# BINARY REPRESENTATION OF XML
# A POSITION

Rick Marshall
11 Aug 2003

From the XML 1.0 specification:

“The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.”

From simple pedantry any “binary” representation is something, but not XML.

But then all those taking part in this discussion no doubt understand that point. So this is a discussion and possible demonstration of how using binary formats can “improve” XML.

The main points of concern appear to be transmission speed and processing speed. In the long run there are more important things to worry about.

This position paper won't present any substantial figures, rather point out the difficulties and contradictions of the various positions.

To paraphrase mark Twain “There are lies, damned lies, and benchmarks” (I don't think this is original just don't know who first used it). I am therefore assuming that other papers will present some fine justifications for all manner of improvements. I have read some material and remain skeptical of the real benefits.

It is my assertion that there is little to be gained and a lot to be lost from this endeavour. I believe the original architects of XML understood this very well and therefore resisted the temptation to do just what is being discussed.

The topics canvassed here:

Readability, Legacy Formats, Tags, UTF-n, compression, binary portability, data presentation, and Moore's Law.

BACKGROUND

Briefly as you won't know me or my interests. 25 years from graduation at University of NSW in Computer Science and Electrical Engineering. For most of that time I have been developing and deploying a 5th generation database system – possibly still the only one of it's kind. This work has involved detailed decisions on data representations, fundamental data structures and algorithms, heuristics, optimisation, multiple CPU support, and because of it's multi-user/multi-site use communications and distributed applications.

At this time the overall most efficient and fastest way to store data remains an ASCII, relational, fixed record length structure. The same things that drove my decisions in this area will mostly apply to XML. Hence this position paper.

Looking back over the development of database technology it's interesting to note the parallels that now face XML.

READABILITY

One of the principle advantages of XML is it's inherent readability – although to look at some of the namespace stuff you might want to argue this point. If we walk away from this we will be doomed to using tools written by someone else to create and consume data. This is not necessarily the case at the moment with XML being written by tools, programs, and editors. I don't want to gloss over the difficulties of UTF encodings, however neither should we lose track of just how important this has been in the development firstly of HTML and now XML (not to mention SGML). We can just as easily consume. This is important to me – I'm not looking forward to debugging a binary format.

LEGACY FORMATS

In the past we have used many formats, some ASCII (eg EDI) and others not. The internet is awash with RFCs describing ASCII/binary interchanges eg POP3. One of the reasons for working with XML is the promise of a single standard that will always suffice. EDI was great except the Europeans and Americans were never going to agree on a format – even the separator characters are different(!), each company put out it's own guidelines on which bit of the standard to use and what it meant, and you need a row of manuals to use it that would make even DEC blush - doomed to tools. But it was a dense format and for commercial transactions there was no compelling reason to go away from it where two parties want to set up a simple and traditional trading relationship. ebXML has a long way to go before it can replace the simple uses that most have for EDI at the moment. It's interesting to note that there are attempts to get a DTD for the EDIFACT and ANSI X.12 documents in place.

We are in danger here of getting a lot of camels instead of horses.

TAGS

This is one of the most contentious and obvious areas that could stand improvement. However it will come at the expense of the goal of simplicity. The two problems I have are 1) finding the end of a tag and 2) not knowing how big a buffer is needed to store the data between tags before it can be processed.

I'd have to say that the present generation of DOM parsers could work faster. It can be very frustrating to have an XML document that will ultimately produce say 10 pages of pdf pass through xmllint, xsltproc, and gs and take about 1.5 seconds on a 1.8GHz P4. gs isn't that slow, although it does take nearly half the time. It takes approximately 0.9 seconds to validate and then translate my 53K document to postscript. I wouldn't want to processing megabytes this way.

So this is an argument for a) binary or pre-encoded tags, b) better tag detection algorithms, c) maybe start and end tags wasn't that simple an idea after all and the standard isn't as good as hoped, d) all the above?

My answer is (b). At least until someone writes a proof that it will never fly.

UTF-n

This is part of the same problem as described for tags.

COMPRESSION

The problem with compression is that it's so data and domain sensitive. Entropy – information content – in the message is the issue.

Domain knowledge might allow us to get some slightly better compression. Eg knowing that every number is less than 256 would allow us to convert all numbers to unsigned bytes first, but this is way too specific for a standard like XML. You'd still have problems with text, but if all you're transmitting is telemetry data this might be a good solution. But then why use XML at all?

One of our customers has a sales order file containing 160,807 multi keyed records. The data file is 106,615,704 ASCII bytes while the index file is 53,729,280 ASCII and binary bytes. Compressed with gzip we get 5,555,290 bytes for the data and 11,387,977 for the index file. Pretty good. Now if we dump the table in what we call attribute format (designed 10 years before XML!) - <attribute>,<value> with a '>' between records and empty fields skipped. Eg

order no,123

sku,ABC

quantity,5

>

we get 75,259,477 bytes. We get 5,790,483 bytes in a storage independent format.

You'll notice the similarity in size between the two compressed files.

Clearly if I trade processing time – table loading, for transmission time I can move a relatively large table quite quickly between machines.

XML is like my attribute format. The entropy of the message will determine the compression. You're not going to get much better compression than say gzip or bzip2.

BINARY PORTABILITY

What is an integer – 16, 32, 64 bits or more. Big endian or little endian. If it's 64 bits are the words big endian or little endian, etc. Floating point? IEEE notwithstanding.

I don't think you can deal with this without starting a new round of processor wars because noone will want their processor disadvantaged by an extra conversion stage in benchmarks.

And a quick note on ASN.1. As with detailed DTD's and Schemas, ASN.1 assumes you know what the message content is going to be. This breaks down completely when the XML is used to encapsulate other messages (XML or otherwise). We use XML extensively in our distributed transaction processor. In this case the content is database and system manipulation functions, only possibly including data.

## DATA PRESENTATION

The simple fact is that for many applications data can stay in a readable format. That's part of how I get good speed from my applications. Eg if you're storing the quantity on order why would you convert it to binary to store it when most of it's use is simply as display. It's easier and quicker to convert it to binary if and when it takes part in a calculation.

## MOORE'S LAW

Microsoft, IBM, Sun and others all have businesses based on the empirical evidence and success of Moore's Law. When these companies design a new interface or service they know they have performance improvements on their side. And while I wouldn't want to accuse them of poor coding practice, it is nevertheless evident that getting product to market is far more important than worrying about the subtleties of coding efficiency. Optimisers, speed improvements, and the ever growing resources of memory and disc allow us to ignore the concerns of the past.

I used to run an 11 shop retail chain on less than 2MB of disk and 64K or RAM – Z80 processor – data and programs. Now I can't even get a single program down to that size. We got by on 1200/75 or 300 baud modems for comms.

Hours, days, weeks were spent making things work in these tight constraints.

Moore's Law will take care of our processing and communications needs. We would be better focused on delivering better applications and tools than wasting time getting temporary advantages that will be rapidly overtaken by our friends in hardware.