card elements that we developed, such as `<range/>` and `<any-element/>`, in the XML documents, buyers and sellers can express flexibility with respect to aspects of the transaction. These are resolved by the system to finally provide a fully defined transaction description in XML.

And yet, despite the strong fit between our objectives and those of the XML initiative, our experiences with XML have been mixed at best. Our frustrations weren't limited to the obvious problems: the lack of tools for a new technology and the surrounding technologies' lack of stability during the last two years. (One example of such frustration leaps to mind – the lack of stability of the XSL transfor-

mation now called XSLT. I once wrote to an XSL tool provider to complain that his tool didn't even recognize the most basic XSL expression `<xsl:process-children./>`, and this had cost us several critical days of development. He wrote back that the working draft had recently changed and this expression was renamed `<xsl:apply-templates/>`!) This article concentrates on inherent issues with the XML standard.

## Semistructured Data

XML defines a syntax for semistructured data centered around the `<xxx>...</xxx>` syntax for elements. *Semistructured* here indicates that specific XML schemas can impose typelike constraints on documents but need not rigorously fix the elements and their types. This offers considerably more flexibility than a programming language object conforming to a class file or a database record conforming to a table definition. Arguably, however, an XML document has considerably less flexibility than a graph of objects or a scheme of database tables.

Coping with the differences in paradigms between object graphs, relational database tables and XML documents will provide employment for software professionals for years to come in a way that only COBOL Y2K-bugs and chasing C/C++ pointers have done up to now. Therefore, XML should be judged on the basis of a cost/benefit analysis as well as on its efficacy. The cost of an entirely new data paradigm is exceedingly high

(witness database vendors struggling to come out with a usable package to store XML documents efficiently in a database without too much human intervention in mapping XML schemas to relational database schemes).

One important innovation (arguably the only innovation) that XML does offer is a formal syntax for defining the semistructured schema to be satisfied by documents for particular applications. This provides flexibility that's useful in allowing variation in data structure from subcategory of goods to subcategory of goods, user to user or system to system. Of course, this flexibility always comes at the expense of making data structures less predictable and processing more complicated.

The use of a formal language to describe the schema allows mechanized parsers to check the validity of an XML document. This schema syntax has hitherto been an awful syntax called DTD, which wasn't even an XML language (supposedly the ultimate language for any semistructured data), soon to be replaced by a slightly less awful syntax called XML Schema, a well-formed XML (allowing lots of recursive fun with an XML Schema defining the syntax for XML Schema). XML Schema will be widely backed (which probably means that the only name not reserved for a type that someone wants will be `<designed-by-committee/>`).

XML provides a reasonable framework for semistructured data and a standard language (at least two "standard" languages) for defining schemas. The rest of the hype can safely be ignored. For

## AUTHOR BIO

Zvi Schreiber is the founder and CTO of Tradeum Inc., which pioneered the concept of dynamic B2B electronic commerce and adopted XML in 1998. Zvi holds a PhD in theoretical computer science from Imperial College, London. He lives in Jerusalem, Israel.

example, XML isn't self-describing in any meaningful way. You can call the element that describes color `<color>` (or, giving away my roots, `<colour>`), but this means nothing to the computers – the document's intended audience – at least, nothing more than having a variable name "color" in your object or a field "color" in your database table definition – no one ever called these self-describing. So XML and the associated schemas are really about syntax, not semantics. The semantics are still described in plain language.

Is XML at least the perfect data language for the specific purpose of the exchange of semistructured data? Unfortunately not, for the simple reason that it wasn't designed for this purpose or, at best, it was derived from languages that weren't designed for this purpose. XML is a subset of SGML and a generalization of HTML, both largely designed for publishing.

Here are some examples of where the publishing heritage of XML gets in the way of the applications for which XML is now touted.

### Attributes or Textual Elements?

We had several debates at Tradeum on whether a range should look like

```
<range min="1" max="10" />
```

or

```
<range>
  <min> 1 </min>
  <max> 10 </max>
</range>
```

The XML standard doesn't provide any guidance on when to use attributes and when to use textual elements inserted in child elements. The reason is that in the publishing world it was clear: anything that didn't appear on the page (for example, choice of font) went in an attribute and anything that appeared as text on the page (including elements such as the page header) deserved a textual element. When applying XML to abstract data this becomes an annoying matter of taste. Needless to say, we ended up supporting both formats, which probably adds a few microseconds to the processing time of every document passing through our system.

### Ordering of Elements

Another area of confusion is the ordering of elements. Are the element lists `<a/><b/>` and `<b/><a/>` identical? XML specifies that attributes aren't ordered so:

```
<aa b="1" c="2" />
```

and

```
<aa c="2" b="1" />
```

are certainly the same, but in general:

```
<a/><b/>
```

and

```
<b/><a/>
```

are different. This can be useful if we want them to list the order in which items should be shipped:

```
<shipping-order>
  <item> ... </item>
  ...
  <item> ... </item>
</shipping-order>
```

However, it's more common to specify attributes by name, and the order is irrelevant:

```
<software-engineer>
  <experience> ... </experience>
  <languages> ... </languages>
  <favorite-pizza> ... </favorite-pizza>
</software-engineer>
```

What difference would it make if the order of the child elements was changed? What counts here is the element name, not its position. However, XML doesn't provide a general way of indicating that data should be viewed as unordered name/value pairs instead of an ordered list. For Tradeum this means that our software has no general way of deciding whether to match a buyer and seller who specify XML documents that are compatible except for the order of elements.

This lack of attention to whether elements have an order isn't surprising, since in publishing everything is naturally ordered by the order in which it's supposed to be read! The concept of unordered elements referenced by name didn't arise in the main applications of SGML and HTML. (On the other hand attributes, which in publishing represent data that wasn't part of the document as noted above, don't have a natural ordering and were declared to be unordered name/value pairs; just what we'd expect from XML elements in most nonpublishing applications!)

This dual and uncontrolled use of XML for ordered and unordered lists makes it awkward to talk formally about an XML document. This forces XSLT and XQL to provide two sets of formal ways to reference an element: using an element name path such as `software-engineer/languages` or using a position such as [2].

(Quiz: How do you mix the two methods and refer to the element in an element list that follows the element named `<start/>`? Answers on postcards please...)

### White Space Issues

XML has other annoyances. The language, which is supposed to be used for communication between computers, is typically formulated with lots of white space to make it indent nicely for human readers. This is a manifestation of the tenth declared design goal of the XML Specification: "Terseness in XML markup is of minimal importance." (One can't help wondering whether this rather negative design goal was added to pad out the number of objectives to 10 – a good example of terseness not being important.) This, of course, necessitates a set of confusing rules determining which white space is part of the text and which is to be interpreted as padding. (At Tradeum every time we thought we understood how XSLT dealt with white space, the standard changed. We've spent half our lives taking `normalize()` functions in and out of XSLT scripts. This said as much about the confusing issue of white space in XML as it did about the instability of the XSLT working draft.)

So there it is. XML offers a data format that's reasonably well suited for human reading (in a data language supposedly designed for computer-to-computer interaction and at the expense of annoying white space issues, etc.), a choice of powerful but somewhat clumsy formal languages for defining semistructured schemas (at the expense of forcing all data into a tree hierarchy and making processing complex) and the ability to handle semistructured data flexibly (but no inherent ability to distinguish between different types of data collection such as ordered lists or name/value pairs). Whether these benefits, such as they are, are worth the disruption of an entirely new data paradigm that's difficult to map to the relational or object models is a matter of opinion.

I'd like to conclude with a positive side to this story. XML has inspired many industry consortia to announce their intention to create standards for exchanging data in their industries (inevitably, on many occasions, several standards per industry). If half the standards announced to the press come to fruition (and a number already have), all the effort and disruption will have been worthwhile and our early adoption of XML vindicated. 🌐