Closing Keynote, XML 2006
Jon Bosak, Sun Microsystems
Boston, 7 December 2006

Good evening, friends and fellow conference attendees.

When David Megginson asked me to give you this closing keynote tonight, I think he knew that I couldn't refuse an opportunity to come back to the same conference, in the same hotel, at which I had the honor of introducing the SGML community to a product initially known as "SGML for the Web" but by then repackaged under the name "XML."

It's hard to remember now just how different this community was ten years ago. There were only a few hundred SGML experts in the world, a goodly number of whom came to this conference every year. Very few groups in existence today could boast the level of intelligence, the breadth of interest, and the depth of independence, not to say downright weirdness, evidenced by the SGML community of a decade past.

In my role as master of reminiscence this evening, I've been trying to figure out how to give those of you who weren't part of that early SGML community some sense of what it was like to work within it. I'm going to give you one extended example that sticks in my memory as emblematic of that period and its people.

To understand this anecdote, you're going to have to absorb two pieces of context.

The first piece is the text of a very famous seventeenth-century poem by Ben Jonson titled "To Celia," which many of us know set to music under the name "Drink to Me Only with Thine Eyes." Here's the text of the original poem for those of us who need a refresher:

> Drink to me only with thine eyes,
> And I will pledge with mine;
> Or leave a kiss but in the cup
> And I'll not look for wine.
> The thirst that from the soul doth rise
> Doth ask a drink divine;
> But might I of Jove's nectar sup,
> I would not change for thine.
>
> I sent thee late a rosy wreath,
> Not so much honouring thee
> As giving it a hope that there
> It could not wither'd be;
> But thou thereon didst only breathe,
> And sent'st it back to me;
> Since when it grows, and smells, I swear,
> Not of itself but thee!

You'll see why I'm reminding you of this in a moment.

The second piece of context, which in itself demonstrates how different things were ten years ago,

was the issue that was raging in the extended group of SGML experts that was known as the W3C SGML Working Group. That issue was: Does every XML document have a DTD? Notice that I didn't say: *Can* every XML document have a DTD (because that was never in question); the question was, *Does* every XML document have a DTD.

By DTD we meant, of course, the traditional notation used in SGML to specify the legal structure and vocabulary of a document. This is what came generically to be called a schema, and probably most of you know that DTDs are still part of XML, even though most people today implement the function of DTDs using one of the constraint languages that are themselves expressed in XML, such as W3C Schema or Relax NG, rather than something that uses its own special syntax.

The debate on this subject, which grew quite intense, hinged on the concept of validity. In the SGML world, documents were of two kinds: valid and invalid. Valid was good. Invalid was bad.

This made a certain kind of logical sense if the DTD was considered part of the document, which technically it was. But very few people actually used DTDs that way, and very few people thought about them that way; in practice, a DTD was something that stood outside individual documents and could be applied to all documents of a given type.

But still people clung to the idea that a DTD had to be associated with every document, because a document could only be valid if it had a DTD, and validity was good. Furthermore, it was possible to change the DTD in a way that gave you a different information set even though the document in question still validated. And there were some other peculiarly SGMLish reasons for requiring a DTD, such as the fact that a DTD was needed to tell you what the delimiters were and which elements had to have end tags. So the idea of a very tight association between a specific document and a specific DTD was ingrained in SGML consciousness at that point. It was not until XML started being widely deployed that people began to realize that "validity" is not a category that can properly be predicated of a document, if a document is considered to be something separate from a DTD, which everyone had long acknowledged in practice to be the case. In practice, validity is something that's predicated of the relationship between two things, namely a specific document and a specific DTD with which it has been associated somehow.

This was true even in SGML, notwithstanding the fact that the DTD was technically part of the document. So in reality, it was never conceptually (as opposed to technically) correct to say that a document was valid in an absolute sense; you could only say that it validated against this DTD rather than that DTD. In other words, the document was sometimes valid and sometimes not, depending on which DTD you tried to associate it with.

Once considered fairly, this thought, while deeply upsetting to many SGML experts, led to the observation that there *were* actually two kinds of SGML documents: ones that could be valid against *some* DTD and ones that could never be valid against *any* DTD. I don't recall that there was an official name for this second kind, though perhaps there was one somewhere in the SGML standard; perhaps the most common term was "broken."

As fate would have it, a number of people working with very early pre-world-wide-web SGML-based online publishing systems had already discovered a very useful aspect of this way of looking at validity. These were people who, like me, were responsible for implementing publishing systems based on early versions of DynaText, a product of Electronic Book Technologies.

The story, as I remember it, is that Steve DeRose, the engineer responsible for the original EBT

code, didn't have time to implement a full SGML parser in the engine that compiled SGML input to produce the indexed and structured files used at run time — so in that first release, he just left out the part that validated the document against the DTD. The makebook program accepted the DTD, and in versions later than the first it actually *did* check the DTD, but in that first release it just didn't bother with that kind of validation at all. What that first version did check was a certain kind of structural consistency that was the unnamed constituent of every SGML document that *could* be validated against a DTD. This suchness that inhered in every validatable document was what we later called "wellformedness."

Those of us who worked with the early EBT products discovered that the lack of a true SGML validator didn't make a damn bit of difference if the document had already been validated by a true SGML parser further up the line. And the reason it didn't matter was that the SGML parser had incidentally found the document to be what we later called "well formed" in the process of validating it.

So without thinking about it very hard, or in the terms that I've just laid out for you, some of us working in the trenches had gotten used to the thought that DTDs were something separate from documents and that documents had a goodness or badness independent of any particular DTD.

To the SGML theorists, however, this decoupling of document from DTD was anathema. And no one struggled harder against it than Charles Goldfarb, the IBM computer scientist who is rightly called "the father of SGML" and who would be called "the father of XML" as well if history had gone just a little bit differently. (As I'm sure most of you know, XML did not invent anything; it was basically a subsetting and repackaging of the SGML standard, ISO 8879.)

Charles knew everything there was to know about SGML, and for various technical and theoretical reasons, he found the concept of an SGML document without a DTD completely unacceptable. Realizing, however, that the people working on XML were determined to allow documents without DTDs explicitly attached, he finally was forced to adopt the position that every well-formed document *does* have a DTD, *even if you can't see it.*

The controversy on this point (and there were several issues that aroused equal passion) reached a peak when Charles posted a message that read in part as follows:

> Questions like this are more easily analyzed if we avoid the confusing term "optional DTD". A missing DTD is really an implied DTD (like SGML's implied SGML Declaration) in which all element types have mixed content consisting of "(#pcdata | any element type)*" and all attributes are CDATA #REQUIRED (except for special conventions for ID, etc. that we might adopt).

To which I responded:

> There is no "implied DTD" of this nature in XML. It is possible for a well-formed XML document to be parsed in the absence of a DTD. In such a case, the DTD does not maintain a noumenal existence; it just really isn't there at all.

I was referring, of course, to the Kantian distinction between noumena and phenomena.

Charles answered with a typically careful and detailed analysis that ended, in part, with the following remarkable statements:

> Postulating, as has been done on this list, that the "style sheet" will know what to do with unspecified attributes or references to undeclared entities JUST MEANS THAT THE DTD IS IN THE STYLE SHEET rather than in the standardized declarations used in SGML (and XML).
>
> I believe it would be useful (perhaps essential) for an XML document to be parsable without *reference* to its DTD, or with reference only to the internal subset, or both. But that is very different from saying that the DTD "just really isn't there at all".
>
> There is always a DTD, it is a law of nature. The issue is whether the DTD is represented by standardized declarations, or in a proprietary style sheet, or (as with HTML "extensions"), buried in the browser code. For XML, a standardized representation of DTDs is the only way to go.

To which I could not resist replying with a message that ended as follows:

> If what you are saying is that for every document, you can imagine a DTD to which that document conforms, then you are understating the case. For any document, one can imagine an infinite number of DTDs to which it conforms. You are free to imagine as many of these DTDs as you like. That still doesn't make any of them part of the XML specification.
>
> Even if we grant for purposes of argument that the infinite number of DTDs to which any document is conformant have some kind of existence (in the mind of God, say), their ontological status is equal; there is nothing to distinguish any one of this infinitude as more real than the rest of them.
>
> It may be that there is a practical purpose to be served by inventing the idea of an implied DTD and specifying rules whereby it is to be constructed from the document, but you can't make such a thing pop out of the void by saying that nature will provide it every time a document is created.

Lee Quin, who's been here at the conference this week, was moved to contribute the following:

> ...if you examine the byte stream after the structovator has run & made a DTD, you may well have valid SGML. A philosopher might opine that
>
> For every DTD-less XML file
> There existeth a Document Divine,
> Which conformeth even to the uttermost mile
> Of Iso Eight Eight Seven Nine.
>
> Well it doesn't scan very well, but it parses :-)

I hope you're starting to get a sense of what it was like to supervise a project involving these people.

Now you have the minimum necessary background for understanding a message I received from Michael Sperberg-McQueen, one of the XML editors, a few days before the SGML conference here in Boston at which the first XML draft was released.

This message, sent at about 2 a.m., listed a series of issues that we wanted to settle in the conference call that was to take place the next morning. I'm not going to read you Michael's typically thorough exposition of those issues, but to give you an idea of the scope of the discussion at that point, I'll note that the points we were trying to resolve a few days before the conference

included the following:

> Determinism in element content models
>
> Behavior of the XML processor when encoding is not recognized
>
> Whether on not processors should be required to notify applications of entity boundaries or merely allowed to do so
>
> Whether a rule we had just adopted for white space processing was required of all XML parsers, or just validating parsers
>
> The syntax of the doctype declaration
>
> The list of registered IANA charset values that could appear in the encoding section

This set of agenda items was not atypical of our weekly conference calls.

In a previous message to the group, Michael had said

> Ever since Lee's posting with the bit of verse speaking of documents divine, I can't get the tune of "Drink to me only with thine eyes" out of my head.

And in the late-night message I'm referring to, he finally offloaded this conceit with the following contribution to English literature:

> *The Validating Client to the File Server:*
>
> Serve to me only docs with tags
> And I will parse just fine.
> Or leave a declaration there
> For entities condign.
> The numinous Doc Type I infer
> may match a Type Divine.
> But might I see thy Real DTD
> I'd rather parse that, than mine.
>
> *The File Server's Reply:*
>
> I sent thee late a well-formed doc
> Not so much hon'ring thee
> As giving it a hope that thou
> Would'st check its validity.
> But thou thereat didst only sniff
> And send'st it back to me,
> Since when it weeps and cries "If you loved me
> You'd give me a real DTD."
>
> With apologies to Ben Jonson, to Julia Flanders (the originator of the Empty Element's Lament: if you really loved me, you'd give me content), and to you, for trying your patience. You must come by and try mine sometime.

Now what's my point with all this? Partly a reflection on the past and partly a reflection on some differences between then and now.

Here's what I got up and showed you all that day ten years ago when we introduced the first draft of XML — 27 pages produced in about eleven weeks of committee work, first meeting to first draft.

What's remarkable about this artifact today is the way it was physically produced and what that says about the people who produced it.

The draft was authored by its two editors, Tim Bray and Michael Sperberg-McQueen, using an XML markup invented for this one document.

The RTF file from which the copy was printed was generated by a DSSSL RTF formatting engine written by the technical lead of the XML committee, James Clark. (For those who don't go back that far: DSSSL was a scheme-based language architected by James as what in retrospect can be seen as a warmup exercise for XSLT and XSL-FO.)

James's DSSSL engine was driven by a DSSSL stylesheet designed by the chair of the committee (yours truly) for this one document.

And the same XML document was also rendered into a richly annotated and hyperlinked online version using a perl-based HTML formatter scripted by one of the document's editors, Tim Bray.

So in other words, not only was no proprietary software used in creating the online and printed versions of this publication (with the exception of what we used to print out the generated RTF), but several essential pieces of the infrastructure that produced it were created especially for this one document.

What's really notable about all this is that the process I've just described was one that the people involved took for granted. Yes, they knew they had to demonstrate to their audience that the stuff they were peddling that day actually worked, so of course all that effort had a purpose. But the whole process of design was one that everyone involved had engaged in a number of times on other projects. And as a result, this was a group that did not take the limitations of any one piece of software as carved in stone and certainly wasn't going to be constrained by the limitations of anybody's proprietary software. I'll get back to that point in a bit.

If you weren't there at the time, which was November 1996, you might be forgiven for thinking that what we're engaged in tonight is a formality observing the occasion of another formality. I mean, how exciting can the release of a draft standard be? But the fact is, we didn't know the day that XML was introduced here in Boston whether it would be greeted warmly or shouted out of the hall.

In anticipation of a rocky reception, we came up with a couple of ideas for how to present this radically stripped down version of SGML to the SGML community that I think helped carry the day.

To start with, we introduced the members of the W3C SGML Editorial Review Board by bringing the perps up on stage, one by one, so that the assembled SGML experts would be forced to recognize each participant in the ERB as a well-known member of their own community. This was followed by a brilliant piece of theater in which Tim Bray strode to the edge of the stage and tossed a copy of this slim little spec into the audience and said as it fluttered down, "If that had been a copy of the SGML Handbook, it would have taken out the first couple of rows."

Since XML basically represented what most sensible SGML workers were already doing, selling the XML concept wasn't all that hard once people were persuaded to take a look at it. But I'd like to observe a couple of things about the people involved in that occasion.

The first thing I'd like to observe is that the group that created the initial draft of XML succeeded because it wasn't afraid to think independently and try something new. And the second thing I'd like to observe is that the group to which the idea was presented wasn't afraid to be critical of it. Nobody involved was the least bit inclined to be overawed by anyone else, including the software vendors and the SGML installed base. Everyone was mindful of creating as little disruption as possible, but no one was afraid of it.

How far we've drifted from the attitudes that created XML was demonstrated to me recently in the revision to the OASIS UBL Standard for business documents, which is the product of the OASIS technical committee I chair. We had a big technical problem to solve for UBL 2.0, which by the way should be announced as an OASIS Standard in the next week or so, and that problem was how to handle code lists. You all know code lists — those collections of little two- and three-letter codes like "FR" and "EUR" that stand for the names of countries and currencies and so on. In documents designed to support cross-border trade, there are a lot of codes.

In UBL 1.0, which uses W3C Schema as its only constraint language, the standard code lists were all translated into lists of enumerations that are incorporated into the schema at run time. In practice, we found several problems with this approach.

To begin with, there's no standard format for publishing code lists. So right there you've got a big maintenance problem as these things are updated by the various agencies that publish the standard versions of these code sets. This frequently involves a lot of handwork to get the revised lists into whatever form they're referenced by the software.

Then you've got problems inherent in using XSD enumerations. It's very common to want to subset or extend standard code lists to fit specific trading partner relationships. You can't do that if the code lists are implemented as schema fragments without eroding the concept of a standard schema, and of course you've got to fudge the namespaces so that the rest of the schema space doesn't know what you've done to the original lists.

But the problems don't end there. The reason for modifying a code list is often to make the schema validator perform simple business logic by filtering based on code acceptance. For example, a business in the U.S. might want to restrict its country subentity list to just those codes representing U.S. states, thus enforcing an implicit business rule that it's only going to accept purchase orders from inside the U.S. This is actually a potentially very powerful declarative way of stepping around some procedural business logic.

Pretty soon, though, you want to start applying different subsets of the code list at different places in the document. To take an example from the UBL 2.0 spec, a business in Canada might agree with a business in the United States to use a set of code lists that allow the Buyer to be associated with either a U.S. state or a Canadian province but restrict the Seller to just U.S. states — that is, to apply a code list subset containing state and province codes in one place in a document instance and a different code list subset containing just state codes in another place in the instance. This kind of thing is basically impossible to achieve using the XSD enumeration construct.

After four years of futile efforts to resolve these problems, Ken Holman (who among his other roles

is one of the UBL editors) came up with a solution: use a single XML format, namely the genericode format for code lists developed by Tony Coates and presented at this conference in 2004, and compile the code values represented in various genericode files into a single XSLT script that would drive an XSLT engine to check all the code values in an instance.

In other words, you treat instance validation as a two-step process: first you check the instance using the standard UBL schema for the given doc type, and then you check it again against the standard XSLT file that has the code list values. The first validation phase makes sure that the structure, vocabulary, and data typing of the instance is correct, and then the second one makes sure that all the code values are correct. This not only solves all of the problems we started with, it also helps deal with the related issue of schema customization simply by making a lot of schema customization unnecessary.

But the benefits don't stop there. It turns out that once you've got that second validation phase, you've also got a mechanism for implementing a layer of declarative business rule checking, because you can use Schematron to produce your own version of the stock XSLT file that drives the second part of the process. So you're not just saying things like "the buyer has to come from one of these countries"; you're also saying things like "if the buyer comes from France, then the price has to be in euros," or, "if the value is in yen, then the amount has to be greater than 100." And you get this fabulous amount of flexibility without changing the standard schemas, so the whole question of what it means for a business document to conform to a standard schema becomes much clearer: the instance is standard UBL if it validates against the standard UBL schema for that document type — period. The rules about what values that instance can convey are a separate issue, and we give you a mechanism for taking care of that.

In the process of developing this solution, we achieved what I view as a real conceptual breakthrough: we came to the realization that schemas are good for checking structure and vocabulary and data typing, but they are not the right tool for checking instance data. To do that, you need to add another tool — XSLT.

Well, I don't mind telling you that I was really pleased about this development. It seemed to me that Ken and the rest of the UBL TC had solved a huge problem not just for UBL but for a number of other business-oriented XML standards as well. So imagine my surprise when this approach was not greeted with the hosannas of praise I had been expecting but rather with the objection that software vendors could not implement it. Why not? I asked. And the reply came back: Because XSD validation is built into the existing business software, and your solution would require them to add something else.

It didn't matter that what we were adding was the implementation of a W3C recommendation with the same status as W3C Schema and the XML specification itself. It didn't matter that XSLT was a pretty well established technology that didn't require vendors to adopt something new like the SchemaPath proposal we heard about this morning from Paolo Marinelli.

Our code list solution was out of bounds because solving this enormous and critical set of problems would require the vendors of business software to add something to their products.

This news came as a big shock to me. I thought the way it worked was that we customers would tell software vendors what we wanted and then the software vendors would include what we wanted in their products. I thought that's what we were paying them for.

But no. Now that we've got business systems that support XML and XSD, that's the end of the line. Adding XSLT support was going too far.

Furthermore (I was told), business users will never adopt a solution that depends on an additional XSLT pass because it would require them to learn something new (never mind that this "new" thing had been widely employed in other contexts for years).

What I said then, and what I'll say now, is that if we had adopted that attitude ten years ago, there wouldn't be any XML.

But you know, I'm still the kind of person that starts looking for an open-source solution if the commercial vendors don't have the wit or the nerve to provide the capabilities I'm looking for. So I asked Ken to implement our new model using freely available open-source software, and after I dumbed down Ken's shell and batch scripts to the point where even I could understand what was going on, we cooked the whole thing right into the UBL 2.0 specification.

We included the runtime module from xerces to do the XSD part, and Norm Walsh's xjparse driver to feed xerces the files, and saxon to do the XSLT part, and some scripts to automate the two-part process, and we put everything in a folder in the UBL 2.0 distribution. So now all you have to do if you've downloaded and unzipped the 2.0 package is to go to the val directory and type in "validate (path to instance) (path to schema)" and it does the whole thing for you. With no setup. At no cost. So if the business vendors find all this just too complicated for their programmers to handle, to hell with 'em.

I learned yesterday that at least two popular XML tools demonstrated here at the conference already have the capabilities needed to implement our code list methodology built right into them, so the established business software vendors may find themselves sitting ducks for more agile companies that aren't scared to go beyond what seems to have become the establishment picture of what constitutes XML.

The lesson that apparently needs to be learned in the data-centric XML world is this: stability and preservation have always been central to the SGML/XML concept, but it's the *data* we're trying to preserve, not the software we might be using at the moment to process it.

So that was how I found out that things had changed a lot out there in XML data land since I started focusing on ecommerce infrastructure six years ago. But the document aspect of the UBL spec itself turned out to be almost as problematic.

Somewhere along the line, for reasons that probably made sense when standards were unitary documents, OASIS, the organization we all know and love, adopted a rule that all specifications submitted for OASIS standardization have to be provided in PDF. Never mind that at this point, OASIS can proudly claim not one, but two standard document formats of its own: DocBook for format-independent publishing and ODF (which is now a published ISO/IEC standard) for word-processed documents; those are optional. The required format is PDF.

Consider for a moment what this means for the just-released UBL 2.0 package. It contains 124 megabytes of material, including:

35 JPEG files for the artwork

42 ASN.1 modules

94 genericode files for the code lists

94 XSD versions of those same files for people who want to use the old enumerated code list approach

A set of CSS stylesheets for viewing HTML

A set of DocBook XSL stylesheets for viewing DocBook

A PDF file containing our Naming and Design Rule checklist

32 .ods files containing the spreadsheets that define the UBL data model, including one spreadsheet thirty columns wide by more than one thousand rows deep that specifies our reusable data components

32 .xls files containing the Excel versions of those same spreadsheets

A set of 230 HTML and JPEG files presenting the entire UBL data model in an interactive form that lets the user freely navigate the model

A set of open-source software modules that lets the user run xerces, xjparse, and saxon together with scripts to provide the test harness for our two-step validation process

13 sample UBL documents, which of course are XML files

32 XSD document schemas that together define almost 2000 individually named elements and occupy almost three megabytes of space all by themselves

and

32 more schemas with all the comments stripped out to help with parser performance

The OASIS requirement is to publish all this in PDF.

Guess what: that doesn't work very well.

The way we actually publish the package is as a reasonably well-structured directory tree together with an HTML file that explains the contents and links all this stuff together. The original of this HTML file is a DocBook version from which the HTML is generated using OASIS XSL stylesheets designed by — you can probably guess who — Ken Holman. I'm pleased to report, by the way, that the DocBook original, which we include in the downloadable version of the package, is viewable in both Firefox and Internet Explorer, so you can browse the DocBook version directly, which I think is pretty cool. Yuri Rubinsky would have been pleased.

There's actually a name for the kind of stucture we adopted for publishing UBL: it's called a *hypertext*. The document by which the UBL spec is served out over the web is written in the Hypertext Markup Language and the mechanism by which it's transmitted over the web is called the Hypertext Transfer Protocol.

Our experience trying to publish UBL (and I'm not going to tell you how we solved the PDF problem — you'll have to download the spec and see for yourself) made me realize that in addition to losing a certain spirit of innovation, the larger XML world seems in its rush toward commercial success over the last decade to have forgotten something else as well.

There's a reason that XML was originally called "SGML for the web." A large part of the

motivation for creating XML was to further the vision of pioneers like Doug Engelbart, Ted Nelson, Tim Berners-Lee, and Yuri Rubinsky. Somewhere in attempting to realize this vision we've gotten hung up in implementing the very first step along the way. We've wandered off into the weeds of commercialization and forgotten that the web we've got is the most primitive form of hypertext that could be imagined — which is why it works, and I don't want to deny that. But this focus on the money to be made right at the start has led us into an explosion of XML applications that focus purely on the exchange of data between computer systems. We've lost track of the human aspect of this to the point where even an organization whose very purpose is the advancement of XML considers it unsuitable for human consumption and requires its specifications to be issued in forms tied to the printed page.

The application of XQuery to web publishing applications described by several speakers at this week may signal the beginning of a trend back toward the human-oriented dimension of XML. I certainly hope so.

But while much has changed in the ten years since we introduced XML to this community, an equally surprising amount has stayed the same. Listening to the presentations earlier in the conference — and of course I wasn't able to attend more than a fraction of them — I was struck by how many of the issues troubling today's XML adopters are exactly the same ones we faced in the SGML days. For example, the choice between creating XML documents using unstructured style-driven word processors and creating them using native XML tools. As far as I can tell, our understanding of this problem has not changed an iota in the last ten years.

The reason this hasn't changed is very simple: the issues involved have much more to do with people than with technology. The very first presentation I ever gave on a technical subject was at the 1992 TechDoc conference (run by the same organization responsible for this one), and the subject of that presentation was the horribly painful process by which Novell put its printed documentation online using SGML. We used the "style-driven authoring converted to SGML" approach. Our authoring tool was the old pre-SGML Framemaker, and our converter was Avalanche FastTag, but the picture would have been exactly the same had we been using OpenOffice and XSLT.

My message to the audience at TechDoc was: don't try to create SGML using unstructured authoring tools; use a native SGML editor. But subsequently I spent several years in the Tech Pubs department at Sun Microsystems getting a close-up look at the agony of training authors to use native SGML tools and finally realized that there *is* no easy answer to this problem. Nowadays I'm inclined to recommend the structured approach, not because I think it makes the overall job of implementation any easier, but simply because it allows the authors to blame the software rather than the poor schlemozzle who's been stuck with the job of style Nazi.

Another ancient subject that seems to be popping up again is the idea of modular document creation. This is one of those concepts that comes through about once a decade, seduces all the writing managers with the prospect of greater efficiency, takes over entire writing departments for a couple of years, and then falls out of favor as people finally realize that document reuse is not a solvable problem in document delivery but rather an intractable problem in document writing — which is, how to retain any sense of logical connection between pieces of information while writing as if your target audience consisted entirely of people afflicted with ADD.

I could go on at length about this, but instead I'll simply leave you with the observation that my personal love affair with modular documentation occurred in 1978 and that I haven't seen a thing since then that would change the conclusions I reached about it almost thirty years ago. This is not to say that I'm trying to discourage the technical writing community whence I came from their enthusiasm for the modular authoring technology *du jour,* since engagement in such efforts is virtually guaranteed to buy tech writers a few years in which they can act like software engineers and present themselves as engaged in cutting-edge informational technology development rather than plain old technical writing. That strategy has worked great for some of us.

In closing, I'm happy to report that among the things that haven't changed in the last ten years are some of the reasons I fell in with this community to begin with: a sense of respect for one another; a tradition of intellectual honesty; a culture that values the common good above the agenda of one's employer of the moment; and a belief in the power of open standards to free knowledge workers from the tyranny of proprietary software. As long as we stick to those principles, we'll continue to evolve toward the goal we set ourselves so many years ago.

Thank you, and good night.