GOOD Publishing System: Generic Online/Offline Delivery

Short Paper

Jacek Radajewski
Distance and e-Learning Centre
University of Southern Queensland
Queensland 4350 Australia
jacek@usq.edu.au

Sally MacFarlane
Distance and e-Learning Centre
University of Southern Queensland
Queensland 4350 Australia
macfarla@usq.edu.au

Stijn Dekeyser
Mathematics and Computing
University of Southern Queensland
Queensland 4350 Australia
dekeyser@usq.edu.au

Abstract GOOD *is a tailor-made, fully integrated publishing system that creates output documents for multiple media types used in both online and offline teaching modes at the University of Southern Queensland. It is used in the Distance and e-Learning Centre of USQ to create course material for thousands of on-campus, online and external students. Among the end products generated from a single XML input document containing study material for a specific course are study books, introductory books, and web sites in a variety of formats. Future end products currently being investigated include voice rendering. The* GOOD *system is entirely based on open standards such as* XML, XSLT, DOM, *and* XSL:FO *and implemented with* JAVA/J2EE *technology. Among its features is a smart editing client to allow technically non-proficient staff to edit their own course material.*

Keywords Document Management and Publishing, XML authoring.

1 Introduction

USQ is among the leading Australian universities catering to international and off-campus students. The Distance and e-Learning Centre (DeC) provides services to lecturers in publishing educational material both in print form and for web based systems.

Aside from using the GOOD system presented in this paper, course material at USQ is typically written using commercial word processing or publishing software, or LaTeX in the university's mathematics and computing department. Academics working with DeC staff collaborate on a wide range of file formats. The main drawback of this approach is that the aim of "write once, publish in any format" is difficult and expensive to reach. To this end, the DeC in 2000 started work on a generic system to author course material in XML and deliver output in a wide range of formats for both online and offline delivery.

Proceedings of the 9th Australasian Document Computing Symposium, Melbourne, Australia, December 13, 2004.

Related Work

To our knowledge, a complete, fully integrated publishing system specifically for authoring and delivery of course material based on XML technology did not yet exist when work commenced on the one described here. The well-known DocBook [15] system is too complex for use by technically non-proficient academics. Furthermore, DocBook was designed for authoring IT-specific documentation, making its DTD unsuitable for course material.

Additionally, commercial and other XML authoring clients currently assume some knowledge of XML and tend to offer only DTD-based checking of structure. The absence of a more intuitive interface to write course material in XML format is a significant limiting factor for the adoption of such generic editors.

As well as not being an XML-based system (thus making conversion from it to any other format non-trivial in general), LaTeX often is too complex for non-computing oriented staff.

2 GOOD Document Type

To specify the document type for GOOD, the DTD technology was chosen instead of the more expressive XML Schema [9] (XSD) because at that time XSD's specification was not yet a W3C recommendation. Furthermore, types are not so important in this very document-oriented application. Even so, adoption of XSD is planned for the future (see Section 6).

The GOOD DTD [11] consists of four main sections. These are meta-data, introductory material, study modules and selected readings. Cross-referencing is permitted between all sections. We briefly describe each

of these sections. The *meta-data* section holds information about the course, for example course code, year and semester offered. The *introductory* section includes information about assessment, general study resources and faculty policies, as well as the course introduction. The *study modules* section consists essentially of any number of modules, which can be grouped into parts. Modules include the basic content of the course, as well as learning objectives, recommended references and activities. Modules can be further divided into topics, sub-topics and ideas. Arbitrarily deep nesting of topics is not allowed for typesetting and pedagogical reasons. Other elements which are optional in this section include 'executive overview', 'index', 'glossary' and 'appendix'. Finally, the *selected readings* section contains any number of selected readings which may be grouped by module, or be independent of modules. Each reading may include a scanned reading or a reference to a URL.

Elements of varying degrees of granularity, from paragraph up to module, can be marked for conditional delivery. For example, some content may only be intended for print delivery and is required to be excluded from the web delivery. This is implemented as a group of optional attributes on specific elements. Finally, the DTD provides support for images, standard LaTeX equations, and various types of media, such as Macromedia Flash.

2.1 Meta DTD

The GOOD DTD is contained within a "MetaDTD" XML file [11]. Naturally this file is governed by a DTD itself. The MetaDTD document contains 'help' information for each element in the actual GOOD DTD, as well as the DTD definition for each element. This enables developers to easily update the help information as elements are added or altered. From the MetaDTD, using Java, DOM [10] and LaTeX, we render HTML help, PDF documentation, and the GOOD DTD itself. Use of our MetaDTD tool will also facilitate the eventual migration to XML Schema.

2.2 DTD updates

The GOOD DTD is constantly evolving, with attributes and elements being added, deleted and updated. This means that occasionally the DTD is changed in such a way that existing documents become invalid with respect to the new version of the DTD. Each such DTD change is released with an accompanying updater, essentially a SAX [5] parser which is run over existing documents to make any necessary changes. This prevents users from being exposed to validation errors caused by DTD changes.

3 Client: XML Authoring

As mentioned in the introduction, authoring documents conforming to a specific DTD using current editors typically is not very intuitive. For this reason, the GOOD system comprises an editor (Arbortext's Epic [2]) that has been extended with specific GUIs to make editing of course material by end-users much easier. Even so, the GOOD system's architecture is neutral to whatever XML editor is used. Thus, sophisticated end-users may use their preferred tools.

Epic displays the traditional two views of the XML document: a tree-based map view, and a styled-up view using FOSI [6]. The FOSI stylesheets enable automatic numbering of elements such as modules, topics and readings; display of grouped data in html tables; and display of information from cross-referenced elements. Thus, the use of Epic allows authors to get a swift, if inaccurate, feel for how the final product will look.

In the next three sections, we describe two additional features of the extended Epic editor. The first is specific to Epic, while the second and third comprise our own extensions.

3.1 Change Tracking

It is important to the workflow of GOOD documents that users can see the changes made to the document by themselves and other users. Epic implements change tracking using elements (such as add and delete) within a namespace. These elements which are specific to change-tracking remain in the document until the change is either accepted or rejected by a user through Epic.

Epic automatically handles the validation of documents containing change-tracking with respect to the DTD. Outside Epic, however, documents containing change-tracking cannot be validated with respect to the DTD. All the XML parsers used as part of the rendering process are for this reason non-validating, leaving the validation to the editor. As part of future work, we will investigate the development of an XML Schema that would accept both forms of the document.

3.2 Custom-built Java Swing GUIs

There are also a number of custom-built GUIs, written in Swing [8], which are used to simplify certain operations for the user.

The GOOD DTD supports some twenty reference types, including book, database, journal and email. The user enters information through a reference GUI which clearly indicates mandatory fields and other requirements, based on the reference type. These GUIs are based on a common framework so that it is relatively simple to change the requirements for existing reference types or add new ones. Fields or groups of fields that are commonly used across various reference types, for example URL or group of authors, are represented by Java classes.

The GOOD DTD specifies a number of cross-reference types, each referencing different elements, including textbook, reading, resource and module, as well as a generic cross-reference that can reference any type of element. The cross-referencing GUI

displays a tree of elements, filtered depending on the cross-reference type, from which the user selects a cross-reference. This enables users to easily select an element to cross-reference, with validation performed by the application.

There is also a GUI to enable the user to perform *check in/out* operations, without knowing anything about the version control system in use. Access to documents is restricted by access control lists. Also, the user is able to view the HTML help for any element in the GOOD DTD. This help is produced from the MetaDTD presented in Section 2.1. The LaTeX *viewer* GUI enables the user to preview LaTeX equations. Finally, the *render* GUI allows the user to specify exactly what they wish to produce, for example web, print, draft, cd label or print cover pages.

3.3 Document Checker

Some constraints on document contents cannot be specified by DTD or schema. These constraints include restrictions on empty elements, duplicate references and invalid URLs. The *document checker* GUI performs this validation, reporting errors and warnings back to the user, with a cross-reference to the location of each error so that the user can easily rectify the problem.

4 Rendering Servers

The server side of the GOOD system is mainly concerned with managing course documents and rendering them to any of the supported output formats. The rendering phase is somewhat different for print products than for web publishing. However, both have a number of steps in common, as described next.

4.1 Common Rendering Steps

The rendering process essentially consists of a number of sequential DOM and SAX parses and transformations of the document, after which the resultant XML is transformed using XSLT to the desired output type. As mentioned previously, all of these XML parsers are non-validating.

The *SAX Document Builder* performs initial processing of the document. This includes normalizing filenames and converting images to different formats depending on the target. For PostScript output, high quality EPS images can be used, while for PDF or web output these images are converted to a lower quality image format. The SAX Document Builder also dereferences non-ASCII character entities.

Next, the *Delivery Filter* filters out content which is marked not to be delivered to the render target. This is done before pre-processing so that excluded sections do not affect the numbering or format of the final output.

To enable users to view the effect of their changes on the rendered output, GOOD has two options for rendering documents containing change tracking. Users can render the document "without" change tracking, that is render the document as though all changes had been accepted. There is also an option to render "with" change tracking, that is render the document with added content highlighted in green and deleted content in red. The change tracking filter makes the necessary changes to the document.

The *Pre-processor* uses both XSL and Java to number certain elements, collate reference lists, normalise image height and width attributes and populate element titles.

4.2 Print Rendering Steps

After pre-processing, the print render process is simple. To generate the XSL:FO [1] document, the appropriate XSLT stylesheet is chosen, depending on the document type being rendered (study book, introductory book, selected readings or solutions manual). XEP [16] is then used to render either PDF or PostScript from the FO document. The table of contents, including bookmarks, is done by the XSL. Cross-references are rendered as internal links.

4.3 Web Rendering Steps

Unlike print output, web output consists of any number of separate pages, with cross-references rendered as links within and between these pages. A SAX parser processes the cross-references, modifying the links so that they will work once the content is broken up into separate pages. The content is then broken up into chunks, each chunk representing a discrete section of the course, for example module, reading or course overview. Each chunk is then passed to an XHTML XSL transformer to generate a separate HTML page. The same chunk is passed to an XSL:FO transformer to produce a PDF version of the same page.

The navigation for the site, including a JavaScript menu and site map, are also generated using XSL. Images, CSS and other content are then packaged with the generated XHTML files to make a complete web site. For output of hypertext files to a hybrid cd, an ISO-9660 CD image file is created.

5 Implementation Issues

We now turn to a few implementation-related issues in the GOOD system. Since this is a short technology paper, this section is necessarily brief. More information about the system's design and implementation can be found in [13].

5.1 Scalability

At present, the GOOD system is used primarily by DeC staff and a small number of academics from different faculties. As the number of users of the GOOD system grows, scalability is increasingly important. A number of users may be rendering concurrently, or simultaneously require access to the document repository. The following two sections describe how the GOOD system is designed to deal with such issues.

5.1.1 JMS render queue

To enable multiple users to render concurrently, the render process is implemented as an asynchronous process. The client sends a render request which is queued on a JMS [14] queue of render jobs. There are a number of distributed rendering servers, each with a Message Driven Bean [14] monitoring this queue. The Message Driven Bean picks up the job and initiates the render. This solution is scalable since while each rendering server executes only one render at a time, any number of rendering servers can connect to the same queue.

The rendering server sends progress messages to the client, at various stages of the render process. When the render completes, the output file is automatically opened on the user's machine.

5.1.2 XEP memory use

XEP uses a significant amount of memory, particularly for rendering images. Initially some documents were causing out of memory errors on the rendering servers. There is now a separate queue and rendering server for large render jobs.

5.2 Version control

GOOD currently uses CVS for version control. CVS commands are run from shell scripts called from Java. There are some transaction issues with this, since it can be difficult to tell whether the shell script has failed.

6 Future Work

While GOOD is already in operation, it has a sizable wish list attached. We classify each of these future features based on a time-frame for their implementation.

Short Term

- *Role-based access* to course documents (Read-only, Write, Team Leader).
- *Document preferences* allowing a user to set up preferences relating to document structure, so that based on these preferences we can create a course document skeleton. Also for example, when they add a module, the editor will be able to set attribute values and create required elements based on these preferences.
- *Access to* GOOD from outside USQ.

Medium Term

- *Migration to XML Schema* [9] instead of DTD.
- *Voice rendering* using VoiceXML [12] to support visually impaired students.
- *Closer integration* with other university systems such as calendar and course specifications.
- *Cross-platform support*.

Long Term

- *Migration to XML Databases* [3] instead of CVS.
- *Concurrent Authoring* either by use of CVS or more elaborate mechanisms [3, 7].
- *Import/Export* to and from LaTeX and to and from the emerging Open Document Standard [4].

References

[1] S. Adler, A. Berglund, J. Caruso et al. Extensible stylesheet language (XSL) version 1.0. Recommendation, World Wide Web Consortium (W3C), October 2001. http://www.w3.org/TR/xsl.

[2] Arbortext. Epic editor overview. Arbortext.com Web Article, 2004. http://www.arbortext.com/html/epic_editor_overview.html.

[3] R. Bourret. XML and databases. Website, 2004. www.rpbourret.com/xml/XMLAndDatabases.htm.

[4] M. Brauer, G. Edwards, D. Vogelheim et al. Open office specification 1.0. Committee draft 1, Oasis, March 2004. http://www.oasis-open.org/committees/office/.

[5] D. Brownell. *SAX 2*. O'Reilly, January 2002. ISBN: 0-596-00237-8.

[6] G. Charlebois. Standard generalized markup language (SGML): Overview and new developments. *Network Notes*, Volume 3, 1994. http://www.collectionscanada.ca/9/1/p1-202-e.html.

[7] S. Dekeyser, J. Hidders and J. Paredaens. A transaction model for XML databases. *World Wide Web Journal*, 2004.

[8] R. Eckstein, M. Loy, D. Wood et al. *Java Swing*. O'Reilly, second edition, 2002. ISBN: 0-596-00408-7.

[9] D. Fallside. XML Schema. Recommendation, World Wide Web Consortium (W3C), May 2001. http://www.w3.org/XML/Schema.

[10] A. Le Hors, P. Le Hegaret, L. Wood et al. Document object model (DOM) level 2. Recommendation, World Wide Web Consortium (W3C), November 2000. http://www.w3.org/DOM/DOMTR.

[11] S. MacFarlane, J. Radajewski et al. GOOD and MetaDTD DTDs. Technical report, Distance and e-Learning Centre, USQ, 2004. http://www.sci.usq.edu.au/staff/dekeyser/Good/index.php.

[12] S. McGlashan, D. Burnett, J. Carter et al. Voice extensible markup language (VoiceXML) version 2.0. Recommendation, W3C, March 2004. http://www.w3.org/TR/voicexml20/.

[13] J. Radajewski et al. The GOOD System. Technical report, Distance and e-Learning Centre, USQ, 2004.

[14] B. Shannon. Java 2 platform, enterprise edition (J2EE). Specification, v1.4, Sun Microsystems, 2003. http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf.

[15] N. Walsh and L. Muellner. *DocBook: The Definitive Guide*. O'Reilly, October 1999. ISBN: 1-56592-580-7.

[16] xAttic. XEP XSL Rendering Engine. xAttic.com Web Article, 2003.