The Myth of Self-Describing XML.

Eric Browne - Sept. 2003

The strange perception that XML is the panacea for system interoperability, or even information representation, is seriously affecting how systems are being designed and specified. I believe it is encumbent upon those of us who understand and are experienced in the domain of information representation and processing to try to explain the issues to those less experienced. Otherwise, the hype surrounding XML will exact its toll on us over the years - a toll the information industry can ill-afford.

I intend to make a start on this by describing the role of XML and by illustrating its features and shortcomings through two examples. One example is the transfer of simple demographic information from a hospital system to a GP system. The other is the potential use of XML for helping to transport the Sydney Opera House from Sydney to Perth.

XML stands for eXtensible Markup Language, and is a language framework for structuring and tagging pieces of data for transmission from one source system to one or more other systems. As such, its primary aim is to “serialise” and “format” the data. By formatting, I mean encoding the data value from its native representation in the source system into a character-based representation in the serialised stream. So a binary floating-point number might be represented as “32.467”, or a binary-coded[1] decimal number might be represented as “32.468”. If such a number represented, say, the age of a patient, it might appear tagged in XML as:-

```
<patient_age>32.468</patient_age>
or
<patient_age>”32years, 5 months”</patient_age>
```

This encoding/formatting aspect of XML has visual appeal and “allows” for interpretation or decoding by the receiving system. At the same time, however, it places a processing burden on sending and receiving systems.

Serialising of data is the process of creating a stream of the data items that need to be transmitted, such that one data item follows another sequentially in the message, eventually being transmitted as a sequence of ‘0’s and ‘1’s and ultimately as a sequence of electrons or photons that can be restructured into components by the receiving system. The analogy with transporting the Sydney Opera House hinges on the assumption that the Opera House cannot be transmitted as a complete building, and has to be broken down into smaller parts and placed onto a series of pallets for transfer via truck or rail. The parts could simply be placed arbitrarily on their pallets, or could be annotated and grouped in some fashion to allow the Perth builders to identify the constituent parts. Either way, serialisation refers to the process of placing the parts onto a sequential stream, that may, or may not, reflect their functional, spatial, or other relationships.

Relationships and groupings between data items can be represented by XML. Containment (e.g. asprin contains acetyl salicylic acid) and “is-a” (e.g. an analgaesic is a medication) relationships are readily expressed in XML, seductively so for containment relationships, since they can be visually represented because of XML’s start and end tags, and the oft-used “tabbing” of fields when illustrating snippets of XML in documentation or XML-editors. The use of start and end tags is also a useful mechanism for XML-parsers. Thus, we can express and visualise a simple XML representation of the Sydney Opera House as:-

[1] binary-coding is a storage method that allows for decimal numbers to be stored exactly to represent the number as input or written - i.e. each digit is stored separately.

```
<Building>
  <name>”The Sydney Opera House”</name>
  <sails>
    <front_left>
      <tiles>
        <tile id=1 material=”ceramic”>23.6</tile>
        <tile id=2 material=”ceramic”>23.6</tile>
        ...
      </tiles>
    </front_left>
  <front_right>
          ...
  </front_right>

       ...
  </sails>

     ...
</Building>
```

As part of its formatting role, XML can be used to format and annotate a data stream. In its simplest form, the **formatting** consists of delimiting individual data items using **<>** tags and, optionally, encoding binary data into characters that represent binary data, such as numbers, dates, etc. In its simplest form, the **annotation** consists of giving common language names to the delimiting <> tags, to tell the receiving system that the next piece of data in the stream is a name or a date or a diagnosis code etc. The annotation can be enhanced to provide additional qualifiers on each data item, such as name (at birth) or date (when diagnosed) or (code set for) diagnosis code. These are termed attributes of the data item/element. Data streams have been delimited and encoded in various ways since the inception of computers. Annotation has been less common, and is the aspect that now leads many people to suggest that XML data streams are self-describing.

Thus, the tagging and qualifying (via attributes) of data, allows the sending system to express values of complex concepts. Tagged data elements may be ordered and grouped to express values of even more complex concepts. Furthermore, the XML standard has evolved over the years to allow the expression of relationships between one data item (an instance of a concept), and another data item somewhere else in the data stream. A given data stream can have its formatting and annotation subject to a set of rules, that state which tags and attributes are permitted under what circumstances. These rules can be placed in a separate document and sent independently to receiving systems to help those systems parse and decode all data streams formatted according to those rules or “schema”.

It is partly the potential richness of the annotation of the data stream, together with the adherence to some predefined set of rules(schema), that has led to a misconception, by many in the IT industry, that data streams expressed in XML are, somehow, self describing in a way that their meaning can always be discerned by the receiving system. Nothing could be further from the truth! But the issue that probably leads most people astray is the confusion caused by the visual representation of XML on the one hand, set against its intended purpose of interpretation of data by receiving machines.

Consider the following:-

```
<Patient>
  <firstname>”Eric”</firstname>
  <lastname>”Browne”</lastname>
  <age>18.6</age>
  <medicare>
  <number>1234567</number>
  <join_date>”31/2/98”</join_date>
  </medicare>
</Patient>
```

Because we humans can readily comprehend this message, we tend to prescribe a similar facility to machines. But XML is **not** intended to be read primarily by humans! We should consider how such messages would appear to a machine, which doesn't have our level of knowledge and understanding. The above example should more appropriately be expressed something like this:-

```
<pzagg><ffgf>"Eric"</ffgf><pskk>”Browne”</pskk><a1>18.6</a1><
mzdcyy><nnb>1234567</nnb><jzndd>”31/2/98”</jzndd></mzdcyy></p
zagg>
```

[Even when reading the above stream, as humans, we tend to impute meaning from the data itself - e.g. we infer that “`Eric`” is the name of a person. This is not the case for machines.]

When the receiving system receives such a message, it has to do something with it! It has to “parse” it and it has to “process it”. It does the former using generic software that merely understands the structure and syntax. It does the latter by applying what is often described as “business logic” or “business rules” to the data items, so that the data can be validated, placed in appropriate locations in the receiver's persistent store (for later processing), or shipped out to a user interface or ..., etc. This business logic has to process the data in the messages according to their meaning or semantics. This business logic must exist *a priori* - it must have been created from some conceptual model that describes the domain to which the message belongs. The business logic cannot be constructed *post priori*! The meaning of a `<pzagg>` cannot be supplied along with the message. [or if it were, the messages would be unworkably huge, and would still need to be expressed in terms of an agreed set of lower-level components from which the business logic is built]. The business logic processes the message by comparing patterns in the received messages, to patterns it **already knows about.** When a pattern match occurs, the data associated with those patterns is then processed according to one or more rules. Both the patterns and the rules have to be known a priori! That is the essence of business logic. That is the essence of almost any software that needs to process messages. If the receiving system receives an XML tag that is not in its vocabulary, it can do very little with it. It cannot magically impute meaning to it, just as we humans cannot impute meaning to a `<pzagg>` or a `<mzdcyy>`.

Consider now, our hospital system, sending its patient demographic data to a General Practitioner's (GP) system. The hospital might store information thus:-

```
name
address line 1
address line 2
address line 3
address line 4
address line 5
employment category
```

The GP system might store demographics as:-

```
firstname
lastname
middle initial
street number
suburb/town
postcode
employment status
```

No amount of XML formatting is going to make these systems interoperable. The business logic at each end needs to change. They both need to conform to a common conceptual model. Even though the **GP** might be able to, the GP **system** cannot simply "understand" XML messages of the form :-

```
<name>"Rev. Eric Browne"</name> <address_line_1>"c/o
Bournemouth Caravan Park"</address_line_1> <address_line_2>
"Camden South"</address_line_2>
<employment_category>"baker"</employment_category>
```

Even where an XML-schema supposedly "describing" the content of such messages is supplied, there is still a major gulf between one system and the other. This "impedance mismatch" cannot be overcome without recourse to a common conceptual model, to which both systems conform.

Finally, consider the transport of the Sydney Opera House to Perth, as described earlier. Now unless the Perth builder has the correct notion of the concepts that the tagged parts arriving from Sydney represent, she is not likely to reconstruct the Opera House in Perth, the way it was in Sydney. The Perth conception of `<front_left>` may not match the Sydney convention, and Perth could end up with a much deformed version of the famed Sydney building. So even though the XML is being directly interpreted here by an intelligent human, there is still potential for misunderstanding.

How much worse could the result of such misunderstandings be for patient care!

It is critical that we consider the effect of embarking on a new methodology for data message construction, when the characteristics of such a methodology are often misunderstood and misrepresented by its proponents.

XML is **not** intended for consumption by humans, but by machines!

And machines are **not** humans!

Why the Hype?

So why is it that XML has garnered such an unwarranted following as a panacea for interoperability. Well, in order to explain this, it is necessary to describe some of the history of XML. First and foremost, XML has ridden on the back of HTML (HyperText Markup Language), which allows for documents to be marked up and delivered to web browsers on client machines. Rather than simply tagging parts of a document for formatting purposes, as is the case with HTML, it was considered desirable to markup, or describe the document in terms of its content, so that receiving systems could "process" the information in more flexible ways. XML was developed to allow flexibility in the way documents could be marked up. The tags to be used for annotating parts of the document, could be specified in a DTD (Document Type Descriptor) which could be used by receiving systems to "understand" the document structure, and to process it according to the receiving system's requirements.

The advent of business to business e-commerce required a mechanism for data and process interoperability, based on exchange of information using web (HTTP) protocols. XML seemed like a good option. It had already been used for limited exchange of web-based resources, using RDF (Resource Descriptor Framework) and RSS (RDF Site Summary). However, the number of concepts that needed to be represented was very small. As XML started to be used to cover a broader range of more complex information structures, the schema syntax became more complex. XML-Schema is a particularly complex specification, and although endorsed and promoted by W3C (World Wide Web Consortium), is only one of a number of schema representations in use. Yet its very existence, and the fact that it substantially overcomes many of the shortcomings of DTDs, has led to a belief by many that it "solves" the semantic interoperability problem, by adequately describing almost any document structure, even if it only does this in the **semantic frame of reference of the source system**.

It is this last rider, that is overlooked by many proponents of XML.

XML has been largely promoted by programmers and systems architects, because such people are able to "see", and therefore "understand" the semantics of DTDs and XML-Schemas. It is programmers who write the business logic that processes the parsed data streams, and since every example in every text book, and in every classroom uses elements and attributes that convey implicit meaning to the reader, such programmers and developers forget that such meaning is implicitly unknowable by the machines themselves. The machines need to be instructed what to do with each element and attribute, if they are to "understand" and act on the semantics of the message.

XML is very useful for describing the structure of data streams; it can tag individual items of data; it can qualify such items; it can associate individual items with others in the stream; it can group items together; and, augmented with XML-Schema or its alternatives, it can allow for constraining, validation, data-typing and other structural niceties, that hithertofor have been difficult to achieve across a range of operating systems and APIs (Application Programming Interfaces). There are many tools available to developers. It looks somewhat like HTML. It is supported by a well-funded independent international standards organisation. In short, XML is a very useful and widely adopted technology.

Adoption and acceptance of a technology, when it reaches some critical threshold, automatically induces further acceptance, irrespective of the merits of the technology. One invariably hears “We should adopt this technology because it has become a *de facto* standard”. But what is XML a *de facto* standard for? What should we adopt it for? It may well be appropriate for formating and serializing data streams for exchange between systems, but, by itself, it certainly is not adequate for semantic interoperability amongst heterogeneous systems devoid of a common conceptual model of their domain.

References:

Cover R. , XML and Semantic Transparency, 1998 http://www.oasis-open.org/cover/xmlAndSemantics.html

Smith H, Poulter K., The Role of Shared Ontology in XML-Based Trading Architectures, Ontology.Org *Included within Communications of the ACM, special issue on Agent Software*, 1999 http://www.ontology.org/main/papers/cacm-agents99.html

Simeon J. & Wadler P. The Essence of XML, *ACM Symposium on Principles of Programming Languages* **(***POPL'03)*., 2003,