The Joy of X: The design of an XML system

Dave Kennedy
Christchurch Polytechnic Institute of Technology
Christchurch, NZ
kennedyd@cpit.ac.nz

The two main uses of xml are data exchange and as a central source that can be extracted and displayed in multiple ways. This paper describes the design and development of an xml-based system for course outlines that uses xml for data exchange and as a central repository. The central repository is constructed from a number of base xml documents that have been extracted from various disparate sources. The central repository is used to produce a range of different outputs in different formats. The design considerations, for the system, the schema and the xsl, are discussed.

Keywords

XML Design, XSLT, XSL-FO

1. INTRODUCTION

1.1 Motivation

The current course outline system (Kennedy & Lance, 2001) is based on a set of separate MS Word documents that are maintained by each course co-ordinator. There is a standard template, produced by the school administrator each semester, which contains common paragraphs e.g. sections on recognition of prior learning, aegrotat applications and dishonest practices and a table for the course diary that contains the week numbers and dates.

Course descriptors are a separate set of MS Word documents that are maintained independently of course outlines.

An updated course outline is typically produced by a series of cut and paste operations from the existing course outline and the latest course descriptor. Timetable information is added, staff details are updated if required, and course diary and assessment summary details are entered.

There is much redundant data and plenty of scope for errors.

Web pages that display much of the same content as course outlines are maintained independently of this system.

Most of the information that changes from one semester to the next is contained in the student management system (A Jade based system named Jasper).

1.2 The problems

Although there is a standard template for course outlines it is often changed from one semester to the next and each course co-ordinator further modifies it to suit their requirements.

The data redundancy inherent in multiple independent systems creates problems with maintenance and leads to out-of-date versions. This has been commented on by monitors.

Course outlines are more narrative-oriented documents than data-oriented. Each one is similar in structure but can also be very different in terms of the number of staff involved, texts and resources, pre-requisites, outcomes, assessments, and course diary information.

1.3 An XML Solution

The solution appeared to lie with an XML based system for the following reasons:

a) XML is a method of mark-up for data – in particular the data contained in documents.

b) XML has the flexibility to allow for many variations within a standard document schema i.e. the number of staff involved, a range of class hours and times and types of session, a variable number of aims, outcomes, assessments, texts, and many variations on the content of the course diary.

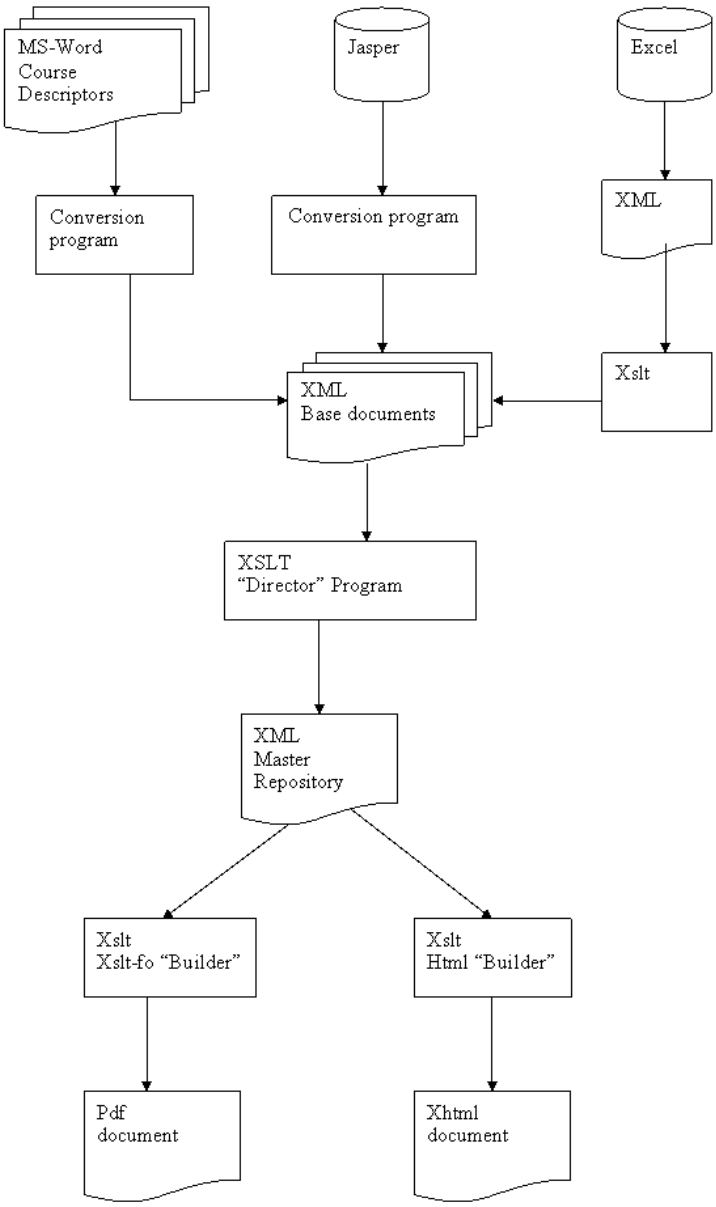

Figure 1. The XML System Design.

c) An important aspect of XML design is the separation of content and presentation (Hoenisch, 2002b; Miller, 2003; Neugebauer, 2003; Ryan, 2002). It should be possible to build a central XML repository that contains all the course outlines for a particular semester and use that to publish the data in many different formats e.g. web page, printed output, marketing information and other reports such as textbook lists (Kumar & Langley, 2002; Ponisio & Rossi, 1999; Souri, 2002). This way all output would be in sync and up-to-date.

d) XML has become widely used as a data exchange format and as a method for extracting data from disparate sources(Ponisio & Rossi, 1999).

1.4 The Challenge

The challenge was to:

a. extract the base data from a number of disparate sources and build a master document

b. write xslt programs to produce the required outputs

c. use 'best practice' techniques for the system design, the schema design, the xslt design

d. at all stages aim to separate the data from the presentation

1.5 Requirements

The main system requirements were:

a. Minimise human intervention in the production of course outlines i.e where there are no changes in texts, assessments or staff from one semester to the next the system will automatically produce the course outlines with updated class timetable and course diary. The assessment schedule may require assessment dates to be adjusted in the light of holidays and semester breaks.

b. Staffing changes will be made by the Programme Leader to a single base document.

c. If changes are made to a course descriptor (via faculty academic procedures) these will automatically be incorporated.

d. The faculty administrator will use an Excel spreadsheet to build an annual calendar containing week number, week beginning dates and term and semester dates. This information will be linked to the course outlines.

e. Produce course outlines in printed form and as a web page.

2. SYSTEM DESIGN

2.1 Initial Design

The basic idea was to extract data from Jasper etc and build a master repository of XML documents, one for each course outline. From there xslt programs would be written to produce the required outputs. (see fig 1)

This design was based on:

a) a flow of xml documents using xslt to transform one xml document to another (Harold, 2003; Miller, 2003; Ponisio & Rossi, 1999).

b) The use of RelaxNG as the schema language (Harold, 2003) to ensure structural completeness.

c) A master repository approach (Kumar & Langley, 2002; Ryan, 2002)

d) XML design patterns e.g XML Mediator, Director, Builder (Ponisio & Rossi, 1999; Sun, 2003).

e) Use of xml standards such as vCard, iCalendar and docBook (Kumar & Langley, 2002).

2.2 The Base Documents

Because the data comes from a number of different sources it was decided to extract from each source to a base document and then write xslt to combine the base documents and so produce the master repository. The base documents were identified as:

```
staffMembers (staff details – office,
phone, email)
timetables (day, times, room and
sessionType details for each occur-
rence)
courseDescriptor (hours, aims, pre-
requisites, objectives, assessments
etc)
courseInfo ( course controller,
teaching staff, assessment dates, and
session details for all sessions)
calendar (dates for semester, term,
holidays, breaks)
programme (programme details – code,
faculty, school, exam dates, resit
procedures, etc)
```

The base documents were defined using an ELM tree diagram. These were then written as RelaxNG Compact (rnc) schemas and tested using sample documents constructed from existing course outlines(Kennedy, 2003).

The vCard standard was used within the staffMembers base document and docBook biblioentry tags were used for describing referenced material(Iannella, 2001; Walsh & Muellner, 1999). At this stage the use of such standards has no impact on this system but it obviously facilitates data exchange with systems that use these standards.

A JADE program was written to convert the MS-Word course descriptors to XML(Lance, 2003). This could also have been done using OpenOffice or MS-Office to convert each document to low-level XML followed by an xslt program to change this into our courseDescriptor XML(Implementing the Online Edition of the Chicago Hittite Dictionary, 2002).

XML systems that use the write once – publish many paradigm are usually publishing static documents. In this case the course outline documents are semi-static in that we want to recreate and update them each semester. Most of the base documents require little or no change from one semester to the next. When this system was first developed the timetable information was extracted from Jasper each semester and the timetables base document recreated. From 2004 timetable information is no longer included in course outlines. Without the timetables.xml document the base documents are relatively static and the major design problem is to enable the development of xslt that can create the master reposi-

tory for any given semester (1, 2 or summer school) and xslt that can generate the required output for a given course code. The most changeable parts of a course outline are the assessment dates and the dates within the course diary. For semester and term classes the diary is in weeks, for summer school classes it may be in weeks or days. The courseInfo base document was specified with a courseDiary element that had attributes of unit (week or day) and unitNo. The assessment dates were similarly specified. As a consequence if the sequence of sessions for a course did not change from one semester to the next then the base document did not need to be changed.

The calendar base document is generated by first creating an equivalent Excel spreadsheet. The spreadsheet is saved as an xml document and an xslt program converts this to the calendar base document.

2.3 The Master Repository

A master repository is created each semester that contains an xml document for each course outline. It is possible to create a single xml document that contains all the course outlines in which case some normalisation (Provost, 2002) is possible e.g. staff details. However the xslt processing model uses an in-memory node tree so it is faster to use small source documents (Bonneau, Kohl, Tennison, & Williams, 2003). Each course outline is fully resolved in terms of assessment and diary dates, exam date and breaks.

3. SYSTEM DEVELOPMENT

The concept of extracting data from the various sources to each of the base documents has been proved using sample course outlines. However this part of the system requires further development to create a working system. The bulk of the work to date has been in creating the xslt to:

a) Combine the base documents and build the master repository.

b) Output a course outline in html format.

c) Output a course outline in xsl-fo format that can then be rendered as a pdf document.

3.1 The Excel transformation

The standard polytechnic calendar for programme, class, term and semester start and finish dates is organised by week no. i.e. for 2004 week no 1 starts 29 Dec 2003. The easiest way to create such a calendar is by using a spreadsheet. The courseDiary base document was designed with unit (week or day) and unitNo attributes. The director program uses an offset parameter (e.g. 29 for semester 2) which when added to the unitNo gives the weekNo for that diaryLine. The date for that weekNo is found via a lookup to the calendar.xml document. This means that the courseDiary base document does not have to be changed each semester to reflect the new dates.

The calendar created using Excel was saved as an xml document. This document is low-level xml with detailed elements and attributes that fully describe the workbook and spreadsheet e.g. Author, Created, LastSaved, WindowHeight, as well as attributes such as Style, DataType and Formula for each cell. A simple rule-based (Kay, 2001) xslt program, was required to transform the Excel xml to the calendar.xml base document. The calendar.xml document contains a collection of date elements each with attributes for weekNo, semester, term and startDate.

3.2 The xslt "Director" program

The director program combines xml structures and data from the base documents and creates the master repository document for a given course. The program has been generalised by the use of parameters(Bonneau et al., 2003) for programme, semester, term and an offset. The offset parameter is added to the week numbers in the course diary to give an absolute week number which is then used to lookup the start date for that week. The input document is a particular course descriptor.

The documents were processed using saxon, an open source implementation of xslt (Kay, 2001). Saxon is a command line program that accepts various parameters: most commonly an xml input, a stylesheet, and an output filename.

```
E.g. saxon courseDescriptorIS301.xml
combine.xsl thisSemester="2"
thisOffset="29" -o
courseOutlineIS301.xml
```

A series of such commands can be run as a batch file which enables the pipelining of transformations

i.e. the output from one transformation can be used as the input for a further transformation (Bonneau *et al.*, 2003). Saxon also has a trace option which is useful as a debugging aid.

An xsl:variable can be an xml document

```
 e.g. <xsl:variable name
="docProgramme"  select
="document(programme.xml)" />
```

This construct was used to open the base documents and make them available to the xsl program which could then extract the data required.

The programme document uses staffid attributes for programme leader and head of school. The program uses this id to lookup the staffMember details in the staffMembers.xml document using a statement such as

```
<xsl:variable name ="thisStaff"
 select ="$docstaffMembers/
staffMembers/
staffMember[@staffid=$thisStaffid]" /
>
```

(Bonneau *et al.*, 2003)

This statement is indicative of the high-level, functional programming that xslt provides. The appropriate xml fragment is retrieved and assigned to the variable thisStaff.

The director programme is mostly about combining the xml from a number of documents into the one master document. It uses a number of xsl:copy-of instructions. This instruction is a "deep" copy i.e. it copies a node set and all underlying sub-trees to the output file (Kay, 2001).

3.3 The xslt "Builder" programs

The builder programs that extract the data for a particular course outline and create an html or xsl:fo document are extracting the same data so the xsl code to do this is the same in each program. In line with the fundamental principle of XML design – the separation of presentation and content – the "look and feel" was contained in a separate file. (Harold, 2003; Hoenisch, 2002a; Neugebauer, 2003)

The html builder wraps this code in html tags that includes a link to a css which handles most of the presentation aspects. The xsl:fo builder wraps the extraction code in xsl:fo tags and imports an attribute-set file. Again the attribute-set file controls the "look and feel" aspects of the final pdf document.

The builder programs were designed to be a general program that would accept a parameter for the course code and then create the required course outline document. This parameter was an xml file that contained rdf and Dublin core elements for the course outline.

Xslt can be modularised by using separate templates to process different nodes (Hoobler, 2002). If these templates are stored in separate files then they can be imported into other programs as required. The builder programs were designed to be highly modular. A separate file was created for the templates for each section. The main program used seventeen import statements to bring all the code together. This makes for much easier debugging and maintenance and maximises reusability.(Bonneau et al., 2003)

3.4 Producing Pdf output

A builder program was used to transform the master xml document into an xsl-fo representation of a course outline (Neugebauer, 2002; Pawson, 2002). The xsl-fo document is in turn processed by a java program called FOP (Formatting Objects Processor) to produce a pdf document (Souri, 2002). The main program sets up the xsl-fo page masters and page sequence, creates the header and footer and then uses a for-each loop to process the various sections of the course outline such as staff members, handbooks, etc. Each section has its own template which has been imported into the main program. These section templates create the xsl-fo tags as required and extract the data using the same xsl instructions as is used in the html builder program. The presentation aspects are contained in a separate file which is referenced by an xsl:use-attribute-sets instruction (Neugebauer, 2003). It is the xsl-fo equivalent of a css file. However because xsl-fo is concerned with more detailed presentation aspects than is html the attribute-set file is more complex than the css file. The FOP program is used to produce a pdf document (see fig 2).

4. FURTHER WORK

It should be possible to save the course Descriptor documents in xml format and then use xslt to create the courseDescriptor base documents. This would enable a working system for the production of course outlines each semester.

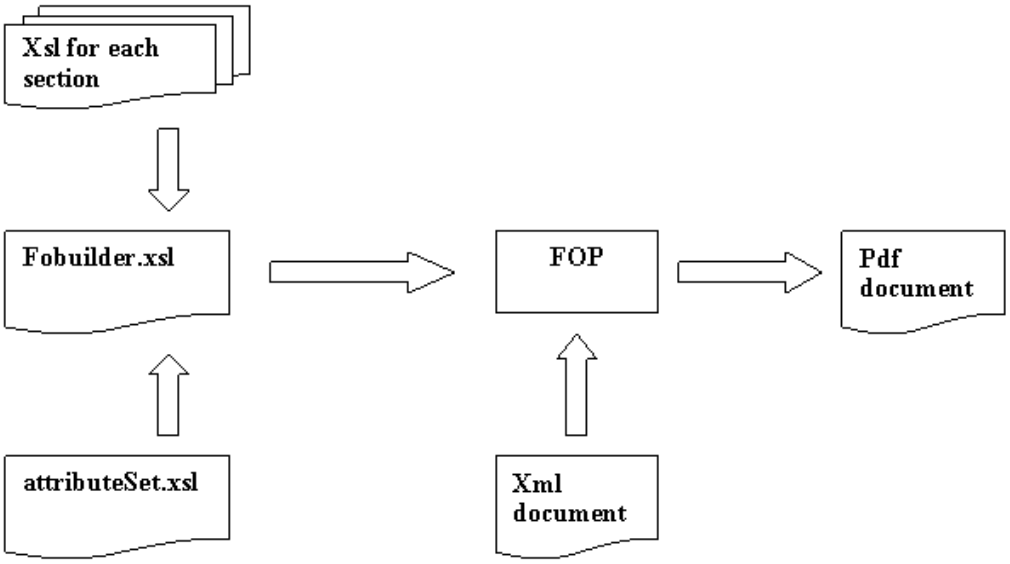

Figure 2. using fop to produce pdf output.

Additional builder programs can be written to produce other output e.g. marketing information.

Because of memory constraints it is not possible to store all the course outlines within a single document. If a native XML database was used to store the master repository documents then reports on all course outlines would be possible e.g. a list of texts for a group of courses.

5. CONCLUSIONS

Today xml is often used for data exchange, to which it is well suited. However the real power of xml is seen in systems that build, store and transform documents: documents that have a defined structure that is also highly flexible and extensible (Miller, 2003). To allow for such flexibility using a traditional database design would make for a complex database structure and complex programming. An xml design for such a system is much more straightforward as is seen in the design of this system which creates a set of course outline xml documents and displays them in multiple formats.

The use of elm tree diagrams to describe the xml structures provides a useful diagram from which the rnc schema can be constructed and is an aid to writing xsl programs.

Relax NG compact is an easy to use and powerful schema language which can be used to validate the xml documents.

Good xml design is based on:

a) The separation of content from presentation at all stages e.g. the use of css and attribute-set files.

b) A flow of documents through a series of simple transformations rather than one complex transformation.

c) Modular xsl programs that use parameters for ease of writing, debugging, re-use and maintenance.

This system showed that Excel documents saved in xml format can easily be transformed to a specific xml format.

This system also demonstrated the use of xsl-fo and the use of a program such as FOP that can render xsl-fo documents as pdf or other formats.

Acknowledgemets

Thanks are due to Jason Ions and Stephen Wilson for the work they did in developing this system as a BICT programming project (level 7, 45 credits).