KUNGL
TEKNISKA
HÖGSKOLAN

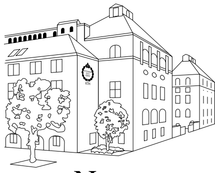
Royal Institute of Technology
Dept. of Numerical Analysis and Computer Science

Presenting XML documents on different media with stylesheets

practial use and web server integration

by
Mikael Ståldal

TRITA-NA-E0051



NADA

Nada (Numerisk analys och datalogi)
KTH
100 44 Stockholm

Deptartment of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, SWEDEN

Presenting XML documents on different media with stylesheets

practial use and web server integration

by
Mikael Ståldal

TRITA-NA-E0051

Master's Thesis in Computer Science (20 credits)
at the School of Computer Science and Engineering,
Royal Institute of Technology year 2000
Supervisor at Nada was Viggo Kann
Examiner was Stefan Arnborg

Abstract

XML is a useful framework for storing and processing structured documents. By using stylesheets, automatic presentation on different media can be achieved. I have evaluated different stylesheet technologies and concluded that XSLT is suitable for high-quality presentation of the same document on several media.

I have also investigated the integration of XML and XSLT in a web server and the performance issues it raises. Caching is necessary to achieve acceptable performance. I have studied a new approach of doing XSLT processing with caching, implemented a prototype for it and concluded that it can give better performance than the usual approach with a web server extension.

Sammanfattning

Svensk titel: *Presentation av XML-dokument på olika medier — praktisk användning och webbserverintegration*

XML är ett användbart ramverk för lagring och behandling av strukturerade dokument. Genom att använda stilmallar så kan man uppnå automatisk presentation på olika medier. Jag har utvärderat olika stilmallstekniker och kommit fram till att XSLT är lämpligt för att med hög kvalité göra presentationer av samma dokument på flera medier.

Jag har också undersökt integration av XML och XSLT i en webbserver och de prestandaproblem som det ger upphov till. Cachning är nödvändigt för att få acceptabel prestanda. Jag har studerat ett nytt sätt att göra XSLT-bearbetning med cachning, implementerat en prototyp och kommit fram till att det kan ge bättre prestanda än det vanliga sättet med en webbserverutökning.

Foreword

This report is the result of a master's project in computer science at the Department of Numerical Analysis and Computer Science, at the Royal Institute of Technology in Stockholm, Sweden. The work has been done at my employer, Entra Internet Solutions.

I want to thank my supervisor at NADA, docent Viggo Kann, and my supervisor at Entra, Kristoffer Lindberg, for their valuable support during my work.

Contents

1 Introduction **9**
1.1 Background 9
1.2 Current practice 9
1.3 XML - A new approach 10
1.4 Relations to other work 10

2 XML and related technology **11**
2.1 XML 11
2.1.1 Background 11
2.1.2 Basics and well-formed documents 11
2.1.3 Document classes 12
2.1.4 Valid documents 12
2.1.5 Namespaces 12
2.1.6 XML Schema 13
2.2 XHTML 13
2.3 Cascading Style Sheets 14
2.4 Extensible Stylesheet Language 14
2.4.1 XSLT 14
2.4.2 XSL:FO 15
2.5 XML APIs and software components 15
2.5.1 DOM 15
2.5.2 SAX 15

3 Analysis and design issues **17**
3.1 Stylesheet technology 17
3.1.1 CSS and XSL 17
3.1.2 Handling of different output media 17
3.2 XML API 18
3.3 Web server integration and caching 18
3.3.1 Different types of caching 19
3.3.2 Static and dynamic content 19
3.3.3 Caching possibilities 19
3.4 Different approaches to web server integration 20
3.4.1 Web server extension 20
3.4.2 Offline content generation 20

4 Practical use of stylesheets **21**
4.1 Tools 21
4.1.1 XML parsers 21
4.1.2 XSLT processors 21
4.1.3 Formatting object processors 22

4.2 Case study: Technical reports 22

5 XotW - XML transformations for the web **23**

5.1 How XotW works 23

5.1.1 The sitemap and producers 23

5.1.2 Extra feature: wildcards 24

5.1.3 Extra feature: splitting 24

5.1.4 Program structure 24

5.1.5 Running XotW 25

5.2 Performance test of XotW 25

5.2.1 Test environment 25

5.2.2 Test scenario and setup 26

5.2.3 Error sources 26

5.2.4 Test results 27

5.2.5 Test conclusions 27

6 Conclusions **29**

Bibliography **31**

A DTD for technical reports **33**

B Stylesheet for technical reports to HTML **35**

C Stylesheet for technical reports to XSL:FO **39**

D Stylesheet for technical reports to LaTeX **43**

E UML class diagram for XotW **45**

F DTD for XotW sitemap **47**

G Example of XotW sitemap **49**

Chapter 1

Introduction

1.1 Background

Consider having a document, for instance this report, that is meant to be presented on two different media, paper and computer screen. When printing it on paper, it is feasible to have clever pagination with page headers and a table of contents with page references. When reading it with a computer program (such as a web browser), it is feasible to have continuous scrollable text and a table of contents with hypertext links.

Technical reports are only one class of documents that one want to present on several different media. In electronic commerce, there are invoices and several other kinds of documents. Even if the ultimate goal is to never have to print such documents at all, the need to do so is not likely to disappear entirely in the near future. There may also be a need to present documents on both traditional desktop computer screens and the small screen on a handheld device (such as a mobile phone or a PDA).

Common to all these cases is that the document is to be stored in *one* well-defined and structured format (otherwise maintenance of the document would be unnecessarily difficult). The presentation of the document is to be tailored to make use of the specific features of each medium. The presentation on each supported medium shall be performed with minimal, preferably no, manual intervention. For some document classes, it is feasible to be able to easily perform other automated processing than just presentation, such as registering invoices in a bookkeeping system.

1.2 Current practice

For technical reports and similar documents that are to be presented on computer screen and printed, several practices exist.

The document can be created in a format suitable for screen viewing, such as HTML, and printouts can be made from the same HTML document. However, HTML browsers do not produce optimal printouts.

The document can be created in a format suitable for printing, such as LaTeX, and a PDF file can be made for screen viewing. However, PDF viewers do not give optimal screen viewing.

Two different versions of the document can be created, one for printing in LaTeX, and one for screen viewing in HTML. However, this gives maintenance problems.

An ad-hoc application can be written in a general programming language (such as C++, Java or Perl) to translate from LaTeX into HTML (or vice versa). However, this will not work with document classes that are very different from technical reports, and writing such ad-hoc applications for each document class is not feasible if there is a need to work with many different document classes.

1.3 XML - A new approach

XML gives a framework for storing structured documents of several classes in a well-defined way. XML documents are easy to process with a computer. By using a technology called stylesheet, it is possible to automatically produce presentations on several different media.

The purpose of this project is to investigate the possibilities to make a general tool for automatic presentation of XML documents on several different media using stylesheets. The integration of such a tool with a web server and how that affects performance will also be investigated.

With a general presentation tool based on XML and stylesheets, each document is stored as XML and one stylesheet is defined for each combination of document class and presentation medium. Defining a stylesheet is not trivial, but considerably easier than developing an ad-hoc application in a general-purpose programming language.

1.4 Relations to other work

A major part of the work has already been done by the World Wide Web Consortium (W3C) by developing and publishing the specifications for XML and several related standards (for stylesheets and many other things). More information about W3C can be found on their web site at `http://www.w3.org/`.

Lindholm [1] has made investigations on a related subject, limited to screen viewing and conversion from XML to HTML in a web server. This report will extend the study to other media than computer screen, in particular paper, and stress the importance of presentation of the same document on several media. This report will also continue the study of the performance problem when doing XML processing on a web server.

Chapter 2

XML and related technology

This chapter contains a brief introduction to XML in general and some related technologies used in this report.

2.1 XML

2.1.1 Background

SGML [2] is a meta-language for structured documents, standardized by ISO. HTML [3] is the markup language used on the web and it is defined as an application of SGML. As HTML is a specific application of SGML, it lacks the flexibility to be adapted for different document classes. The SGML standard is big and complicated and most tools for SGML processing are expensive, this prohibits widespread use on the web. W3C thus developed XML as a subset of SGML, simple enough to gain widespread use. XML is fully defined by the freely available specification [4], and several free tools for XML processing (see section 4.1) exist.

A good overview of XML, its history and how it is related to SGML and HTML can be found in [5].

2.1.2 Basics and well-formed documents

An XML document consists of *elements*. An element is delimited by a *start-tag* (`<element>`) and an *end-tag* (`</element>`), and may contain text and other elements. An element may have associated *attributes*, given in its start-tag (`<element attr="value">`). If an element does not have any content, its end-tag may be omitted if a special form of start-tag is used (`<element/>`). There must be exactly one top-level element (called the *root element*) which contains the rest of the document.

An XML document that meets some basic criteria defined in the XML specification [4], is said to be *well-formed*. In particular, it is required that all elements are explicitly closed (this is different from SGML and HTML where an element in some cases can be implicitly closed) and properly nested. Properly nesting means that the last opened element must be closed first, i.e. `<a><b>...</b></a>` is properly nested but `<a><b>...</a></b>` is not.

Almost all XML tools produce and expect well-formed XML documents. See figure 2.1 for an example of a well-formed XML document.

```
<?xml version="1.0"?>
<!-- This is a comment -->
<document>
<title>Well-formed XML document</title>
<body>
<p>This is a <emph>well-formed</emph> XML document.</p>
<img file="xml.png" /> <!-- An empty element with an attribute -->
</body>
</document>
```

Figure 2.1: Example of a well-formed XML document

2.1.3 Document classes

A document class is an (infinite) set of documents that are similar in structure. Examples of document classes are technical reports, invoices, press releases and minutes.

With XML, a DTD (Document Type Definition) is defined for each class of documents. A DTD defines all elements that can be used along with their attributes and the order that they may appear in. An XML document can be automatically validated against the formal rules in a DTD, this is done by some XML parsers. In companion with the formal DTD, it is useful to have a description of the semantic meaning connected to all the markup tags and guidelines on how to use them to build well-structured documents.

2.1.4 Valid documents

If a well-formed XML document has an associated DTD and complies with the structural rules defined in this DTD, it is said to be *valid*. Most XML tools do not require XML documents to be valid.

It is important to realize that well-formedness is a property of the XML document itself, independent of any DTD. Validity, on the other hand, is a relation between an XML document and a DTD. An XML document valid with respect to one DTD is probably not valid with respect to another DTD. Figure 2.2 contains an XML document valid with respect to the DTD in figure 2.3.

2.1.5 Namespaces

In many applications, it may be useful to have XML documents containing elements and attributes defined for several different purposes (and thus defined in different DTDs).

For example, when designing a document class for failure reports, a rigid structure for the time, location and type of failure is necessary, but a part for describing the problem is to be free-form text. In the problem description part, it is useful to allow some existing standard for general text markup to be used for paragraphs, emphasis and other general text properties.

Mixing elements and attributes from different standards in the same document gives rise to some problems. Name collision (two elements with different purposes have the same name) is handled by the standard for Namespaces in XML [6]. Using elements defined in several DTDs in the same document does not work well with the usual validation process in XML [4].

```
<?xml version="1.0"?>
<!DOCTYPE invoice SYSTEM "invoice.dtd">
<invoice>
<no>126</no>
<date>2000-02-14</date>
<from>The Green Garden Inc.</from>
<to>John Doe</to>
<main>
<row><qty>1</qty><item>Lawnmover</item>
     <price currency="USD">120</price></row>
<row><qty>2</qty><item>Gloves</item>
     <price currency="USD">15</price></row>
</main>
</invoice>
```

Figure 2.2: Example of a valid XML document

```
<!ELEMENT invoice (no,date,from,to,main)>
<!ELEMENT no (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT main (row)+>
<!ELEMENT row (qty,item,price)>
<!ELEMENT qty (#PCDATA)>
<!ELEMENT item (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST price
          currency NMTOKEN #REQUIRED>
```

Figure 2.3: DTD for the valid XML document in figure 2.2

2.1.6 XML Schema

Due to the limitation of the DTD model (especially its inability to handle namespace mixing), it has become common not to perform any validation and perhaps not even define a complete formal DTD. While XML can be used without DTDs and validation, many applications can benefit from automatic validation against formal structure rules. To remedy this problem, W3C is working on a more powerful and flexible model for XML document validation, called XML Schema [7]. The XML Schema standard is not yet finished, but is expected to replace DTD in the future.

2.2 XHTML

HTML is an application of SGML, and uses some features of SGML not present in XML. Thus, an HTML document is not (usually) a well-formed XML document. To overcome the gap between HTML and XML, W3C has developed XHTML [8], which is a reformulation of HTML in XML. XHTML is as powerful as HTML and has the same features, but the syntax is somewhat different (and more strict). However, the

XHTML specification contains guidelines for a syntax that is both well-formed XML and accepted as HTML by most recent web browsers.

Future development of HTML by W3C will be XHTML based, and thus XML compatible.

2.3 Cascading Style Sheets

Cascading Style Sheets (CSS) is a way to add formatting properties to elements in HTML or XML documents.

CSS level 1 (CSS1) [9] was designed for adding formatting properties to elements in HTML documents. These properties are primary applicable for presentation on a continuous (non-paged) visual medium (such as a common graphical web browser). Figure 2.4 contains some CSS.

```
H1, H2, H3 { text-align: left; font-weight: normal; }
UL LI { font-size: 12pt; }
UL UL LI { font-size: 10pt; }
```

Figure 2.4: Example of CSS

CSS level 2 (CSS2) [10] can be used for XML as well as HTML. CSS2 has a notation for different media and includes formatting properties for paged visual media (printing) and aural media (speech synthesis).

2.4 Extensible Stylesheet Language

The Extensible Stylesheet Language (XSL) consists of two parts, XSL Transformations (XSLT) and XSL Formatting Objects (XSL:FO). XSL was originally developed as one standard, but has now diverged into two separate standards. XSLT and XSL:FO can be used together, but can also be used independently of each other.

2.4.1 XSLT

XSLT [11] is a language for specifying how a class of XML documents is to be transformed into other documents. The output document can be XML (usually another document class than the input), HTML or plain text. XSLT is very powerful and allows the content of the input document to be filtered, reordered, sorted and duplicated. Arbitrary static content can be inserted anywhere in the output document. The output may have a completely different structure then the input

Since XSLT usually is used for formatting and presentation of XML documents, a transformation specification is called a stylesheet. An XSLT stylesheet is stored as an XML document.

XSLT uses a language called XPath [12] to select the information to be included in the output document. In addition to the main input document, information can be fetched from other XML documents identified by URLs.

2.4.2 XSL:FO

XSL:FO [13] is a language used to describe the formatting and presentation of information on some medium. Currently, XSL:FO supports the same set of media as CSS2.

XSL:FO data is usually generated by an XSLT transformation, directly interpreted to produce the desired presentation and then discarded. But XSL:FO can also be stored as an XML document.

2.5 XML APIs and software components

There exist two quite different standardized APIs for handling XML documents, DOM and SAX. It is possible to construct XML applications by composing several software components communicating using DOM and/or SAX.

There exist several common types of components for XML handling. An *XML parser* reads an XML document from a file or network connection (usually an URL) and provides its data to another component for further processing, the document can optionally be validated. An *XSLT processor* takes one stylesheet and one input document and produces one transformed output document. A *Formatting object processor* takes XSL:FO data and performs the presentation on some medium, either directly or by producing an intermediate format (such as Postscript or PDF for printing). An *XML serializer* takes XML data from another component and writes it to a file or network connection.

2.5.1 DOM

DOM (Document Object Model) [14] is an API for XML and HTML documents defined by W3C. The idea is to store the entire document in some data structure in primary memory, and DOM provides an interface to access the data in a tree-like way. The in-memory document representation can be both inspected and modified.

A major usage area for DOM is to view and manipulate a document in an interactive browser (such as "dynamic HTML" in a web browser). However, DOM can also be used as a general API to connect different components in an XML application.

2.5.2 SAX

SAX (Simple API for XML) [15] is quite different from DOM. The idea is to process the XML document sequentially by generating a stream of events while the document is read. With SAX, the entire document does not have to be stored in primary memory.

The main usage for SAX is for an XML parser to provide the XML data to some other component, but it can also be used to communicate XML data between other components.

Chapter 3

Analysis and design issues

In this chapter, different technologies and approaches for stylesheets, XML APIs and web server integration (including caching) will be discussed.

3.1 Stylesheet technology

3.1.1 CSS and XSL

I have evaluated two stylesheet technologies, CSS2 [10] and XSL (consisting of XSL Transformations, XSLT [11] and XSL Formatting Objects, XSL:FO [13]).

The support for different media and set of formatting properties are quite similar in CSS2 and XSL:FO. In fact, many XSL:FO properties are identical to CSS2 properties.

The main difference between CSS and XSL is that CSS only assigns formatting properties to the elements in the source document, while XSL also gives you transformation (XSLT). CSS2 has a quite powerful model for selecting elements, although not as powerful as XPath used by XSLT. With CSS2, you can filter out elements by assigning a "display: none" property to them, and you can insert static content between elements by assigning a "content" property to them. However, CSS2 does not allows you to reorder, sort or duplicate elements in the source document. CSS also cannot transform XML documents into HTML or some other XML format (such as WML [16]).

The ability of XSLT to reorder, sort and duplicate source elements (in addition to filter out source element and insert static content) is the main reason for me to choose XSL (or rather XSLT, as explained below) as the stylesheet technology. Without such features, it would not be possible to, for example, automatically generate a table of contents for a report.

Even though XSL is chosen as the main stylesheet technology, it can be useful to add CSS formatting when transforming into HTML.

3.1.2 Handling of different output media

The input documents are always transformed using XSLT. However, the transformation does not necessary generate XSL:FO. What format to generate depends on the output medium. Because of this, it is justified to say that the main stylesheet technology to use is XSLT (not XSL), and XSL:FO is just one possible output format.

For desktop computer screens, it seems like the best format is HTML with CSS formatting. Theoretically XSL:FO could be used, however, I am not aware of any implementation that presents XSL:FO on computer screens in a reasonable way. When using HTML, a common web browser can be used, and almost all network connected PCs and workstations do have a web browser installed.

For hand-held devices (such as mobile phones or PDAs), either HTML (probably different than for desktop computers) or WML [16] can be used, depending on the device's capabilities.

For printing, XSL:FO can be useful. There do exist some implementations that processes XSL:FO for printing (usually by producing PDF). Another alternative is to generate LaTeX code (using the plain text output method of XSLT). It is also possible to use HTML with CSS2 formatting (different than for screen viewing), but current web browsers generate printouts of very poor quality and do not implement the paged media support of CSS2.

3.2 XML API

There are two standardized APIs for connecting XML software components, SAX and DOM. It might seem like SAX is more efficient since the whole document does not have to be stored in primary memory, and processing can be done in an incremental way. However, XSLT processing usually cannot be done incrementally (because its ability to arbitrary reorder the source document) so the whole document is likely to be stored in primary memory anyway, inside the XSLT processor. However, the XSLT processor might prefer an internal XML representation different from DOM. Also, some other kinds of XML processing can be done incrementally and benefit from the SAX model.

I guess that SAX in most cases will be at least as efficient as DOM.

3.3 Web server integration and caching

Implementing transformation of XML documents in a web server raises some performance issues. The results in [1] show that the performance is unacceptably poor if the transformation is done for each request on a heavily loaded server, however, if caching is used the performance is greatly improved. This leads to the conclusion that it is not feasible to solve the problem by optimizing the transformation process, the principal solution is to develop a clever caching system so that the transformation does not have to be done for each request.

Today it is common to have dynamically generated documents, i.e. a request starts a process that generates the document. This is the current situation for the HTML based web, and the need for dynamically generated documents will remain also when using XML. Dynamically generated documents makes caching more difficult, but not always impossible.

To be useful in the context of a web server, an XML transformation system must include support for caching, even for dynamically generated documents. It is not acceptable to do the full XSLT transformation and possibly post-processing (such as transforming XSL:FO to PDF) for each request, since that would degrade performance significantly when the server is heavily loaded.

3.3.1 Different types of caching

Caching of web content can be done in several places, in the server, in the client or in a proxy somewhere between the client and the server. The purpose of server caching is to reduce the time it takes for the server to handle a request. The primary purpose of client and proxy caching is to limit the use of network bandwidth, but it will also reduce the load on the server. In this report, I will only consider server caching.

If the content generating process has several steps, it is possible to cache an intermediate result. It is also possible to cache parts of the content and add some missing information during each request. However, I will only consider the simple case of caching the complete final result (i.e. exactly what will be sent to the requesting client).

3.3.2 Static and dynamic content

The basic functionality of a web server is to send the content of a regular file stored on disk as the response of a request, this is called static content. The other alternative is to start a process that generates the response for each request, this is called dynamic content. The use of dynamic content can be divided into several categories as follows.

Pseudo-dynamic where the requested document is generated by composing information from several files and/or from a database. The information in the files and in the database is static, i.e. updated in a controlled manner and not very frequently. This is called "pseudo-dynamic" since the produced document is a function of static information only. In principle, this use of dynamic content is not necessary since the updates could have been done in static content. The main reason for using this approach is easier maintenance and updates. Technologies such as SSI[1], ASP[2], PHP[3] and JSP[4] are used for this.

Real-time data where the generated document depends on some information that is updated frequently and outside the control of the web server. Technologies such as CGI[5] and Servlets (sometimes also ASP, PHP and JSP) are used for this.

User interaction where the generated document depends on parameters in the request and/or state information from previous requests. Technologies such as CGI and Servlets (sometimes also ASP, PHP and JSP) are used for this.

It is also possible to have combinations (such as both real-time data and user interaction).

3.3.3 Caching possibilities

When the content is static, caching is straightforward. When the transformation is done, the result is saved and reused for subsequent requests. The transformation has to be redone only when the input document (or stylesheet) is updated, which in most cases happens much less frequently (perhaps twice a day) than requests (possibly several per second). Updates can be detected by inspecting the timestamp on the input files.

[1]Server Side Includes
[2]Active Server Pages
[3]`http://www.php.net/`
[4]Java Server Pages
[5]Common Gateway Interface

Pseudo-dynamic content can be handled in the same way. The content generation and transformation is done only when the information has been updated. If the input only consists of regular files, this is easily done by examining the timestamps of the files. The XSLT transformation can be seen as just a step in the pseudo-dynamic content generation process.

Real-time data is more difficult to handle. One way is to perform the transformation at regular intervals, where the choice of interval is a tradeoff between performance (short intervals can degrade performance) and how old information the user will accept. Another way to handle this is to actually detect when the information is updated and perform the transformation each time.

User interaction is the most difficult case. Since the response is a function of request parameters, each response may be unique and cannot be cached (if only the complete final result is cached).

3.4 Different approaches to web server integration

3.4.1 Web server extension

The usual approach to integrate dynamic content processing and XML transformations with a web server is to have a web server extension (such as a Servlet). The web server extension is invoked each time a request is made and has to generate the desired response, either by performing the necessary processing and/or transformation, or by using previously cached data. This is probably the only reasonable way it can be done when user interaction is involved. However, for pseudo-dynamic content (including XML transformations) and some cases of real-time data, there is another, possibly better, approach.

3.4.2 Offline content generation

The other approach is to let the web server do what it does best, send the content of regular files. The files are generated by a content generating process, separate from the web server and independent of the requests to the web server. The content generating process performs the necessary dynamic content processing and XML transformations, and stores the result in regular files. The content generating process is only executed when necessary, i.e. when the information is updated. Real-time data can also be handled if there is a way to get notified each time it is updated, or by periodically processing the data. Caching is inherent with this approach. This approach can give better performance than the web extension approach since the web server only has to send the content of a regular file for each request. If the content generation has to be done often, it may affect the performance of the web server. However, it is easy to move the content generating process to another machine, and let it communicate with the web server using a distributed file system (or possibly FTP).

Chapter 4

Practical use of stylesheets

In this chapter, some useful tools for XML and stylesheet processing will be listed, and a simple case study will be given.

4.1 Tools

In order to work with XSLT stylesheets, some tools are needed. The basic requirements are an XML parser and an XSLT processor. A serializer is used to get the output on a file (however, most XSLT processors come with a built-in serializer). When using XSL:FO, a formatting object processor is used to process the produced XSL:FO and print it or transform it to some easily printable format, such as Postscript or PDF.

To view HTML output on screen, a common web browser is used. Preferably, a browser with good support of CSS1 should be used, such as Internet Explorer 5.x. Netscape Navigator 4.x also works, but its output quality is not as good.

4.1.1 XML parsers

Xerces, by The Apache XML Project[1], is a free XML parser written in pure Java and thus portable. It can deliver the XML data as SAX or DOM and optionally validate the input. The Xerces distribution also includes a serializer, supporting SAX and DOM input and can output as XML, HTML, XHTML and plain text. Xerces is suitable as a component for building XML applications in Java. Xerces is almost the same product as IBM XML4J.

As Xerces works fine, I have not investigated the market further. However, another popular parser is XP by James Clark[2].

4.1.2 XSLT processors

Xalan, by The Apache XML Project, is a free XSLT processor written in pure Java and thus portable. It can either use its own special interface to Xerces, or take the input as SAX or DOM. The output can be produced as SAX, DOM or written to a file using the

[1] `http://xml.apache.org/`
[2] `http://www.jclark.com/xml/`

built-in serializer (which can output XML, HTML and plain text). Xalan can be used as a component in an XML application, but it also comes with a command line interface so it can be used to transform files without any additional Java programming. Xalan claims to be a complete implementation of the XSLT recommendation [11] and is thus capable to do any transformation. Xalan is almost the same product as LotusXSL.

As Xalan works fine, I have not investigated the market further. However, another popular XSLT processor is XT by James Clark.

4.1.3 Formatting object processors

FOP, by The Apache XML Project, is a free Formatting Objects processor written in pure Java and thus portable. FOP produces PDF output. It can be used as a component in an XML application and take input as SAX or DOM, but it also comes with a command line interface so it can be used to process XSL:FO files (in that case it needs an XML Parser, such as Xerces). Currently, FOP is not a finished product, it does not implement the latest XSL:FO working draft and it produces output of poor quality.

PassiveTeX by Sebastian Rahtz[3], is a free macro package for TEX. To use PassiveTeX, you need a working TEX system. It is difficult to get PassiveTeX to work properly. PassiveTeX cannot handle SAX or DOM, it expects the XSL:FO data from a file. PassiveTeX produces output of quite good quality, but does not implement all features of the latest XSL:FO working draft.

I have not been able to test any other formatting object processors. However, the XSL:FO specification is still a working draft. When W3C has finalized the XSL:FO specification, I expect that some Formatting Objects processors that produce reasonable quality output will appear.

4.2 Case study: Technical reports

I have designed a simple DTD for technical reports, see appendix A. A real-world application would probably need a more complex DTD, but this serves well as an example. I have also designed three stylesheets for this DTD.

One stylesheet transforms to HTML with CSS formatting for screen viewing, see appendix B. The second stylesheet transforms to XSL:FO for printing, see appendix C. The third stylesheet transforms to LATEX code for printing, see appendix D.

[3] http://users.ox.ac.uk/~rahtz/passivetex/

Chapter 5

XotW - XML transformations for the web

To test if the offline content generation approach works and give better performance in practice, I have developed a prototype and tested it against an implementation of the usual web server extension approach. I call the offline content generation prototype "XML on the Web" (or XotW for short).

5.1 How XotW works

XotW is an implementation of the offline content generation approach (see section 3.4.2).

5.1.1 The sitemap and producers

XotW is based on a *sitemap*, which is a file describing the structure of the website. The sitemap is in XML format, see appendix F for the DTD and appendix G for an example. The sitemap has entries for directories and files, each directory entry contains other file and directory entries, and each file entry describes how that file should be created.

A file is created with a pipeline of connected *producers*. A producer is a component which produces a stream of bytes (implemented using a Java OutputStream) or a stream of XML data (implemented using SAX events). A producer may additionally take a stream of bytes or XML data as input (if it does not, it is a *source producer*). A pipeline is a chain of connected producers with a byte producer in one end and a source producer in the other end.

XotW has defined four types of producers:

Format which inputs XML data and outputs bytes. Will typically be used to apply a serializer as the last step in the pipeline.

Transform which inputs XML data and outputs XML data. Will typically be used to perform XSLT transformations.

Source which is a source producer outputting XML data. Can be a XML parser that reads a source file.

Copy which is a source producer outputting bytes. Will typically be used to copy a source file unchanged (useful for e.g. images).

Parameters can be passed to a producer, which is useful for e.g. giving the name of the stylesheet to a XSLT processor. Each file entry has an input file, which can be read by the source producer (however, a source producer may instead obtain the data from a database or some other source).

When XotW is started, the sitemap is parsed and a pipeline is set up for each file entry. XotW is now ready to build the website, which can be done several times since the pipelines are reusable.

The website building is performed by recursively traversing all directory entries, and processing each file entry encountered. A file entry is only processed if necessary, i.e. if any source data has been updated since the last time it was processed. A file is processed if the target file does not exist or if the source file was modified more recently than the target file. In addition, XotW asks each producer in the pipeline whether processing is necessary (most producers ignore this and always answer "no", but a XSLT processor can make use of this to force processing if the stylesheet is updated).

5.1.2 Extra feature: wildcards

You can specify similar treatment of several files by using a wildcard in the source filename. XotW will enumerate all source files matching the given wildcard pattern, and process all of them in sequence. If the target filename also contains a wildcard, it will be instantiated with the same string used to match the source pattern, i.e. if the source pattern is `*.xml` and the target is `*.html`, then the source file `book.xml` will generate `book.html`.

5.1.3 Extra feature: splitting

There is a special kind of producer called *split* which works as a transform, but can generate several output files from one single source. The pipeline after a split producer will be executed once for each part the split producer generates (it has to generate a filename for it as well).

5.1.4 Program structure

XotW is written in pure Java and consists of the packages `xotw`, `xotw.core` and `xotw.processors`. See appendix E for an UML class diagram of the `xotw.core` package.

The `xotw.processors` contains a set of fundamental producers. This includes parsing XML from a file, copying the contents of a file, applying XSLT transformation, serializing into different formats, interpreting formatting objects and a simple way of splitting an XML stream by outputting each element of a specified type to its own file.

5.1.5 Running XotW

XotW requires a Java 1.1 compatible runtime environment and makes use of the XML tools Xerces, Xalan and FOP from Apache. It has been tested on OS/2 Warp with IBM JDK 1.1.8, RedHat Linux 6.1 with Sun JDK 1.2.2 and Windows NT Server 4.0 with Sun JDK 1.3. Everything works fine except that the splitting feature does not work properly on JDK 1.3 on Windows. This is probably because the splitting feature is the only part of XotW that makes use of multithreading. Since XotW is only a prototype, I did not find it worthwhile to fix this problem.

XotW is used by instantiating the class `xotw.core.XotWProcessor`, this makes it easy to integrate XotW with another Java application. There is also a simple command line interface in the class `xotw.XotW` which is used to run XotW standalone. The command line interface can be used to build the website either only once, or to set up for periodically building at a specified interval.

5.2 Performance test of XotW

Cocoon[1] is an implementation of the web server extension approach (see section 3.4.1) in the form of a Servlet. Cocoon is a popular tool for XML transformation in a web server and it is free.

I have measured the performance of XotW in comparison with Cocoon 1.7.3. Like XotW, Cocoon is a pure Java program which makes use of external components for XML parsing and XSLT transforming. I was able to test XotW and Cocoon in the same Java environment and with the same XML parser and XSLT processor. However, one important difference is that XotW uses SAX, while Cocoon uses DOM.

5.2.1 Test environment

The test server is a IBM PC330 with Intel Pentium 100 MHz CPU and 80 MB RAM. It has the following software configuration:

Operating system RedHat Linux 6.1

Web server Apache server 1.3.9

Java runtime Sun JDK 1.2.2

Servlet engine Apache JServ 1.1

XML parser and serializer Apache Xerces 1.0.3

XSLT processor Apache Xalan 1.0.1

Linux is a popular operating system for servers, and Apache is one of the most used web server. All software used in my test are gratis.

The test was performed, using Apache JMeter 1.4, from another computer connected to the server with an 10 Mbit Ethernet LAN. No other computers were connected to the LAN.

It should be noted that a real web server is much faster than my test server.

[1] `http://xml.apache.org/cocoon/`

5.2.2 Test scenario and setup

The test scenario is to display stock quotes in real-time on the web. I was not able to get real stock quotes into the test environment, so I wrote a test program that simulates changing stock quotes. The program keeps an internal database of a number of stocks and each second it makes some changes to the quotes of some stocks. Every 120th second, the test program writes the stock quote data onto three XML files on disk. This test program runs in the background on the server. In a real application, this program would obtain real stock quotes from a connection to a stock exchange.

When testing Cocoon, the server is set up to allow requests of these XML files, and let Cocoon handle the request by applying a XSLT stylesheet which transforms to HTML.

When testing XotW, the test program is modified to invoke XotW each time the XML files are written. XotW transforms the XML files into HTML (using a similar XSLT stylesheet as above) and writes three corresponding HTML files which can be requested normally via the web server.

As a reference, I also tested the web server performance by requesting a static HTML file without the test program running in background. This static HTML file was created by a previous run of XotW, so this can be considered as pseudo-dynamic content.

In all test runs, one of the three stock quote lists is requested. The size of the requested data is about 20 KB.

5.2.3 Error sources

During some preliminary testing, I have identified and eliminated a number of error sources when measuring the performance of a web server.

DNS lookup Sometimes the time to perform DNS lookup will affect the test result. I have eliminated this by using an explicit IP address in the URL.

Irregular network load If the load on the network between the client and the server is irregular, this may affect the test result. I have eliminated this by performing the test on an isolated LAN.

Other processes on the server Other processes on the server which consumes considerable resources, such as OpenGL screen savers on Windows NT, may affect the test result. I have tried to eliminate this by not running any unnecessary processes on the test server, and not using any screen saver.

When I did some preliminary testing without eliminating these error sources, the results became very unstable and not reproducible, thus not suitable for comparing two different server configurations. After eliminating these error sources, the test result became quite stable and reproducible. However, this also makes the tests unrealistic, because the normal web user will experience the effects of slow DNS lookup, irregular network load and sometimes servers with irrelevant processes running.

However, the purpose of this test is to compare two different server configurations, not to obtain some “absolute” value of the performance. So this unrealistic test setup is justified.

5.2.4 Test results

For each test setup, I made two test runs, one “light” with one thread doing a request every other second, and one “heavy” with ten threads doing a request every second. A real-world web server can be much more heavily loaded than in this test case. However, since the test server is considerably slower than a normal web server, I choosed a load it can handle.

The results, as average response time in seconds after 6 minutes of testing, is summarized in the table below.

	Light load	Heavy load
static HTML	1.6	2.1
XotW	2.2	2.5
Cocoon	3.9	8.7

As expected, the static HTML gives the best performance. The results also clearly show that XotW gives better performance than Cocoon.

5.2.5 Test conclusions

The results show that the offline content generation approach is an interesting alternative to the usual web server integration approach for handling real-time data. We can not conclude that it is always superior though, since shorter update interval or other different conditions might give other results.

It should, however, be clear that the offline content generation approach is superior for pseudo-dynamic content. As mentioned above, the static HTML can be considered as pseudo-dynamic as it was created by XotW.

Chapter 6

Conclusions

XML provides a useful framework for storing and processing structured documents. By using stylesheets, XML documents can be automatically presented on different media. XSLT is a powerful stylesheet technology for producing high-quality presentations on different media by transforming XML documents into some format suitable for presentation. Suitable presentation formats are HTML with CSS for desktop computer screens and HTML or WML for hand-held devices. However, it is not clear how to handle the important case of printing. XSL:FO is a promising technology, but not yet finished and current implementations give poor quality. Transforming into LaTeX source might be an alternative.

It is easy to start doing XML and XSLT processing; a set of free and ready-to-use tools can be obtained from The Apache XML Project. However, it is difficult to get printing of reasonable quality.

It is useful to integrate XML and stylesheet processing with a web server, but it must be done carefully not to degrade the performance. It is essential to have a clever caching system to avoid doing stylesheet processing for each request. For static, pseudo-dynamic and even some cases of real-time content, my proposed offline content generation approach gives better performance than the usual web server extension approach. This approach is useful also for pseudo-dynamic content processing not involving XML.

My initial goal of making a general tool for stylesheet processing turned out to be both too trivial and too difficult. Making a tool for output to HTML or some XML based format is a trivial composition of Xerces and Xalan (and a such is included with Xalan). However, making a tool that also can generate high-quality printouts is currently difficult because I have not found any useful formatting object processors and writing a new is way beyond the scope of this project.

Bibliography

References to W3C are official recommendations from the World Wide Web Consortium and can be found at `http://www.w3.org/TR`.

[1] Lindholm, David: *Displaying data stored as XML — a study of different possibilities to display XML-documents*, NADA/KTH 1999, TRITA-NA-E9969.

[2] ISO (International Organization for Standardization): *ISO 8879:1986(E). Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML).* First edition – 1986-10-15.

[3] W3C: *HTML 4.01 Specification*, REC-html401-19991224.

[4] W3C: *Extensible Markup Language (XML) 1.0*, REC-xml-19980210.

[5] Bradley, Niel: *The XML companion, second edition*, Addison-Wesley 1999, ISBN 0-201-67486-6.

[6] W3C: *Namespaces in XML*, REC-xml-names-19990114.

[7] W3C: *XML Schema Requirements*, NOTE-xml-schema-req-19990215.

[8] W3C: *XHTML 1.0: The Extensible HyperText Markup Language*, REC-xhtml1-20000126.

[9] W3C: *Cascading Style Sheets, level 1* REC-CSS1-19990111.

[10] W3C: *Cascading Style Sheets, level 2* REC-CSS2-19980512.

[11] W3C: *XSL Transformations (XSLT) Version 1.0*, REC-xslt-19991116.

[12] W3C: *XML Path Language (XPath) Version 1.0*, REC-xpath-19991116.

[13] W3C: *Extensible Stylesheet Language (XSL) Version 1.0* (working draft), WD-xsl-20000327.

[14] W3C: *Document Object Model (DOM) Level 1 Specification*, REC-DOM-Level-1-19981001.

[15] Megginson, David: *SAX 2.0: The Simple API for XML*, `http://www.megginson.com/SAX/`, 2000-05-05.

[16] WAP Forum: *Wireless Markup Language Specification Version 1.2*, SPEC-WML-19991104.

Appendix A

DTD for technical reports

```
<!-- DTD for technical reports -->

<!-- Text in a paragraph (or similar) -->
<!ENTITY % inline "(#PCDATA|em|cite|footnote)*">

<!-- Root Element Type -->
<!ELEMENT report (front, body, back)>

<!ELEMENT front (author, title)>

<!ELEMENT author (#PCDATA)*>

<!ELEMENT title %inline;>

<!ELEMENT body (chapter+)>

<!ELEMENT chapter (title, p*, section*)>

<!ELEMENT section (title, p*, subsection*)>

<!ELEMENT subsection (title, p*)>

<!ELEMENT p %inline;>

<!ELEMENT em %inline;>

<!ELEMENT back (bibliography?)>

<!ELEMENT bibliography (bibitem+)>

<!-- Bibliographic item -->
<!ELEMENT bibitem %inline;>
<!ATTLIST bibitem
    id ID #REQUIRED
    href CDATA #IMPLIED>    <!-- Optional external link -->

<!-- Bibliographic citation -->
<!ELEMENT cite EMPTY>
<!ATTLIST cite
    ref IDREF #REQUIRED>

<!ELEMENT footnote %inline;>
```

Appendix B

Stylesheet for technical reports to HTML

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!-- XSLT stylesheet for transforming the technical reports
     to HTML for screen viewing -->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<!-- Output as HTML -->
<xsl:output method="html" encoding="iso-8859-1"/>

<xsl:template match="/">
  <!-- Build the main structure of the produced HTML -->
  <html>
    <head>
        <title><xsl:value-of select="report/front/title"/></title>
        <meta name="Author" content="{report/front/author}"/>

        <!-- The produced HTML will have an embeeded CSS stylesheet -->
        <style type="text/css">
            h1.main { margin-bottom: 0.2em; }
            p.author { font-style: italic; margin-top: 0; }
            ul.toc { list-style: none; margin-top: 0.2em; }
            p.toc { margin-bottom: 0; font-weight: bolder; }
        </style>
    </head>
    <body>
        <h1 class="main"><xsl:value-of select="report/front/title"/></h1>
        <p class="author">by <xsl:value-of select="report/front/author"/>
        </p>
        <hr/>
        <h2>Table of Contents</h2>
        <!-- Build a TOC from chapters, sections and subsections,
             with hypertext links -->
        <xsl:for-each select="report/body/chapter">
            <xsl:variable name="chapnum">
                <xsl:number count="chapter" format="1"/>
            </xsl:variable>
            <p class="toc"><a href="#chap{$chapnum}">
            Chapter <xsl:value-of select="$chapnum"/> -
```

```
            <xsl:value-of select="title"/></a></p>
            <ul class="toc">
            <xsl:for-each select="section">
                <xsl:variable name="sectnum">
                    <xsl:number count="chapter|section"
                                level="multiple" format="1.1"/>
                </xsl:variable>
                <li><a href="#sect{$sectnum}">
                <xsl:value-of select="$sectnum"/><xsl:text> </xsl:text>
                <xsl:value-of select="title"/></a></li>
                <ul class="toc">
                <xsl:for-each select="subsection">
                    <xsl:variable name="subsectnum">
                        <xsl:number count="chapter|section|subsection"
                                    level="multiple" format="1.1.1"/>
                    </xsl:variable>
                    <li><a href="#subsect{$subsectnum}">
                    <xsl:value-of select="$subsectnum"/>
                    <xsl:text> </xsl:text>
                    <xsl:value-of select="title"/></a></li>
                </xsl:for-each>
                </ul>
            </xsl:for-each>
            </ul>
        </xsl:for-each>
        <hr/>
        <!-- The main part of the report -->
        <xsl:apply-templates select="report/body"/>
        <hr/>
        <xsl:if test="report/body//footnote">
            <h2>Footnotes</h2>
            <!-- Put the text of all footnotes here -->
            <xsl:for-each select="report/body//footnote">
                <xsl:variable name="footnotenum">
                    <xsl:number count="footnote" level="any"/>
                </xsl:variable>
                <a name="footnote{$footnotenum}"><p>
                    <sup><xsl:value-of select="$footnotenum"/></sup>
                <xsl:text> </xsl:text><xsl:apply-templates/></p></a>
            </xsl:for-each>
            <hr/>
        </xsl:if>
        <xsl:apply-templates select="report/back"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="chapter">
    <xsl:variable name="chapnum">
        <xsl:number count="chapter" format="1"/>
    </xsl:variable>
    <a name="chap{$chapnum}"><h2>Chapter <xsl:value-of select="$chapnum"/>
        <br/>
    <xsl:value-of select="title"/></h2></a>
```

```
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="section">
    <xsl:variable name="sectnum">
        <xsl:number count="chapter|section"
                    level="multiple" format="1.1"/>
    </xsl:variable>
    <a name="sect{$sectnum}"><h3><xsl:value-of select="$sectnum"/>
    <xsl:text> </xsl:text><xsl:value-of select="title"/></h3></a>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="subsection">
    <xsl:variable name="subsectnum">
        <xsl:number count="chapter|section|subsection"
                    level="multiple" format="1.1.1"/>
    </xsl:variable>
    <a name="subsect{$subsectnum}"><h4><xsl:value-of select="$subsectnum"/>
    <xsl:text> </xsl:text><xsl:value-of select="title"/></h4></a>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="p">
    <p><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="em">
    <em><xsl:apply-templates/></em>
</xsl:template>

<!-- Insert a footnote reference -->
<xsl:template match="footnote">
    <xsl:variable name="footnotenum">
        <xsl:number count="footnote" level="any"/>
    </xsl:variable>
    <sup><a href="#footnote{$footnotenum}">
    <xsl:value-of select="$footnotenum"/></a></sup>
</xsl:template>

<xsl:template match="title"/>

<!-- Insert a bibliographic citation, with a hypertext link -->
<xsl:template match="cite">
    <xsl:variable name="cref">
        <xsl:value-of select="@ref"/>
    </xsl:variable>
    <a href="#{@ref}">[<xsl:for-each select="//bibitem[@id=$cref]">
        <xsl:number/></xsl:for-each>]</a>
</xsl:template>

<!-- Build the bibliography -->
<xsl:template match="bibliography">
    <h2>Bibliography</h2>
```

```
    <table>
    <xsl:apply-templates/>
    </table>
</xsl:template>

<!-- Each bibliographic item -->
<xsl:template match="bibitem">
    <tr>
    <td><a name="{@id}">[<xsl:number/>]</a></td>
    <td>
    <xsl:choose>
    <xsl:when test="@href">
        <a href="{@href}"><xsl:apply-templates/></a>
    </xsl:when>
    <xsl:otherwise><xsl:apply-templates/></xsl:otherwise>
    </xsl:choose>
    </td>
    </tr>
</xsl:template>

</xsl:stylesheet>
```

Appendix C

Stylesheet for technical reports to XSL:FO

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!-- XSLT stylesheet for transforming the technical reports
     to XSL:FO for printing -->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                version="1.0">

<xsl:template match="/">
  <!-- Build the main structure of the produced XSL:FO -->
  <fo:root>
     <fo:layout-master-set>
      <fo:simple-page-master page-master-name="one" margin-left="3cm"
                                                    margin-right="3cm"
                                                    margin-top="2cm"
                                                    margin-bottom="2cm">
       <fo:region-before extent="12pt"/>
       <fo:region-body margin-top="20pt" margin-bottom="20pt"/>
       <fo:region-after extent="12pt"/>
      </fo:simple-page-master>
     </fo:layout-master-set>
    <fo:page-sequence>
    <fo:sequence-specification>
    <fo:sequence-specifier-single page-master-name="one"/>
    </fo:sequence-specification>
        <fo:flow>
        <fo:block font-size="24pt" text-align="centered" space-after="6pt">
            <xsl:value-of select="report/front/title"/>
        </fo:block>
        <fo:block text-align="centered">
            by <xsl:value-of select="report/front/author"/>
        </fo:block>
        </fo:flow>
    </fo:page-sequence>

    <fo:page-sequence>
        <fo:sequence-specification>
        <fo:sequence-specifier-repeating page-master-first="one"
                                         page-master-repeating="one"/>
```

```
</fo:sequence-specification>

<fo:static-content flow-name="xsl-before">
    <fo:block font-size="10pt">
        <xsl:value-of select="report/front/title"/>
    </fo:block>
</fo:static-content>

<fo:static-content flow-name="xsl-after">
    <fo:block font-size="10pt" text-align="centered">
        <fo:page-number/>
    </fo:block>
</fo:static-content>

<fo:flow font-size="12pt" font-family="serif">
<fo:block font-size="20pt" space-after="24pt">Table of Contents
</fo:block>
<!-- Build a TOC from chapters, sections and subsections,
     with page references -->
<xsl:for-each select="report/body/chapter">
    <xsl:variable name="chapnum">
        <xsl:number count="chapter" format="1"/>
    </xsl:variable>
    <fo:block>
    Chapter <xsl:value-of select="$chapnum"/> -
    <xsl:value-of select="title"/>...
    <fo:page-number-citation ref-id="chap{$chapnum}"/>
    </fo:block>
    <fo:list-block>
    <xsl:for-each select="section">
        <xsl:variable name="sectnum">
            <xsl:number count="chapter|section"
                        level="multiple" format="1.1"/>
        </xsl:variable>
        <fo:list-item><fo:list-item-label><fo:block/>
        </fo:list-item-label><fo:list-item-body>
        <fo:block>
        <xsl:value-of select="$sectnum"/><xsl:text> </xsl:text>
        <xsl:value-of select="title"/>...
        <fo:page-number-citation ref-id="chap{$sectnum}"/>
        </fo:block></fo:list-item-body></fo:list-item>
        <fo:list-item><fo:list-item-label><fo:block/>
        </fo:list-item-label><fo:list-item-body>
        <fo:list-block>
        <xsl:for-each select="subsection">
            <xsl:variable name="subsectnum">
                <xsl:number count="chapter|section|subsection"
                            level="multiple" format="1.1.1"/>
            </xsl:variable>
            <fo:list-item><fo:list-item-label><fo:block/>
            </fo:list-item-label><fo:list-item-body>
            <fo:block>
            <xsl:value-of select="$subsectnum"/><xsl:text> </xsl:text>
            <xsl:value-of select="title"/>...
```

```
                    <fo:page-number-citation ref-id="chap{$subsectnum}"/>
                    </fo:block></fo:list-item-body></fo:list-item>
                </xsl:for-each>
                </fo:list-block>
                </fo:list-item-body></fo:list-item>
            </xsl:for-each>
            </fo:list-block>
        </xsl:for-each>

        <!-- The main part of the report -->
        <xsl:apply-templates select="report/body"/>
        <xsl:apply-templates select="report/back"/>
        </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>

<xsl:template match="chapter">
    <xsl:variable name="chapnum">
        <xsl:number count="chapter" format="1"/>
    </xsl:variable>
    <fo:block break-before="page" font-size="20pt" id="#chap{$chapnum}">
        Chapter <xsl:value-of select="$chapnum"/>
    </fo:block>
    <fo:block font-size="20pt" space-after="24pt">
        <xsl:value-of select="title"/></fo:block>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="section">
    <xsl:variable name="sectnum">
        <xsl:number count="chapter|section"
                    level="multiple" format="1.1"/>
    </xsl:variable>
    <fo:block font-size="14pt" font-weight="bold" id="#sect{$sectnum}">
        <xsl:value-of select="$sectnum"/><xsl:text> </xsl:text>
        <xsl:value-of select="title"/>
    </fo:block>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="subsection">
    <xsl:variable name="subsectnum">
        <xsl:number count="chapter|section|subsection"
                    level="multiple" format="1.1.1"/>
    </xsl:variable>
    <fo:block font-weight="bold" id="#subsect{$subsectnum}">
        <xsl:value-of select="$subsectnum"/><xsl:text> </xsl:text>
        <xsl:value-of select="title"/>
    </fo:block>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="p">
```

```
    <fo:block><xsl:apply-templates/></fo:block>
</xsl:template>

<xsl:template match="em">
    <fo:inline-sequence font-style="italic">
        <xsl:apply-templates/>
    </fo:inline-sequence>
</xsl:template>

<!-- Insert footnote reference and text
     (XSL:FO will place it in the page footer automatically) -->
<xsl:template match="footnote">
    <xsl:variable name="footnotenum">
        <xsl:number count="footnote" level="any"/>
    </xsl:variable>
    <fo:footnote>
    <fo:footnote-citation><xsl:value-of select="$footnotenum"/>
    </fo:footnote-citation>
    <fo:block><xsl:value-of select="$footnotenum"/> <xsl:apply-templates/>
    </fo:block>
    </fo:footnote>
</xsl:template>

<xsl:template match="title"/>

<!-- Insert a bibliographic citation -->
<xsl:template match="cite">
    <xsl:variable name="cref">
        <xsl:value-of select="@ref"/>
    </xsl:variable>
    [<xsl:for-each select="//bibitem[@id=$cref]">
        <xsl:number/></xsl:for-each>]
</xsl:template>

<!-- Build the bibliography -->
<xsl:template match="bibliography">
    <fo:block font-size="20pt" space-after="24pt" break-before="page">
        Bibliography
    </fo:block>
    <fo:list-block><xsl:apply-templates/></fo:list-block>
</xsl:template>

<!-- Each bibliographic item -->
<xsl:template match="bibitem">
<fo:list-item>
    <fo:list-item-label id="{@id}"><fo:block>[<xsl:number/>]</fo:block>
    </fo:list-item-label>
    <fo:list-item-body><fo:block><xsl:apply-templates/></fo:block>
    </fo:list-item-body>
</fo:list-item>
</xsl:template>

</xsl:stylesheet>
```

Appendix D

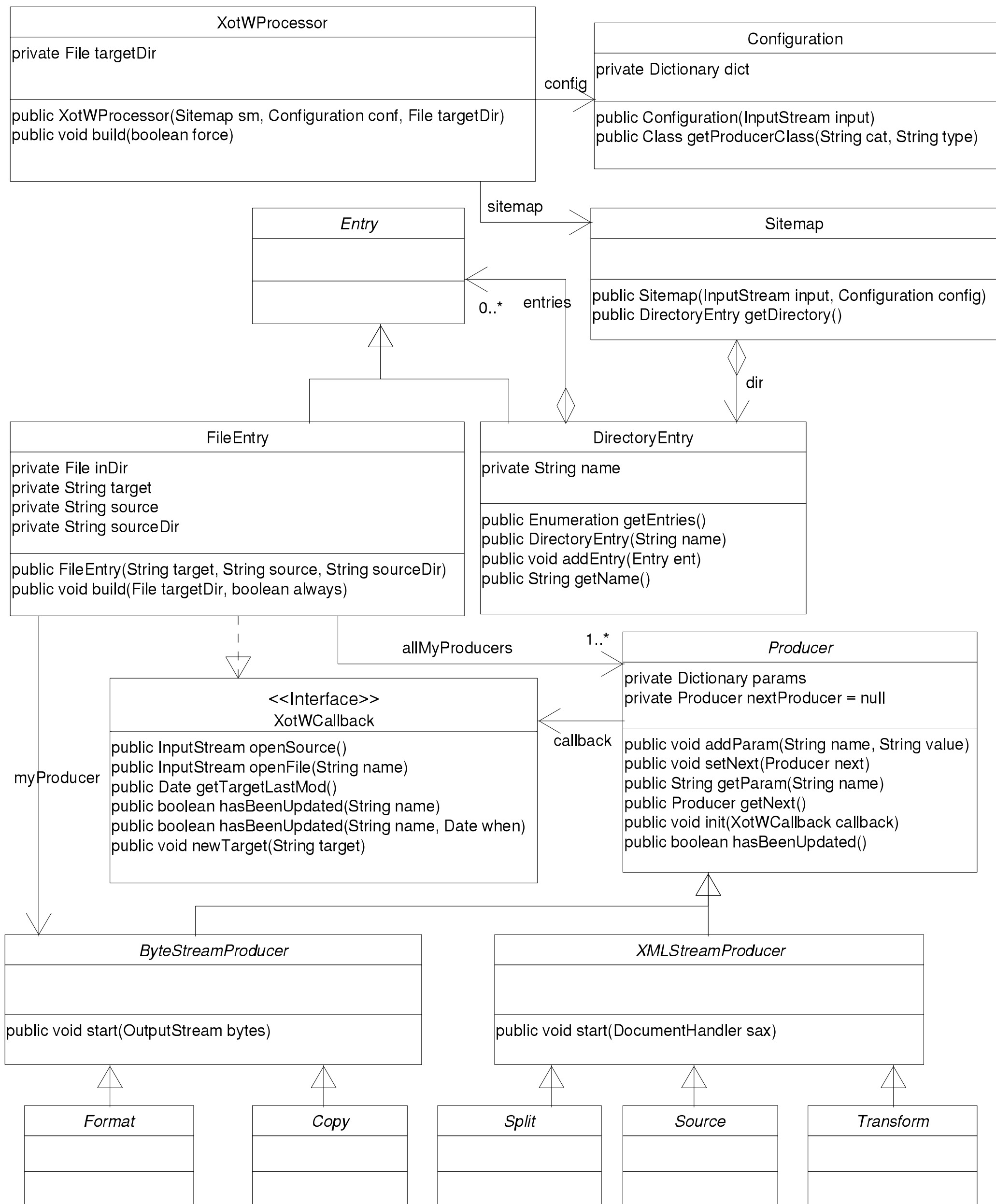
Stylesheet for technical reports to $\LaTeX$

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!-- XSLT stylesheet for transforming the technical reports
     to LaTeX for printing -->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<xsl:output method="text"
encoding="iso-8859-1" />

<xsl:template match="/">\documentclass[a4paper,10pt]{report}
\usepackage[T1]{fontenc}
\usepackage[latin1]{inputenc}

\author{<xsl:value-of select="report/front/author"/>}
\title{<xsl:value-of select="report/front/title"/>}

\begin{document}

\maketitle

\tableofcontents

\setlength{\parindent}{0pt}
\setlength{\parskip}{1ex plus 0.5ex minus 0.2ex}

<xsl:apply-templates select="report/body"/>
<xsl:apply-templates select="report/back"/>

\end{document}
</xsl:template>

<xsl:template match="chapter">
\chapter{<xsl:value-of select="title"/>}
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="section">
\section{<xsl:value-of select="title"/>}
```

```
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="subsection">
\subsection{<xsl:value-of select="title"/>}
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="p">
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="em">\emph{<xsl:apply-templates/>}</xsl:template>

<!-- Insert footnote reference and text -->
<xsl:template match="footnote">\footnote{<xsl:apply-templates/>}</xsl:template>

<xsl:template match="title"/>

<!-- Insert a bibliographic citation -->
<xsl:template match="cite">\cite{<xsl:value-of select="@ref"/>}</xsl:template>

<!-- Build the bibliography -->
<xsl:template match="bibliography">
\begin{thebibliography}{99}
<xsl:apply-templates/>
\end{thebibliography}
</xsl:template>

<!-- Each bibliographic item -->
<xsl:template match="bibitem">
\bibitem{<xsl:value-of select="@id"/>} <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

Appendix E

UML class diagram for XotW


XotWProcessor
private File targetDir
public XotWProcessor(Sitemap sm, Configuration conf, File targetDir)
public void build(boolean force)
config
Configuration
private Dictionary dict
public Configuration(InputStream input)
public Class getProducerClass(String cat, String type)
sitemap
Entry
Sitemap
public Sitemap(InputStream input, Configuration config)
public DirectoryEntry getDirectory()
0..*
entries
dir
FileEntry
private File inDir
private String target
private String source
private String sourceDir
public FileEntry(String target, String source, String sourceDir)
public void build(File targetDir, boolean always)
DirectoryEntry
private String name
public Enumeration getEntries()
public DirectoryEntry(String name)
public void addEntry(Entry ent)
public String getName()
allMyProducers
1..*
Producer
private Dictionary params
private Producer nextProducer = null
public void addParam(String name, String value)
public void setNext(Producer next)
public String getParam(String name)
public Producer getNext()
public void init(XotWCallback callback)
public boolean hasBeenUpdated()
<<Interface>>
XotWCallback
public InputStream openSource()
public InputStream openFile(String name)
public Date getTargetLastMod()
public boolean hasBeenUpdated(String name)
public boolean hasBeenUpdated(String name, Date when)
public void newTarget(String target)
callback
myProducer
ByteStreamProducer
public void start(OutputStream bytes)
XMLStreamProducer
public void start(DocumentHandler sax)
Format
Copy
Split
Source
Transform

Appendix F

DTD for XotW sitemap

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!-- DTD for XotW sitemap -->

<!--
format: xml-stream -> byte-stream
transform: xml-stream -> xml-stream
source: ? -> xml-stream
copy: ? -> byte-stream
split: xml-stream -> *xml-stream
-->

<!ELEMENT sitemap (directory) >
<!ATTLIST sitemap
    source CDATA #REQUIRED >

<!ELEMENT directory (file|directory)* >
<!ATTLIST directory
    name CDATA #IMPLIED > <!-- must be omitted for root dir,
                               included otherwise -->

<!ELEMENT file (format|copy) >
<!ATTLIST file
    target CDATA #REQUIRED
    source CDATA #REQUIRED
    force (force) #IMPLIED >    <!-- always rebuild this file -->

<!ELEMENT format (param*,(source|transform|split)) >
<!ATTLIST format
    type NMTOKEN #REQUIRED >

<!ELEMENT transform (param*,(source|transform|split)) >
<!ATTLIST transform
    type NMTOKEN #REQUIRED >

<!ELEMENT split (param*,(source|transform)) >
<!ATTLIST split
    type NMTOKEN #REQUIRED >

<!ELEMENT source (#PCDATA) > <!-- the #PCDATA is taken as a param with
                                  name "name" -->
<!ATTLIST source
    type NMTOKEN #REQUIRED >
```

```
<!ELEMENT copy (#PCDATA) >   <!-- the #PCDATA is taken as a param with
                                  name "name" -->
<!ATTLIST copy
    type NMTOKEN #REQUIRED >

<!ELEMENT param (#PCDATA) >
<!ATTLIST param
    name NMTOKEN #REQUIRED >
```

Appendix G

Example of XotW sitemap

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE sitemap SYSTEM "sitemap.dtd">

<sitemap source="theSourceDirectory">

<directory>

<file target="index.html" source="index.xml">
    <format type="html">
        <transform type="xslt">
            <param name="stylesheet">style/index.xsl</param>
            <source type="file"/>
        </transform>
    </format>
</file>

<directory name="html">
    <file target="*.html" source="pages/*.xml">
        <format type="html">
            <transform type="xslt">
                <param name="stylesheet">style/Page_html.xsl</param>
                <source type="file"/>
            </transform>
        </format>
    </file>
</directory>

<directory name="pdf">
    <file target="*.pdf" source="pages/*.xml">
        <format type="fop">
            <transform type="xslt">
                <param name="stylesheet">style/Page_pdf.xsl</param>
                <source type="file"/>
            </transform>
        </format>
    </file>
</directory>

<file target="picture.png" source="picture.png">
    <copy type="file"/>
</file>
```

```
</directory>
</sitemap>
```