Regulatory Publishing with XML: A Case Study

By John S. Barker,
Metaformix Information Systems Inc.
August 31, 2000

INTRODUCTION

XML gained its recent popularity because of its ability to represent database information in a compact, interchangeable form, which makes it ideal for web-based applications with database back-ends.

But XML is also suited to more traditional document production environments. There are, and will be for many years yet, paper and paper-like documents, and a need for their distribution and management. This in itself is not new, or even interesting. What becomes interesting, however, is how modern information interchange and storage techniques can coexist in a world of “different dimensions”: A4, B5, Letter, Ledger and so on.

The vast majority of today’s legal, legislative and regulatory documents continue to exist as paper. And it is also true that the processes behind the creation, management and distribution of these types of documents is ingrained in legislation and regulations. Those processes absolutely demand rigid controls designed to manage the traditional page. So the crux of the issue for XML in such environments is simply: What is a page? What does it mean?

Regulatory documents rely on pages and page management practices. XML is a structured information markup language. Structured information is one thing. Managing it when its destination is a page is quite another.

THE PROBLEM OF PAGES

I think we have to ask a generally philosophical question: What exactly *is* a page? It is not an information structure, nor is it a format structure. The answer, I think, is that the page is a medium on which information is carried and transmitted, much the same way that tape, CDs, or diskettes are media. If we look at the page as the information carrier, and separate it from issues of structure, then we can begin to see how XML might actually be used in page-oriented documentation processes.

A case in point regarding this whole issue is Bell Atlantic, which is a large, East-Coast US based telecommunications provider. In November of 1997, Bell Atlantic presented to Open Text Corporation a request for information on a new document management, publishing and distribution system for its ageing platform, which it required for the authoring, management and production of telecommunications tariffs.

(At the time, I was employed by Open Text as a Senior Project Manager in their Professional Services organization.)

A “Tariff” is a document approved by a regulatory body, such as the FCC in the United States or any of the state telecommunications regulatory authorities. Each state has such an authority, which controls the products, services and prices available within its jurisdiction. Tariff documents state what these products, services and prices are.

Whatever technologies were proposed as part of the solution, they had to meet the following key requirements:

1. **The document content had to be stored as XML compliant SGML**; up until then, Bell Atlantic, a regulated telecommunications company, managed its information in proprietary word processing formats. The business case for XML and SGML was that it would free future technology migrations from having to proceed through a tedious and error-prone data conversion process. Because XML was an emerging standard, Bell Atlantic did not want to adhere to an implementation which would later limit its use in an XML application.
2. **The documents had to be distributed through both newer technology means as well as traditional paper means**; and, as part of this requirement, the presentation of each had to match. That is, the electronic form of the document had to match the paper form. The information consumers were state and federal regulators who not only required certain strict rules of page formatting and management to be applied, but who in fact required this because of state and federal legislation which made it the law!
3. **The document publishing process had to be capable of generating and managing the information as loose leaf documents for eight different regulatory agencies**; and here is where traditional processes and new technologies collide, because on the one hand, XML and SGML structure information know nothing about pages, while on the other, the product (the documents) are managed at the page level. What is important when managing information presented on pages is such things as where a page starts, and ends, what its page number is, what revision it is, when it was issued, and when it comes into effect. While you could tag data in XML with revision information, you cannot deal with the structure of the page—because the page is a storage medium, not an information structure.
4. **The document management process had to be simple, and web-based**; a good portion of the inefficiency in loose leaf management has to do with the fact that it is several full-time jobs’ worth of effort to keep track of the changes, the page revision histories, the review comments from the regulators and all the internal reviewers of the changes.

This last point is perhaps the most crucial from a business benefit point of view, and I’d like to take a bit of time to describe why this is so.

For those who may not be aware, loose leaf publishing is a highly demanding and disciplined form of document authoring, production, management and distribution. The goal is pretty straightforward: as changes are made to the content of a document, treat each page essentially as a separate entity, with its own revision history, and reissue only those pages which have changed.

This is not unlike managing updates to a software development process, where, as revisions are made to individual components, updates to a software product may only include a fraction of the total of the components in the product. In the case of documents, it's a bit more complicated, because it's like issuing each software component on its own CD. So, for example, if you were going to distribute each file which made up Windows NT on a separate CD, and then manage the updates to each file by reissuing those files which had changed by reissuing those CDs, you can see that the scope of the management increases dramatically.

For large document sets, such as new drug applications, power plant operation and safety manuals, or telecommunications tariffs, all of which may span anywhere from ten to tens of thousands of pages (even more), there is a tremendous advantage to being able to distribute only those pages which have changed, regardless of whether you distribute paper or electronic files.

Well, it seems innocent enough, perhaps, on the surface. Let's take a look at the process:

1. Publish the document
2. Receive changes
3. Locate pages where changes occur
4. Incorporate the changes
5. Adjust the page revision information
6. Create change instructions
7. Build a package of the changes and instructions
8. Publish the change package
9. Distribute the change package
10. Hope that the information consumer receives the changes, and actually follows the instructions on how to incorporate the changes into the document set.
11. Repeat steps 2 through 10 until such a point that the document integrity is questionable, at which point you republish the document.

For something like a nuclear submarine, a power plant, or a large telecommunications tariff, this was always an intensely manual process. The big advances in the 1970s and 1980s all dealt with the formating and publishing aspects of document production, not the authoring, management or distribution aspects. It really wasn't until the early 1990s that the benefits of electronic document management to the large document life cycle started to be understood and appreciated. But even then, there was a fundamental question: How do you manage the pages? Given a Word Perfect or Word file, how do you sensibly go about managing, tracking, and implementing revision management methodologies on each of those pages?

The answer, most of the time, was the same: break the files into individual pages, and manage each as a separate document. It might seem drastic, but it makes a lot of sense, because for loose leaf documents, each page has its own life cycle. And as new information on a page is added, it must be able to be incorporated without causing renumbering ripples throughout the entire document set. You change a page number, you are revising it. You revise it, you must reissue it. You reissue it, you must get approval to reissue it, through both internal and external approvers. Similarly, delete a page, you cannot allow all the other pages to ripple back.

That's the basic problem. Bell Atlantic has this problem, as does every regulated industry. And that was a tremendous force in motivating Bell Atlantic to seek better ways to manage loose leaf documents.

THE VISION

When I first came to this problem with Bell Atlantic, I had recently done some significant application work for two key clients in similar areas: the Canadian Department of Justice's legislative branch, known as the Statute Revision Commission (SRC), and Hughes Aircraft of Canada Systems Division (HCSD), who had been contracted by Transport Canada to develop a computer automated air traffic control system. The SRC is a government department tasked specifically for managing the updates to the Statues of Canada as enacted by Parliament. At that time, this amounted to thirteen volumes of information, taking up about 2 meters of shelf space. HCSD, on the other hand, was developing massive system specifications according to the government software and documentation standard DOD-2167A. Both were highly regulated. The SRC was primarily a paper-based process. HCSD was primarily electronic. The SRC needed a loose leaf system; Transport Canada would have preferred it for HCSD, but because all of the documents were machine assembled, we managed to side-step the issue. The SRC's loose leaf system took baby steps to provide document-oriented loose leaf management, rather than one-page-per-file non-management.

Bell Atlantic needed what amounted to a blending of both types of solutions: automated document management and production for loose leaf document sets.

Well, the consensus of everyone at the time (except the sales folks) was that it couldn't be done. XML and SGML were pageless. You could not sensibly impose rigorous page revision methodologies on XML or SGML data.

What was needed was a couple of new approaches. First, we had to recognize that the pages were not simply going to disappear. In fact, the Gartner Group, in a study commissioned by the New York Public Utilities Commission, one of Bell Atlantic's regulators, told the Commission to throw away the page. The answer was a flat, "No".

The governments involved would not enact changes to regulations to accommodate the fact that XML didn't know much about pages, and couldn't really be taught without significant effort. XSL itself was still developing, and even cascading style sheets didn't offer much of a solution, because these are purely format-oriented, and again, pages are not, strictly speaking format objects.

So, rethinking this a bit, we had to find a way to "lay down" structured information with an adequate set of management devices, not part of the data stream, on the page. The XML data can be pageless, but you at least need a way to control where the pages start and stop, much the way you need a way to format a floppy disk. Second, we had to provide an adequate level of abstraction so that authors didn't need to know (but could know if they chose) what pages were in a chapter, and which page they were working on at any one time.

Ultimately the vision was simple: provide a way for authors to write their documents in a more natural, pageless way, in XML, and let the electronic document manager deal with the rest, including all the messy page "stuff".

(Easier said than done.)

THE TECHNOLOGY

The document management backbone of the system was Open Text's Livelink. Not surprising, since Bell Atlantic specifically went to Open Text with its request for information. Bell Atlantic wanted Livelink because of its web-based architecture and collaborative tools. We wrote a special DTD for Bell Atlantic's tariff documents, to ensure that the structural needs of the tariff information were being met. Since at that time XML was still not well supported by application developers, we implemented the solution as an XML compliant SGML solution. Arbortext Adept Editor, an SGML editor, was the authoring tool, but we have used many different editors, including SoftQuad's XMetaL, to provide the authoring front end to these documents, without any problem.

We had authoring, we had a document manager, we had no publisher.

We spoke with several vendors of SGML typesetting solutions, but for a variety of reasons, none was acceptable. In desperation, I posted a question to the comp.text.sgml newsgroup, and got a response back about TopLeaf, from Turn-Key Systems, here in Sydney.

As we discovered, TopLeaf was designed to manage loose leaf documents, using XML and SGML as its input. After a few weeks of negotiation, Bell Atlantic agreed that TopLeaf was the better technology for what they wanted.

THE ARCHITECTURE

By May of 1998, Turn-Key had produced the first cut of an API for TopLeaf. This was critical to the project, because without an API, Livelink, the document manager, had no way to talk to TopLeaf. So, regardless of the fact that TopLeaf could format, manage, publish and print XML loose leaf documents, we had no way of making it do anything until we built a Livelink-based customization which could tell it to do those things, and to do that, TopLeaf needed to have an API.

Open Text had agreed to bear the cost of developing the interface between Livelink and TopLeaf, as part of the custom solution for Bell Atlantic. Turn-Key had agreed to bear the cost of developing the API.

In fact, Turn-Key developed both a command-line interface (CLI) and a DLL which could be called from Livelink. We opted to use the CLI, because it meant we could easily script calls to the CLI, and execute those scripted calls through a single, common entry point in Livelink.

At the same time, Turn-Key was busy developing the typesetting control files which would control TopLeaf. TopLeaf uses four control files to specify the behaviour of the typesetting engine in response to the XML tags it reads in the input data stream:

- a page style file, which defines things such as page layout characteristics, data areas, margins, and so on,
- a mapping file, which maps XML or SGML mark-up tags to typesetting macros which drive the typesetting engine
- a macros file, which encapsulates the bulk of custom typesetting functionality in response to data in the data stream
- the DTD.

The typesetting requirements for Bell Atlantic’s documents are extraordinarily detailed. Each regulator has its own set of regulations, and they are all similar but different. The control files originally were built to handle the documents produced for the FCC’s jurisdiction. (There is a four-volume set of books which define the regulations pertaining to the layout and management of documents targetted for the FCC.) Over time, the intention was to enhance these control files and provide some kind of switching mechanism so that Livelink would be able to say to TopLeaf “this document is for New York”, or “this document is for Rhode Island”, without requiring additional control files.

All along, we had envisioned that somehow, either in Livelink’s database, or in TopLeaf’s control files, we would develop a method for storing information specific to each page, which could be programmatically queried and manipulated to deal with the paper-oriented issues of managing revision, date and status information. Up until this point, we had more or less been able to shunt this issue to the back burner, because just getting Livelink to control TopLeaf the way it needed to had been problem enough. By October of 1998, we came to realize that TopLeaf would not do this without additional work. We also came to realize that TopLeaf was where it did in fact need to be done.

Unbelievably, within four weeks, Turn-Key had developed the ways and means to do this and the idea of "leaf status information" was introduced to TopLeaf. (In double-sided documents a page is composed of two leaves, a right and left-hand leaf. In single-sided documents, a page and a leaf are essentially the same thing. Bell Atlantic's documents are all single-sided, but TopLeaf handles both types.)

The position we kept coming back to was this: It is incorrect to attempt to encode this type of information in the XML document. This information does not intrinsically belong to the XML data, but rather to the page on which the data appears. By providing for a variety of other, non-XML-data-driven data types, TopLeaf could "stamp" the page (in the appropriate place of course) with page-oriented revision information. By associating these bits of extra stuff with each composed page, we could change these leaf indicators independently of the XML data. And, since the leaf indicators were stored with the actual composed leaf images, and not the XML, the document structure did not have to accommodate any page-oriented attributes.

So how does TopLeaf do this magic of managing XML documents as loose leaf pages? In a nutshell, it goes like this. When an XML document is deposited into TopLeaf's repository, TopLeaf uses its control files to typeset the document and create a composed image. When the XML data is updated, a difference engine determines where content changes have occurred, recomposes those pages, and flags those pages as being changed. Other types of data (such as leaf indicators) might also cause a page to be flagged as changed even though the input data has not changed. After this is done, TopLeaf allows the user to select for printing only those pages which have been touched in the update. In the background, TopLeaf is in fact creating separate page files for each composed page. It then sets status information to "flag" a page as being touched in the update or not. From the user's point of view, they simply edit a continuous file. TopLeaf typesets, paginates, and manages all the other loose leaf issues.

PROCEDURES

From the user's perspective, the system is simple to operate through a handful of funtions, all done through the web browser. TopLeaf is driven completely through its API, and is installed in the back office, away from any direct user control.

From the information processing side, things are not quite so simple. The procedures revolving around how changes to regulatory documents can be implemented are quite strict. There are only a select few who can request a change to an existing document. The reasons for changes can be such simple things as correcting errors, but they may also involve introduction of a new product or service into the regulated jurisdiction. The regulators, too, can request changes.

Each jurisdiction has its own rules regarding how the changes should be incorporated as well. For example, for some jurisdictions, all changes must be done against the version of the page currently in effect. For other jurisdictions, the changes must be done against the highest version of the page, regardless of whether it is in effect or not.

Let me unpack these a bit. For the first kind of jurisdiction (changes against currently effective), for each person who might simultaneously be working on the same page, a copy of the chapter being worked on is made. Each person works independently, and each changed page package may be submitted to the regulator without regard to the other.
This can lead to the situation where you have multiple copies of the same document floating around, but with different changes in each. At some point, one of those changes will become effective, and the other outstanding changes will have to be reworked to bring the new information into that changed document. For the second kind of jurisdiction, where changes are made against the highest version, another change management problem is introduced: multi-streaming. This has proven to be an extremely difficult issue to tackle and implement. The issue is: how do you flag a changed page as belonging to one group of changes rather than another? For example, let’s say you have a document which is being updated by two change requests. One change request touches pages 1 and 3, the other pages 1 and 2. A method is needed to be able to determine whether or not a page belongs to one change request or the other.

One solution is to implement this as tracking attributes in the XML. In this solution, the author would be required to input a value against at least one change on a given page (assuming they knew where they pages were), and the application would be able to “pick up” on this value, and deliver a set of loose leaf pages specifically related to that change package. We choose, however, to stay away from introducing these types of attributes in the XML source, as their use is prone to user error.

Another way to do it is through TopLeaf’s “leaf indicators”, the same mechanism we are using to manage other page-related information. At the time of writing, this is being implemented. It has required additional functional enhancements to TopLeaf to allow it to be queried properly to deliver a set of pages having a leaf indicator of value “X”, for example, as well as enhancements to the differencing process to be able to detect which pages “ought” to appear in an update for one group of changes.

PROGRESS

By May 1999, the New England jurisdictions were supported by the application. These regulatory jurisdictions include Rhode Island, Vermont, New Hampshire, Maine and Massachussetts. Each regulatory jurisdiction has its own rules about how page revision and status information is to appear on the page (and where).

Bell Atlantic’s Tariff Management System has been in production since June of 1999, and this system has shown that it is feasible to move traditional, regulated document publishing processes successfully to XML.