XML Publishing in a Post-Standards World

Geoff Nolan
Senior Systems Engineer
Turn-Key Systems

The early days of XML saw a great deal of effort expended in the development of standards. These included not only XML itself, but related standards such as XPath, XSLT, XSL-FO, MathML, Schemas and a host of others. As these standards mature, they become less relevant for the end user. This is not because they cease to be valuable. Many standards become universally accepted and incorporated into software tools. After all, who nowadays hand codes their HTML? Almost no-one needs to, because all web browsers and authoring tools follow the standard as a matter of course.

The XML standard itself has now achieved a similar universal status, but whether to adopt associated standards such as XML Schemas or XSL-FO is a more difficult question. XML is now approaching a level of maturity where the key issues in publishing are no longer which standards to support, but which tools will provide the required functionality most effectively and economically.

Standards — What and Why

In the beginning (ie. around 1995) there was SGML, itself an ISO standard, and its semi-legitimate offspring HTML. Between them, these two illustrate many of the advantages and drawbacks of the things we call standards.

For our purposes a standard can be defined as a set of rules and/or procedures which, when properly followed, confer some advantage on the user. Typically, standards are proposed by bodies which themselves have sufficient prestige and influence that their recommendations are likely to be followed. Such bodies include:

- **ISO** (the International Standards Organisation) — approved the original SGML standard in 1986, and is in the process of ratifying a number of subsidiary standards such as Relax NG and Schematron.
- **W3C** (the World Wide Web Consortium) — the premier XML standards body has provided most of the key standards such as XML itself, XSLT/FO, XPath etc.
- **OASIS** (Organization for the Advancement of Structured Information Standards) — has produced a large number of standards in specific areas such as e-Business and document management (DocBook).

However, some standards have arisen around a single tool or concept. One example is Schematron XML verification.

In choosing whether to adopt a standard, a company (or an industry) must consider a number of issues:

- **Applicability** — Is the standard relevant to my business?
- **Utility** — Does the standard provide some tangible benefit?
- **Complexity** — What is the likely cost of adopting the standard?
- **Acceptability** — Is the standard universally (or at least widely) accepted?
- **Consistency** — Is the standard supported in a consistent way by all users?

Let's examine SGML and HTML in this light. SGML was the first significant vendor-neutral mechanism for coding, validating and exchanging structured data. As such it had the potential to provide enormous benefits to its users. It also scored high marks for consistency, inasmuch as a vast range of markup styles were all consistent with the standard.

However, this enormous flexibility eventually led to its decline. The cost of designing software tools to handle all its options was far too high, and SGML data from one source was often completely incompatible with (equally valid) data from another source. As a result SGML use was largely confined to government and major corporations where the benefits were sufficient to offset the high costs involved.

HTML on the other hand began life as a bare bones screen-based publishing standard. While notionally it was just SGML conforming to a predefined simple DTD, its very simplicity made it ideal for early web development. Quite rapidly however software manufacturers took to padding out its limited facilities with custom add-ons. Consistency problems plagued HTML through the mid to late 1990s.

Nevertheless its near universal adoption in an explosively growing market made HTML the most successful standard in structured data. And in recent years its consistency has greatly improved (at the cost of some extra complexity) as the standard has matured.

The main weakness of HTML is in the area of applicability. While well suited to Web pages, there is a vast quantity of structured data for which the limited tagset of HTML is quite inappropriate. It was this need which drove key figures in the SGML world to opt for a simpler but far more powerful standard, XML.

Standards for the Publisher

We'll take the XML standard itself as a given (otherwise you wouldn't be reading this). XML addresses the shortcomings of SGML by using a much more tightly specified syntax, and eliminating functionality which add complexity for little advantage (INCLUDE, CONREF and so on). XML's subsequent success in largely taking over the SGML market while carving out vast new areas of application (eg. e-Business) is testimony to the effectiveness of its initial design.

At least as important as XML itself is the host of subsidiary standards which have grown up around it. These extend the functionality and applicability of XML without raising the complexity of the standard, as we are free to adopt only those additional standards which are directly useful to us.

From a publishing perspective there are several classes of potentially useful standards:

- **Specification** — XML, Unicode, XInclude, Namespaces, Catalogs
- **Navigation** — XPath, XPointer, XLink, DOM
- **Validation** — DTD, W3C Schema, Relax NG, Schematron
- **Processing** — XSLT, DOM, SAX
- **Applications** — XHTML, DocBook, MathML etc.
- **Formatting** — XSL-FO, CSS, SVG, RTF!!!

Of course a number of these standards apply over several categories. MathML for example has definite processing and formatting aspects.

Given the huge number of standards available, why does this paper refer to a “Post-Standards World”. This theme will be taken up in the following section.

The Decline (but not necessarily Fall) of Standards

In the early days of HTML, debate raged hot over what functionality should be supported and how. As the body of HTML documents grew into the billions, and consistent browser support became essential, the HTML standard settled down. It was no longer acceptable for applications to add non-standard functionality, and the battle moved on to areas like CSS and SVG support. This is the natural evolutionary path of a successful standard.

An alternative fate lay in store for SGML. Already burdened by its own complexity, it was further disadvantaged by not being able to fully partake of the wealth of XML standards or the resulting software tools. As a result SGML continues to be used mainly for legacy systems, and even there the non-XML features have fallen largely into disuse. So while SGML still exists, very few applications these days would consider SGML conformance an important issue.

Thus both SGML and HTML are these days seldom considered in application design or selection. HTML is so universally accepted that every application takes it for granted, while SGML has been largely superseded.

The XML standard has itself undergone a similar fate. Changes to the the definitions of XML itself are made rarely and only to meet some critical requirement (such as Unicode support). In other areas, particularly as regards schemas/validation, the battle still rages. But for the modern XML Publisher, there are only a handful of essential standards:

- **XML** (including namespaces where necessary).
- **Unicode** — absolutely necessary for both authoring and publishing tools, unless you only ever deal with legacy documents. It is not yet mandatory to use encodings like UTF-8 in your own data, but the advantages of doing so are considerable (even if you only use the occasional ∑ or smiley face ☺).
- **DTD/W3Schema/RelaxNG/Schematron** (validation is important, the exact method isn't).

This is not to say that the other standards cannot be extremely valuable *for specific applications or document classes*, but except for the above a publisher should use a tools-based rather than a standards-based approach when setting up a production system.

Standards-based Tools

An XML publishing system will consist of a number of components, typically including those discussed below.

Author/Editing

A number of good tools exist. These include XMetaL, XML Spy, and Arbortext's Epic. Alternatively, an authoring application may be built into a Content Management or Publishing system. While the selection of an authoring tool is not trivial, it will generally be based on criteria other than standards support.

Storage

Once again, document storage requirements are not by and large overtly standards dependent. Facilities such as boilerplating, version control and collaborative authoring will determine whether the best fit is a million dollar CMS or a simple file system.

Conversion Tools

While conversion from legacy data *into* XML is rarely concerned with standards, conversion *from* XML to HTML, XSL-FO etc is a different matter.

The primary standards here are DOM, SAX and XSLT. When producing XSL-FO, the choice of an XSLT based tool/program is almost automatic. Indeed the original XSL working group tried to develop a single conversion/formatting standard, and XSLT/FO is often spoken of as a single unit. More on this below.

For other output types (HTML, RTF etc) any of the three (or none) can be used. SAX is a simpler and faster, but less powerful protocol. DOM and XSLT take a similar approach, but DOM (Document Object Model) is basically aimed at programmers whereas XSLT is a means of specifying transformations in an implementation independent XML syntax.

But using any of the above methods requires considerable programming/design skills. Producing a non-trivial XSLT script by hand can be a brain-breaking exercise even for the expert. Of course we could devise tools to help us produce such scripts, but this reintroduces the proprietary factors that the standards are supposed to have removed.

Composition/Rendering

This is the area in which the standards issue becomes murkiest. If we are simply displaying XML on screen then CSS is a possibility. It is simple enough for non-specialists to use, especially when a CSS development tool is supplied (as is the case with many authoring systems).

However, CSS has two severe limitations. Firstly its support for paginated output (PDF/paper) is quite limited, though this is being addressed to an extent in CSS-3. Second, it has almost no capacity for rearranging the output. If you want (say) notes printed at the end of the current chapter then you must do a pre-transform of the data — so it's back to XSLT and friends.

The other alternative in the standards arena is XSLT/FO. The XSLT component allows arbitrarily complex rearrangement of the data, while XSL-FO specifies a set of formatting objects which support very sophisticated page layout. But the setup and maintenance costs are very high. Which begs the question, what are the costs/benefits of a standards based approach.

Rendering — What Price Standards?

All XML rendering is done via tools. Some, such as FOP/RenderX, are entirely standards based. Some, like 3B2, can work with or without standards. Others, like TopLeaf (the Turn-Key composition engine) are not based on any recognised standard. Each approach has its costs and benefits.

The total cost of a rendering package is a combination of purchase price and stylesheet setup/maintenance costs. The following table indicates these costs for each alternative, along with the degree of interoperability (ie. how easily stylesheet components can be utilised by other rendering tools).

Product	Purchase Price	Stylesheet Setup / Maintenance	Interoperability
RenderX	Low	High	High
3B2	High	Variable	Variable
TopLeaf	Low	Low	Low

As you can see, 3B2 supports all alternatives, but the cost of this flexibility and its numerous other features is a very high purchase price. RenderX and TopLeaf each support only one publishing method and are correspondingly cheaper.

A second fact which stands out is that Setup/Maintenance costs mirror Interoperability. This is due to the fact that XSLT/FO rendering is an inherently costly operation because:

- You need an XSL-FO expert to specify the formatting objects required for your desired output format.
- You need an XSLT expert to prepare the conversion scripts from your own XML markup to the FO format.
- Even the simplest output requires a radical conversion of your entire document. This means there's a large "flag-fall" to pay before seeing any output at all.

TopLeaf has been developed in-house over many years in response to customer requirements. It's a bit like the architect who designed a university campus without footpaths. A year later he came back and paved over the areas of grass which had been worn flat by passing feet. In both cases development is in response to real world requirements.

To be sure this has meant that there are some typesetting features supported by FO which cannot (currently) be matched by TopLeaf. For example, FO allows arbitrary nesting of boxes. TopLeaf only allows a single level of box-within-box, mainly because no-one has ever asked for more. If the need did arise, the feature would be added.

The upside is that TopLeaf styles are entirely controlled by a simple GUI. All options are given reasonable defaults, so it's quite possible to have a very nice looking output document (such as the one you're now reading) after an hour or so. And no special expertise (programming or typesetting) is required. Standard production editors can, and do, use the system to produce sophisticated effects after a few hours training. Output to paper, PDF, RTF and HTML are all supplied from the same stylesheet, and not a standard in sight.

The downside of course is you can't take your TopLeaf styles and plug them into 3B2 or RenderX. *But this would be the case even if TopLeaf used XSLT and XSL-FO behind the scenes!* The reason is simple. In order to allow styling via a GUI interface, we would have to restrict ourselves to particular subsets of both XSLT and XSL-FO. If a target application had similar tools to simplify stylesheeting, it would not recognise the particular idiom used by TopLeaf.

The conclusion is that full interoperability is only possible if the stylesheets being interchanged are manipulated directly by human experts. Any attempt to introduce tools to simplify the stylesheeting process negates the universal non-proprietary nature of the underlying standard. The wide use of (non-standard) extension functions in XSLT conversions is another indication that true interoperability is not so easy to achieve in practice.

This is not to say that systems such as RenderX or 3B2 should not be considered. They each have powerful features that would make their selection a no-brainer for the right user. It's just that adherence to XSLT/FO should not *of itself* be a major consideration in product choice. As I say, it's a post-standards world out there.

Conclusion

I'd like to emphasise that the point of this paper is not to minimise the importance of standards, nor to class them as obsolete.

Standards are more like scaffolding which lends structure and support to a building under construction. But as the building nears completion the support is built into the structure of the building itself, and the unwieldy tubing on the outside can be safely removed.

In the same way, as XML matures, an *overt* reliance on standards becomes less necessary. The software tools we use to manipulate our documents either have standard functionality built in, or provide an appropriate equivalent.

After all, both standards and tools are only a means to an end, namely the timely and profitable publication of documents. And *that* should be the primary consideration for any XML publisher.