Keynote

XML Five Years On: A Review of the Achievements So Far and the Challenges Ahead

Michael H. Kay
Software AG
Michael.Kay@softwareag.com

ABSTRACT
This is an extended abstract of the talk given by Michael Kay in the keynote address of the DocEng2003 symposium

Categories and Subject Descriptors
I.7.2 [**Document and Text Processing**]: Document Preparation – *format and notation, languages and systems, markup languages, standards*. I.7.1 **[Doument and Text Processing]**: Document and Text Editing – *document management*. H2.3 **[Database Management]**: Languages – *data description languages, query languages.*

General Terms
Management, Design, Standardization, Languages.

Keywords
XML, XSLT, XQuery.

1. INTRODUCTION
XML was launched on the world in 1998, and it seems to be a good time to take stock of what it has achieved, and what it has not yet achieved. It's important that we understand the factors that made it successful and let to its adoption, so that we can learn from the experience and try to reproduce these conditions as we move forward. It's also important that we understand that there are things that XML was simply never designed to do..

2. THE ORIGINAL GOALS
The XML 1.0 specification [1] is unusual in that it starts off, right at the beginning, with a statement of the design goals. This is an excellent list of design principles, for example:

- XML documents should be human-legible and reasonably clear
- The design of XML shall be formal and concise

DocEng'03, November 20–22, 2003, Grenoble, France.

No doubt these design principles contributed a great deal to XML's success. What they don't tell us, however, is what purpose XML was intended to serve: they don't constitute a statement of requirements in that sense. Perhaps this indicates that the requirements were obvious to everyone, which might be another factor behind XML's success.

The primary motivation behind the development of XML was the recognition that maintaining content for the web in HTML was a really bad idea, because it failed to separate content from presentation, and because HTML browsers were full of proprietary features that inhibited interoperability. There were any number of products that attempted to solve these problems, but because they weren't standardized, they only added to the chaos.

The people who developed XML knew that SGML offered a way forward but they also knew that SGML was far too complicated and expensive. The decision that XML should be a subset of SGML was made partly, I suspect, to gain acceptability, and partly because some people believed this would lead to quicker availability of XML software. In fact the main benefit of the decision was that it reduced the space of design possibilities and thus led to a quicker and smaller spec. Although SGML features were savagely removed, it is clear in retrospect that more could have gone without being seriously missed. Probably the biggest technical mistake was the retention of DTDs: these have always been the most difficult part of XML to learn and use effectively, and if they had been left out from the first version we would probably today have a much better schema language than we now have.

3. WHO IS USING XML?
It's easy if you are using XML to look around you and imagine that everyone else is using it too, for everything. We often make the mistake of imagining that the communities we work in are typical of the rest of the world, and very often we are wrong.

XML has made a serious impact on web publishing, without any doubt. It hasn't displaced direct authoring in HTML, and it hasn't displaced all proprietary tools, but it has made a serious impact. This was the field that XML was designed for, and I think we can rate it as a success in that area. XML is proving particularly important, I think, for the many sites that rely on content syndication. This is where the real value of separating content from presentation starts to become apparent.

In the wider publishing world, the impact is less marked. My current book is being produced using Microsoft Word and Quark Express. XML is starting to appear around the edges of these

toolsets, but it isn't yet mainstream. Where XML is being used in conventional publishing, it is usually because there is a strong requirement for re-purposing (publishing the same information in more than one form, typically in print and online), or because the publications are highly structured and benefit from XML's ability to manipulate the information by sorting and selection before it is committed to paper.

The other area where XML has made its mark is in data interchange between applications. This is perhaps a little surprising, since I can see little evidence that it was consciously intended for this purpose. But it reflects the fact that there was a pent-up demand for better solutions in this area. Before XML came along, there seemed to be four main approaches to the problem:

- EDI standards. These were generally regarded as extremely complicated and therefore extremely expensive to deploy. EDI required a serious investment decision, and even then the standards were seen as inflexible.
- ASN.1. This specification came out of the networking community and was quite widely used in specifying technical protocols. It never really caught on at the application level, mainly for the same kind of reasons as much of the OSI initiative failed: it was too complex for the average commercial programmer, and the tools were marketed at silly prices, meaning that the decision to use it had to be made at a level of management where the benefits were far from obvious.
- Application-specific interchange formats. Some industries or application domains established their own interchange formats, sometimes based around the products of a single vendor, sometimes around a true "industry standard". Many initiatives to create such standards failed, often because they got bogged down in the details of syntax and character encoding, rather than concentrating their efforts on creating the data model for the industry at a semantic level.
- Ad-hoc hackery. Any number of practical working data interchange solutions were knocked up by programmers using variations of file formats such as comma-separated values. Sometimes they worked well, but usually were not engineered with sufficient robustness to be extensible beyond the very narrow area for which they were originally designed.

The real reason for XML's success in this area is its accessibility. It is accessible to the ordinary commercial programmer (a character often referred to among the XML cognoscenti as the Desperate Perl Hacker, or DPH) for two reasons. Firstly, the specs are simple to read and understand (if you ignore DTDs, which you can). Secondly, the tools are readily available. You can download them and play with them without first producing a business case for investment. (How many technologies have failed because the programmer who needed them didn't know enough about them to be able to produce the business case needed to get hold of them?)

In addition, the XML tools went beyond mere parsing, and extended to transformation. None of the other approaches to data interchange has offered a high-level transformation language comparable to XSLT. This is critical, because it is a great mistake to imagine that data interchange would work successfully if only the whole world agreed on a single schema. For data interchange to work successfully, you need flexibility, the ability to create local variations and enhanced versions, so that the standard does not inhibit the ability of two parties to communicate what they need to communicate. And as soon as you have versions and variants, you need transformation capability.

4. WHERE ARE WE NOW?

At present we seem to be at a stage where the XML standards family is acquiring layers of complexity: a schema language whose specification is impenetrable to mere mortals, a query language whose specification is split into seven separate documents, a vast repertoire of supposedly general-purpose features such as XML Include, XML Base, XML Pointer, and XML Link whose importance it is difficult for users to assess, version 1.1 specifications for XML and XML Namespaces that create incompatibilities without offering any new features that the average user will recognize as something they need, and a layer of "semantic web" specifications that seem at times to live in the world of metaphysics rather than information technology.

Sadly, this is the price we pay for success. When a simple technology like XML becomes widely adopted, lots of people jump on the bandwagon, and decide to use it as a vehicle for their extensive technological ambitious; and the more people who are involved, the more complex the specifications become. Not all of the new raft of specifications will succeed, but I don't claim to be able to predict which will flourish and which will wither.

In terms of exploitation of XML, there are two key areas that until now have hardly been touched by XML, namely the traditional "corporate database" and the office desktop. These appear to be locked into a time-warp: the 1970s relational database with its rigid rows and columns, and the 1980s era of personal (that is, undisciplined) computing. Most of the information held by the average enterprise is either locked into a rigid structure modelled on 19th century ledger books, or it is held in ad-hoc whimsical documents and spreadsheets understood only by the person who created them.

We need to ask whether the new layers of standards actually have anything to contribute to solving this problem.

This is a big question, and I will concentrate on just one part of it: the question of the XML database, which happens to be a rather central part of my own company's product line.

5. THE XML DATABASE

One of the things I find fascinating about XML is the fact that it spans the spectrum from highly-structured information to information with no structure at all, and it is the first technology that attempts to do so. Before XML, document technology and data technology were different worlds. The need to integrate them became apparent with the web: if you are trying to sell holidays, or books, or hotels, or pensions, then the web acts both as your product catalogue and as your transaction processing system, and they cannot be kept apart.

The requirement for a database technology that can handle this full spectrum is therefore fairly obvious, but the goal remains elusive. With XML Schema [2] as a data description language (and despite its faults, people are using it successfully) and

XQuery [3] as a query language, the basic technology components are coming into place. There is a furious spat, of course, between vendors like Oracle and Microsoft who will tell you that you need a relational database with XML extensions, and vendors like Software AG who will tell you that you need a native XML database engineered to handle XML from the ground up, but that quarrel is really just a sideshow. While the vendors are exploring the best way to build a database that holds XML, the real challenge for the industry is to learn how to use the technology when it becomes available. The answers aren't obvious.

Really there are two competing approaches. One is to treat an XML database as a filing cabinet in which you store XML documents. The documents are designed as just that: messages exchanged between people or applications designed to convey information from the sender to the recipient. Storing the document is a way of getting a historical archive, a way of enabling the XML documents to be found easily, and a way of getting aggregate and summary information by analysing the documents.

The second approach is to design the database as a store of information, not just a store of documents. Here you start with the traditional information management discipline of data modelling, asking yourself what entities, attributes, and relationships exist in the space that you want to hold information about, and then designing representations of this information appropriate to an XML solution.

Looking at it from the viewpoint of the data user, the second approach has much to commend it. The user wants to ask questions about the state of the world (How many tons of tomatoes were exported last year from Uruguay?) not questions about the messages exchanged between people or applications that might have a bearing on the subject (Find me all the bills of lading for ships leaving Uruguay with a cargo of tomatoes). But the practical difficulties are immense. The point about XML is that it covers the spectrum from highly-structured to highly-unstructured information. Unstructured information has a great deal of value, and if you try to model everything, you are imposing structure, and thereby losing the ability to hold information that doesn't fit the model.

So I think that XML databases really have a very exciting future in the next five years of XML exploitation; I think they have real potential to combine the benefits and remove the limitations of the current rigid relational database and uncontrolled desktop applications, which today inhabit different worlds. But we are only at the beginning of the road in terms of learning how to exploit this potential.

6. REFERENCES

[1] Extensible Markup Language (XML) 1.0: http://www.w3.org/TR/1998/REC-xml-19980210

[2] XML Schema Part 1: Structures: http://www.w3.org/TR/xmlschema-1/

[3] XQuery 1.0: An XML Query Language: http://www.w3.org/TR/xquery/