

Using XML in Web Services

Vision of the Future

Timi Soinio

University of Tampere
Department of Computer and
Information Sciences
Master's thesis
June 2000

University of Tampere

Department of Computer and Information Sciences

Timi Soinio: Using XML in Web Services – Vision of the Future

Master's thesis, 97 pages + 20 appendix pages

June 2000

Abstract

World Wide Web (Web, for short) services have traditionally been produced with techniques that combine the content and presentation of the services. Dynamic content produced by Web applications also mixes programmatic logic into those services. For example, ASP (Active Server Pages) and JSP (Java Server Pages) are techniques where program code is embedded in HTML (Hypertext Markup Language) pages.

The combination of these three layers – content, logic, and presentation – poses some challenges to the production, delivery, and consumption of the Web services. The resulting services are often Web device dependent, have poor content re-use, and use the network throughput inefficiently. This thesis presents some solutions to overcome these challenges.

The solutions focus on Extensible Markup Language (XML) and its related techniques. XML can be used to describe pure content; the presentation is left to discrete style sheets. The majority of the current Web technologies have not properly supported the complete separation of content and logic. This thesis introduces one service platform where this problem has also been solved.

In order to make device independent and network-friendly Web services, it is necessary to have mechanisms to define the Web device and network properties. This thesis also presents some of these mechanisms.

Tiivistelmä

Tämän työn aiheena on World Wide Web (WWW)-palvelut. Nykyiset tekniikat WWW-palvelujen toteuttamiseksi ovat eräiltä osin ongelmallisia. Ne yhdistävät tiukasti toisiinsa kolme osa-aluetta – sisällön, muodon ja toiminnallisuuden. Tämä yhdistäminen aiheuttaa haasteita niin palvelujen tuottamiseen, levittämiseen kuin kuluttamiseenkin. Palvelut ovat usein suunnattuja tietyille päätelaitteille, niiden sisältö on huonosti uudelleenkäytettävää tai ne saattavat kuluttaa verkon kapasiteettia tarpeettoman informaation välittämiseen.

Työssä esitetään ratkaisuja näiden haasteiden voittamiseksi. Ratkaisut perustuvat XML-kieleen (Extensible Markup Language) sekä siihen liittyviin tekniikoihin. XML:ää voidaan käyttää pelkän informaation, sisällön, kuvaamiseen kun taas muoto eli ulkoasu voidaan ilmaista erillisten tyylimääryityksien avulla. Toiminnallisen logiikan erottaminen sisällöstä ei suurimmassa osassa nykyisistä järjestelmistä ole täydellisesti mahdollista. Tässä työssä esitellään kuitenkin eräs järjestelmä, jossa tämäkin pulma on ratkaistu.

Pelkät mekanismit laiteriippumattomien ja verkon käyttöä optimoivien palvelujen toteuttamiseksi eivät vielä riitä, tarvitaan lisäksi tapoja selvittää, millä perusteilla palvelut pitää kussakin tapauksessa muodostaa. Työssä esitetään myös joitakin tekniikoita, jotka on kehitetty WWW-sisällön muuntamiseksi vastaamaan asiakkaiden ja heidän käyttämiensä päätelaitteiden toiveita ja ominaisuuksia.

Keywords: Web services, Web applications, Web publishing, XML, XSL, HTML, WML

Acknowledgements

I greatly thank Nokia Mobile Phones for giving me the opportunity to do this work in the first place. Without the resources and support they have given me during the process, this thesis would not have been finished in its present form. I appreciate all the help and advice I have received from my colleagues. Special thanks go to Hannu Hakala and Tommi Ojala who have given me instructions on this work.

During the development and testing of the Schedule Board service, I have had valuable help and suggestions from other people in my working community. I would like to thank Hannu Lehtimäki, Hannu Mähönen, Salla Laurikka, and Anna Mari Salonranta for their time and comments.

A word of thanks goes to Tuija Sonkkila at the library of Helsinki University of Technology for inspiring me to write this study in a structured format. Based on the pointers she gave me, I chose the DocBook XML V3.1.3 for the DTD of this work. I am grateful to Jyrki Nummenmaa and Pertti Järvinen, the supervisors of this thesis at the University of Tampere. Many thanks also to Virginia Mattila for help and advice on language.

And finally, I thank my fiancée, Terhi, for her support and love during all those months I have struggled with this study.

Table of Contents

1. Introduction	1
1.1. Background	1
1.2. Research Problem and the Goals of the Thesis	1
1.3. Research Structure and Methods.....	2
2. Web Publishing and Web Applications.....	4
2.1. Web Publishing	4
2.2. Web Applications.....	5
2.2.1. Dynamic Content Creation Techniques	5
2.3. Existing Problems	6
2.3.1. Misuse of Structural Markup Languages	6
2.3.2. Device Dependency.....	6
2.3.3. Maintenance Problems	7
2.3.4. Poor Searchability	7
2.3.5. Accessibility Problems.....	8
3. Structured Information	9
3.1. Different Perspectives of Markup	9
3.2. XML - Extensible Markup Language	10
3.3. XML Structure and Syntax	11
3.3.1. Elements	12
3.3.2. Entity References	12
3.3.3. Comments	13
3.3.4. Processing Instructions.....	13
3.3.5. CDATA Sections.....	13
3.3.6. Document Type Declarations.....	14
3.4. Validity of XML documents	14
3.4.1. Well-formed XML	14
3.4.2. Valid XML	15
3.5. XML vs. SGML	15
3.6. Applications of XML	16
3.7. The Three R's of XML	18
3.7.1. Re-publish	18
3.7.2. Re-use.....	19
3.7.3. Re-purpose	20
3.8. Why Use XML in Web Applications?	21
3.8.1. Simplicity	21
3.8.2. Richness of Data Structure.....	21
3.8.3. International Character Handling.....	23
4. Styling XML Documents	24

4.1. DOM and CSS.....	24
4.1.1. DOM - Document Object Model	24
4.1.2. CSS - Cascading Style Sheets	25
4.1.3. Style with Programming.....	26
4.2. XSL - Extensible Stylesheet Language.....	27
4.2.1. XSLT - XSL Transformations.....	28
4.2.2. XSL Formatting Vocabulary.....	29
4.2.3. "Formatting Objects Considered Harmful".....	30
4.2.4. Using XSLT for Client-side and Server-side Styling.....	32
4.2.5. XPath.....	33
4.3. Using XSL and CSS Together	33
4.4. Rendering Languages.....	34
4.4.1. XHTML - Extensible Hypertext Markup Language	35
4.4.2. WML - Wireless Markup Language	36
4.4.3. VoiceXML	37
5. Customisations and Personalisations	40
5.1. Client-Specific Web Services by Using User Agent Attributes.....	40
5.2. RDF - Resource Description Framework.....	42
5.3. CC/PP - Composite Capabilities / Preferences Profile.....	43
5.3.1. CC/PP Framework.....	43
5.3.1.1. The CC/PP Data Model.....	43
5.3.2. CC/PP Exchange Protocol over HTTP.....	44
5.4. UAProf - User Agent Profiles	45
5.5. PIDL - Personalized Information Description Language.....	46
5.5.1. Progressive Storage of Processed Content	47
5.5.2. Compact Storage of Processed Content	49
5.6. P3P - Platform for Privacy Preferences.....	50
5.7. Conclusions	51
6. XML Clients and Servers	53
6.1. Client-side Formatting.....	53
6.1.1. Microsoft Internet Explorer.....	54
6.1.2. DocZilla.....	56
6.2. Server-side Formatting.....	57
6.2.1. XML Enabler.....	58
6.2.2. Rocket.....	59
6.2.3. Cocoon	60
7. Cocoon	62
7.1. What is Cocoon?	62
7.2. Cocoon's Infrastructure	63
7.2.1. Architecture.....	63
7.2.2. Cocoon Cache System.....	65
7.3. Dynamic Content in Cocoon.....	66

7.3.1. Servlet Chaining vs. Servlet Nesting.....	66
7.3.2. Producers and Processors	67
7.4. Cocoon's Advantages and Disadvantages	69
7.4.1. Why is Cocoon a good solution?	69
7.4.2. Has Cocoon any disadvantages?	70
8. Cocoon Service Implementation: Schedule Board.....	71
8.1. Background	71
8.2. Requirements.....	72
8.3. Implementation.....	73
8.3.1. Hardware and Software Environment	73
8.3.2. Schedule Board Service Design	74
8.4. Conclusions	80
8.4.1. The Nature of the Application.....	80
8.4.2. Room for Further Improvement	81
8.4.3. The Value of This Exercise.....	81
9. Summary	82
References	84
Glossary	92
A. Trilogy DTD.....	98
B. XSL and CSS.....	99
C. Inline CC/PP	101
D. CC/PP with Indirect References	102
E. Cocoon XSP.....	104
F. Cocoon DCP	105
G. Selected Schedule Board Source Files	106

List of Tables

3-1. Predefined Entities in XML	13
---------------------------------------	----

List of Figures

3-1. The Relation of XML to Its Applications in theUML Notation.	17
3-2. Re-publishing Information [Holman, 1999b].	19
3-3. Re-using Information [Holman, 1999b]	20
3-4. Re-purposing Information [Holman, 1999b].	20
3-5. An XML Document as a Tree Structure [Maruyama <i>et al.</i> , 1999, p. 43].	22
4-1. Styling by Transforming and Rendering.....	24
4-2. Using DOM and CSS on a Browser [Holman, 1999c].	27
4-3. Using DOM and CSS on a Web Host [Holman, 1999c].	27
4-4. The XSL Processing Model Overview [Holman, 1999a].	29
4-5. The Ladder of Abstraction.	31
4-6. Using XSL on Both the Server and the Client [Holman, 1999d].	32
4-7. XSLT and CSS.....	34
4-8. Building Applications with XML [(modified from) Connolly, 2000].....	35
4-9. The Deck and Card Metaphor in WML.	37
4-10. The Document and Dialog Metaphor in VoiceXML.	38
5-1. A Basic Idea for Customised Web Services	40
5-2. Progressive Storage of Processed Content [PIDL, 1999]	48
5-3. The Advantage of Progressive Storage [PIDL, 1999]	48
5-4. Compact Storage of Processed Contents [PIDL, 1999].....	49
6-1. Using HTTP to Serve XML Documents from a Web Server [(modified from) StLaurent and Cerami, 1999, p. 28].	53
6-2. Converting Stored Information to XML Using a Servlet [(modified from) StLaurent and Cerami, 1999, p. 33].	53
6-3. Beyond HTML on MSIE.	55
6-4. Beyond HTML on Netscape.	56
6-5. A Sample XML document on DocZilla.....	57
6-6. Using XML on the Server but Sending HTML to the Browser [(modified from) StLaurent and Cerami, 1999, p. 31].	57
6-7. XML Enabler Receives a Request [Tidwell, 1999].	59
6-8. XML Enabler Sends the Processed Response [Tidwell, 1999].	59
6-9. Typical Interaction Between a Browser and a Web Server Enabled with Rocket [Floyd, 2000, p. 44].	60
7-1. Cocoon Schema [Cocoon, 2000, <code>guide.html</code>]	64
8-1. An Office Personnel Schedule Board.	71
8-2. File Relationships in Schedule Board Requests.....	74
8-6. File Relationships in Update Requests	79
B-1. XML Transformed to HTML.....	99
B-2. XML with XSL and CSS Styling in MSIE5.5	100

List of Examples

3-1. A Sample XML Document	11
5-1. User Agent Attributes in a HTTP Header	40
A-1. Trilogy DTD	98
B-1. Source document <code>sote.xml</code>	99
B-2. XSLT style sheet <code>sote.xsl</code>	99
B-3. CSS style sheet <code>sote.css</code>	100
C-1. A sample profile for a hypothetical smart phone [CC/PP, 1999].	101
D-1. User agent profile [CC/PP, 1999].	102
D-2. Hardware profile [CC/PP, 1999].	103
D-3. Web browser profile [CC/PP, 1999].	103
D-4. Mail application profile [CC/PP, 1999]	103
D-5. Calendar application profile [CC/PP, 1999]	103
E-1. Embedded XSP logic [Cocoon, 2000, <code>xsp.html</code>].	104
F-1. A Sample Document Containing DCP Processing Instructions [Cocoon, 2000, <code>dcp.html</code>].	105
G-1. Source code of <code>board.xml</code>	106
G-2. Source code of <code>board-xsp.xsl</code>	106
G-3. Source code of <code>board-xsp-html.xsl</code>	108
G-4. Source code of <code>board-xsp-generalhtml.xsl</code>	109
G-5. Source code of <code>board.css</code>	111
G-6. Source code of <code>update-xsp.xml</code>	112

Chapter 1. Introduction

1.1. Background

Electronic content distribution has witnessed a huge revolution in the last decade of the 20th century. Two major factors in this revolution have been the birth of the World Wide Web (the Web, for short) and the growth of digital cellular telephony usage, both of which have given content providers new channels to distribute their products to customers.

Since its conception in the early 1990's, the Web has become a critical component in the strategic thinking of content providers [Lie and Saarela, 1999]. More recently, Short Message Service (SMS) and Wireless Application Protocol (WAP) have provided means of distributing content in wireless telephony networks. Since WAP offers a chance to access Web content via wireless telephones, there is a demand to draw these two media together and make their creation process as uniform as possible.

Online information today is mainly Web-oriented. The content and its formatting are designed to be viewed on a Web browser. WAP, however, imposes its own demands on the content due to the more limited capabilities of the browser devices. Not even Web browsers are all alike in their capabilities; Web content can also be accessed with e.g. text-only displays, speech synthesizers or Braille devices.

To put this research into a framework, two areas must be covered. One is electronic publishing, especially aforementioned multipurpose Web publishing, the other is Web applications. In this thesis, the combination of these two is sometimes called by a common name *Web services*. Since wireless telephony networks and the Internet are gradually converging, Web services will eventually be accessed with many different terminals, including mobile phones.

1.2. Research Problem and the Goals of the Thesis

Håkon Lie and Janne Saarela have written about multipurpose Web publishing. In their article [1999], they discuss the questions that Web publishing poses, including “How should content be represented to support device independence, searchability, and efficient network throughput?” They write about *multipurpose publishing*,

where the same content is presented on a range of Web devices ¹. The concept of *publishing* also forms the basis of this thesis; here the subject matter is expanded to include online *applications*. Still, the *main problem* in this study can be outlined as in Lie and Saarela's article, i.e.

how to deal with the most important challenges of delivering content on the network: device independence, content re-use, and network-friendly encodings.

This thesis aims to study the techniques to overcome those challenges. The solutions presented are based on Extensible Markup Language (XML) and other members of the XML specification family, including Extensible Stylesheet Language (XSL), XSL Transformations (XSLT), Extensible Hypertext Markup Language (XHTML) and Wireless Markup Language (WML). Using XML as a generic source language for all content enables the information providers to separate content and presentation, and the same content source can be combined with necessary presentations.

One goal of this thesis is to examine and implement a working Web service. The idea is to make it accessible with both a Web browser and a WAP phone.

1.3. Research Structure and Methods

This thesis is structured into nine chapters, the first one being this Introduction. In Chapter 2, the present situation of Web services and its existing problems are discussed.

The next three chapters present the theory of some solutions to improve Web services. Chapter 3 examines structured information, focusing on one particular technology, XML. Techniques for styling and representing XML are examined in Chapter 4. Chapter 5 presents various techniques to customise the content to suit a particular user and Web device. These include specifications like CC/PP, UAProf, and PIDL that enable the consumers to choose and access just the content they want.

The next three chapters present some applications and other software that make use of the presented solutions in practice. Chapter 6 introduces some existing client and server software that supports XML as the content format. Chapter 7 presents one specific publishing platform capable of serving different consumers with different needs. The platform is called *Cocoon* [Cocoon, 2000] and it is built on open source solutions by Apache Software Foundation. Chapter 8 presents the implementation part of this thesis, the electronic Schedule Board built on Cocoon platform.

¹ Along with the concept “multipurpose publishing”, Lie and Saarela present another useful term: as in their article [1999], “Web device” is used here to denote any hardware or software through which a user accesses Web content.

Chapter 9 concludes the study and gives the final summary.

Chapter 2. Web Publishing and Web Applications

The World Wide Web has become an important channel of information distribution during the past few years. Since its conception at the beginning of 1990's, the essence of Web content has shifted from static documents to dynamically created pages. In this study, Web content is seen as both static and dynamic. The definition originates from Web technology: either the pages are created in advance (static content) or they are generated programmatically at request time (dynamic content).

Static content is associated with existing documents and electronic publishing, while dynamic content is associated with Web applications. This is the key classification in this chapter.

2.1. Web Publishing

The publication of static documents over Web has traditionally been based on document description languages such as Hypertext Markup Language (HTML), Portable Document Format (PDF), and PostScript (PS). For example, the online version of Web Techniques magazine ² provides HTML versions of all of its articles (free of charge). PDF and PS documents have often been created originally for print media but they have also been made accessible online. A recent addition to content description formats is Wireless Markup Language (WML) for WAP content.

Dynamic elements in static HTML and WML documents, and especially in PDF and PS documents, are scarce. The content has been created once and it is published as it is, no modifications are made afterwards. To support various browsers and user demands, content providers may have offered the document in multiple formats. Some publishers are offering their content in both HTML and PDF, for example.

During the mid-1990's, the era also known as the Browser Wars ³, some Web sites were offering their content optimized for one specific browser ⁴. This meant that while those Web pages were taking full advantage of one browser's presentational features, the presentation on other browsers was likely to be poorer. Sites that leant

² <http://www.webtechniques.com/>

³ "Browser Wars" was the era when the makers of the two most popular Web browsers, Netscape and Microsoft, were fighting hard for the dominant market position. It was characterised by the introduction of new, exotic tags that were often incompatible with the rival browsers.

⁴ Alas, this habit is still a reality even today. Take a look with a search engine of your choice and search for pages with keywords "best viewed with netscape" or "best viewed with internet explorer".

heavily on graphical content (images and animation), were almost unreadable with e.g. text-only browsers such as Lynx. This meant that content providers who wanted their sites to be accessed by as wide as possible an audience would have to design their pages more wisely.

Luca Passani [2000] has written that the Browser Wars have divided Web developers into two groups. While one camp insists on producing a unique version of each HTML page to be viewed with each of the different browsers, the other group has sought out an alternative idea. It is to use server-side techniques to tweak up pages dynamically for different Web devices.

Relying mainly on static content is still convenient for some purposes. It is useful for creating content that does not need to be changed or updated frequently. For example, online versions of printed articles are published as they are. Online publications often offer archives of their earlier documents. Archived documents are seldom changed, they may even have a historical value. An example of such an archive is World Wide Web Consortium's (W3C) *Web History* section, containing documents about the evolution of the Web ⁵.

2.2. Web Applications

Web applications rely on dynamic content. Dynamic information is created in real time, according to each user's needs ⁶. Generated content may depend on individual user's parameters and preferences, the capabilities of the user's Web device, the available transmission bandwidth, etc. For example, it is quite common for a Web site to offer pages in two or more versions: text based, graphic enhanced, with or without frames, etc. The point is to give the Web device or, preferably, the user the chance to determine the accessed content.

In his article *Where the Web Leads Us*, O'Reilly [1999] discusses the role of Web sites now and in the future. He notes that Web sites can be thought of as applications: when people are buying books and CDs, they want to use Amazon.com, not “the Internet” or “the Web”. Web sites as applications represent an entirely new breed, something one might call an “information application”, or perhaps even “infoware”.

2.2.1. Dynamic Content Creation Techniques

Conventional techniques to generate dynamic Web content can be divided into two groups. One group can be characterised as “content-in-logic” techniques, the other

⁵ W3C's Historical Archives (<http://www.w3.org/History/>)

⁶ With Web *applications*, it is here more appropriate to write about *users* than viewers or readers. Applications normally imply some kind of *interaction* between a system and its user.

as “logic-in-content” techniques. The terms are not established, they are created for this thesis only in the absence of generally used terminology.

“Content-in-logic” means that the content generator is a functional component, e.g. a CGI (Common Gateway Interface) program or a Java servlet, that either has the content elements hard coded in it or retrieves the information from an external source.

“Logic-in-content” stands for techniques that take the opposite approach: program code is embedded in static content, e.g. HTML elements. Microsoft's ASP (Active Server Pages) and Sun's JSP (Java Server Pages) are examples of these.

Although both techniques may help in the separation of content and logic, it is not enough. The output of CGI programs, servlets, ASP and JSP pages still have the content and its presentation mixed together. This is enough to cause some problems, as seen in the following section.

This study will not dwell on different dynamic content generation techniques more profoundly. They are not the primary subjects here but they are mentioned in order to show their deficiency.

2.3. Existing Problems

HTML and WML, created either manually or automatically, specify both the content structure and the format of the documents. It is this key element in content creation process, the mixing of structure and presentation, which causes a lot of problems.

2.3.1. Misuse of Structural Markup Languages

HTML and WML are by definition *markup languages*. Marking up a document is describing its structure using metadata, also called tags [Siegel, 1997]. HTML was originally created to describe scientific documents, especially their structure, not their presentation [Lie and Saarela, 1999, p. 96]. During the development of the Web, HTML has expanded also to include presentational markup. For example, `<I>` (italics) and `` (font formatting) are tags which describe *how* the information is presented on the Web device. The role of HTML has changed from a structural markup language to a presentational markup language.

2.3.2. Device Dependency

When Web pages are created with markup that also contains presentational instructions, pages are created for a specific medium. Generally, this has not been a serious problem, as the majority of content has originally been created to be viewed with a personal computer and a graphical Web browser. This has led the role of

HTML and the Web itself to shift substantially: The Web today is strongly a *visual* medium [Siegel, 1997].

However, there have always been situations where it is hard or even impossible to access visual content with available Web devices. Text that has been laid out attractively on the computer screen using the `<TABLE>` tag causes problems when rendered with a browser that does not support tables ⁷. As the variety of different Web devices is more likely to increase than decrease along with mobile Internet devices, device dependency is no longer acceptable. It is both the consumers' and the content providers' benefit that the same content can be accessed with all kinds of terminals.

2.3.3. Maintenance Problems

Mixing content, its presentation, and sometimes even the programmatic logic together can cause the content provider maintenance problems. Suppose that a company Web site has a large collection of pages which has a consistent visual design. Overall page layout with headers, footers, fonts, colours, navigational elements, etc. stays the same on every page although the content on each page varies. If every single page has the style information coded along with the content, making changes to layout means that the changes must be included in all pages. When the number of pages is high, this work can be very tedious and error prone, especially if the work is done manually.

Tightly bundled content and logic can cause similar problems. For example, modifying a JSP file that contains program logic in the document structure would require changes to both the structure and the logic. Sometimes the responsibilities for creating and maintaining content, layout and logic are divided between different persons. Maintaining documents that have all these elements mixed together would require efforts to ensure that changing one element does not break other elements.

2.3.4. Poor Searchability

Search engines (or rather: their indexing engines) go through Web pages and index their content automatically. When textual information in a page is indexed, its context is lost ⁸: indexing engines have no way to absolutely define the *meaning* of textual elements. The semantics in HTML and WML is sparse. HTML was influenced by presentational document formats, including PostScript. Both HTML and WML contain such elements as `` (for bold) and `<I>` (for italics) that encode document presentation rather than structure. [Lie and Saarela, 1999, pp. 96-97] and [WML, 1999, pp. 55-56]

⁷ Such a browser is for example in Nokia Communicator 9000.

⁸ This also concerns visual and aural information as well.

Due to the nature of a visual medium, a part of the information on a Web page comes from its visual layout. A human reader can generally see what the various elements on a page represent, at least in a case where the layout of the page is well designed. Normally, a number followed by the character “°” (e.g. 25°) is interpreted as a temperature and a number preceded by the character “\$” (e.g. \$4.95) is interpreted as a price. A human *knows* these and many other visual notations by convention but automatic indexing engines do not. It makes the indexing even harder when the notations vary among information creators. An indexing engine can be taught the notation conventions in some specific information system but to cover all possible notation variations is not reasonable.

As a result of non-contextual indexing, the searchability of information becomes inefficient. For example, if a reader wants to search for information about films where Woody Allen is mentioned as an actor, not a director, regular search engines have no ways to distinguish these two cases.

2.3.5. Accessibility Problems

Using markup languages intended primarily for graphical Web browsers or mobile phones poses a serious accessibility problem when the same information needs to be presented on a different kind of user interface. The problem is an issue not only for disabled people with special types of Web devices, such as aural browsers or Braille devices, but also for anyone who is temporarily unable to use all his senses. Situations where a person's normal senses may be limited are e.g. dark or noisy places, or when the person is driving a car.

Accessibility problems also concern the users of visual Web devices. Screen sizes and resolutions may vary from a small-screen, low-resolution display of a handheld Personal Digital Assistant (PDA) device, through a 28" low-resolution display such as WebTV, to a 15" - 21" high-resolution display of a desktop computer. The job of rendering HTML using a modality or display resolution different from which it was originally designed for is a process that usually produces medium- to low-quality output. This is because it is difficult to extract the relevant content from HTML documents and separate it from the display directives such as columns, tables, frames and JavaScript widgets [Flammia, 1997].

Chapter 3. Structured Information

This chapter presents some solutions to the problems posed in the previous chapter. Generally speaking, the answer lies in structured information. Computer encodings of documents have long concentrated on preserving the final form presentation, such as a nicely laid-out paper document. Structured document formats take a different approach; rather than preserving the final form of presentation, they encode the document's logical structure. In order to achieve device independence, the layout formatting must be separated from the information content. The separation also improves document searchability and information re-use in general [Lie and Saarela, 1999, p. 96].

When delivered to multiple media or to readers with different needs, structured information can be automatically tailored for particular demands. It can be filtered, sorted, or re-organised in some other way for various uses and it can be given an appropriate presentation. Having the original content in the native format of a particular medium, e.g. an HTML file, is not very effective. Such a format contains the information only in the structure for one medium and nothing to make the information accessible in other media or by individual user needs. A generic format is a better choice for expressing the structure of information. Relational databases and markup languages are important means for adding this kind of structure [Baker, 1998, pp. 318-319].

The focus of this thesis is on one markup language standard, Extensible Markup Language (XML), and its related techniques. Database solutions are not discussed here.

3.1. Different Perspectives of Markup

Holman [1999b] has noted that “*markup* is everything in a stream of information that is used to describe the data but isn't the data itself”. In other words, markup can also be called *metadata*, information about the data. With markup, it is possible to express that e.g. the text string “Timothy Zahn” means the *author* of a book in an online bookstore, not for example the publisher or the title of the book.

Holman [1999b] defines a *markup language* as “a formal syntax for markup to be recognized by (parsed by) a computer program”. For instance, HTML and WML are good examples of these. Normally, HTML is parsed by a Web browser and WML is parsed by a WAP phone microbrowser. Pardi [1999, pp. 10-12] calls these *specific* markup languages.

Holman's definition for a *meta-markup language* is “a syntax description mechanism to formally describe different markup languages”. Two of the most

important meta-markup languages are SGML and XML. Both of them use a very similar looking syntax for marking up information, using markers “<” and “>” to denote markup elements, tags. Pardi's [1999, pp. 10-14] definition of these kind of languages is *generalized* markup languages.

3.2. XML - Extensible Markup Language

XML began its life in 1996 when the World Wide Web Consortium (W3C) working group started to specify a subset of Standard Generalized Markup Language (SGML) suitable for the Web. The specification gained the status of a W3C Recommendation⁹ in February 1998 [Lie and Saarela, 1999].

XML is not just a language for defining other markup languages, it is a family of languages. The specifications of all essential XML technologies are coordinated by W3C, an international industry consortium founded in 1994 to guide the development of the Web [W3C, 1997]. The specifications released by W3C (also known as W3C Recommendations) are not official standards in the same sense that e.g. ISO standards are¹⁰. However, in the Internet community, there are certain types of documents that are commonly considered as “standards”, especially IETF¹¹ RFCs (Request for Comments) and W3C Recommendations. Specifications under development, like IETF Internet Drafts, W3C Proposed Recommendations, and W3C Working Drafts are often considered as guiding specifications, although their status is not yet final. XML.com, a Web site dedicated to XML resources, is referring to all these types of documents in its XML-related standards list [XML.com, 2000]

When XML was being developed, the design goals were clearly stated. They were included in the 1.0 specification:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.

⁹ Available on W3C's website: <http://www.w3.org/TR/1998/REC-xml-19980210>

¹⁰ SGML is an ISO standard: ISO 8879: 1986.

¹¹ Internet Engineering Task Force. <http://www.ietf.org/>

9. XML documents shall be easy to create.

10. Terseness in XML markup is of minimal importance [XML, 1998].

Of course, some of these goals may seem somewhat vague and informal. How is it possible to determine the “easiness” of creating XML applications and documents, for example?

Walsh [1998] has commented the goals in his technical introduction to XML. For instance, he notes that while the specification was being developed, the colloquial way of expressing the fourth goal was that it ought to take about two weeks for a competent computer science graduate student to build a program that can process XML documents. The easiness of creating XML documents (goal number 9) arises from the fact that XML is textual: XML documents can be created with text editors, shell and Perl scripts, etc. Eventually there will be more sophisticated tools and editors available for XML.

3.3. XML Structure and Syntax

This section introduces the logical structure and syntax of an XML document. The purpose is not to give a thorough description of every detail in the language. Rather, the principal components that form an XML document are explained. We use a sample XML document (Example 3-1) as a reference.

Example 3-1. A Sample XML Document

```

01: <?xml version="1.0"?>
02: <!DOCTYPE trilogy SYSTEM "trilogy.dtd">
03: <?my-pi do-some-processing-here?>
04: <!-- This is a sample document presenting
05:      some key aspects of XML          -->
06: <trilogy title="The Bounty Hunter Wars">
07:   <author>K. W. Jeter</author>
08:   <part>
09:     <title>The Mandalorian Armor</title>
10:     <image src="armor.gif"/>
11:   </part>
12:   <part>
13:     <title>Slave Ship</title> &sl;
14:   </part>
15:   <part>
16:     <title>Hard Merchandise</title>
17:   </part>
18: </trilogy>

```

XML documents are composed of markup and content. There are six kinds of markup that can occur in an XML document: elements, entity references, comments, processing instructions, marked sections (CDATA sections), and document type declarations [Walsh, 1998]. Each of these markup concepts is introduced in the following subsections.

3.3.1. Elements

Elements are the most common form of markup. They are identified by markers (tags) that surround the content in the document, expressing the nature of it. Elements form one part of the metadata nature of XML; they give the content within them a *meaning*. For example, `<author>` in our sample document indicates that “K. W. Jeter ” is the author of the sample trilogy. Elements can also be empty (such as `<image>` in our example). They have to be marked with either a trailing `</>` in the tag (`<element/>`) or an end-tag right after the start-tag (`<element></element>`).

Attributes

Attributes are name-value pairs that occur within start-tags after the element name. They refine the nature of the element by defining its characteristics. For example, `title="The Bounty Hunter Wars"` (line 6) is an attribute in the `<trilogy>` element that states the title of our trilogy.

3.3.2. Entity References

Entities are predefined content that can be added to documents with entity references. In XML, some characters have been reserved to identify the start of markup. For example, the left angle bracket (“`<`”) normally identifies the beginning of an element start-tag or end-tag. Entity references are a way to insert these characters into the document content. They can also be used to represent often repeated or varying text and to include the content of external files.

Entity references are marked with an ampersand (“`&`”), entity name, and a semicolon (“`;`”). For example, `<` represents the left angle bracket and `&s1;` (line 13 in our example) represents the string “Sponsored by Kuat Drive Yards”. Table 3-1 shows the five predefined entities in the XML specification [XML, 1998]. The `s1` entity is declared in the Document Type Definition (DTD) of our example (Appendix A).

Table 3-1. Predefined Entities in XML

Entity Reference	Character
<	< (opening angle bracket)
>	> (closing angle bracket)
&	& (ampersand)
'	' (apostrophe)
"	" (double quotation mark)

3.3.3. Comments

Comments are the way to add human readable explanations in XML documents. They begin with “<!--” and end with “-->” (as on lines 4 and 5 in our example).

Comments are not a part of the textual content of an XML document. The XML specification does not require XML processors to pass them to applications.

3.3.4. Processing Instructions

Processing Instructions (PIs) can be used to provide information to applications. Like comments, they are not a part of the document character data but XML processors are required to pass them to applications (line 3 in our example).

PIs have the form `<?pitarget pidata?>`. The `pitarget` is used to identify the application to which the instruction is directed. The `pidata` part is optional, it is meant for the application that recognises the target.

The target names “XML”, “xml”, and so on are reserved for XML standardisation [XML, 1998]. The first line in our example is a very special kind of PI, an *XML declaration*. While it is not required, its presence explicitly identifies the document as an XML document and indicates the version of XML to which it was authored [Walsh, 1998].

3.3.5. CDATA Sections

CDATA sections instruct the XML parser to literally interpret all characters in them. They are used to escape blocks of text that would otherwise be recognised as markup. CDATA sections begin with the string “<![CDATA[” and end with “]]>”.

Our example does not have any CDATA sections but one example could look like this:

```
<![CDATA[<anthology>Tales from Jabba's Palace</anthology>]]>.
```

3.3.6. Document Type Declarations

XML documents may contain a document type declaration (line 2 in our example) which specifies the document type and the grammar for using the markup. This grammar is known as a *document type definition* (DTD). The document type declaration can point to an external subset containing markup declarations, or it can contain the markup declarations directly in an internal subset, or it can do both. The DTD for a document consists of both subsets taken together [XML, 1998].

A DTD has its own, XML-incompatible, syntax. It is not important to present the syntax in detail here but Appendix A shows the DTD of our example document.

3.4. Validity of XML documents

The XML specification defines two categories of XML documents: well-formed and valid. The following subsections discuss them both.

3.4.1. Well-formed XML

According to XML specification, a textual object is a well-formed XML document if it obeys the syntax rules of XML [XML, 1998]. Walsh [1998] lists the rules in his technical introduction to XML. Since some of the rules rely on concepts in XML that are not discussed in this study, only some of them are described below.

- The document instance must conform to the grammar of XML documents. In particular, some markup constructs are only allowed in specific places. The document is not well-formed if they occur elsewhere, even if the document is well-formed in all other ways.
- No attribute may appear more than once in the same start-tag.
- Attributes must be declared without ambiguity, notably attribute values must be enclosed between two similar quotation marks.
- Non-empty tags must be properly nested: any non-empty element must be closed by its end-tag before its ancestors.
- Empty element tags must be closed or they must contain a slash “/” just before the end bracket as shown below:

`<EMPTYTAG ATTR1="..." ATTR2='...' />` [Walsh, 1998] and [Ouahid and Karmouch, 1999].

By definition, if a document is not well formed, it is not XML. This implies that there is no such thing as an XML document that is not well-formed. XML processors are not required to parse such documents [Walsh, 1998].

In well-formed documents, it is allowed to add any element with any attribute in any hierarchy, on the condition that the rules for well-formedness are obeyed. However, XML provides a mechanism to define constraints on the logical structure of the document, thus forcing the XML files to follow a strict, predefined format. This mechanism is described in the following subsection.

3.4.2. Valid XML

A valid document is a well-formed document that obeys the rules and the model defined in an associated Document Type Definition (DTD). A DTD specifies all the elements that the associated elements may contain, their hierarchy, their types, and their attributes [Ouahid, 1999, p. 678]. *Validation* is a process for ensuring that documents conform to structures defined in the DTD [StLaurent and Cerami, 1999, p. 90].

The concept of a DTD is borrowed from SGML. SGML documents are *required* to have an associated DTD, XML documents are not. While validation is optional for XML, it can be a useful tool e.g. in gathering conforming information from individual content creators. For example, if a Web site publishes news stories or articles created by independent news agencies or editors, it can require that the delivered content conforms to a mutually agreed DTD.

In valid XML, the model of the document is explicit in the set of declarations (DTD), while in well-formed XML it is implicit in the hierarchy of the data [Holman, 1999b].

3.5. XML vs. SGML

XML is generated as an application profile of SGML. This means that any fully conformant SGML system will be able to read XML documents. The conformance is not required the other way around, though: XML systems are not required to be capable of using and understanding the full generality of SGML. Roughly speaking, XML is a restricted form of SGML [Walsh, 1998]. The main differences between the languages originate from these restrictions in XML.

Clark [1997] has written a note about comparison of SGML and XML. The following list presents some of the most notable differences.

Optional Document Type Declaration

All SGML documents have to be validated against a DTD. For XML documents, DTDs are optional.

Case Sensitivity

In SGML, case sensitivity can be controlled by the document declaration. In XML, case sensitivity is always in use.

Element Minimisation

SGML elements can be minimised, i.e. their end-tags can be omitted, if the DTD allows it. For example, the following code fragment is possible in SGML:

```
<UL>
  <LI>List item 1
  <LI>List item 2
</UL>.
```

In this case, the start of another element also marks the end of the previous element. Element minimisation is not possible in XML. Instead, all elements must be properly closed:

```
<UL>
  <LI>List item 1</LI>
  <LI>List item 2</LI>
</UL>.
```

Empty Elements

Empty SGML elements can have the form: `<emptyelement>` but in XML, they have to be marked as presented in subsection 3.3.1.

Attribute Values

Attribute values in XML are always required to be enclosed in quotation marks. In SGML, the quotes can be omitted if it does not affect the readability of the attribute value. For example, the following markup is valid SGML but not valid (or even well-formed) XML: `<e attr=123>...</e>`.

3.6. Applications of XML

As an *extensible* markup language XML allows the content creators to define new sets of elements, new types of documents for specific needs. This means that XML can encode semantics more accurately than HTML and WML [Lie and Saarela, 1999, p. 97]. It is used for defining the content and structure of documents, not their presentation.

Since 1998, XML has been a base for several document type specifications, also known as “XML applications”. As St. Laurent and Cerami write [1999, p. 18], an application of XML or SGML is a particular document structure specified through

the use of tools like document type definitions (DTDs). HTML has originally been an application of SGML (at least in its standards) but on January 2000 W3C released a recommendation of XHTML, a reformulation of HTML 4 in XML [XHTML, 2000]. Wireless Markup Language (WML) is another application of XML defined by WAP Forum [WML, 1999]. Voice eXtensible Markup language (VoiceXML) is an application of XML for audio applications, developed in partnership with AT&T, IBM, Lucent, and Motorola [VoiceXML, 2000a].

It may sometimes be confusing when SGML and XML experts talk about “tagging information in XML” when they actually mean “tagging information in a language defined by XML” (e.g. WML). The shorter expression comes from the need for simplicity and practicality but it is still important to know the underlying concept. The matter can be drawn an analogy with object oriented analysis. The relationship between XML and its applications is a classic example of *generalisation* [Booch *et al.*, 1999, pp. 64-65]. XML itself can be seen as an abstract base language whereas e.g. WML is a specialised form of XML (language). Therefore it is reasonable to say that a document is written in XML when the actual document type used is WML; a WML document “is-an” XML document. Figure 3-1 shows the relationship between XML and some of its applications in the Unified Modeling Language (UML) notation.

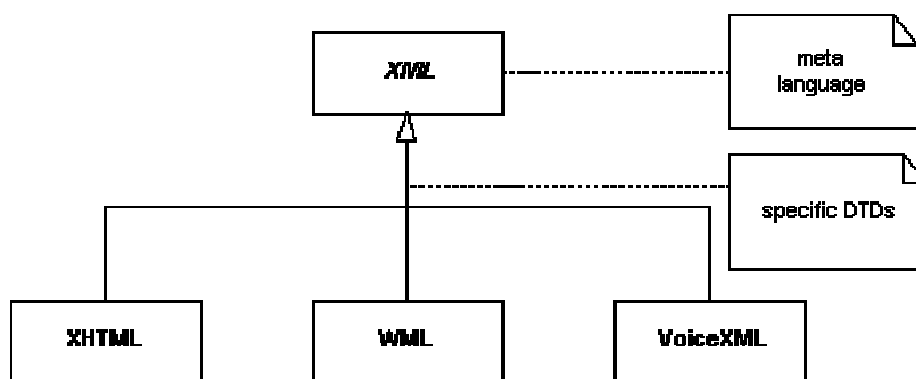


Figure 3-1. The Relation of XML to Its Applications in the UML Notation.

Using XML to describe information content and separating the presentational aspects apart solves the problems that were posed in the earlier chapter. XML documents can act as the “source” of all content. They can be coupled with several different presentations, *style sheets*, that specify how to render each field of XML-encoded information on various types of Web devices [Flammia, 1997].

According to Baker [1999, p. 318], taking a piece of information created for one medium and attempting to use it in another medium is done by converting it from one structure to another. However, every method of content adapting is not based on modifying the *structure* of information. For instance, using Cascading Style

Sheets for rendering XML (or HTML) content on different media does not change the general structure of documents, it only changes the way the content is *formatted* [Holman, 1999a]. Therefore, instead of calling the process of adapting content “a conversion”, it is more convenient to talk about “formatting”. This further harmonises with the juxtaposition of *content* and *format*.

In a network environment, the process of formatting can basically be done on two places: the Web device (client) or the Web server. Of course, it is also possible to have certain middle-tier servers, proxies, to handle it but the techniques for that do not significantly separate from the two basic cases. The following sections discuss the client-side and server-side approaches in more detail, noting the use of proxies where applicable.

3.7. The Three R's of XML

Holman [1999b] has presented three roles for the usefulness of XML. These roles, “The Three R's of XML”, are especially applicable in the field of this study, Web content. They address the very problems stated in the introduction: how to achieve device independence, content re-use and efficient network throughput in multipurpose Web publishing.

3.7.1. Re-publish

When XML documents are the common source of all information content, they can be re-published to multiple information products (Figure 3-2): paper, CD-ROM, Web, proprietary format files, etc. [Holman, 1999b]. This is still useful, even if the products were targetted for the Web alone. As stated earlier, Web content can not be seen as homogeneous data, accessed only with devices with uniform capabilities. By adjusting the content for different types of Web devices, the properties of devices and networks can be taken into account. For example, on a news provider's front Web page, it may be reasonable to offer only headlines of news items to a WAP phone but also short abstracts to a Web browser with larger display screen. Further information can be linked to other pages.

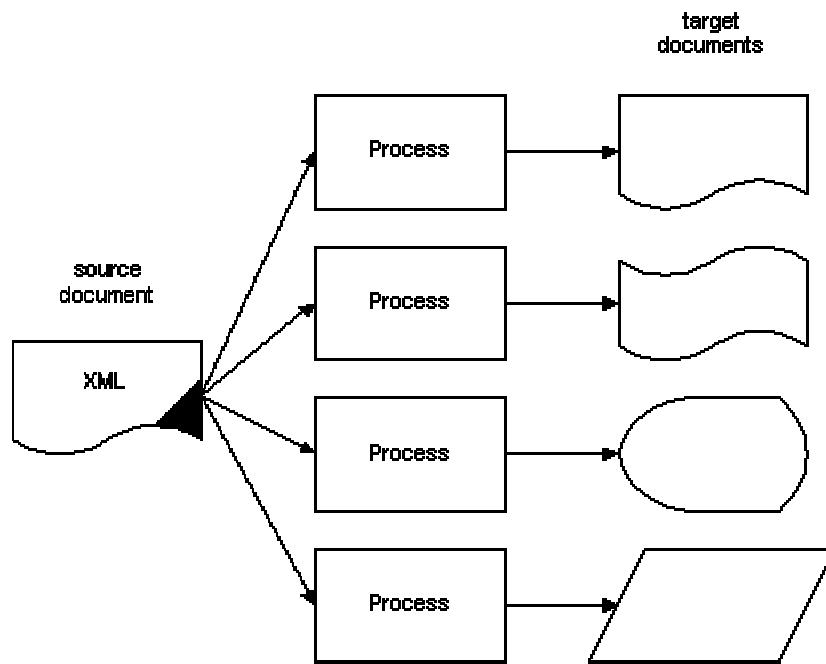


Figure 3-2. Re-publishing Information [Holman, 1999b].

3.7.2. Re-use

Using XML, it is possible to use fragments of included documents in the content of other documents (Figure 3-3). This helps to decrease the production costs, promote consistency and enforce corporate standards and use of common material (e.g. legal statements) [Holman, 1999b]. Here again the benefits also affect Web services: it is very advisable for Web sites that their pages have a consistent look. Consistency, after all, is one of the most basic principles in usability engineering [Lewis *et al.*, 1989].

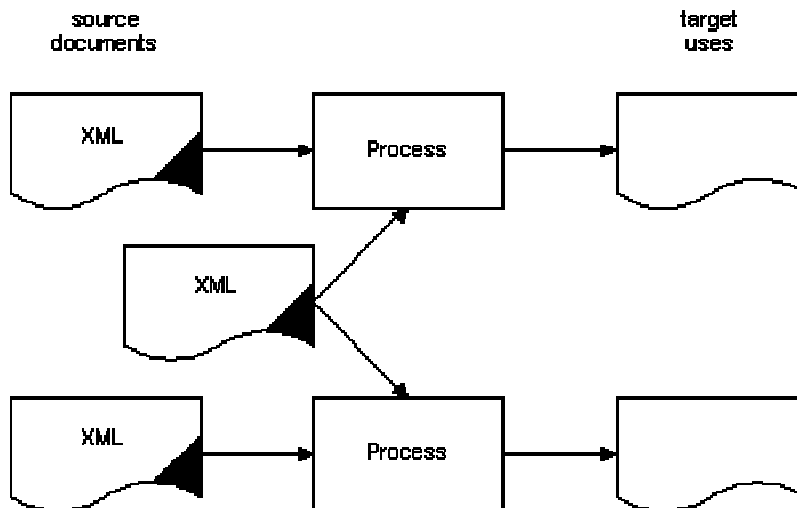


Figure 3-3. Re-using Information [Holman, 1999b]

3.7.3. Re-purpose

The third role concerns the contentual aspect of information. Using XML, it is possible to translate a given document into multiple renditions, each applicable to a different audience (Figure 3-4). This enables the different levels of creators (writers, editors, technical editors, etc.) to have different views of the content. What is perhaps more important in the Web environment, it is also possible to offer different orientations for different kinds of Web devices [Holman, 1999b]. Another consequence is the possibility to create personalised information for various targets.

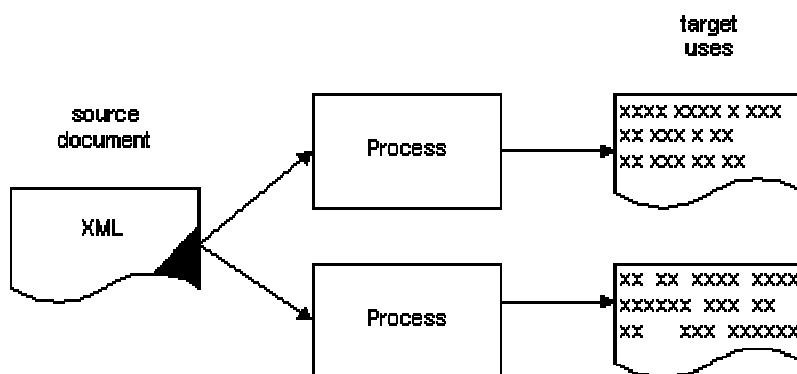


Figure 3-4. Re-purposing Information [Holman, 1999b].

According to Holman, re-purpose differs from re-publish in that here the same information is processed for different purposes and presentations by using *the same medium*. Both of these two roles represent a marketing strategy called *versioning* where the virtually same content can be slightly altered, re-packaged, and sold to multiple customer segments [Shapiro and Varian, 1998].

3.8. Why Use XML in Web Applications?

As e.g. Maruyama *et al.* [1999, pp. 21-22] point out, XML may not be the only or the most efficient way to achieve the goals of device independence, content re-use, and network efficiency. For example, relational databases have been commonly used on server side to contain structured information. Expressing content data in a binary format instead of a verbose textual format would reduce the amount of data to be transferred, while using e.g. Internet Inter-ORB Protocol (IIOP) or a Remote Procedure Call (RPC) instead of HTTP would be much more efficient in terms of communication bandwidth and computation power [Maruyama *et al.*, 1999, pp. 21-22].

Maruyama *et al.* present three of the benefits of using XML in the area of Web applications: simplicity, richness of data structure, and international character handling.

3.8.1. Simplicity

Unlike binary formats, XML is human-readable. It is possible to read, create, and modify the document structure and content data with a simple text editor. Displaying and editing binary data would require specific software tools. Implementing such tools would require bit-by-bit understanding of the encoding format [Maruyama *et al.*, 1999, p. 22].

Although XML as a character-based format is more verbose than binary formats, this has not been considered a drawback. According to Bray's Annotated XML Specification [AXML, 1998], clarity always takes precedence over brevity in XML. Further, the documents and messages encoded in XML can often be compressed for transmission. This is exactly the approach used in Wireless Application Protocol where the data communication over narrowband channels is encoded in binary format. The format is defined in WAP Forum's *Binary XML Content Format Specification* [WBXML, 1999].

3.8.2. Richness of Data Structure

Although XML is a textual notation, it is still powerful enough to express complex data structures. The intrinsic data structure of an XML document is a rooted tree (Figure 3-5). However, as Maruyama *et al.* [1999, pp. 23-24] point out, other structures, such as tables and graphs, might be better suited for certain types of data. Tables represent the logical data structure of relational databases, and graphs can represent shared elements and cycled paths.

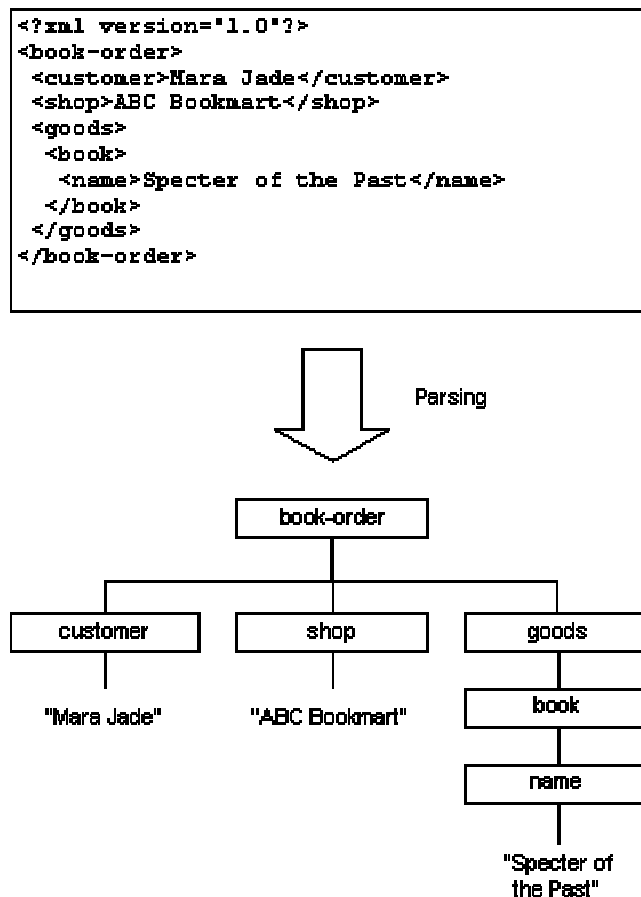


Figure 3-5. An XML Document as a Tree Structure [Maruyama *et al.*, 1999, p. 43].

Maruyama *et al.* claim that for many applications a tree structure is still general and powerful enough to express fairly complex data. This is mostly true in the area of *Web* information where the majority of available content has traditionally been HTML documents. HTML pages tend to have at least some degree of tree structure in them, although the browsers have been quite forgiving about the compliance with HTML syntax rules. With XML, the tree structure is always present in well-formed documents.

The richness of data structures implied in the title of this subsection comes from the possibility to also express other kinds of data in a tree structure. There are even ways to represent graphs and tables in a tree, although they may not be the most efficient solutions. An important characteristic of XML is its ability to conform to object databases. St. Laurent and Cerami [1999, pp. 36-37] note that XML can be parsed into object structures and further stored in object databases. Applications can rapidly access elements and attributes of XML document *objects* without requiring the loading and parsing of a sequential file. This would reduce the time required for processing data on the server side.

3.8.3. International Character Handling

The character set chosen for XML is Unicode (ISO/IEC standard 10646) which has the benefit of handling international character sets [XML, 1998]. As the Web is truly a medium without national borders, the possibility of having content also in foreign alphabets is extremely important. Unicode has currently some 39 000 built-in letters (and plenty of room for expansion) [Fuchs, 1999], so virtually all the characters used today all over the world are legal characters [Maruyama *et al.*, 1999, p. 24].

Chapter 4. Styling XML Documents

Holman [1999a] has considered the *styling* of structured information as comprising of two major processes: *transforming* the source information into the organisation desired for rendering, and *rendering* the organised information into the presentation desired for consumption (Figure 4-1) ¹².



Figure 4-1. Styling by Transforming and Rendering.

This section examines two approaches for styling: programmatic and declarative. First comes the programmatic approach, concerning DOM and CSS, then the declarative method with XSL and XSLT. These two methods are not totally unrelated, they can also be used together.

4.1. DOM and CSS

This subsection discusses two recommendations that have been created for the W3C User Interface Domain: Document Object Model and Cascading Style Sheets. They have been originally developed for HTML-oriented Web but both of them are fully applicable with XML, too.

4.1.1. DOM - Document Object Model

The W3C Document Object Model (DOM) is a platform- and language-neutral application programming interface (API) for XML and HTML documents. It defines the logical structure of documents and the way a document is accessed and manipulated [DOM1, 1998]. The logical structure is modelled as a tree (as seen in Figure 3-5).

The DOM interface can be used to create documents, navigate their structure, and manipulate their elements and content. Thus, an instant of a document tree can be transformed to another tree structure by adding, changing, and deleting elements (also known as nodes). For instance, an XML document can be transformed to a WML document.

¹² He has further divided the rendering of information into two major components: the semantics of style interpreted by the rendering engine and the expression of those semantics.

DOM requires a program to read the entire document structure in the memory before any processing can be done. There are also APIs that have an alternative, event-driven approach. *Simple API for XML* (SAX) is an example of these ¹³. An event-driven API allows the document to be manipulated with a one-pass process, which means it can sometimes be more efficient than DOM. On the other hand, if the processing requires random access to different parts of the document tree, DOM API is more useful than a one-pass process [Maruyama *et al.*, 1999, p. 68]. Both methods have their pros and cons and the reasonable choice depends on the specific application.

4.1.2. CSS - Cascading Style Sheets

Cascading Style Sheets (CSS) is a mechanism for adding style (e.g. fonts, colours, spacing) to Web documents. The first release, CSS level 1 (CSS1), came out as early as December 1996. At that time it was aimed only for HTML documents. The second level (CSS2), released on May 1998, also included XML in its target languages. CSS2 was adding media-specific formatting objects that enabled the content to be made presentable with visual browsers, aural devices, printers, Braille devices, handheld devices, etc. Currently a W3C working group is developing the third level of Cascading Style Sheets (CSS3), adding more features to style formatting.

Cascading style sheets separate content from presentation. In fact, the style sheets *are* the presentation instructions that can be applied to one or more instances of documents. For example, instead of using HTML markup like

```
<h1 align="center">
  <font color="blue">First-level heading</font>
</h1>
```

for all first-level headings, one could write them as

```
<h1>First-level heading</h1>
```

and associate the document with a separate style sheet which could contain the following instruction:

```
h1 { text-align: center; color: blue }.
```

Using CSS2, the content could also be presented on different media. For example, a style sheet for aural rendering of the preceding heading could contain:

```
@media speech {
  h1 { volume: loud }
}.
```

¹³ SAX is a rare exception in the XML related technologies because it does not originate from W3C. It is developed by David Megginson and a number of people on the *xml-dev mailing list* on the Web [Maruyama *et al.*, 1999, p. 42].

Style sheets make documents simpler to write and maintain. A Web site author can use a single style sheet for several documents [Korpela, 1998]. This is one example of re-use mentioned in subsection 3.7.2, only here the re-use is concerning merely style, not content. On the other hand, content can be re-used by associating one document with multiple style sheets. This allows the same document to be used in several publications, without the need to modify the content for particular media [Palola, 1999].

As a CSS style sheet is typically processed in a Web device, it is an example of client-side formatting (discussed in more detail in Section 6.1). However, to save bandwidth for mobile handheld devices, it may be convenient to let a selected portion of the formatting to be handled on a stationary proxy server. This could be accomplished by instructing the proxy not to pass e.g. images and other bandwidth consuming elements to the client:

```
@media handheld {
  IMG { display: none }
  P { display: none }
  P.ingress { display: block }
}
```

[Lie and Saarela, 1999, p. 99].

CSS technology has still its deficiencies. Currently it is only applicable to most HTML and XML-aware browsers, WAP phones do not understand style sheets. In addition, as Holman [1999a] points out, CSS is not aimed at the transformation of the source information prior to its rendering. While it has limited capabilities to attach ornamental text and images to the document tree before rendering, it *does not* have any document manipulation capabilities. CSS does not give one the power to rearrange components of the document, collate and sort selected components of the document, or decorate the rendered tree with rich information. In addition, legacy browsers do not support CSS. Even the recent versions of major browsers¹⁴, including MSIE and Netscape, do not support the completed CSS specifications in their entirety [Meyer, 2000].

To accomplish this kind of content manipulation with CSS, the style formatting must be programmatically executed. This topic is discussed in the following subsection.

4.1.3. Style with Programming

According to Holman [1999c], styling is “the application of both transformation and formatting of information to a presented result”. The two W3C recommendations described above work together to fulfil these two objectives. DOM can be used to transform information obtained from a source into a particular

¹⁴ At the time of writing this thesis, the latest version of MSIE is 5.5 and Netscape is 4.72.

reorganisation while CSS is used for specifying and interpreting formatting semantics for the presentation of information in different media.

Using DOM requires programming. Programs can be implemented in various languages and for various execution environments. For example, DOM programs used on browsers are generally coded with scripting languages such as JavaScript or VBScript. They can handle the transformation and apply style sheets, as seen in Figure 4-2.

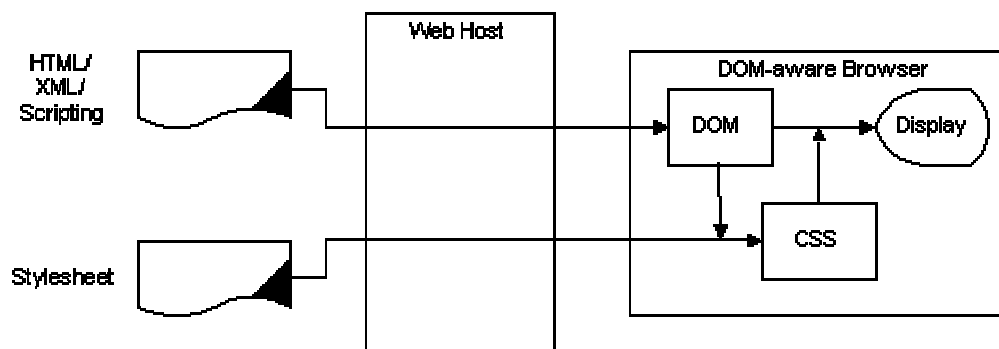


Figure 4-2. Using DOM and CSS on a Browser [Holman, 1999c].

On the other hand, if the browser is not capable of interpreting scripts, the transformation can be handled on the server (Figure 4-3).

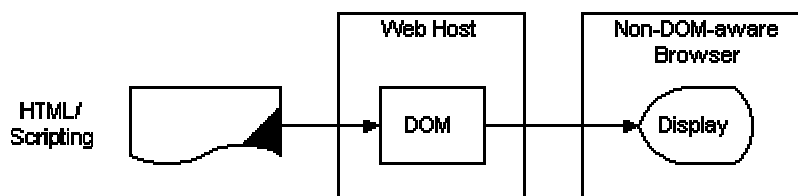


Figure 4-3. Using DOM and CSS on a Web Host [Holman, 1999c].

4.2. XSL - Extensible Stylesheet Language

This section reviews an alternative technique for styling XML documents. Extensible Stylesheet Language (XSL) is another language created by W3C User Interface Domain for expressing style sheets. It consists of two parts: a language for transforming XML documents (XSLT) and an XML vocabulary for specifying formatting semantics (XSL). Note that the term “XSL” has actually two meanings: one is the formatting semantics vocabulary (defined by one specification [XSL, 2000]), the other is the whole Extensible Stylesheet Language concept (defined by

two specifications: [XSLT, 1999] and [XSL, 2000]¹⁵). To avoid confusion, the formatting semantics part of the XSL language is called XSL:FO in this thesis (“FO” for Formatting Objects). Although that is not an official acronym, it is generally used in some sources.

While DOM and CSS are applicable with both HTML and XML, XSL is targeted solely at XML. In particular, XSL is suitable for highly-structured, data-rich documents that require extensive formatting [XSL, 1997, #FAQ].

The development of XSL style sheet language begun with one single specification (released in August 1998). In April 1999 the work was split in two as the XSLT and XSL:FO parts were separated. Later in July 1999, the XSLT specification was further divided in two parts as the XPath addressing notation was separated into its own specification¹⁶. All three specifications are discussed in the following subsections, the main focus laying in the first two, XSLT and XSL.

4.2.1. XSLT - XSL Transformations

As the name of the specification suggests, XSLT is meant for the *transformation* part of styling (depicted in Figure 4-1). The specification defines the syntax and semantics of XSLT, a language for transforming XML documents into other XML documents. XSLT is also designed to be used independently of XSL:FO [XSLT, 1999].

The inputs for the transformation process are the source document to be transformed and the XSL(T) style sheet, both being themselves XML documents (Figure 4-4). The style sheet node tree consists of templates which describe how certain structures of the source node tree are used in the result node tree. The transformation is carried through matching nodes of the source tree to templates in the style sheet, processing the nodes according to the style sheet rules and building the result tree from the transformed nodes. It should be noted that the output of the process is not a document: the result tree is built totally in the XSL Engine. When it is finished, it *may* be serialised to target formats (e.g. files) if wanted.

¹⁵ Or three, depending on whether XPath [XPath, 1999] is counted as a part of XSL or not.

¹⁶ The evolution of the specifications can be examined in the W3C Web site by following the links to previous versions to the documents.

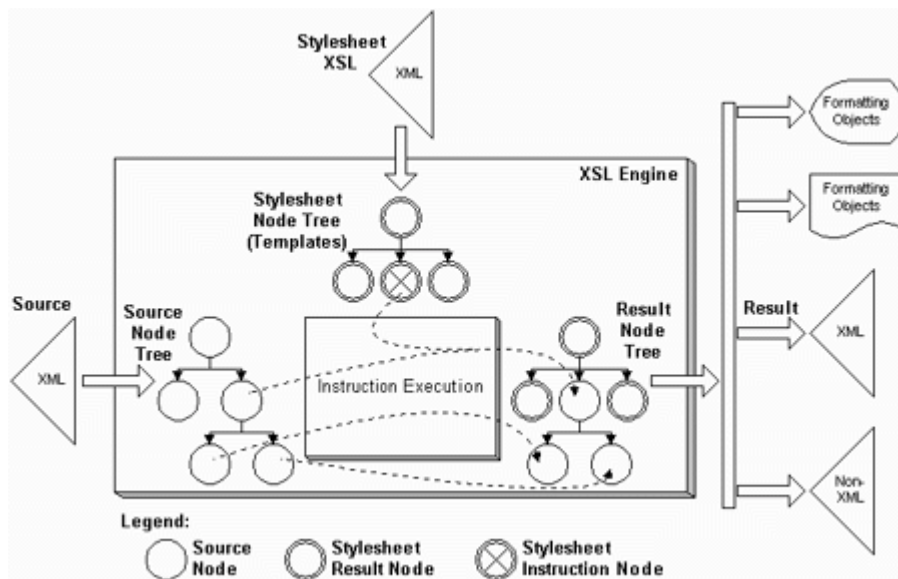


Figure 4-4. The XSL Processing Model Overview [Holman, 1999a].

Transformation allows the structure of the result tree to be significantly different from the structure of the source tree. While it is possible to transform an XML tree into a WML or an HTML tree, it is also possible to add a table-of-contents to the document, sort some elements in it, etc. The tree transformation process may add the formatting information to the result tree [XSL, 2000]. This is where XSL:FO vocabulary comes in. The following section discusses the XSL:FO language and also gives an example of an XSL template.

4.2.2. XSL Formatting Vocabulary

XSL:FO is a language that brings *formatting* to the styling process depicted in Figure 4-1. An XSL:FO style sheet can be used to express how an XML file is presented; i.e. how the source content should be styled, laid out, and paginated on some presentation medium.

It should be noted that an XSL:FO document is not absolutely required to be produced by using the XSLT transformation on another XML document. For example, it is quite possible to use a converter application that reads TeX and PDF files and translates them into XSL formatting objects [Harold, 1999].

According to the XSL:FO specification, “formatting is enabled by including formatting semantics in the result tree. Formatting semantics are expressed in terms of a catalog of classes of *formatting objects*. The nodes of the result tree are formatting objects. The classes of formatting objects denote typographic abstractions such as page, paragraph, table, and so forth. Finer control over the presentation of these abstractions is provided by a set of formatting properties, such as those controlling indents, word- and letter-spacing, and widow, orphan, and

hyphenation control. In XSL[:FO], the classes of *formatting objects* and *formatting properties* provide the vocabulary for expressing presentation intent [XSL, 2000].”

For example, a single XML element

```
<Heading1>The Headline</Heading1> ,
```

transformed with a XSL style sheet template, such as

```
<xsl:template match="Heading1">
  <fo:block font-size="1.3em" margin-top="1.5em"
    margin-bottom="0.4em">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

would produce into

```
<fo:block font-size="1.3em" margin-top="1.5em"
  margin-bottom="0.4em">
  The Headline
</fo:block>.
```

The resulting markup is all about formatting; it consists now of nothing but instructions of *presenting* the content. An XML document that is styled with XSL formatting objects does not contain the richness of data structure that was present in the original XML source. Replacing the document semantics with presentational properties decreases the level of abstraction in the content. The result has no indication of the meaning of the original content [Lie, 1999].

4.2.3. “Formatting Objects Considered Harmful”

In this section we follow Lie's article [1999].

XSL:FO has caused some criticism, too. Lie warns about the misuse of formatting objects. His main concern is that the use of XSL:FO in the Web is a threat to accessibility, device-independence, and a semantic Web. This is due to the descent on “the ladder of abstraction from semantics to presentation” (Figure 4-5), caused by delivering content with pure formatting. Lie states that it is not formatting objects per se that are harmful; the harm is done when they are stored and shipped over on the Web.

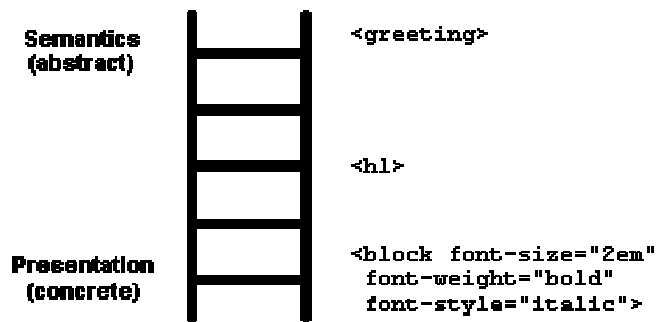


Figure 4-5. The Ladder of Abstraction.

Lie speaks for the use of style sheets on the client side. That would gain the benefits of both rich applications and rich presentations. He is sceptical about the transformations to XSL:FO on the server: although that would seem a good solution in theory, it does not work in practice. For example, to successfully present content aurally, there are four prerequisites:

1. there must be a specification for aural formatting objects,
2. there must be implementations of aural formatting objects,
3. the fact that the user has an aural Web device must be known to the server, and
4. all Web sites must install XSLT sheets to transform content into aural formatting objects.

In the framework of this study, Lie's example could be expanded to cover *all* possible Web devices: browsers, WAP phones, aural devices, Braille devices, etc. According to Lie, the first two prerequisites will require much time and work. The third is undesirable, while the fourth is impossible in practice. In other words, it would not be realistic to expect all Web sites to install XSLT sheets to transform content into formatting objects for all Web devices.

Lie argues that although XSL:FO is not intended to be used in the web, it is unlikely that it can be prevented. However, this is a statement where we disagree: we do not see that HTML or WML would be replaced by XSL:FO. They both are extremely useful in their own domain: presenting Web content on specific client types. Web browsers are very likely to have HTML as their content encoding format in the foreseeable future, too.

We argue that HTML and WML are simpler to learn and use than XSL formatting objects. For example, a single headline element, presented in HTML as e.g.

```
<h1>The Headline</h1>
```

would require considerably more formatting in XSL:FO, like:

```
<fo:block font-size="2em" font-weight="bold">
```

```

    The Headline
</fo:block>.

```

More generally, XSL:FO has more formatting objects and object properties concerning the overall layout than HTML. This is because HTML still *has* some implicit semantics: e.g. `<h1>` is interpreted as a first-level-headline and it is *usually* presented with enlarged, boldface font type. As browsers already have a default rendering style for all known elements, there is no need to describe them again in a style sheet (unless the default is wanted to be overwritten).

As Lie notes, the abstraction levels in HTML and WML are “high enough” so that device independence and accessibility is preserved. They are, and continue to be, good formats for delivering content from a server to Web devices. XML and XSL style sheets can be used effectively on the server side.

4.2.4. Using XSLT for Client-side and Server-side Styling

The optimal method to use XML and XSL could be a combination of server-side and client-side XSLT processing (Figure 4-6). First, the Web server must be able to detect the level of XML/XSLT-support of browsers. For legacy browsers, it can perform the XSLT processing by itself and send them HTML and WML as a response. The server can send both the XML content and the XSLT style sheet to those browsers that support XML and XSLT, thus distributing the burden of transformation to the browsers [Holman, 1999d]. This method is exactly the one used in the XML server *Rocket*, introduced in section subsection 6.2.2 [Floyd, 2000].

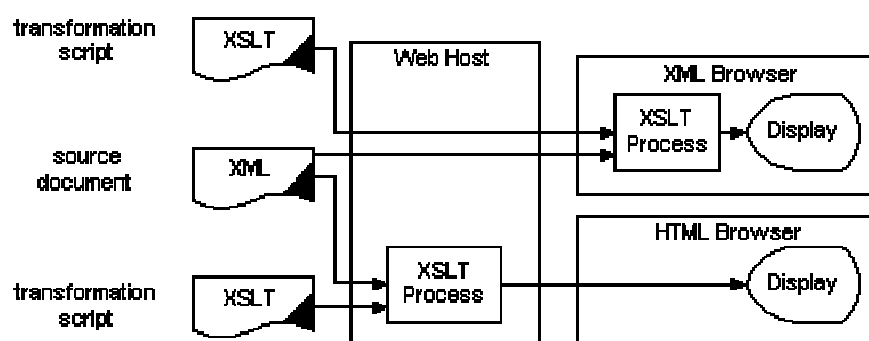


Figure 4-6. Using XSL on Both the Server and the Client [Holman, 1999d].

This architecture allows information to be maintained in XML and still be available to all users. In addition, there may be good business reasons to *always* perform server-side transformations, sending clients only presentation-oriented information instead of semantically rich XML. “Semantic firewall” is a concept where the

investment in rich markup is protected from being seen where not desired [Holman, 1999d].

4.2.5. XPath

XPath is a W3C Recommendation developed alongside with the XSLT language. It provides a common syntax and semantics for functionality shared between XSLT and XPointer (fragment identifier for XML documents). The primary purpose of XPath is to address parts of an XML document with an abstract, logical structure model. In support of this primary purpose, it also provides an expression language for manipulation of strings, numbers and Booleans. XPath uses a compact non-XML-syntax to facilitate the uses of XPath expressions within URIs and element attribute values. The language is called XPath for its use of path notation as in URLs for navigating through the hierarchical structure of an XML document [XPath, 1999].

Both XSLT and XPointer build upon XPath as a core and add extended functionality specific for each domain. XPath expressions are used in XSLT for:

- selecting nodes from the source tree for processing in an instruction,
- specifying matching conditions for choosing between alternative ways to process a given node (conditional processing), and
- generating text to be included in the result tree [Holman, 1999d].

For example, the XPath expression “/book-order/customer” in the XSL template

```
<xsl:template select="/book-order/customer">
  <xsl:text>The Real Heroine: </xsl:text>
  <xsl:apply-templates/>
</xsl:template>
```

would evaluate to the `customer` node-set when the template is applied to the XML document in Figure 3-5.

4.3. Using XSL and CSS Together

In some ways, XSL has similar goals with CSS. Both languages define a vocabulary for defining formatting semantics to separate content, although with different syntaxes. Each one provides a mechanism for selecting elements and specifying how the selected elements are presented. However, they are not competitors; they are likely to co-exist since they meet different needs. XSL is more powerful than CSS in many ways, but it is also more complex. For Web content encoded in HTML, CSS will be the easiest solution. For XML, the

manipulative power of XSL with its transformation capabilities will be required [Walsh, 1999].

The different scopes of the style sheet languages suggest a solution where both of them are used co-operatively (Figure 4-7). Web content can still remain in XML on the server, where it is transformed into presentational markup languages for browsers. The result can be e.g. basic (X)HTML, without any special formatting, or WML¹⁷. Resulted HTML files can further contain associations with separate CSS style sheets which provide additional formatting and decoration. This kind of approach is suitable with most current browsers, at least in some degree (see subsection 4.1.2). The process of styling by transformation and rendering is distributed among the server and the client.

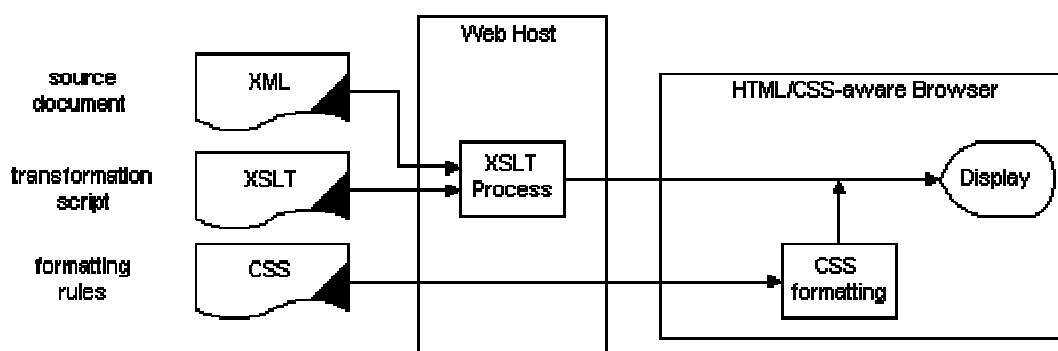


Figure 4-7. XSLT and CSS

Still, even if the content is accessed with a browser that does not support CSS (or supports it only partially), the information can be presented in *some* way. It is just styled with the default formatting built in the browser.

If the content is accessed with an XML/XSL/CSS-aware browser, the situation resembles the one presented in subsection 4.2.4. The difference is that such a browser performs both the XSLT transformation and CSS formatting. An example of that kind of situation is presented in Appendix B.

4.4. Rendering Languages

There are many other activities going on around XML in the W3C (and in other forums as well). Since the release of the *XML 1.0 Recommendation* in February 1998, separate working groups have completed or are currently working on new applications of XML. These include: XML namespaces, XML Queries, XML Schemas, and XML Linking [Connolly, 2000]. They all have a place and

¹⁷ In fact, it is *not* necessary to only use HTML that avoids any markup supported by a specific browser. As the *Viewable With Any Browser* campaign advises, content should be designed to gracefully degrade from state-of-the-art browsers to the more simple ones [Burstein, 2000].

significance in their own domain; for instance, query and schema rules are useful in XML database applications and XML linking enables the creation of hypertext documents. However, they are not discussed here as they are not considered to be in the focus area of this thesis.

Device independent, multipurpose publications and Web services can be achieved by separating content and presentation. While *XML* was considered to present the “content” part and *styling* the way to join content and format, one area still needs to be examined: “format” (i.e. the final rendering formats). Languages that describe how content is to be presented to the reader are called *rendering languages* [Martin, 2000b]. The vocabularies of these languages represent formatting objects that are transformed by *rendering engines* to visual or audible objects. “Rendering engines” are used to be called “browsers” or “viewers” but Martin [2000b] wants to call them with a more general name *rendering language interpreters*.

XSL:FO (discussed in subsection 4.2.2) is one example of XML-based rendering languages. The following subsections describe briefly three other of them: XHTML, WML and VoiceXML.

4.4.1. XHTML - Extensible Hypertext Markup Language

As mentioned earlier, the Extensible HyperText Markup Language (XHTML) is a reformulation of HTML in XML. *XHTML 1.0*, released in January 2000, is W3C's recommendation for the latest version of HTML, following on from earlier work on HTML 4.01, HTML 4.0, HTML 3.2 and HTML 2.0. Basing the future development of HTML on XML is in consistency with W3C's tendency to support XML as a common base for various technologies, as seen in Figure 4-8.

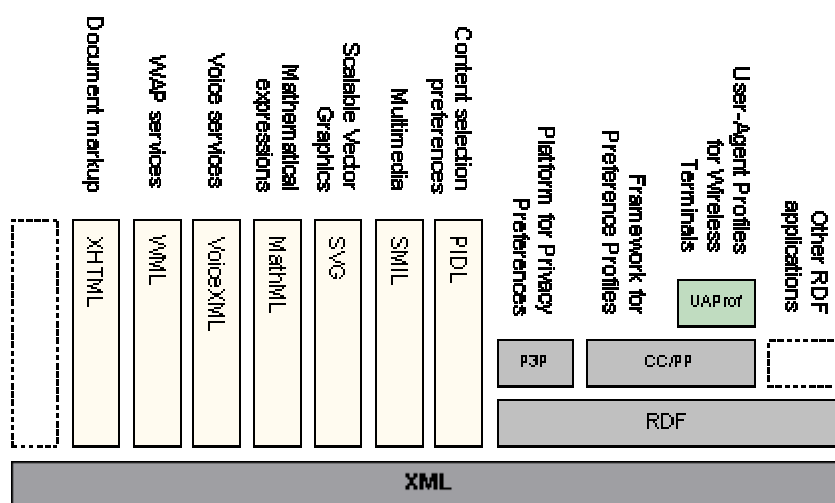


Figure 4-8. Building Applications with XML [(modified from) Connolly, 2000]

As XML means *Extensible* Markup Language, XHTML is also extensible. XHTML could be combined with MathML markup, or developers and content creators can add to it new sets of tags for specific uses. Of course, browsers (or “rendering language interpreters”) may not be able to understand proprietary vocabulary extensions but this is not a problem on the server side. For instance, Martin [2000a] gives us an example where he uses XSLT with extended XHTML to aggregate content from external documents into a single XHTML document – all on the server.

XHTML is modularised into a series of smaller element sets. These sets can then be recombined to meet the needs of different communities. This makes it possible to use only subsets of the language for specific target uses. Not all of the XHTML elements are needed in every Web device; for example a handheld device or a cellular phone may only support a subset of XHTML elements [XHTML, 2000].

4.4.2. WML - Wireless Markup Language

The Wireless Markup Language (WML) specification is one of the many products of WAP Forum. It defines an XML-based markup language that is intended for expressing content and user interfaces for narrowband devices. The domain of narrowband devices, including cellular phones and pagers, sets certain constraints on the content and user interface, including:

- small display and limited user input facilities,
- narrowband network connection, and
- limited memory and computational resources [WML, 1999].

Small display screens with low resolution put an obvious constraint on the type and amount of the content to be presented. For example, most current mobile phones can only display a few lines of text, and each line can contain only 8-12 characters. The input capacities of mobile phones are typically limited by a numeric keypad and only a few additional function-specific keys. The computational resources are often limited by a low power CPU, a small memory and power constraints [WML Reference, 1999].

The characteristics of WML can be grouped into four major areas:

- text presentation and layout,
- deck/card organisational metaphor,
- inter-card navigation and linking, and
- string parameterisation and state management [WML, 1999].

WML offers text and image support, with a variety of formatting and layout commands. As mentioned earlier, e.g. boldface and italics formatting are allowed. The image format developed specially for WAP is Wireless Bitmap (WBMP).

All information in WML is organised into a collection of *cards* and *decks*. A card represents a logical unit of an information container, a “page” that the user reads and interacts with. Cards are grouped into decks. A WML deck is a “physical” information container. It is similar to an HTML page in that it is identified by a URL and it is the unit of content transmission. Figure 4-9 illustrates the deck/card metaphor along with a sample WML deck.

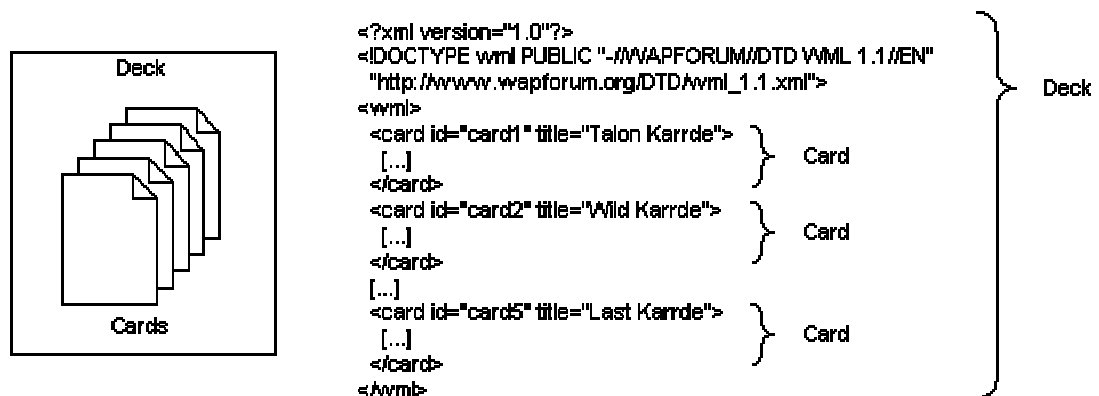


Figure 4-9. The Deck and Card Metaphor in WML.

WML offers support for managing navigation between cards and decks. It includes commands for event handling which may be used for navigational purposes or to execute scripts. WML also supports anchored links, similar to those used in HTML.

Parameters can be set for WML decks using a state model. Variables can be used in the place of strings and are substituted at run-time. This parameterisation allows for more efficient use of network resources [WML, 1999].

The problem with WML is that there is no style guide for its rendering. Any WML microbrowser implementor is free to choose its own way to render WML elements. This means that a WAP application optimised for one specific WML browser may have a poor interface and presentation in some other browser. This unfortunate situation resembles the one experienced in the Web community a few years ago; content providers have to adapt their WML pages for each desired target WAP device. Although this can be done with XML and XSL style sheets, the testing and tweaking can cost a great deal of money [Gegersen and Bilstrup, 2000].

4.4.3. VoiceXML

Numerous companies, including Motorola, IBM, AT&T, and Lucent, have been developing their own versions of markup languages for voice browsers. So far, none of those proposals has gained the status of a de facto standard or a clear leading position. The situation may change with the foundation of an industry

organisation called VoiceXML Forum and its standard proposal for W3C, *Voice eXtensible Markup Language* (VoiceXML). The forum is founded by the four aforementioned companies and it has gained over 100 supporters by April 2000 [VoiceXML, 2000b, [supporters_1.html](#)]¹⁸. Its purpose is to explore public domain ideas from existing work in the voice browser arena.

The information model in VoiceXML is based on *documents* and *dialogs*. A document represents the container unit of one or more dialogs, and dialogs represent the interaction between a user and the system¹⁹. There are two types of dialogs: *forms* and *menus*. Forms present information and gather input; menus offer choices of what to do next [VoiceXML, 2000a]. Figure 4-10 presents the document/dialog metaphor and a sample document with one form dialog.

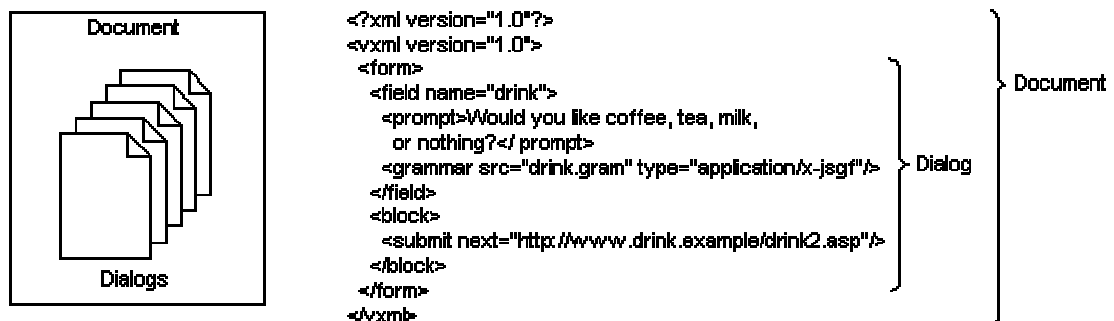


Figure 4-10. The Document and Dialog Metaphor in VoiceXML.

The VoiceXML language is based on the earlier work of the four founder members. It has the similar goal as WML: to make Internet content accessible to telephones. The difference is that while WML is dealing with *visual* content mostly based on text, VoiceXML addresses the *aural* content (mostly) based on voice and sound. According to the language specification, it provides means for handling:

- output of synthesized speech (text-to-speech),
- output of audio files,
- recognition of spoken input,
- recognition of DTMF (numeric keypad) input,
- recording of spoken input, and

¹⁸ As a matter of fact, to be on VoiceXML Forum's (or on some other similar consortium's) "supporter list", does not necessarily imply that an organisation is particularly involved in the forum's work. Sometimes a mere request for further information is reason enough for a consortium to add an inquiring organisation on its list of "supporters". This is one way for industry consortiums to advertise their acceptance and popularity.

¹⁹ The model is analogous to WML and the deck/card metaphor.

- telephony features such as call transfer and disconnect [VoiceXML, 2000a].

The importance of aural medium is not to be underestimated. It is the *only* means of accessing information with telephones without data support (WAP, SMS, etc.). Furthermore, as described in subsection 2.3.5, there will always be situations, either temporary or permanent, where users cannot rely on a visual medium.

While both WML and VoiceXML can present content originating from the World Wide Web, the nature of their content would differ from that presented in XHTML. This is due to the different features of each environment and their Web devices. Having XML documents as the source of the information and transforming them to suit the requirements of different environments help content providers to avoid producing the same information individually for each medium. The point is to have only one “content” and give it several *presentations*.

Chapter 5. Customisations and Personalisations

The previous two chapters discussed the issues of content and presentation. The presented solutions described how XML can be used as the original content format and how it can be transformed to formats supported by various Web devices. An important part of serving Web information is still missing. How to decide what kind of content should be served? How to determine how it should be presented? This chapter addresses these questions and presents some techniques that are developed for the adaptation of Web services.

5.1. Client-Specific Web Services by Using User Agent Attributes

W3C has released a note [Kamada and Miyazaki, 1997] concerning client-specific Web services. It presents a simple framework for realising such services, based on user agent (i.e. Web device) attributes. The idea behind the framework is very straightforward, as seen in Figure 5-1.

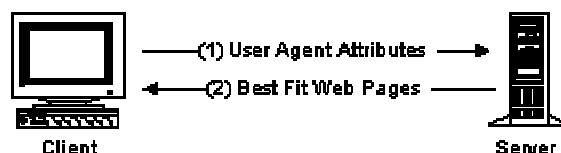


Figure 5-1. A Basic Idea for Customised Web Services

The client reports useful “User Agent Attributes” to the server with the content request. The server recognises these attributes, tailors the content to suit the client, and sends the customised content to it.

The attributes are transferred to the server within the Hypertext Transfer Protocol (HTTP) User-Agent header, as its comment field. Example 5-1 [Kamada and Miyazaki, 1997] shows what that header for an Internet-TV could look like.

Example 5-1. User Agent Attributes in a HTTP Header

```
User-agent: AVE-Front/2.0  
(BrowserInfo Screen=640x480x256; InputMethod=REMOCON,KEYBOARD;  
Page=512K; Product=XXXX/Internet-TV; HTML-Level=3.2;  
Language=ja.JIS; Category=TV; CPU=SH2; Storage=NO;)
```

The User-Agent header field is originally defined to inform the Web server about the type and model of the Web device. The server can read this information in the request and tailor responses to avoid particular user agent limitations [RFC2068, 1997]. The plain user agent type and model may not always give enough information about the Web device; many of the devices have a lot of features that can be modified and customised to meet the needs of different users. Such features can be hardware-related (screen size, amount of memory, etc.), software-related (level of HTML support, Java and JavaScript support, etc.) or user-related (various user preferences, including language, sound, etc.). This is why W3C have presented an extension to User-Agent header: the space for additional comments about the user agent is used to carry structured and detailed information. It should be noted that the extension is *not* a recommendation of any kind. The note, and all other W3C Notes as well, are published for discussion only.

As seen in the previous example, the information is presented in simple property-value format. The predefined properties describe the following features of the Web device:

- screen size and the number of colours,
- input method,
- page (memory capacity for Web page buffer),
- client product information,
- HTML-level,
- preferred language,
- category of the client product,
- CPU type, and
- the amount of secondary data storage (e.g. flash memory or floppy disk) [Kamada and Miyazaki, 1997].

As such, the presented user agent properties are somewhat limited. They focus on the client device (hardware and software) and leave the majority of user preferences outside. For example, a user can not specify whether she wishes to have sound option on or off. What is more relevant in the context of the previous chapters, the clients are all presumed to have at least some kind of HTML support. With WML, VoiceXML and “some-other-ML” now in the set of possible rendering languages, this presumption is no longer justified. The authors of the note admit these limitations, of course, and present a possibility to extend the attribute set by adding new properties and values to the definition.

However, since the release of the note better and more flexible frameworks have been introduced. As they are based on RDF, an application of XML, they offer far

more powerful methods to express all necessary user agent properties and user preferences.

5.2. RDF - Resource Description Framework

Resource Description Framework (RDF) is a framework for expressing metadata, “data about data”, about resources in the Web. It is aimed to provide a foundation for a domain-neutral mechanism that can be utilised in a variety of application areas, for example:

- in *resource discovery* to provide better search engine capabilities,
- in *cataloging* for describing the content and content relationships,
- by *intelligent software agents* to facilitate knowledge sharing and exchange,
- in *content rating*,
- in describing *collections of pages* that represent a single logical “document”,
- for describing *intellectual property rights* of Web pages,
- for describing *privacy preferences* of a user as well as the
- *privacy policies* of a Web site,
- for describing *Web device properties* and
- *other user preferences* [RDF, 1999].

The definition of the mechanism is kept free of any particular application domain. Different application areas complete the framework by specialising it to their domain-specific needs. This is done by defining a collection of classes, also called a *schema*, much like in many object-oriented programming and modelling systems. Further refinement of the schemas for more detailed application areas is possible by subclassing. RDF supports the reusability of metadata definitions through the shareability of schemas.

In essence, RDF is a *model* of metadata which can have a syntactic presentation. The syntax introduced in the RDF specification is based on XML (as seen in Figure 4-8) although that is not necessarily the only choice. It is possible to also introduce alternative ways to represent the same RDF data model.

The following sections discuss the application areas where RDF is used to express Web device capabilities and user preferences. Examples of the RDF syntax in XML are presented there.

5.3. CC/PP - Composite Capabilities / Preferences Profile

W3C has a working group which aims to develop an RDF-based framework for the management of device profile information. The name of the group is CC/PP (Composite Capabilities / Preferences Profile ²⁰) Working Group.

5.3.1. CC/PP Framework

The CC/PP Framework [CC/PP, 1999 and CC/PP-ra, 2000] is an application of RDF that is meant to provide general, yet extensible base for describing user preferences and device capabilities. It addresses the same area as the method presented in Section 5.1 but with a broader applicability. The use of XML-based RDF implies that the framework can be further refined for more elaborate purposes; it is designed to be easily extensible.

The CC/PP note describes a model called Composite Capabilities / Preferences Profile (CC/PP) which means a collection of the capabilities and preferences associated with the user and her user agent(s). The user agent information includes the hardware platform, system software and applications used. Choosing common technologies (XML and RDF) as the base of the framework was expected to encourage the adoption of the technologies and simplify the metadata in the Web [CC/PP, 1999]. RDF also seems to be a logical choice because user agent capabilities and preferences can be thought of as *metadata* or descriptions and attributes *of the user* and the *user agent*.

From the point of view of any particular network transaction, the only property or capability information that is important is whatever is “current”. A network transaction is not interested about the differences between defaults or persistent local changes, the only information it cares about concerns the current transaction [CC/PP, 1999]. Thus, within the same network session, unchanged attributes do not need to be re-transmitted. Subsequent information requests can contain only additions and modifications to the previous profiles.

5.3.1.1. The CC/PP Data Model

The basic data model for a CC/PP is a collection of tables. Each table roughly compares to a significant hardware or software component, such as the general hardware platform and the individual software applications. A CC/PP can be organised to contain all profile descriptions inline, as seen in Appendix C, or it can

²⁰ Even in the CC/PP-WG's own Web documents (<http://www.w3.org/Mobile/CCPP/>), the CC/PP abbreviation has been given different interpretations: 1) Client Capabilities / Preferences Profile, 2) Composite Capability / Preference Profile, 3) Composite Capabilities / Preference Profiles, and 4) Composite Capabilities / Preferences Profile. In this study, the fourth form is used.

have indirect references to them, as seen in Appendix D. Instead of enumerating each set of attributes within the CC/PP, a remote reference (URI) can be used to name a collection of attributes such as the hardware platform defaults. This reduces the amount of data to be transferred from a client to a server (or to a gateway or a proxy), which means better response times in cases where the connection is slow [CC/PP, 1999].

5.3.2. CC/PP Exchange Protocol over HTTP

One W3C Note [CC/PPex, 1999] has been written to cover the CC/PP exchange protocol on HTTP networks. It admits the verbosity of the CC/PP format and suggests two optimisation strategies for it. One is to use compressed form of XML [WBXML, 1999] (introduced in subsection 3.8.1), and the other, a complementary strategy, is to use references.

The *CC/PP Exchange Protocol based on HTTP Extension Framework* specification (CC/PP-HTTP, for short) defines two elementary terms:

CC/PP description

A CC/PP description consists of the device capabilities and user preferences which are described in the CC/PP framework. A CC/PP description is intended to provide information necessary to adapt the content and the content delivery mechanisms to best fit the capabilities and preferences of the user and her agents.

CC/PP repository

A CC/PP repository is an application program which maintains CC/PP descriptions. The CC/PP repository should be HTTP/1.0 or HTTP/1.1 compliant. The CC/PP repository is not required to comply with the CC/PP exchange protocol [CC/PPex, 1999].

The primary protocol strategy is to send a request with profile information as little as possible using references (URIs). When an origin server receives the request, it inquires of CC/PP repositories the CC/PP descriptions using the URIs in the request. CC/PP repositories containing default CC/PP descriptions can be stored on the origin server or they can be distributed on the hosts of the hardware and software vendors. Only those user agent attributes that the user has added or changed locally on her Web device, either permanently or temporarily, need to be sent in the request.

As the name of the specification suggests, the transferred profiles (containing references or modifications) are included in the HTTP/1.1 header fields. The request can contain two types of headers: `Profile` (for references) and `Profile-Diff` (for modifications). One header, `Profile-Warning`, is specified for the

responses and it is used to carry warning information. If any of the CC/PP repositories is not available, the server might not obtain the fully enumerated CC/PP descriptions or it might not get up-to-date CC/PP descriptions. In these cases, the server should respond to the client with the `Profile-warning` header field and inform the user of the inadequate response [CC/PPex, 1999].

5.4. UAProf - User Agent Profiles

UAProf is the result of WAP Forum's development to enable user agent profiles in wireless terminals. In the UAProf specification [UAProf, 1999], profiles are also referred to as *Capability and Preference Information* (CPI). UAProf uses the CC/PP model as a framework and defines a set of components and attributes that WAP-enabled devices may convey within the CPI ²¹. The CPI may include, but is not limited to:

- hardware characteristics (screen size, colour capabilities, image capabilities, manufacturer, etc.),
- software characteristics (operating system vendor and version, support for MExE ²², list of audio and video encoders, etc.),
- application/user preferences (browser manufacturer and version, markup languages and versions supported, scripting languages supported, etc.),
- WAP characteristics (WML script libraries, WAP version, WML deck size, etc.), and
- network characteristics (bearer characteristics such as latency and reliability, etc.) [UAProf, 1999].

The scope of UAProf is to provide origin servers with device preference information such as that presented in the list above. The information in the user agent profile is used for content *formatting* purposes. According to the WAP Forum's specification, a user agent profile is distinct from a *user preference profile* that would contain application-specific information about the user for content *selection* purposes. For example, a user preference profile might designate whether the user is interested in receiving sport scores and, if so, the particular teams. The specification of user preference profiles is beyond the scope of WAP Forum's UAProf document [UAProf, 1999].

²¹ The implementors can, but are not restricted to, use the components and attributes defined in the UAProf specification. They are free to provide any additional components and attributes with their CPI. Of course, those extensions may not be properly interpreted by most origin servers and proxies.

²² MExE (Mobile Station Application Execution Environment) is a framework on mobile phones for executing operator or service provider specific applications. Essentially, it is the incorporation of a Java virtual machine into the mobile phone, enabling full application programming. MExE is developed by the European Telecommunications Standards Institute (ETSI) [Thompson, 2000].

In addition to defining the components and attributes for the CPI, the UAProf specification also includes a section about the exchange of them between the client and the server. It defines a protocol very similar to the one specified for CC/PP and HTTP, using the WAP-equivalent to HTTP: Wireless Session Protocol (WSP). The protocol, called CC/PP-WSP, is needed between the client and the WAP gateway. The gateway bridges CC/PP-WSP and CC/PP-HTTP representations of the User Agent Profile. As the origin servers are normally accessed with HTTP, this bridging has to be made.

There is one problem concerning the deployment of UAProfs (and CC/PPs in general): the lack of support. The specifications rely on mechanisms (such as the HTTP 1.1 Extension Framework and the CC/PP-HTTP) that are not currently deployed widely over the Internet. The UAProf specification recognises this problem and proposes the use of interim proxies. Although origin servers may not currently support UAProfs, proxies that do support them could use the information to adapt the content received from the servers.

It may also be possible that the clients themselves do not support profiles. UAProfs were not in the WAP specification suite till version 1.2 so WAP products conforming to the earlier specifications do not support them. So far, none of the major Web browsers have announced to include CC/PP (or UAProf) support in their products. However, the UAProf specification presents a solution for these clients, too. They can have indirect profile support by a gateway. It may be possible to have a profile provisioned at the gateway by a carrier or service provider. This profile would be presented to gateways, proxies, and servers on behalf of the device(s) involved. The profiles could be static (shared by all users) or dynamic (customised by individual users), chosen with an appropriate user identification method.

5.5. PIDL - Personalized Information Description Language

As CC/PP and UAProf were primarily developed for expressing profile and preference information for content *formatting* purposes, other languages have been created for the complementary part of service customisation: selecting content. W3C has released its own proposition, *Personalized Information Description Language* (PIDL), that currently is only in its initial state²³ [PIDL, 1999].

The purpose of PIDL is “to facilitate personalization of online information by providing enhanced interoperability between personalization applications”. The language tries to provide a common framework for applications to process contents and offer personalised versions of it for individual users. According to the note,

²³ At end of April 2000, only the initial draft of the W3C Note has been released. It is dated on 09 Feb 1999.

PIDL supports the personalisation of different media (plain text, structured text, graphics, etc.), multiple personalisation methods (such as filtering, sorting, and replacing), and different delivery methods (SMTP, HTTP, IP-multicasting, etc.) [PIDL, 1999].

Numerous Web sites, mailing-list maintainers, and other content providers offer personalised content today. It is a common feature on modern Web portals to have service and layout customisation mechanisms so that their customers can have “their own versions” of those portals. So what is a language like PIDL needed for? What new can it offer? According to the PIDL document, it is an attempt to move away from ad hoc implementations to general frameworks that provide efficiency and interoperability.

PIDL is a XML-based language for expressing content. It claims to make personalisation applications simple by realising the interoperability among such applications. If content is originally created as PIDL documents, processed contents of one application can be incrementally processed by other applications. The language features are:

- it encapsulates both the original contents and the progressively processed personalisations in a single XML document,
- it can contain personalised contents for multiple users in a single XML document, allowing distribution of personalised content over 1-to-many connections (such as IP-multicasting), and
- it supports incremental storage of personalisation results in order to keep the overall document size small [PIDL, 1999].

Instead of having only the original content stored somewhere on the server and applying different personalisation processes to it again and again, PIDL documents contain both the original content and the results of each personalisation step in a single document. Having the raw, non-customised content included in the document along with the personalised versions allows later, independent processes to continue or alter the initial personalisation. This progressive personalisation by multiple, independent processes is described in the next subsection.

5.5.1. Progressive Storage of Processed Content

The result of each personalisation step is stored progressively in a PIDL document as *processed content*. The blocks of such processed content can be used for further processing independently and/or progressively, by building upon the results of previous processes. Each personalisation process appends its results to the end of the document, instead of altering the earlier blocks, as in Figure 5-2.

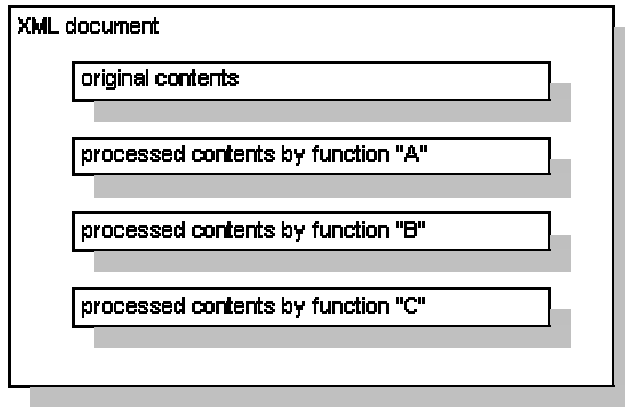


Figure 5-2. Progressive Storage of Processed Content [PIDL, 1999]

The advantage of this kind of progressive storage is that each personalisation step has to be done only once, even if it was required for several users. For example, let us suppose a company intranet for marketing division has two users, X and Y, and each of them have their own customisation preferences. X wants to get her information personalised by

1. only showing contents related to online marketing,
2. highlighting words that match the keywords she has registered.

Employee Y has her own customisation preferences, like:

1. only show contents related to online marketing,
2. shorten the contents so that they can be shown on a PDA.

With these personalisations, the PIDL document containing news items could look like the one in Figure 5-3.

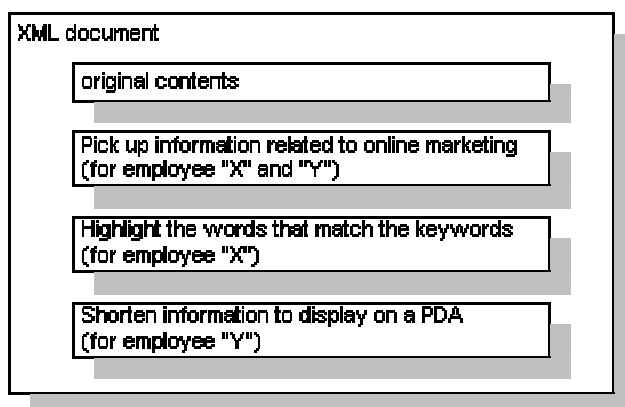


Figure 5-3. The Advantage of Progressive Storage [PIDL, 1999]

5.5.2. Compact Storage of Processed Content

The progressive storage of processed content could easily result in a huge document containing hundreds of copies of almost identical content for each subscribing user. To solve this problem, PIDL documents can store only the *processing method* used and the *personalisation data* used for processing, not the full content of each personalisation step.

For example, a PIDL document with newspaper articles could store a personalisation process with only a set of flags for each user indicating whether a particular article from the original content is relevant to the user (Figure 5-4).

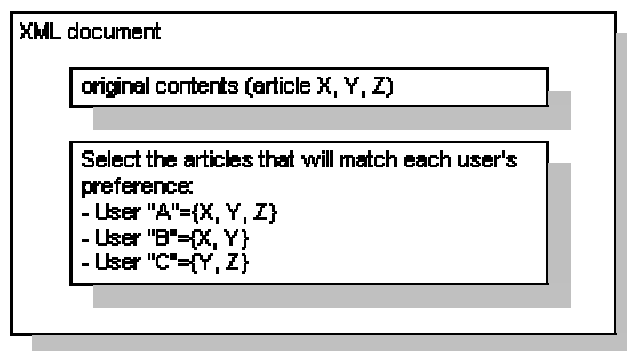


Figure 5-4. Compact Storage of Processed Contents [PIDL, 1999]

The PIDL specification suggests that in order to create the personalised documents out of such a compact presentation, a client-side PIDL document reader would parse and display the document with only the wanted articles. However, this does not seem a good solution. Is it reasonable to expect PIDL-aware clients to emerge in the ever expanding variety of different Web devices? In the short term: not likely. It has been a struggle to get a decent support for even existing standards to clients: CSS, XML, etc. Another format atop of that is not likely to be reality in a long time. In the long term: in some selected platforms, such as in PC browsers, perhaps (provided that PIDL gets accepted as a W3C Recommendation). The capabilities of small, handheld devices are likely to remain stripped-down versions of their larger scale counterparts, such as desktop computers.

The fact that a PIDL document has content and preferences stored for multiple users in a single file raises an important question: what about user privacy? If every user gets the same document, can they view the source code and look what kind of profiles other users have? The PIDL specification considers this issue by suggesting the use of encryption. Each user's personalisation profiles could be encrypted with their own public keys, which would ensure that only authorised users could access profiled information. Still, this would leave some doubts to the wary users: would they have their personal data, even in encrypted form, to be

freely accessed by anyone interested? Can they be absolutely certain there is not some skilful hacker that can decrypt their data? This question will always remain if one's personal data is released from a (more or less) trusted content provider.

Our opinion is that server-side processing with style sheets would be a more reasonable way to utilise PIDL. All the customisations and content selections could be made on the server, clients would get the processed results in the rendering language they support. This approach solves the concerns about both the lack of client support for PIDL and also the user privacy.

There is yet another standard that is related to personalisation and profiling. It is a supplementary format to PIDL in that it can be used for submitting the data relevant for personalising the documents, while PIDL can be used for personalisation. The format is called P3P [PIDL, 1999].

5.6. P3P - Platform for Privacy Preferences

Platform for Privacy Preferences (P3P) is a protocol designed to enable Web sites to express their privacy practices in a standard format (known as a *P3P policy*) that can be retrieved automatically and interpreted easily by user agents. P3P user agents will allow users to be informed of site practices (in both machine- and human-readable formats) and to automate decision-making based on these practices where appropriate. This frees the users from reading the privacy policies at every site they visit [P3P, 2000].

The P3P specification ²⁴ defines:

- a standard schema for data a Web site may wish to collect, known as the “P3P base data schema”,
- a standard set of uses, recipients, data categories, and other privacy disclosures,
- an XML format for expressing privacy policy ²⁵,
- a means of associating privacy policies with Web pages or sites, and
- a mechanism for transporting P3P policies over HTTP [P3P, 2000].

Although the primary goal of P3P is slightly different from the subject of this chapter, adapting content according to user agent profiles and user preference profiles, it is thought to be useful for those purposes, too. As the PIDL document suggests, PIDL and P3P can supplement each other when creating personalised services in the Web. P3P automates the exchange of personal information between the user and a service and allows users to express preferences about the release and usage of their personal information to services. Thus, P3P could fill the gap left by CC/PP and UAProf: it could be used for expressing *user preference profiles* for

²⁴ The specification is still a working draft in April 2000.

²⁵ Actually, P3P is also an application of RDF, as shown in Figure 4-8.

content *selection* purposes. However, it is still an area for further work to extend the existing P3P base data sets to allow user (and services) to express preferences and capabilities regarding personalisation [PIDL, 1999].

5.7. Conclusions

Having presented various techniques and mechanisms for achieving content customisation and personalisation, we should give some attention to the utilisation of them in general.

Let us assume a user is a subscriber to several Web services (e.g. news and email) and she uses them with two different devices: a PDA and a PC. Where should the preference profiles be stored? For hardware- and software-specific properties, CC/PP and UAProf take the approach where the “the client knows what it is like”. In other words, they have the information about their capabilities in them. Of course, that information can simply be a *reference* to some external profile repository. It may be quicker for a Web server needing the capability information to retrieve default values to hardware and software components from an Internet-based repository than to get them from a mobile phone over a low-speed air connection.

The philosophy in PIDL is to have the content and service selection profiles stored on the network, on each content providers' Web site. This seems to be a rational approach for other similar profiles as well. When the service profiles are stored on the network, it does not matter what devices they are accessed by. For example, a user of a news portal gets always just the content she is interested in, regardless of whether she accesses it with a WAP phone or a desktop computer. It is only the *formatting* that gets changed.

Of course, one could argue that it does not always make sense to bring the same kind of content to every possible medium. For example, large blocks of text, images or video files are not useful in a mobile phone. But is this kind of customisation not content *selection* (rather than content formatting)? When a user prefers to receive “sports news with video clips and audio commentaries”, does he not then select what kind of content he wishes to receive?

The answer is not an easy one. To draw a line between content selection and formatting is something that should be considered case by case. The above example could be seen to represent both situations: choosing *sports news* from the set of whatever news sections the content provider has can be thought of content *selection*. On the other hand, stating “I want to receive video clips and audio commentaries” (in addition to and/or instead of text, images, etc.), can be interpreted as content *formatting*. The content provider can have the “same” content, e.g. a news item concerning Formula-1 races, in multiple content formats: text, images, audio, video. Selecting the appropriate formats, choosing *how it is*

presented can also be interpreted as *what is presented*. In this case, formatting is some kind of selecting. The information that gets selected, however, is a little different by its nature. The first case selects the *kind* of information (sports news instead of anything else), the second case selects the *format* of it.

CC/PP has its rule to give all profiles an order of precedence [CC/PP, 1999]. A profile with the highest precedence overrides all other conflicting profiles. What about conflict situations between different preference formats? Suppose a user has stated in her content selection profile that she wants to also have video clips with her news items. When she happens to access the service with a Web device that cannot play video objects, such as a WAP phone, how should the server respond? The presented techniques give no answer to this, the applications using them are responsible for solving these kind of situations. This is reasonable since it leaves more freedom to service providers to implement their own conflict solving rules.

Chapter 6. XML Clients and Servers

This chapter discusses the principles in client-side and server-side formatting and presents some available software applications that support the use of XML in Web services.

6.1. Client-side Formatting

The first of the models for using XML in Web content is close to the traditional HTML / WML approach. As shown in Figure 6-1, a Web server receives a normal HTTP request from the Web device and responds with an appropriate XML file. The important thing is that the Web device supports XML: it can parse the XML file, determine its document type, and display it with appropriate formatting.

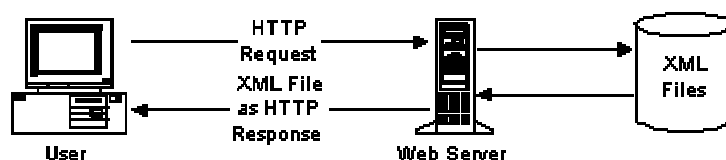


Figure 6-1. Using HTTP to Serve XML Documents from a Web Server [(modified from) StLaurent and Cerami, 1999, p. 28].

Figure 6-2 shows a slightly more elaborate solution where the content is stored in a relational database. The Web server uses an interface to make the queries to the database and convert the results into XML.

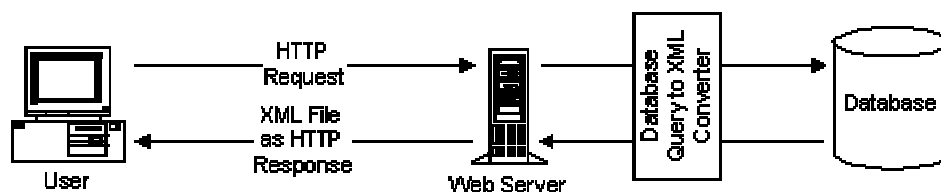


Figure 6-2. Converting Stored Information to XML Using a Servlet [(modified from) StLaurent and Cerami, 1999, p. 33].

Client-side formatting has its advantages. Using XML to create a content-based vocabulary makes it possible to build new client applications by extending the browser. A browser could combine its typical text-and-graphics display with a set of small programs that can do something with the XML data in a document. For

example, a table could be rendered in different forms: tables, diagrams, graphs, etc. Some elements that are intended for display in the browser may also be useful in the context of a spreadsheet, where the user can perform more sophisticated analysis. Applications that can interpret XML can import shared information without losing context, making it possible to exchange information seamlessly [StLaurent and Cerami, 1999, pp. 28-31].

When Web servers are used for only providing clients with the requested XML files (and the corresponding style sheets), the computational burden of formatting is shifted from the server to the clients. This may make the request serving process more effective in a centralised, one-server architecture.

Using XML is convenient for designers and document editors. While they are no longer restricted to use the limited vocabularies of e.g. HTML or WML, they are able to develop style sheets that reflect their own formatting needs. This is a benefit for the server side as well.

Client applications could be used to find and extract specific parts of the content, searches can be limited to particular tag sets or vocabularies. Focusing searches on certain *types* of information gives the users smaller and more focused lists of possible matches, and thus more useful results. This is another advantage that is also gained on the server side.

The problem in serving XML for the Web devices is that there are not many XML-aware clients currently available. In March 1999 Microsoft announced the release of Internet Explorer 5, the first commercially available browser software supporting XML [Microsoft, 1999]. The competitor Netscape Communications has stated it is going to support XML in its future browser version, Communicator 6.0 (also known as *Mozilla*) [Zelnick, 1998]. Since Netscape released the source code of its browser in March 1998 [Netscape, 1998], various third party developers have been able to freely modify the code to make their own improvements²⁶. One such third party developer is a Finnish Company *CiTEC Information* with its own version of Mozilla browser, known as *DocZilla*²⁷. Examples of these browsers are presented in the following sections.

6.1.1. Microsoft Internet Explorer

In Figure 6-3, Microsoft Internet Explorer (MSIE) 5.0 shows the front-page of Michael Floyd's Web site, *beyondHTML*²⁸. This Web site is an example of using XML for multipurpose publishing. It takes advantage of MSIE's XML support by serving the pages for that browser in XML. The browser detection is made

²⁶ Web site for the open-source development of Mozilla: <http://www.mozilla.org/>

²⁷ DocZilla Web site: <http://www.doczilla.com/>

²⁸ <http://www.beyondhtml.com/>

automatically with HTTP's User-Agent request header. As the User-Agent header in MSIE's case includes a text string "MSIE 5", the server can identify the browser.

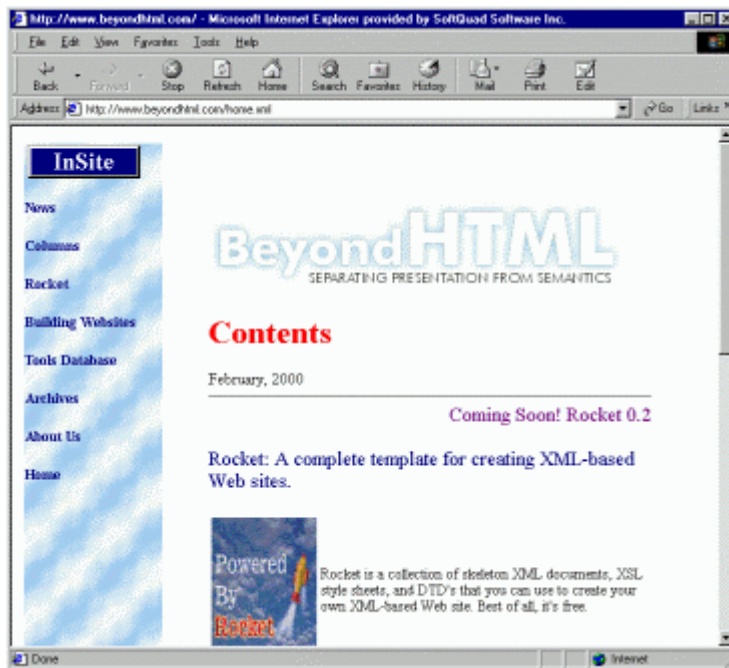


Figure 6-3. Beyond HTML on MSIE.

For the sake of comparison, the same page on Netscape Communicator 4.72 is shown in Figure 6-4. The main difference is invisible to the user: Netscape receives the content in HTML while MSIE transforms the HTML page by itself, using the XML and XSL files it receives. There are also some differences in the layout.



Figure 6-4. Beyond HTML on Netscape.

6.1.2. DocZilla

DocZilla is a browser based on Netscape's open-source Mozilla. On March 2000 the software is still in its alpha phase which means that it has bugs and deficiencies in the feature set [DocZilla, 1999]. The technical manual sample page shown in Figure 6-5 is coded in XML and the formatting information is given in an accompanying Cascading Style Sheet (CSS) file. Since the content oriented XML file does not have any formatting instructions, all information concerning the layout is written in the CSS file.

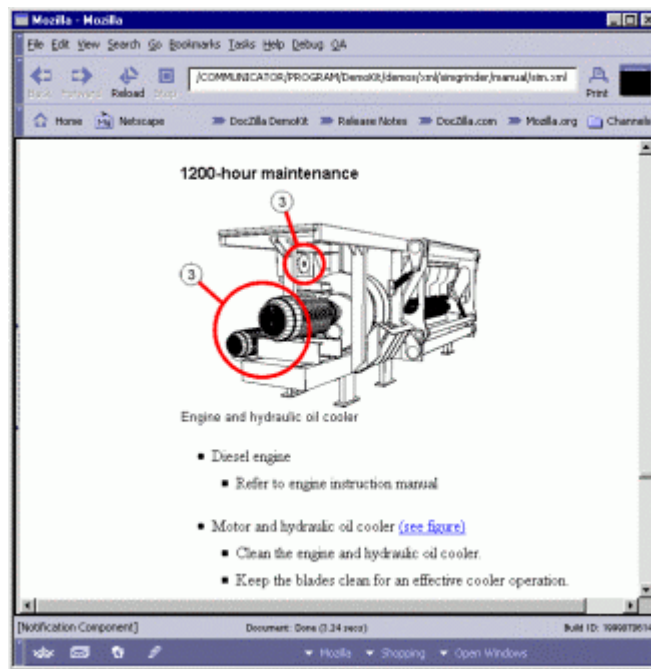


Figure 6-5. A Sample XML document on DocZilla.

6.2. Server-side Formatting

An alternative solution to use XML is to have the Web server handle the conversion from XML to other formats. According to Floyd [1999], client-side processing of XML will always be doomed to inconsistent support. After all, it has been hard for browser developers to get even HTML 4 features to behave consistently in the major browsers.

A situation where the initial content on the server is in plain XML files is shown in Figure 6-6. The server receives an HTTP request from the Web device and retrieves the requested XML file from the repository. After that, it converts the file structure into another format, e.g. HTML, and sends the resulting file to the browser.

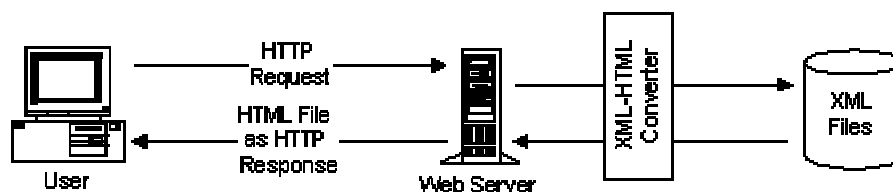


Figure 6-6. Using XML on the Server but Sending HTML to the Browser [(modified from) StLaurent and Cerami, 1999, p. 31].

Processing XML documents on the server, outputting e.g. HTML or WML means that content providers can benefit from the flexibility and power of XML without having to worry about whether a particular client provides XML support or not [Mikula, 1998]. This is a strong benefit of server-side XML processing, it provides true device independence by serving each user and Web device just the information they need and are able to handle. The prerequisite is, however, that the server has the information to do the processing. In other words, it has style sheets for each supported Web device.

The following sections present some of the available software packages for building an XML-based Web site, XML server for short. The first is IBM's *XML Enabler*, the next is Michael Floyd's *Rocket*, and the third is Apache Software Foundation's *Cocoon*.

6.2.1. XML Enabler

XML Enabler is one solution for an XML server, offered by IBM on its alphaWork Web site ²⁹. Basically, it is a Java servlet that converts XML-tagged data to other markup languages, including HTML [XMLEnabler, 1999a].

The architecture and the functional model of XML Enabler are shown in Figure 6-7 and Figure 6-8. XML Enabler uses two pieces of information to transform the XML-tagged data: the type of the Web device, encoded in the User-Agent field of the HTTP header, and a table that maps user-agent values to Extensible Stylesheet Language (XSL) style sheets [XML Enabler, 1999a].

When the user requests an XML document, XML Enabler gets the XML document from the requested URL, determines which XSL style sheet to use for the user's Web device, and uses an XSL Processor to convert the XML-tagged data. Finally, the output from the XSL Processor is sent back to the Web device [XML Enabler, 1999a].

²⁹ <http://www.alphaworks.ibm.com/>

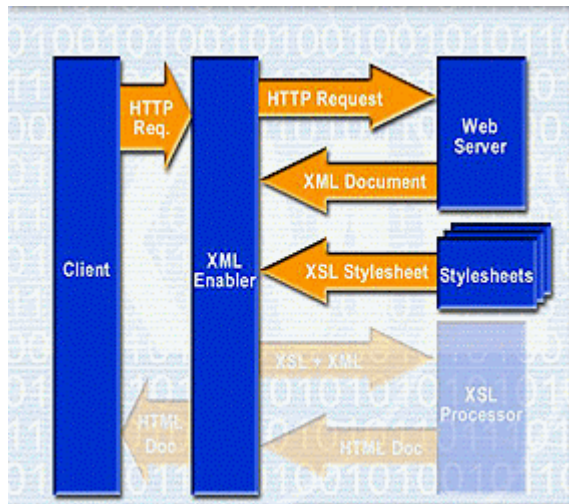


Figure 6-7. XML Enabler Receives a Request [Tidwell, 1999].

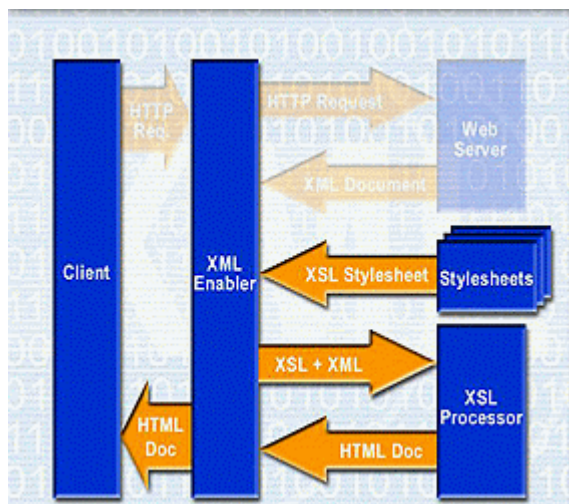


Figure 6-8. XML Enabler Sends the Processed Response [Tidwell, 1999].

Unfortunately, the development of XML Enabler has not been very active. According to the downloadable software's licence agreement, it is not even generally available software (sic!). It is still in experimental state in order to allow developers to evaluate the software [XML Enabler, 1999b].

6.2.2. Rocket

A more recent solution for an XML server is Michael Floyd's effort called *Rocket*. Instead of a Java servlet, it uses Active Server Pages (ASPs) as its interface to perform Web device detection and style processing. Although there is nothing to prevent porting the framework into a servlet or a CGI environment, the gateway programs would have to be rewritten [Floyd, 2000].

Figure 6-9 shows a typical interaction between a browser and a Rocket-enabled server. The interaction follows the pattern presented in the previous section with XML Enabler. Only here, the Gateway Interface is a JavaScript program contained in an ASP file. The default configuration has content formatting support for three different client types: Internet Explorer 5, Netscape Navigator, and generic (i.e. all other) browsers. Examples of a Web page served by Rocket can be seen in Figure 6-3 and Figure 6-4 where Floyd's own Rocket-enabled Web site is viewed in MSIE5 and Navigator 4.72, respectively.

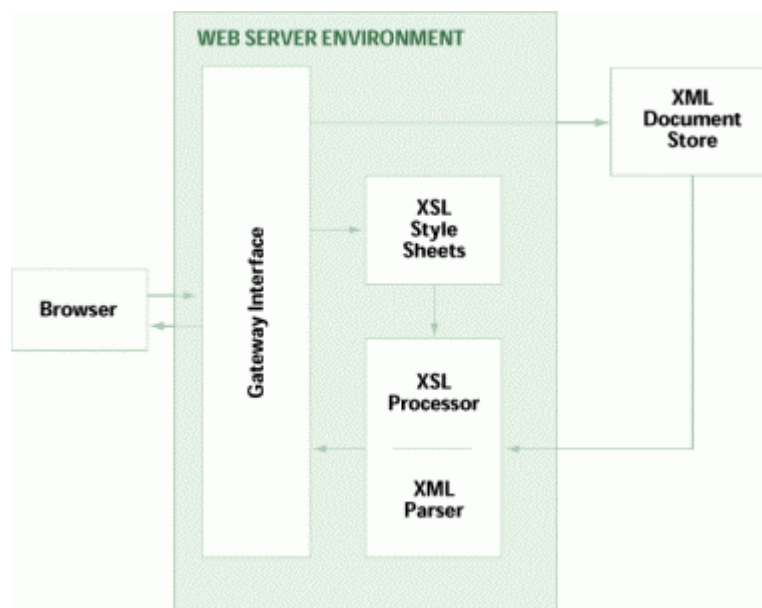


Figure 6-9. Typical Interaction Between a Browser and a Web Server Enabled with Rocket [Floyd, 2000, p. 44].

Although Rocket seems to be slightly better maintained than XML Enabler, it still has a few deficiencies. The fact that it is based on ASP technology makes it highly dependent on Microsoft server environment. Deploying Rocket on other server platforms requires more or less tedious handwork. Furthermore, like XML Enabler, Rocket is still incapable of serving *dynamic* content [Floyd, 2000]. The presented solutions are not sufficient for having a Web server handle both on-the-fly XML conversion and dynamic content generation (like Java servlets and CGI programs do).

6.2.3. Cocoon

Cocoon is the third XML server presented in this study. It is a part of the Apache XML Project, headed by the Apache Software Foundation. The project aims to contribute to the development and utilisation of XML standards through the open source development of XML tools. It currently consists of four subprojects, each

focused on a different aspect of XML. In addition to Cocoon, the project also has sections called: *Xerces* (XML parses in Java, C++, and Perl), *Xalan* (XSLT style sheet processor, in Java and C++), *FOP* (XSL formatting objects, in Java), and *Xang* (rapid development of dynamic server pages, in Java) [Apache, 1999].

Apache XML Project refers to Cocoon as a “publishing platform” or a “framework for XML Web publishing”. Although the general concept conforms to the one depicted in XML Enabler and Rocket (with browser detection and style processing), Cocoon architecture is much more elaborate. It introduces a whole new paradigm to the way Web information is created, rendered and served. The paradigm aims to a complete separation of document content, style and logic. This separation is based of the fact that these three different layers are often created by different individuals or working groups. Dividing the work among different persons allows the layers to be independently designed, created and managed, thus reducing management overhead, increasing work re-use and speeding up publishing schedules [Cocoon, 2000].

Unlike XML Enabler and Rocket, Cocoon also takes into account the generation of dynamic content. This is a strong benefit of this platform as a far majority of today's Web content is created dynamically. According to Sahuguet and Azavant [1999], in 1999 the percentage of database-generated content in the Web was over 80 %. A more detailed discussion of the Cocoon platform is given in Chapter 7.

Chapter 7. Cocoon

This chapter takes a closer look at *Cocoon*, the XML server introduced briefly in subsection 6.2.3. First, we give a general overview of the server platform. Then we examine the server infrastructure and dynamic content generation mechanisms. After that, Cocoon's advantages and disadvantages are considered.

It should be noted that the following discussion concerns the versions 1.x of the Cocoon framework ³⁰. As Cocoon is under constant development, the following major release (Cocoon 2) is expected to have a few modifications in its design [Cocoon, 2000, `cocoon2.html`].

7.1. What is Cocoon?

Cocoon is a publishing platform that relies on some of the technologies presented in previous chapters (such as XML, DOM, and XSLT) to provide Web content. To put it simply: Cocoon uses server-side XSLT transformations to serve client-specific Web content for different client types.

It differs from most of the other server platforms in that it allows the complete separation of the three layers: *content*, *style*, and *logic*. We have earlier discussed mostly the first two layers, content and style, but the logic layer can not be ignored either. It is as important part of Web services as they are. It is hard to imagine a successful Web portal that does *not* have any dynamically created content. Of course, nothing prevents one from using Cocoon to publish only static content; it just means that the logic part is not used.

The separation of the different layers divides the development of Web services in three separate levels:

- XML content creation,
- XML processing, and
- XSL rendering [Cocoon, 2000, `index.html`].

The *content* is *created* by people who have the money, resources, skills, and whatever it takes to generate new content. They do not need to have any knowledge about how the XML content is further processed. The only thing they must be aware of is the particular document type (DTD).

XML *processing* means that the requested XML file is processed by applying the *logic* contained in its logic sheet. This is the main difference to the other dynamic

³⁰ The current version of Cocoon at the time of writing this chapter was 1.72.

content generation techniques discussed in subsection 2.2.1: in Cocoon the logic can be separated from the content file.

The created and processed document is finally *rendered* by giving it a *style*. This is done by applying an XSL style sheet to it and formatting the document to the specified resource type. The presentation formats can be e.g. HTML, XHTML, WML, XML, and PDF.

The distinction of the three layers allows Web service providers to have each level managed by individuals best suited for their particular job. The Cocoon model allows the people working in each context to concentrate in their own tasks, minimising the cross-talks (and cross-work) between different working contexts. Multiply skilled persons who have talents for content creation, programming, and graphical design are hard to find. Even if such persons existed, the distinction of different working contexts to independent layers could help to divide and structure the work logically.

7.2. Cocoon's Infrastructure

Cocoon documentation claims that unlike other XML projects, Cocoon concentrates on solving the publishing infrastructure problem. What Cocoon offers is a transparent way of making the transformations from XML to other markup languages, processing it with programmatic logic on the way, if necessary. The infrastructure handles automatically the calls for XML parsers and XSL processors [Cocoon, 2000, `infrastructure.html`].

The performance of the framework is improved with a memory-based cache system for both static and dynamic pages. The following subsections discuss the Cocoon architecture and cache system.

7.2.1. Architecture

The platform has an engine that is based on the *Reactor* design pattern, illustrated in Figure 7-1.

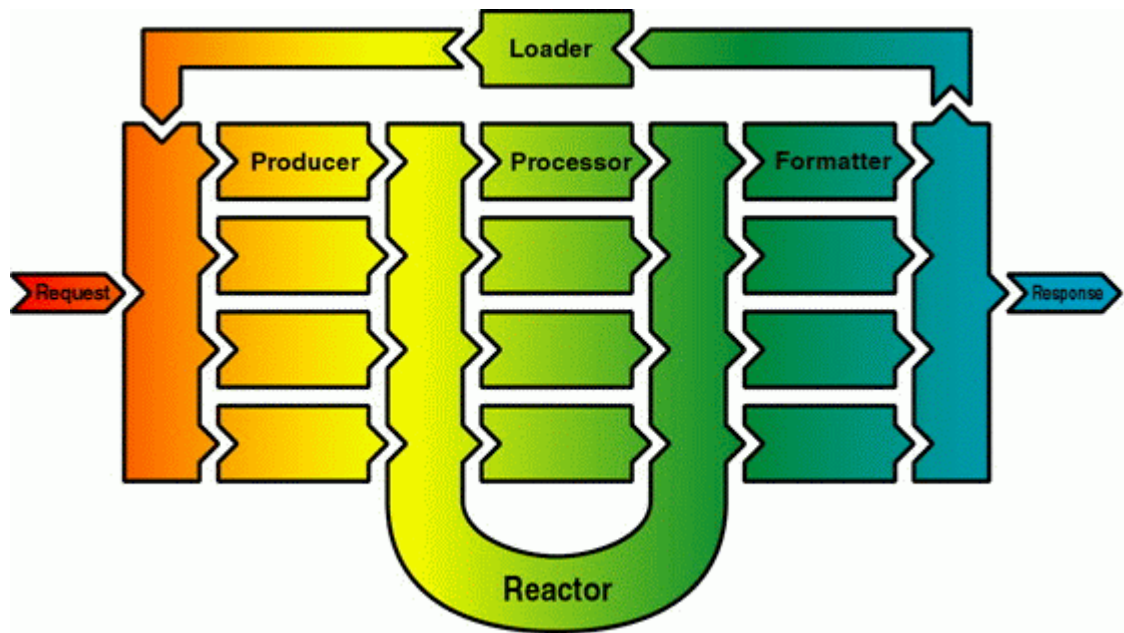


Figure 7-1. Cocoon Schema [Cocoon, 2000, [guide.html](#)]

The components of the schema are described below.

Request

A wrapper around the client's request and all the information needed by the processing engine. The information contained in the request includes: client type (user agent), requested URL, and producer needed to handle the request.

Producer

The handler for the requested URI that produces an XML document. Cocoon itself is implemented as a Java servlet, producers are servlet-like pluggable components that work in this framework. A producer creates the XML document (either by generating it or by reading it from a file) and feeds it into the processing reactor.

Reactor

The component that directs the document to the correct processor, reacting on XML processing instructions. The essence of the architecture, the reactor pattern is different from a processing pipeline in that it allows the processing path to be dynamically configurable. It increases performance since only the required processors are called to handle the document. The reactor is also responsible for forwarding the document to the appropriate formatter.

Processor

A processor transforms a given XML document into some other structure (in the memory). It can either perform a straightforward XSLT transformation or it can also replace portions of the document with dynamically generated content. The Cocoon distribution has a few built-in processors (XSLT, XSP, DCP, SQL, and LDAP) but more about these in Section 7.3.

Formatter

A formatter is responsible for transforming the memory representation of the XML document into a stream that may be interpreted by the requesting client.

Response

An encapsulation of the formatted document along with its properties (such as length, MIME type, etc.).

Loader

The component that loads the formatted output when it is executable code. Instead of sending the output directly back to the client, it gets loaded and executed as a document producer. This part is used for building new producers from compiled server pages (XSPs) [Cocoon, 2000, `guide.html`].

7.2.2. Cocoon Cache System

The Cocoon server environment has a few critical issues concerning performance and memory usage. XML parsing, XSLT transformations, document processing and formatting are all relatively heavy operations that require their share of the processing power on the server. To make Cocoon handle effectively the load of multiple requests the designers implemented a special cache system in its engine. It is able to cache both static and dynamically created pages.

The cache adheres to the following algorithm:

```
when the request comes, the cache is searched
  if the request is found;
    its changeable points are evaluated
    if all changeable points are unchanged
      the page is served directly from the cache
    if a single point has changed and requires reprocessing
      the page is invalidated and the server continues
      as if it was not found
  if the request is not found;
    the page is normally processed
    it is sent to the client
    it is stored into the cache
```

[Cocoon, 2000, `guide.html`]

The pages are processed with the help of many components (XML content files, style sheets, and possibly logic sheets), many of which may change independently over time. This requires that every component that *can* be changed must be checked by the cache system at request time. If, for example, a style sheet or a file template is updated on disk, the whole page is reprocessed.

The cache system is based on a persistent object storage system and it is able to save stored objects in a persistent state that outlives the server environment's Java Virtual Machine (JVM) execution. It is responsible for handling the cached pages as well as the pre-parsed XML documents, including XSL style sheets. Applying pre-parsed style sheets to XML pages that have changed (which supposedly is a rather frequent case) helps to speed up the execution [Cocoon, 2000, `guide.html`].

7.3. Dynamic Content in Cocoon

What makes Cocoon different from most of the other publicly available XML servers, including *XML Enabler* and *Rocket*, is that it has support for the generation of dynamic content ³¹. Java servlet API sets certain limitations on interconnecting several servlets; Cocoon designers have created special techniques to overcome these limitations.

7.3.1. Servlet Chaining vs. Servlet Nesting

Java servlets were introduced by the Java Web Server team as Java-equivalents of CGI-programs, server-side pluggable components to handle dynamic Web content. The need for a componentised request handler was not taken into serious consideration in the design phase. It was only considered afterwards, when at an implementation phase. The ability to *chain* multiple servlets was added in the Java Web Server but unfortunately the servlet API does not include such possibility. The API designers came up with their own solution: servlet *nesting* [Cocoon, 2000, `dynamic.html`].

Servlet chaining (also known as servlet piping)

The servlets are executed sequentially so that the output of one servlet becomes the input to another (Figure 7-2). The output of the *last* servlet is returned to the client [Callaway, 1999, pp. 287-289]. Servlets can be used to filter the output of previous servlets.

³¹ At least the current version of Rocket (0.1) and the soon-to-be-released version 0.2 do not support dynamic content.

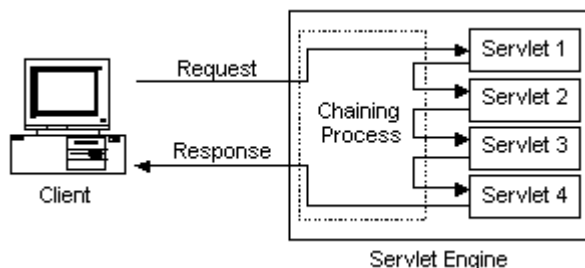


Figure 7-2. The Servlet Chaining Process [Callaway, 1999, p. 288]

Servlet nesting

A servlet executes other servlets inside its own code and includes their output inside its own. The output of the *first* servlet is returned to the client.

The limitation in the servlet API and servlet nesting is that the servlet that gets called within another servlet cannot modify the output of the caller. If no further XML processing is needed on the server side, this is no problem: servlets or JSP files can do the work just fine. However, if this output requires some server-side processing by other servlets (such as XSLT transformations), the servlet API as such is not adequate. *XML Enabler*, *Rocket*, and *Cocoon* are all examples of these “other servlets”.

Although servlet chaining is not supported by the servlet API, it can be managed with special solutions on the servlet engine (“Chaining Process” in Figure 7-2). The Cocoon framework is one implementation of such a chaining process. Rather than turning Cocoon into yet-another servlet engine, thus limiting its portability, its designers came up with alternative solutions. These solutions offer servlet-equivalent functionality with the Cocoon design ideas introduced in subsection 7.2.1 [Cocoon, 2000, `dynamic.html`].

7.3.2. Producers and Processors

Producers are responsible for initiating the request handling phase. They are the Cocoon-equivalents of servlets that evaluate the HTTP request parameters and create XML content for further processing by processors (“Servlet 1” in Figure 7-2). The Cocoon documentation suggests that if a servlet needs many parameters to work, it is more reasonable to write a processor instead. Processors are Cocoon-equivalents of the subsequent servlets in Figure 7-2. The platform distribution has a number of processors that implement common needs and situations. These are explained below.

XSLT Processor

The XSLT processor is probably the most common of the provided processors. As the name states, it applies XSLT transformations to the input document.

XSP Processor

The XSP processor evaluates Extensible Server Pages (XSP) and compiles them into producers. An XSP page is the Cocoon-equivalent of an ASP/JSP page, an XML document containing tag-based directives that specify how to generate dynamic content at request time.

The logic can be embedded in an XML page or it can be separated in discrete modules (e.g. JavaBeans) called with simple method calls. Appendix E shows a sample XSP page with embedded logic and its processed result.

DCP Processor

The Dynamic Content Processor (DCP) represents a little earlier design than XSP in the Cocoon framework. The difference is that while XSP pages are compiled and executed directly as document producers, DCP is interpreted and executed at runtime. This makes DCP somewhat inferior to XSP: interpretation requires some time and the results are not cached. The developers of Cocoon have stated that the further development of DCP is currently limited to bug fixes; it will eventually be deprecated in favor of XSP [Cocoon, 2000, `dcp.html`].

DCP uses XML processing instructions (e.g. `<?dcp-object?>`, `<?dcp-content?>`) in the content files to denote directives for DCP logic. The logic that gets executed is written in separate modules, in e.g. Java or JavaScript. Appendix F shows what a document containing DCP directives could look like.

SQL Processor

The SQL processor acts as an interface to relational databases. It performs SQL queries, translates the result set into an XML fragment and inserts the fragment in the original document. As a considerable amount of data is currently stored in relational databases, a means to utilise that in also XML applications is certainly appreciated. The SQL processor is the means provided in Cocoon's distribution [Cocoon, 2000, `sql.html`].

LDAP Processor

Like the SQL processor, the LDAP Processor is an interface processor. Its functioning is similar to the previous description: it performs LDAP queries,

translates the result set into an XML fragment, and inserts the fragment in the original document.

7.4. Cocoon's Advantages and Disadvantages

This section summarises the advantages Cocoon has over other XML servers and conventional Web servers as well. It also presents some problems still existing in the framework.

7.4.1. Why is Cocoon a good solution?

Cocoon is by no means the only server solution to enable transformations from XML to other markup languages, even with dynamic content generation capabilities. Various software manufacturers, including Oracle, have released similar products as well. Cocoon has still some advantages over other products.

- It is *free*. A working Web server environment with XML support can be built totally on free software, such as Apache Web server, Apache JServ servlet engine, and Cocoon.
- A free Web server is not a requirement, though. It can be installed in *numerous server platforms*, as long as they have a servlet engine.
- It is *open source*. Anyone is allowed to modify the source code to meet her own needs.
- It allows a *complete* separation of content, logic, and style, thus enabling better management of data and human resources.
- It is based on *standard* technologies, such as DOM, XML and XSLT. In the Web community, relying on standards should result to a better chance to be widely accepted (at least in the long run).
- It allows multiple presentations of the same content, e.g. HTML or WML versions of the original XML documents, thus increasing work *re-use*.
- It has a *cache* system that makes the server processing more effective, thus reducing latency time.
- It has been developed for a relatively long time now (in the XML/XSL time scale)³². This along with the fact that many individuals under the open source philosophy have aided its development has made Cocoon a relatively robust framework.

³² Cocoon's creator, Stefano Mazzocchi, claims to have the inspiration for the framework in around Christmas, 1998 [Cocoon, 2000, [faq5.html](#)].

7.4.2. Has Cocoon any disadvantages?

The Cocoon documentation reckons some known problems in the framework. The biggest of those problems is thought to be the lack of XML and XSL knowledge, being relatively new formats. However, even when the public knowledge of those technologies spreads (as it most probably will do), service providers must learn to use them and be willing to fully apply them. Cocoon developers are confident that XML and XSL are going to “do magic” when they gain broader public acceptance [Cocoon, 2000, [index.html](#)]. Whether it does it or not, that remains to be seen.

Another, much more concrete problem concerns the processing complexity. The creators of the framework admit that the kind of operations required to process the document layers are complex and consume a fair amount of computing power on the server side. The problem has been alleviated by the implementation of a page compiler for dynamic pages and a memory-based cache system for both static and dynamic pages. Distribution of computing should also be a solution to lessen the burden on one server. Logic components can be encapsulated in e.g. JavaBeans and distributed on several hosts to take care of raw computing while other servers handle the document serving.

One limitation in the context of the previous chapter is the lack of support for different client adaptation techniques. The decision of what markup language to use (i.e. what XSLT style sheet is used) is based on the HTTP User-Agent header alone. Cocoon does not take into account any of the previously presented formats for preference profiles. On the other hand, one could argue that it is not *Cocoon's* responsibility to adapt the content to meet the preferences of different users and client terminals; it is the *applications* that should take care of that. That may well be true but the server platform could make the job for the applications easier by providing at least some support for the mechanisms. For example, the API for Cocoon producers could have means to access UAProf or P3P attributes.

Chapter 8. Cocoon Service

Implementation: Schedule Board

This chapter describes the implementation of a Web service using XML and Cocoon. The idea is to build an electronic version of an office schedule board. The motivation behind the exercise is to examine the techniques presented earlier and to develop a proof-of-concept demonstration of an XML Web service.

8.1. Background

A schedule board is a tool for any community which needs to keep track of its members' whereabouts. A very common use for such a tool is for example in an office, where employees mark their current status ("in office" / "out of office"), along with some additional, explanatory information. An example of such a board is seen in Figure 8-1.



Figure 8-1. An Office Personnel Schedule Board.

A typical schedule board has a roster of employees and a set of information associated with each employee, including:

- status (in/out),
- schedules for the days of the current week and/or hours of the current day, and
- additional remarks.

The idea is that each employee marks oneself “in” on coming to the office and “out” on leaving it. Writing down some explanatory remarks, such as “Monday to Friday: on holiday”, helps other employees to check one's whereabouts and to decide how and when to contact her (or whether to contact her at all).

The electronic schedule board implemented here could be used in a company intranet. It would be mainly updated with a desktop PCs but a WAP connection would also be possible. This means the schedule board could be checked and updated virtually anyplace and anytime... at least if a WAP connection to the service is possible.

One of the reasons why we chose a *schedule board* as the target of our application is the way its content is created. There are few content providers that offer publicly meaningful, structured data in well-formed XML. For this kind of exercise, we wanted to rely on data that was “home-made”, created by its very users.

8.2. Requirements

This section lists the few requirements our implementation should fulfil.

Data content

The schedule board should contain the following information:

- number of current week,
- name of every employee in the office (must be unique),
- status information (“in” / “out”), and
- optional remarks for each day of the week.

Accessibility

The schedule board should be accessible using a Web browser or a WAP phone. (In the absence of available WAP gateway, the board is tested with a WAP browser emulator running on a PC.)

Implementation

The system should be implemented with tools presented in the earlier chapters, namely XML, XSLT, DOM and Cocoon.

8.3. Implementation

This section describes the technical details of the development. First, we present the implementation platform used in the work. After that, we describe the design of the system.

8.3.1. Hardware and Software Environment

The implementation platform is documented here. The information is admittedly quite technical and may not be of interest to all readers. However, its main purpose is to describe a combination of components which are “tried and true”; i.e. they have been proven to work in at least one platform. This may be valuable information to those readers who are trying to build their own services with Cocoon.

Hardware

IBM ThinkPad 600E laptop computer

Operating System

Microsoft Windows 95

Java Virtual Machine

Sun Java 2 SDK, Standard Edition version 1.2.2

Web Server

Apache HTTP Server v. 1.3.9

Servlet Engine

Apache JServ v. 1.1

Cocoon Servlet Environment

Cocoon v. 1.7.3

Xerces XML parser v. 1.0.1 ³³

Xalan XSL processor v. 0.19.2 ³⁴

³³ The Xerces parser (v. 1.0.3) included in the applied Cocoon distribution was not working with XSP pages in the platform.

³⁴ The Xalan processor (v. 1.0.1) included in the applied Cocoon distribution was not working with XSP pages in the platform.

Web Browser

Netscape Communicator 4.72 (for viewing the HTML output)

WML Browser

Nokia WAP Toolkit v. 1.2 (for viewing the WML output)

8.3.2. Schedule Board Service Design

The structure of the schedule board implementation can be seen from two viewpoints: *logical* and *physical*. The pages that are shown in a Web device are thought to form the logical structure; they affect the way the user understands the service and interacts with it. The physical structure consists of all the files the system uses: two XML files, five XSLT files, and one CSS file. None of them per se are shown to the user, they are only meaningful to the inner functionality of the system. The files and the relationships between them are presented in Figure 8-2.

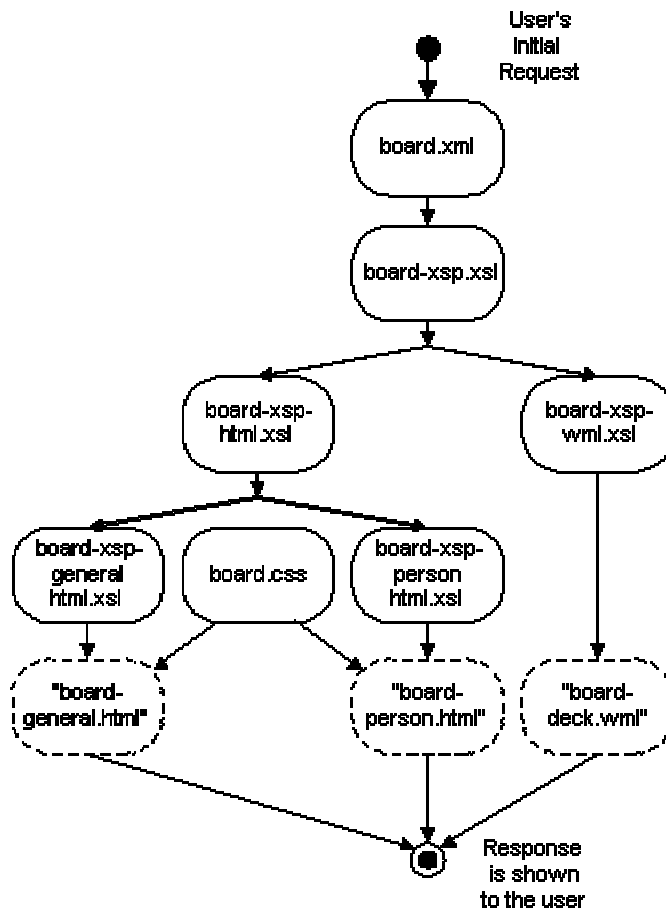


Figure 8-2. File Relationships in Schedule Board Requests

Actually, the pages rounded with dashed line (“board-general.html”, “board-person.html”, and “board-deck.wml”) are not serialised in a file at all. They represent the *logical* view of the service, the pages that get sent to the browser.

Presenting the source codes of all the files would take too much space here. However, as the source codes are perhaps the best way to exactly document the implementation and the functionality of the service, it is essential to present at least some of them in this thesis. A selected set of the source files is shown in Appendix G. They represent one particular chain of files in Figure 8-2, from board.xml to board-xsp-generalhtml.xsl and board.css.

The files on the server and their roles are presented below.

board.xml

This is the XML file that has the data content of the schedule board. It does not contain any logic or presentational information. An excerpt from the file looks like this:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="board-xsp.xsl" type="text/xsl"?>
<scheduleboard>
  <!-- <servlet-parameters/> is needed for the board-xsp.xsl. -->
  <!-- It is replaced by the HTTP request parameters, -->
  <!-- if there are any. -->
  <servlet-parameters/>
  <!-- Person data is omitted here. -->
</scheduleboard>.
```

A more detailed excerpt from the whole file is shown in Appendix G, Example G-1.

board-xsp.xsl

This XSLT file transforms board.xml to an XSP “page” that adds some additional information to board.xml. The added information (HTTP request parameter “personID” and the number of the current week) is generated dynamically upon each request. The resulted XSP “page” is then automatically compiled into a Cocoon producer (_board.class) and stored in Cocoon's repository. Although its source code and the compiled byte code are saved on the hard disk, the producer remains on the server's memory for further processing. The producer is finally responsible for producing the dynamically expanded version of board.xml object (in memory, not as a file).

The XSL template that adds the servlet parameters to the source document is shown below.

```

<xsl:template match="servlet-parameters">
  <xsp:logic><![CDATA[
    Enumeration e = request.getParameterNames();
    if ((e != null) && (e.hasMoreElements())) { ]]>
      <request-params>
        <xsp:logic><![CDATA[
          while (e.hasMoreElements()) {
            String k = (String) e.nextElement();
            String val = request.getParameter(k);
            String vals[] = request.getParameterValues(k); ]]>
          <param>
            <xsp:attribute name="name">
              <xsp:expr>k</xsp:expr>
            </xsp:attribute>
            <xsp:logic> <![CDATA[
              for (int i = 0; i < vals.length; i++) { ]]>
                <value>
                  <xsp:expr>vals[i]</xsp:expr>
                </value>
              }
            </xsp:logic>
          </param>
        }
      </xsp:logic>
    </request-params>
  ]]>
</xsl:template>

```

The entire file source is shown in Appendix G, Example G-2.

board-xsp-html.xsl

This is the XSLT file that does the transformation to HTML. It has an `xsl:include` element to copy two separately stored XSLT files into itself. The included files describe the formatting for two different views on the original XML file. The decision of what view to choose is made dynamically in the `board-xsp-html.xsl` style sheet, based on the now included request parameter.

The following code selects the value of the request parameter and sets it to an XSL variable (“`pID`”).

```

<xsl:param name="pID"
  select="//request-params/param[@name='personID']/value"/>

```

The XSL template that chooses the appropriate view (or *mode* in the XSL terminology) on the document is shown below.

```

<xsl:template match="/">
  <xsl:processing-instruction
    name="cocoon-format">type="text/html"</xsl:processing-
instruction>

```

```

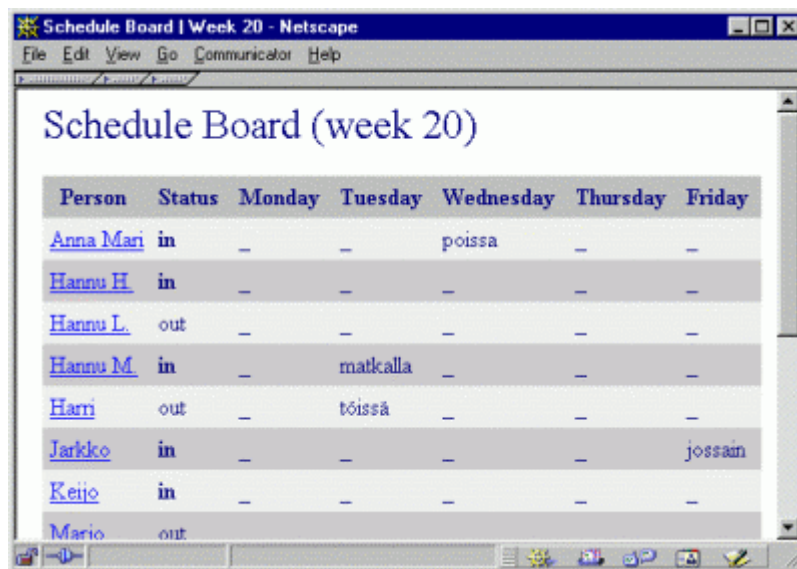
<html>
  <xsl:choose>
    <xsl:when test="$pID">
      <xsl:apply-templates mode="person"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates mode="general"/>
    </xsl:otherwise>
  </xsl:choose>
</html>
</xsl:template>

```

The entire file source is shown in Appendix G, Example G-3.

board-xsp-generalhtml.xml

This is the first of the “view files” for HTML rendering. It generates a table with all the information in the board. This view is chosen if there is no “personID” parameter in the HTTP request. Figure 8-3 shows the generated page (“board-general.html”) in Netscape Navigator.



Person	Status	Monday	Tuesday	Wednesday	Thursday	Friday
Anna Mari	in	-	-	poissa	-	-
Hannu H.	in	-	-	-	-	-
Hannu L.	out	-	-	-	-	-
Hannu M.	in	-	matkalla	-	-	-
Harri	out	-	toissa	-	-	-
Jarkko	in	-	-	-	-	jossaan
Keijo	in	-	-	-	-	-
Mario	out	-	-	-	-	-

Figure 8-3. “board-general.html” in Netscape Navigator

The XSL template that adds the status cell in the HTML table is shown below.

```

<xsl:template match="status" mode="general">
  <td>
    <xsl:choose>
      <xsl:when test="@value='in'">
        <xsl:attribute name="class">in</xsl:attribute>
      </xsl:when>
    </xsl:choose>
    <xsl:value-of select="@value"/>
  </td>

```

```
</xsl:template>
```

The entire file source is shown in Appendix G, Example G-4.

board-xsp-personhtml.xsl

This is the second view to the original XML file. It is chosen if there is a “personID” provided in the request. The result is an HTML document (“board-person.html”) that shows the information of the requested employee. It is also an HTML form which allows the user to update the data. Figure 8-4 shows the page in Netscape Navigator.

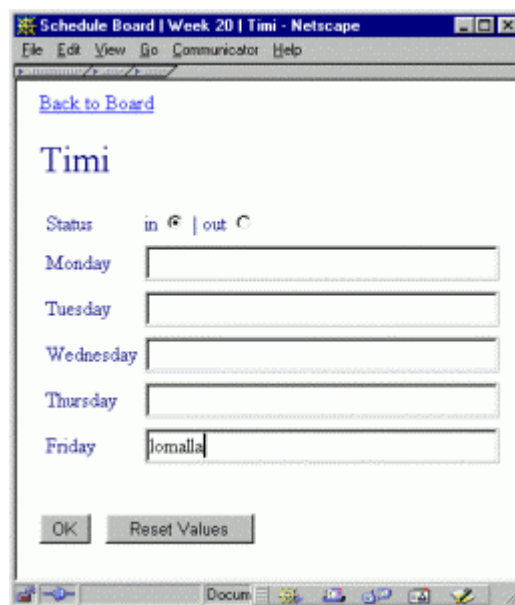


Figure 8-4. “board-person.html” in Netscape Navigator

board.css

This is the common style sheet for the two HTML pages shown on the browser. It has the formatting instructions, “text decorations”, for Web browsers.

The CSS rule for HTML elements of the class “in” is shown below.

```
.in      {
          font-weight: bold;
        }
```

The result is that the status sells (in board-general.html) with the value “in” are rendered with boldface font.

The entire file source is shown in Appendix G, Example G-5.

board-xsp-wml.xsl

This style sheet transforms the XML file to WML. The resulting WML page (“board-deck.wml”) contains one card for the list of all employees and a card for each individual employee. The cards that show each employee's data also have means to update the information. In the implementation environment, the WML page was tested with a WAP phone emulator's WML browser. Figure 8-5 shows the card for the employee list (left) and the card for one employee (right).

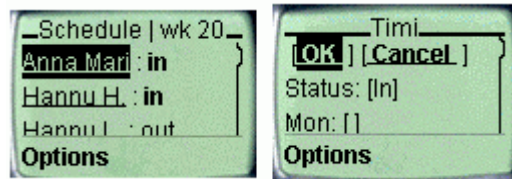


Figure 8-5. Two Cards from “board-deck.wml”

The files in the preceding list are used when the file `board.xml` is requested. There is still one file, `update-xsp.xml`, that is used for updating information. This XSP page contains the Java code that is compiled into a producer by Cocoon. The producer does not generate any substantial output itself; when it has processed the update, it redirects the HTTP request back to the file `board.xml` (Figure 8-6). From there on, the process continues as explained earlier.

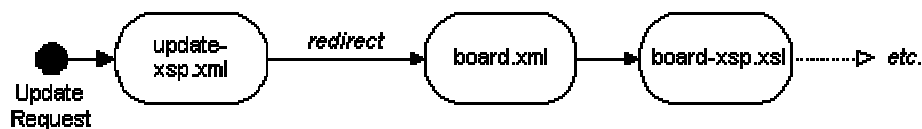


Figure 8-6. File Relationships in Update Requests

An example of a Java function that manipulates the `board.xml` file with the DOM interface is shown below.

```

<xsp:logic><![CDATA[

    /**
     * Return 'null' if first person in not found
     */
    private Node getFirstPerson(Node node) {
        // returns 'null' if first person in not found
        if (node == null || node.getNodeName().equals("person"))
            // return 'null' or 'person' node
            return node;
        else {
            Node child = node.getFirstChild();

```

```

        if (child == null)
            return getFirstPerson(node.getNextSibling());
        else return getFirstPerson(child);
    }
}
...

```

The entire file source is shown in Appendix G, Example G-6.

8.4. Conclusions

This section takes a look at the final application and considers the value of the achieved result.

8.4.1. The Nature of the Application

In general, the possibility to update the data on-line by different types of Web devices may cause some interesting problems. Suppose a Web service created by transforming XML to various presentation formats would also have other kinds of media types to offer: images, audiofiles, binary documents, etc. According to the capabilities of the Web devices, it chooses the appropriate content formats for each device from a repository of existing documents or, alternatively, it can modify the existing documents dynamically to meet the required capabilities (e.g. by converting image files to other formats).

So what happens when such a service is used with a terminal with very limited capabilities? How can the terminal update the content on the server when it is not capable to even handle the same formats the server does? For example, our schedule board application could also have included a means to attach an image to the data of each person. The users could have changed the image; most HTML browsers can be used for uploading images to servers. On the other hand, WAP phones do not have the capability to do that.

In this situation, we would have had to accept the fact that a portion of the service, updating images, would have been accessible by only some of the Web devices. Similarly, in other kind of Web services where the information can also be fed *from* the clients to the servers, some updates can be managed from most clients and some not. In the case of our Schedule Board, the inputted information is only text. The problem does not have any effects here because WAP phones are as capable to send textual information to servers as the HTML browsers are.

Still, updates concerning only textual information may rise a situation that resembles the problem of view update in traditional databases [Date, 1995, p. 472]. HTML and WML represent views on the data originally stored in XML. If XML is considered to have the greatest power of expression, is it possible to retain that

power also in the updates coming from less powerful presentation languages? Our conclusion is that yes it is, provided the transmission mechanism from the client to the server supports the structuring of information. In the case of HTML and WML (and HTTP and WSP), request parameters provide a way to structure the submitted data.

8.4.2. Room for Further Improvement

Although the implemented application is functional and serviceable, it is not polished for real use. For example, it lacks proper error handling, which means the results on incorrect requests may not always be sensible. This is not a serious problem, however, since the work is not intended to be utilised in real office environment.

If wanted, the application could be improved with many additional features. For instance, it could be modified to produce XML output for those browsers who support it. This would allow the browsers to perform tasks to the content without contacting the server, such as filtering the information or sorting the table by different columns.

8.4.3. The Value of This Exercise

Even with a few imperfections, the implementation has presented its worthiness. It is a demonstration of how content, logic, and style(s) can be totally separated. The content (`board.xml`) is fed to a separate logic sheet (`board-xsp.xml`) and the processed output is associated with an appropriate style (HTML and WML style sheets).

It has been clear from the start that the implemented Schedule Board would not be easily applicable in real life. The main reason is that it would be yet another cognitive load for office employees. For the schedule board to be up-to-date at all times, every employee would be required to dutifully update one's information. This would hardly be probable on a day-to-day basis.

The value of the implementation is not so much in the particular resulting application, a schedule board service, as in the proof-of-concept of the applied techniques.

Chapter 9. Summary

Web services, the combination of Web applications and Web content, have been traditionally produced with techniques that mix data content with styling information, sometimes even with programmatic logic. The tight coupling of these elements causes problems that affect the production, delivery, and consumption of the services. Chapter 1 stated the main problem in this thesis as “how to deal with device independence, content re-use, and network-friendly encodings”. This chapter summarises the presented solutions.

Device independence can be achieved by separating the layers of content, logic, and style. Content is pure information, it does not have any attributes that specify what it is used for or how it is presented. It is the job of the applications that process the content to handle those things. Logic, of course, has to know what the content is like in order to be able to process it. However, logic does not have to be tightly coupled with content; they can be only associated with each other. When style (the formatting) is a separate layer, one content object can be given multiple styles. For example, a document can be styled with either graphic-intensive or text-based HTML formatting for Web use.

A general solution to content re-use is structured information. It is a prerequisite for including pieces of information, anything from small fragments to whole documents, in new aggregates. Structured information can be created in many ways, databases and markup languages being two of the most important of them. This thesis focused on one particular technology, Extensible Markup Language (XML). It has some interesting features in its favour, such as: it has been developed specially for the Web, it has its own styling language (XSL), it is extensible and customisable.

Network-friendly encodings mean efficient network throughput. In other words, no network capacity is wasted on transmitting data that is unnecessary or unwanted. Such a situation may occur, for example, when a Web device without a colour display is forced to receive content that uses colourful text or images. Another example is a situation where a user receives information she is not interested in. All these annoyances become even greater when the Web device is behind a slow network connection, such as a telephone line.

Several solutions for this problem were presented in this thesis. Verbose textual content can be compacted by binary encodings, such as Binary XML Content Format in WAP. Content formats for customisation information (Chapter 6) are meant for specifying the capabilities of Web devices and the user preferences. This kind of information can be used to customise the content to better meet the needs

and capabilities of users and their Web devices. One way to perform customisation could be styling mechanisms presented in Chapter 5.

The scope of this study did not cover every aspect of device-independent, customisable Web services. There are still interesting areas left for further research. The following directions concern the practical application of the methods and technologies presented in this thesis.

For a Web service provider, it would be important to determine how the preferences and profiles affect the customisation process. Examples of open questions could be: what facet of the customisation does a particular preference or profile affect? In other words, what preference affects the content, what affects the functional logic, and what affects the styling? What parts of the customisations can be managed with program logic and what parts can be managed with styling?

Another practical issue is the service performance. A service provider planning to build up a huge, device-independent Web portal should take this issue into serious consideration. Can an XML-powered portal effectively substitute for database-driven portals? What parts of the whole service platform could benefit from distributed computing, what parts are better left centralised? To study these kind of things would require systematic testing: setting up test environments and comparing the performance data from varying architecture configurations.

The essence of this thesis was to present XML and relating its techniques as the solution to the problems that exist today in the field of Web services and Web publishing. XML has not yet made its breakthrough as a common format for all Web content, and it is not likely to replace the existing formats (HTML, WML, etc.), either. Services are still going to be implemented in traditional techniques, too. XML can be used as a complementary technique, producing e.g. services that need to be Web device independent.

How realistic is it then to expect content providers to adopt XML as the format for their data? A simplistic answer would be: “it depends on the content provider”. XML requires more strict rules to produce and process the data than traditional HTML. The real world does not always obey rules and specifications; not every content provider is willing to apply techniques that are thought to bring more restrictions and complexity to the current situation. For those content providers who wish to serve customers with a variety of Web devices, and who want to do that in a maintainable and manageable way, XML is a solution to be reckoned with.

References

The reference entries are indexed by the (first) author of the document if the source is a book or an article, and by the title of the document if the source is a specification.

[AXML, 1998] Tim Bray, *The Annotated XML Specification*, XML.com, Apr. 15, 1998, Available online (<http://www.xml.com/pub/axml/axmlintro.html>) Accessed on March 17, 2000.

[Apache, 1999] The Apache Software Foundation, *xml.apache.org: Introduction*, 1999, Web site (<http://xml.apache.org/>) Accessed on March 13, 2000.

[Baker, 1998] Mark Baker, "Using server side XML to create individualized Web pages", 317-319, *Professional Communication Conference, 1998*, IPCC 98, IEEE International, 1998.

[Booch *et al.*, 1999] Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*, 4th Printing, Addison-Wesley, 1999.

[Burstein, 1999] Cari D. Burstein, *Frequently Asked Questions*, Last Modified on January 22, 2000, Online FAQ (<http://www.anybrowser.org/campaign/abfaq.shtml>) Accessed on April 5, 2000, Viewable With Any Browser, Web site (<http://www.anybrowser.org/campaign/>).

[Callaway, 1999] Dustin R. Callaway, *Inside Servlets: Server-Side Programming for the Java™ Platform*, Addison-Wesley, 1999.

[CC/PP, 1999] *Composite Capability / Preference Profiles*, CC/PP, *A user side framework for content negotiation*, World Wide Web Consortium, Edited by Franklin Reynolds, Johan Hjelm, Spencer Dawkins, and Sandeep Singhal, 27 Jul 1999, Latest version available online (<http://www.w3.org/TR/NOTE-CCPP/>) Accessed on June 5, 2000 .

W3C Note.

[CC/PP-ra, 2000] *Composite Capabilities / Preference Profiles: Requirements and Architecture*, CC/PP-ra, World Wide Web Consortium, Edited by Mikael Nilsson, Johan Hjelm, and Hidetaka Ohto, 28 February 2000, Latest version available online (<http://www.w3.org/TR/CCPP-ra/>) Accessed on June 6, 2000 .

W3C Note.

- [CC/PPex, 1999] *CC/PP exchange protocol based on HTTP Extension Framework: A user side framework for content negotiation*, World Wide Web Consortium, Edited by Hideka Ohto and Johan Hjelm, 24 Jun 1999, Latest version available online (<http://www.w3.org/TR/NOTE-CCPPexchange>) Accessed on June 5, 2000.
- W3C Note.*
- [Clark, 1997] James Clark, *Comparison of SGML and XML*, World Wide Web Consortium, 15 Dec 1997, Available online (<http://www.w3.org/tr/note-sgml-xml.html>) Accessed on June 5, 2000.
- W3C Note.*
- [Cocoon, 2000] *Cocoon Documentation | Index*, Apache XML Project, The Apache Software Foundation, 1999, Web site (<http://xml.apache.org/cocoon/>) Accessed on March 13, 2000.
- [Connolly, 2000] Dan Connolly, *Extensible Markup Language (XML) Activity: W3C Architecture Domain Activity Statement*, World Wide Web Consortium, 1998-2000, WWW document (<http://www.w3.org/XML/Activity.html>) Accessed on April 10, 2000.
- [Date, 1995] C. J., *An Introduction to Database Systems*, 6th Edition, Addison-Wesley, 1995.
- [DocZilla, 1999] *DocZilla SGML/XML Module (Alpha3)*, CiTEC Information, 1999, DocZilla Web site (<http://www.doczilla.com/download/index.html>) Accessed on March 9, 2000 .
- [DOM1, 1998] *Document Object Model (DOM) Level 1 Specification*, W3C DOM Working Group, World Wide Web Consortium, 1 October, 1998, Online version (<http://www.w3.org/TR/REC-DOM-Level-1/>) Accessed on March 24, 2000.
- W3C Recommendation.*
- [Flammia, 1997] Giovanni Flammia, “XML and style sheets promise to make the Web more accessible”, 98-99, *IEEE Expert*, **12**, 3, May/June 1997.
- [Floyd, 1999] Michael Floyd, “Building an XML workbench”, Online version (<http://www.webtechniques.com/archives/1999/05/beyo/>) Accessed on March 14, 2000, *Web Techniques*, Miller Freeman, **4**, 5, May 1999.
- [Floyd, 2000] Michael Floyd, “Launching XML Web sites with Rocket”, 44-48, Online version (<http://www.webtechniques.com/archives/2000/02/beyo/>) Accessed on March 21, 2000, *Web Techniques*, Miller Freeman, **5**, 2, February 2000.

- [Fuchs, 1999] Matthew Fuchs, “Why XML is meant for Java”, Online version (<http://www.webtechniques.com/archives/1999/06/fuchs/>) Accessed on March 21, 2000, *Web Techniques*, Miller Freeman, 4, 7, June 1999.
- [Gregersen and Bilstrup, 2000] Claus Leth Gregersen and Bent Bilstrup, “WAP from an XML developer's point of view”, *The Fifth International WAP Developers' Symposium: Conference Proceedings*, May 5, 2000.
- [Harold, 1999] Elliotte Rusty Harold, *XML Bible*, IDG, 1999, An update available online (<http://metalab.unc.edu/xml/books/bible/updates/14.html>) Accessed on April 7, 2000.
- [Holman, 1999a] Ken G. Holman, “What's the big deal with XSL?”, Online article (<http://www.xml.com/xml/pub/1999/04/holman/xsl.html>) Accessed on September 28, 1999, *XML.com*, O'Reilly Network, April 1999.
- [Holman, 1999b] Ken G. Holman, *The XML family of standards, XML Finland'99: SGML Users Group Finland - Conference Paper*, September 23, 1999.
- [Holman, 1999c] Ken G. Holman, *Introduction to DOM and CSS, XML Finland'99: SGML Users Group Finland - Conference Paper*, September 23, 1999.
- [Holman, 1999d] Ken G. Holman, *Introduction to XSLT and XPath, XML Finland'99: SGML Users Group Finland - Conference Paper*, September 23, 1999.
- [Kamada and Miyazaki, 1997] Tomihisa Kamada and Tomohiko Miyazaki, *Client-Specific Web Services by Using User Agent Attributes*, W3C, December 30, 1997, WWW document (<http://www.w3.org/TR/NOTE-agent-attributes.html>) Accessed on April 17, 2000.
- W3C NOTE.*
- [Korpela, 1998] Jukka Korpela, “Lurching toward Babel: HTML, CSS, and XML”, 103-104, 106, *Computer*, IEEE Computer Society, 31, 7, July 1998.
- [Lewis *et al.*, 1989] C. Lewis, D. Hair, and V. Schoenberg, *Generalization, Consistency, and Control: Proc. ACM CHI'89 Conf. (Austin, TX, 30 April 4 - May 5), 1-5.*
- [Lie, 1999] Håkon W. Lie, *Formatting objects considered harmful*, April 15, 1999, An online article published on Håkon Lie's website (<http://www.operasoftware.com/people/howcome/1999/foch.html>) Accessed on June 3, 2000.
- [Lie and Saarela, 1999] Håkon Wium Lie and Janne Saarela, “Multipurpose Web publishing using HTML, XML and CSS”, 95-101, *Communications of the ACM*, 42, 10, October 1999.

- [Lilley, 1997] Chris Lilley and Vincent Quint, *Extensible Stylesheet Language (XSL): XSL information page on the W3C site*, W3C, 1997-2000, <http://www.w3.org/Style/XSL/> Accessed on March 30, 2000.
- [Martin, 2000a] Didier Martin, "Component-based page layouts", Feb. 16, 2000, Online article (<http://www.xml.com/pub/2000/02/16/style/index.html>) Accessed on April 10, 2000, *XML.com: Style Matters*, O'Reilly Network.
- [Martin, 2000b] Didier Martin, "Markup: a family affair", Apr. 5, 2000, Online article (<http://www.xml.com/pub/2000/04/05/style/index.html>) Accessed on April 10, 2000, *XML.com: Style Matters*, O'Reilly Network.
- [Maruyama *et al.*, 1999] Hiroshi Maruyama, Kent Tamura, and Naohiko Uramoto, *XML and Java™: Developing Web Applications*, 2nd Printing, Addison-Wesley, 1999.
- [Meyer, 2000] Eric Meyer, *webreview.com*, Web Review / Miller Freeman, Jan. 4, 2000.
- [Microsoft, 1999] *Microsoft delivers industry's first XML-compliant browser: comprehensive XML support enables developers to build new generation of data-driven applications*, Microsoft Corp., March 31, 1999, Online version (<http://www.microsoft.com/presspass/press/1999/Mar99/XMLISVpr.asp>) Accessed on June 5, 2000, *PressPass: Microsoft's press release archive*.
- [Netscape, 1998] *Netscape accelerates Communicator evolution with first release of next-generation Communicator source code to developer community via Mozilla.org: Industry leading companies support bold move and will participate in early development of the source*, Netscape Communications, March 31, 1998, Online article (<http://home.netscape.com/newsref/pr/newsrelease591.html>) Accessed on June 5, 2000, *Company Press Relations: Netscape Press Release Archive*.
- [O'Reilly, 1999] Tim O'Reilly, "Where the Web leads us", Online article (http://www.xml.com/pub/1999/10/tokyo.html?wwwrrr_19991013.txt) Accessed on June 5, 2000, *XML.com*, O'Reilly Network, Oct. 6, 1999.
- [Ouahid and Karmouch, 1999] H. Ouahid and A. Karmouch, "Converting Web pages into well-formed XML documents", 676-680, *1999 IEEE International Conference on Communications, ICC'99, Conference Record*, IEEE, 1, 6-10 June 1999.
- [P3P, 2000] *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*, W3C Working Draft 04 April 2000 Available online (<http://www.w3.org/TR/P3P/>) Accessed on April 15, 2000, Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, and Joseph Reagle, 04 April 2000.

- [Palola, 1999] Ilkka Palola, "CSS:n käyttö XML-sovelluksissa", 80-88, *XML Finland '99: Conference Proceedings*, SGML/XML Käyttäjäkerho, 23-24.09.1999.
- [Pardi, 1999] William J. Pardi, *XML in Action: Web Technology*, Microsoft Press, 1999.
- [Passani, 2000] Luca Passani, "Building WAP services": *XML and ASP will set you free*, 48-53, Online version (<http://www.webtechniques.com/archives/2000/03/passani/>) Accessed on June 5, 2000, *Web Techniques*, CMP Media, 5, 3, March 2000.
- [PIDL, 1999] *PIDL - Personalized Information Description Language*, Edited by Yuichi Koike, Tomonari Kamba, and Marc Langheinrich, World Wide Web Consortium, 09 Feb 1999, Latest version (<http://www.w3.org/TR/NOTE-PIDL>) Referred version (<http://www.w3.org/TR/1999/NOTE-PIDL-19990209>) Accessed on June 5, 2000.
- W3C Note.*
- [RDF, 1999] *Resource Description Framework (RDF) Model and Syntax Specification*, Ora Lassila and Ralph R. Swick, 22 February 1999, W3C Recommendation 22 February 1999 Available online (<http://www.w3.org/TR/REC-rdf-syntax/>) Accessed on June 5, 2000.
- [RFC2068] *Hypertext Transfer Protocol -- HTTP/1.1: Request for Comments*, R. Fielding, U. C. Irvine, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, IETF, January 1997, Available online (<http://www.ietf.org/rfc/rfc2068.txt?number=2068>) Accessed on June 5, 2000.
- [Sahuguet and Azavant, 1999] Arnaud Sahuguet and Fabien Azavant, "Looking at the Web through XML glasses", 148-159, *Cooperative Information Systems, 1999, CoopIS '99, Proceesings on 1999 IFCIS International Convergence*, IEEE Computer Society, Conference held on September 2-4 , 1999.
- [Shapiro and Varian, 1998] Carl Shapiro and Hal R. Varian, "Versioning": *The smart way to sell information*, 106-114, *Harvard Business Review*, Harvard Business School Publishing, 76, 6, November-December 1998.
- [Siegel, 1997] David Siegel, "The Web is ruined and I ruined it", Published originally on Web Review (<http://www.webreview.com/pub/97/04/11/feature/index.html>). Also published on XML.com (<http://www.xml.com/pub/w3j/s1.people.html>) Accessed on June 5, 2000, *XML.com*, O'Reilly Network, Oct. 2, 1997.
- [StLaurent and Cerami, 1999] Simon St. Laurent and Ethan Cerami, *Building XML Applications*, McGraw-Hill, 1999.

- [Tidwell, 1999] Doug Tidwell, "Tutorial: introduction to XML", Available online in HTML and PDF format. (<http://www-4.ibm.com/software/developer/education/xmlintro/>) Accessed on June 5, 2000, *developerWorks: IBM's Web site for software developers*, Web site (<http://www.ibm.com/developer/>) Accessed on June 5, 2000, July, 1999.
- [Thompson, 2000] Valerie Thompson, "What the hell is... MExE?", 29/03/2000, Online article (<http://www.theregister.co.uk/000329-000007.html>) Accessed on May 29, 2000, *The Register*, Situation Publishing.
- [UAProf, 1999] *Wireless Application Group - User Agent Profile Specification*, WAG UAPROF, Wireless Application Protocol Forum, 10 Nov 1999, Online version in PDF format (<http://www1.wapforum.org/tech/terms.asp?doc=SPEC-UAProf-19991110.pdf>).
- WAP Forum Specifications.*
- [VoiceXMLa, 2000] *Voice eXtensible Markup Language Version 1.00*, VoiceXML, VoiceXML Forum, 07 March 2000, Online version in PDF format (<http://www.voicexml.org/specs/VoiceXML-100.pdf>).
- [VoiceXMLb, 2000] *VoiceXML Forum*, Web site (<http://www.voicexml.org/>).
- [W3C, 1997] *About the World Wide Web Consortium [W3C]: W3C Web page*, Web page (<http://www.w3.org/Consortium/>).
- [Walsh, 1998] Norman Walsh, *A technical introduction to XML*, O'Reilly & Associates, Oct. 3, 1998, Appeared initially in the Winter 1997 edition of the World Wide Web Journal, but revised and XML 1.0 up-to-date edition is available online at XML.com (<http://www.xml.com/pub/98/10/guide0.html>). Accessed on June 4, 2000.
- [Walsh, 1999] Norman Walsh, *The Extensible Style Language*, Online version (<http://www.webtechniques.com/archives/1999/01/walsh/>) Accessed on September 9, 1999, *Web Techniques*, Miller Freeman, 4, 1, January 1999.
- [WBXML, 1999] *Wireless Application Protocol - WAP Binary XML Content Format*, WBXML, Wireless Application Protocol Forum, 4-November-1999, Online version in PDF format. (<http://www1.wapforum.org/tech/terms.asp?doc=SPEC-WBXML-19991104.pdf>) Accessed on June 5, 2000.
- WAP Forum Specification.*
- [WML, 1999] *Wireless Application Protocol - Wireless Markup Language Specification - Version 1.2*, WAP WML, Wireless Application Protocol Forum, 4-November-1999, Online version in PDF format.

(<http://www1.wapforum.org/tech/terms.asp?doc=SPEC-WML-19991104.pdf>)
Accessed on June 5, 2000.

WAP Forum Specification.

[WML Reference, 1999] *WML Reference Version 1.1*, Nokia, September 1999,
Online version at the Forum Nokia Web site (<http://www.forum.nokia.com>)
[Registration required] Accessed on June 5, 2000.

Nokia WAP Developer Forum: Documents.

[XHTML, 2000] *XHTML™ 1.0: The Extensible HyperText Markup Language: A
reformulation of HTML 4 in XML 1.0*, W3C HTML working group, Available
online in multiple formats (<http://www.w3.org/TR/xhtml1/>) Accessed on June
5, 2000, 26 January 2000.

W3C Recommendation.

[XLink, 2000] *XML Linking Language (XLink)*, XLink, Edited by Steve DeRose,
Eve Maler, David Orchard, and Ben Trafford, 21-February-2000, Latest
version (<http://www.w3.org/TR/xlink/>) Accessed on April 7, 2000.

W3C Working Draft.

[XML, 1998] *Extensible Markup Language (XML) 1.0*, REC-xml-19980210,
Edited by Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen, 10-Feb-98,
Available online in multiple formats: HTML, XML, PDF, and PS
(<http://www.w3.org/TR/1998/REC-xml-19980210>) Accessed on June 5, 2000.

W3C Recommendation.

[XML.com, 2000] *XML.com - Standards List*, XML.com, O'Reilly & Associates,
2000, Web page (<http://www.xml.com/pub/stdlist>) Accessed on March 13,
2000.

[XMLEnabler, 1999a] *FAQ for XML Enabler*, IBM Alphaworks, Online FAQ
([http://www.alphaworks.ibm.com/aw.nsf/techFAQS?OpenAgent&FAQXML+
EnablerFAQ](http://www.alphaworks.ibm.com/aw.nsf/techFAQS?OpenAgent&FAQXML+EnablerFAQ)) Accessed on March 10, 2000.

[XML Enabler, 1999b] *alphaWorks License Agreement*, IBM alphaWorks,
Available in a downloadable XML Enabler package on alphaWorks Web site
Web page
([http://www.alphaworks.ibm.com/aw.nsf/frame?ReadForm&/aw.nsf/techmain
/D963C6A8CA1360548825671B00682901](http://www.alphaworks.ibm.com/aw.nsf/frame?ReadForm&/aw.nsf/techmain/D963C6A8CA1360548825671B00682901)) Accessed on March 10, 2000.

[XPath, 1999] *XML Path Language (XPath) Version 1.0*, XPath, Edited by James
Clark and Steve DeRose, 16 November 1999, Online version
(<http://www.w3.org/TR/xpath>) Accessed on June 5, 2000.

W3C Recommendation.

[XPointer, 2000] *XML Pointer Language (XPointer)*, XPointer, Edited by Steve DeRose, Ron Daniel, Jr., and Eve Maler, 6 December 1999, Latest version (<http://www.w3.org/TR/WD-xptr>) Accessed on April 7, 2000.

W3C Working Draft.

[XSL, 2000] *Extensible Stylesheet Language (XSL) Version 1.0*, XSL, W3C XSL Working Group, 1 March 2000, Latest version (<http://www.w3.org/TR/xsl/>) Accessed on March 30, 2000.

W3C Working Draft.

[XSLT, 1999] *XSL Transformations (XSLT) Version 1.0*, XSLT, Edited by James Clark, 16 November 1999, Available online on Online version (<http://www.w3.org/TR/xslt>) Accessed on March 30, 2000.

W3C Recommendation.

[Zelnick, 1998] Nate Zelnick, "Netscape builds XML parser into new version of Navigator", April 6, 1998, Online article (<http://www.internetworld.com/print/1998/04/06/webdev/19980406-parser.html>) Accessed on March 10, 2000, *Internet World*, Penton Media.

Glossary

Here are the essential terms and abbreviations used in this study.

Terms

Unless otherwise mentioned, the definitions are originally created for this thesis.

Content

Information, data.

Dynamic content

Any content that is created by a Web server as a function of request parameters or state of the requested resource [Cocoon, 2000, `dynamic.html`].

Logic

Functionality.

Static content

Information that has been made available on a Web server. Static content remains the same for every request.

Style

Formatting, rendering, presentation.

Web application

An application or system of applications that uses HTTP as its primary transport protocol [Maruyama *et al.*, 1999, p.].

Web content

Any content a Web server sends that to the Web device. Content is the information that is consumed by the user: textual documents, images, sound and video clips, etc. It can be either static or dynamic.

Web device

Any hardware or software through which a user accesses Web content [Lie and Saarela, 1999]. A superconcept of: client (device and/or software), user agent, browser, rendering engine.

Web publication

A Web service that is (mostly) comprised of static Web content.

Web service

The combination of Web application(s) and Web content.

Abbreviations

API

Application Programming Interface

ASP

Active Server Pages

CC/PP

Composite Capabilities / Preferences Profile

CD

Compact Disc

CD-ROM

Compact Disc - Read Only Memory

CDATA

Character data

CGI

Common Gateway Interface

CPI

Capability and Preference Information

CPU

Central Processing Unit

CSS

Cascading Style Sheet

DCP

Dynamic Content Processor

DOM

Document Object Model

DTD

Document Type Definition

ETSI

European Telecommunications Standards Institute

HTML

Hypertext Markup Language

HTTP

Hypertext Transfer Protocol

IEC

International Electrotechnical Commission

IETF

Internet Engineering Task Force

IIOP

Internet Inter-ORB Protocol

IP

Internet Protocol

ISO

International Standard Organization

JSP

Java Server Pages

JVM

Java Virtual Machine

LDAP

Light Directory Access Protocol

MathML

Mathematical Markup Language

MExE

Mobile Station Application Execution Environment

MIME

Multi-purpose Internet Mail Extensions

MSIE

Microsoft Internet Explorer

P3P

Platform for Privacy Preferences

PC

Personal Computer

PDA

Personal Digital Assistant

PDF

Portable Document Format

PI

Processing Instruction

PIDL

Personalized Information Description Language

PS

PostScript

RDF

Resource Description Format

SAX

Simple API for XML

SGML

Standard Generalized Markup Language

SMIL

Synchronized Multimedia...

SMS

Short Message Service

SMTP

Simple Mail Transfer Protocol

SQL

Structured Query Language

SVG

Scalable Vector Graphics

UAProf

User Agent Profile

UML

Unified Modeling Language

URI

Uniform Resource Identifier

URL

Uniform Resource Locator

VoiceXML

Voice eXtensible Markup Language

W3C

World Wide Web Consortium

WAP

Wireless Application Protocol

WBMP

Wireless Bitmap

WBXML

WAP Binary XML

WML

Wireless Markup Language

WWW

World Wide Web

XHTML

Extensible Hypertext Markup Language

XML

Extensible Markup Language

XSL

Extensible Stylesheet Language

XSL:FO

XSL Formatting Objects

XSLT

XSL Transformations

XSP

Extensible Server Pages

Appendix A. Trilogy DTD

This is the Document Type Definition of our sample XML document.

Example A-1. Trilogy DTD

```
<!ELEMENT trilogy (author, part, part, part)>
<!ATTLIST trilogy title CDATA #IMPLIED>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT part (#PCDATA | title | image?)+>
<!ELEMENT image EMPTY>
<!ATTLIST image src CDATA #REQUIRED>
<!ENTITY s1 "Sponsored by Kuat Drive Yards">
```

Appendix B. XSL and CSS

This example shows how XSL and CSS style sheets can be used together with a browser that supports both formats. The browser used here is Microsoft Internet Explorer 5.5.

Example B-1. Source document `sote.xml`

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="sote.xsl"?>

<book>Shadows of the Empire</book>
```

Example B-2. XSLT style sheet `sote.xsl`

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
    <html><head><title>Example</title>
      <link rel="stylesheet" href="sote.css" type="text/css" />
    </head><body>
      <u><xsl:value-of select="book" /></u>
    </body></html>
  </xsl:template>

</xsl:stylesheet>
```

The source document after the XSLT transformation from XML to HTML:

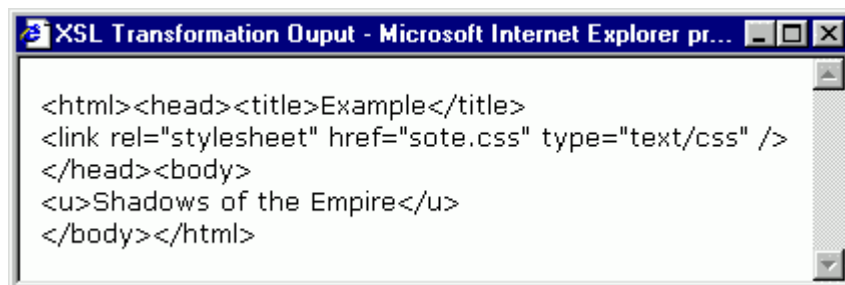


Figure B-1. XML Transformed to HTML

Example B-3. CSS style sheet `sote.css`

```
body {  
    font-style: italic;  
    font-weight: bold;  
}
```

The result viewed with MSIE5.5:

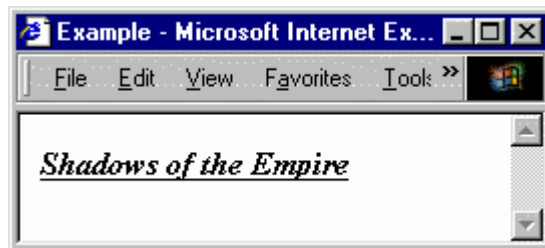


Figure B-2. XML with XSL and CSS Styling in MSIE5.5

Notice that the underlining is done with the XSLT transformation, while italics and bolding is added with CSS style sheet.

Appendix C. Inline CC/PP

This appendix has an example of an inline encoding of a CC/PP.

Example C-1. A sample profile for a hypothetical smart phone [CC/PP, 1999].

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:prf="http://www.w3.org/TR/WD-profile-vocabulary#">
  <rdf:Description about="HardwarePlatform">
    <prf:Defaults
      Vendor="Nokia"
      Model="2160"
      Type="PDA"
      ScreenSize="800x600x24"
      CPU="PPC"
      Keyboard="Yes"
      Memory="16mB"
      Bluetooth="YES"
      Speaker="Yes" />
    <prf:Modifications
      Memory="32mB" />
  </rdf:Description>
  <rdf:Description about="SoftwarePlatform">
    <prf:Defaults
      OS="EPOC1.0"
      HTMLVersion="4.0"
      JavaScriptVersion="4.0"
      WAPVersion="1.0"
      WMLScript="1.0" />
    <prf:Modifications
      Sound="Off"
      Images="Off" />
  </rdf:Description>
  <rdf:Description about="EpocEmail1.0">
    <prf:Defaults
      HTMLVersion="4.0" />
  </rdf:Description>
  <rdf:Description about="EpocCalendar1.0">
    <prf:Defaults
      HTMLVersion="4.0" />
  </rdf:Description>
  <rdf:Description about="UserPreferences">
    <prf:Defaults
      Language="English"/>
  </rdf:Description>
</rdf:RDF>
```

Appendix D. CC/PP with Indirect References

This appendix has the same profile information than the previous appendix but with indirect references.

Example D-1. User agent profile [CC/PP, 1999].

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:prf="http://www.w3.org/TR/WD-profile-vocabulary#">
  <rdf:Description about="HardwarePlatform">
    <prf:Defaults>
      <rdf:li re-source="http://www.nokia.com/profiles/2160"/>
    </prf:Defaults>
    <prf:Modifications>
      Memory="32mB" />
    </rdf:Description>
    <rdf:Description about="SoftwarePlatform">
      <prf:Defaults>
        <rdf:li re-source="http://www.symbian.com/profiles/pda"/>
      </prf:Defaults>
      <prf:Modifications>
        Sound="Off"
        Images="Off" />
      </rdf:Description>
      <rdf:Description about="EpocEmail">
        <prf:Defaults>
          <rdf:li re-
source="http://www.symbian.com/epoc/profiles/epocemail" />
        </prf:Defaults>
      </rdf:Description>
      <rdf:Description about="EpocCalendar">
        <prf:Defaults>
          <rdf:li re-
source="http://www.symbian.com/epoc/profiles/epoccal"/>
        </prf:Defaults>
      </rdf:Description>
      <rdf:Description about="UserPreferences">
        <prf:Defaults>
          Language="English" />
        </prf:Defaults>
      </rdf:Description>
    </rdf:RDF>
```

Then, the profile provided by the hardware vendor.

Example D-2. Hardware profile [CC/PP, 1999].

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description
    Vendor="Nokia"
    Model="2160"
    Type="PDA"
    ScreenSize="800x600x24"
    CPU="PPC"
    Keyboard="Yes"
    Memory="16mB"
    Bluetooth="YES"
    Speaker="Yes" />
</rdf:RDF>
```

Finally, the profiles provided by the software platform and application vendors.

Example D-3. Web browser profile [CC/PP, 1999].

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description
    OS="EPOC1.0"
    HTMLVersion="4.0"
    JavaScriptVersion="4.0"
    WAPVersion="1.0"
    WMLScript="1.0" />
</rdf:RDF>
```

Example D-4. Mail application profile [CC/PP, 1999]

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description
    Version="EpocEmail1.0"
    HTMLVersion="4.0" />
</rdf:RDF>
```

Example D-5. Calendar application profile [CC/PP, 1999]

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description
    Version="EpocCalendar1.0"
    HTMLVersion="4.0" />
</rdf:RDF>
```

Appendix E. Cocoon XSP

Example E-1. Embedded XSP logic [Cocoon, 2000, `xsp.html`].

```
<?xml version="1.0"?>
<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="sample.xsl" type="text/xsl"?>

<xsp:page language="java"
xmlns:xsp="http://www.apache.org/1999/XSP/Core">
  <page title="Time of Day">

    <xsp:logic>
      // Define a variable to hold the time of day
      Date now = new Date();
    </xsp:logic>

    <p>
      To the best of my knowledge, it's now
      <!-- Substitute time of day here -->
      <xsp:expr>now</xsp:expr>
    </p>
  </page>
</xsp:page>
```

Upon XSP and XSLT processing , the page should yield something like:

```
<html>
<head><title>Time of Day</title></head>
<body>
  <h3 style="color: navy; text-align: center">Time of Day</h3>
  <p>It's now Thu Dec 23 20:11:18 PST 1999</p>
</body>
</html>
```

Appendix F. Cocoon DCP

Example F-1. A Sample Document Containing DCP Processing Instructions
[Cocoon, 2000, dcp.html].

```
<?xml version="1.0"?>
<?cocoon-process type="dcp"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="sample.xsl" type="text/xsl"?>

<page>
<?dcp-object name="util" language="javascript" code="test.js"?>

  <title>A Dynamic JavaScript Cocoon Page</title>
  <p>
    Hi, I'm a dynamic JavaScript page generated by Cocoon on
    <?dcp-content method="util.getSystemDate" format="MM/dd/yyyy"?>
  </p>
  <p>
    During my currrent incarnation, I've been hit
    <?dcp-content method="util.getCount"?>
    times.
  </p>
</page>
```

Appendix G. Selected Schedule Board Source Files

Example G-1. Source code of `board.xml`

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Written by Timi Soinio "Timi.Soinio@nokia.com" -->

<?cocoon-process type="xslt"?>
<?xml-stylesheet href="board-xsp.xsl" type="text/xsl"?>
<scheduleboard>
  <!-- servlet-parameters are needed for the board-xsp.xsl -->
  <!-- They are replaced by the HTTP request parameters, -->
  <!-- if there are any. -->
  <servlet-parameters/>

  <person id="AMSa" name="Anna Mari">
    <status value="out"/>
    <week>
      <day name="wednesday">Asioilla</day>
    </week>
  </person>
  <person id="TSo" name="Timi">
    <status value="in"/>
    <week>
      <!-- Note: days do not have to be ordered. -->
      <day name="wednesday">UML training</day>
      <day name="monday">UML training</day>
    </week>
  </person>

  <!-- Other persons are not shown in this example. -->

</scheduleboard>
```

Example G-2. Source code of `board-xsp.xsl`

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Written by Timi Soinio "Timi.Soinio@nokia.com" -->

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

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

```

    <xsl:processing-instruction name="cocoon-
process">type="xsp"</xsl:processing-instruction>

    <xsl:processing-instruction name="cocoon-
process">type="xslt"</xsl:processing-instruction>
    <xsl:processing-instruction name="xml-stylesheet">href="board-xsp-
html.xml" type="text/xml"</xsl:processing-instruction>
    <xsl:processing-instruction name="xml-stylesheet">href="board-xsp-
wml.xml" type="text/xml" media="wap"</xsl:processing-instruction>

    <xsp:page language="java"
xmlns:xsp="http://www.apache.org/1999/XSP/Core">

    <xsp:logic>
        Calendar cal = new GregorianCalendar(new Locale("fi", "FI"));
    </xsp:logic>

    <xsl:copy>
        <xsl:apply-templates/>
    </xsl:copy>

    </xsp:page>
</xsl:template>

<xsl:template match="scheduleboard">
    <xsl:copy>
        <xsp:attribute
name="week"><xsp:expr>cal.get(Calendar.WEEK_OF_YEAR)</xsp:expr></xsp:a
ttribute>
        <xsl:apply-templates/>
    </xsl:copy>
</xsl:template>

<xsl:template match="servlet-parameters">
    <xsp:logic><![CDATA[
        Enumeration e = request.getParameterNames();
        if ((e != null) && (e.hasMoreElements())) {
            <request-params>
                <xsp:logic><![CDATA[
                    while (e.hasMoreElements()) {
                        String k = (String) e.nextElement();
                        String val = request.getParameter(k);
                        String vals[] = request.getParameterValues(k);
                    <param>
                        <xsp:attribute name="name">
                            <xsp:expr>k</xsp:expr>
                        </xsp:attribute>
                        <xsp:logic> <![CDATA[
                            for (int i = 0; i < vals.length; i++) {
                                <value>
                                    <xsp:expr>vals[i]</xsp:expr>
                                </value>
                            }
                        ]
                    ]>
                ]>
            ]>
        }
    ]>
    </xsp:logic>

```

```

        </xsp:logic>
    </param>
}
</xsp:logic>
</request-params>
}
</xsp:logic>
</xsl:template>

<xsl:template match="*|@*|text()|comment()">
    <xsl:copy>
        <xsl:apply-templates select="*|@*|text()|comment()"/>
    </xsl:copy>
</xsl:template>

</xsl:stylesheet>

```

Example G-3. Source code of board-xsp-html.xsl

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Written by Timi Soinio "timi.soinio@nokia.com" -->

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

    <xsl:import href="board-xsp-personhtml.xsl"/>
    <xsl:import href="board-xsp-generalhtml.xsl"/>

    <xsl:param name="pID" select="//request-
params/param[@name='personID']/value"/>
    <xsl:param name="week" select="/scheduleboard/@week"/>

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

    <xsl:template match="text()|@"><xsl:value-of se-
lect="."/></xsl:template>

    <xsl:template match="/">
        <xsl:processing-instruction name="cocoon-
format">type="text/html"</xsl:processing-instruction>
        <html>
            <xsl:choose>
                <xsl:when test="$pID">
                    <xsl:apply-templates mode="person"/>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:apply-templates mode="general"/>
                </xsl:otherwise>
            </xsl:choose>
        </html>
    </xsl:template>

```

```
</xsl:stylesheet>
```

Example G-4. Source code of `board-xsp-generalhtml.xsl`

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Written by Timi Soinio "timi.soinio@nokia.com" -->

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:html="http://www.w3.org/Profiles/XHTML-transitional">

<xsl:output method="html"/>

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

<xsl:template match="text()|@"><xsl:value-of se-
lect="."/></xsl:template>

<xsl:template match="scheduleboard" mode="general">
  <head>
    <title>Schedule Board | Week <xsl:value-of se-
lect="$week"/></title>
    <link rel="stylesheet" type="text/css" href="board.css"/>
  </head>
  <body bgColor="white">
    <h1>Schedule Board (week <xsl:value-of select="$week"/>)</h1>
    <table border="0" cellPadding="5" cellspacing="0">
      <tr class="headrow">
        <th>
          Person
        </th>
        <th>
          Status
        </th>
        <th>
          Monday
        </th>
        <th>
          Tuesday
        </th>
        <th>
          Wednesday
        </th>
        <th>
          Thursday
        </th>
        <th>
          Friday
        </th>
      </tr>

      <!-- tr tags for days of week are in the following template -->
```

```

        <xsl:apply-templates mode="general">
            <!-- Persons do not need to be in alphabetical order. -->
            <!-- Sort by person name: -->
            <xsl:sort select="@name"/>
        </xsl:apply-templates>

    </table>
</body>
</xsl:template>

<xsl:template match="person" mode="general">
    <tr>
        <xsl:attribute name="class">
            <xsl:choose>
                <xsl:when test="position() mod 2 = 0">evenrow</xsl:when>
                <xsl:otherwise>oddrow</xsl:otherwise>
            </xsl:choose>
        </xsl:attribute>
        <td>
            <a>
                <xsl:attribute name="href">board.xml?personID=<xsl:value-of
select="@id"/></xsl:attribute>
                <xsl:value-of select="@name"/>
            </a>
        </td>
        <xsl:apply-templates mode="general"/>
    </tr>
</xsl:template>

<xsl:template match="status" mode="general">
    <td>
        <xsl:choose>
            <xsl:when test="@value='in'">
                <xsl:attribute name="class">in</xsl:attribute>
            </xsl:when>
        </xsl:choose>
        <xsl:value-of select="@value"/>
    </td>
</xsl:template>

<xsl:template match="week" mode="general">
    <td>
        <xsl:choose>
            <xsl:when test="day[@name='monday']">
                <xsl:apply-templates select="day[@name='monday']"/>
            </xsl:when>
            <xsl:otherwise>
                —
            </xsl:otherwise>
        </xsl:choose>
    </td>
    <td>
        <xsl:choose>

```



```

        <xsl:when test="day[@name='tuesday']">
            <xsl:apply-templates select="day[@name='tuesday']"/>
        </xsl:when>
        <xsl:otherwise>
            —
        </xsl:otherwise>
    </xsl:choose>
</td>
<td>
    <xsl:choose>
        <xsl:when test="day[@name='wednesday']">
            <xsl:apply-templates select="day[@name='wednesday']"/>
        </xsl:when>
        <xsl:otherwise>
            —
        </xsl:otherwise>
    </xsl:choose>
</td>
<td>
    <xsl:choose>
        <xsl:when test="day[@name='thursday']">
            <xsl:apply-templates select="day[@name='thursday']"/>
        </xsl:when>
        <xsl:otherwise>
            —
        </xsl:otherwise>
    </xsl:choose>
</td>
<td>
    <xsl:choose>
        <xsl:when test="day[@name='friday']">
            <xsl:apply-templates select="day[@name='friday']"/>
        </xsl:when>
        <xsl:otherwise>
            —
        </xsl:otherwise>
    </xsl:choose>
</td>
</xsl:template>

<xsl:template match="day" mode="general">
    <xsl:number level="any"/>.
    <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>

```

Example G-5. Source code of board.css

```

<!-- written by Timi Soinio "timi.soinio@nokia.com" -->

A          {

```

```

        color: #0000FF;
        font-weight:bold;
        font: 12pt palatino, times
    }
    BODY
    {
        margin-left:10px;
        font: 9pt palatino, times;
        color: #000080;
        background-color: white;
        margin-top: 0px;
        margin-right: 0px
    }
    H1
    {
        font: 22pt palatino, times;
        color: #000080
    }
    TH
    {
        font: 12pt palatino, times;
        font-weight: bold;
        color: #000080
    }
    TD
    {
        font: 12pt palatino, times;
        color: #000080
    }
    .headrow
    {
        background-color: #bebebe;
    }
    .evenrow
    {
        background-color: #efefef;
    }
    .oddrow
    {
        background-color: #cccccc;
    }
    .in
    {
        font-weight: bold;
    }

```

Example G-6. Source code of `update-xsp.xml`

```

<?xml version="1.0"?>
<!-- Written by Timi Soinio "timi.soinio@nokia.com" -->

<?cocoon-process type="xsp"?>

<xsp:page language="java"
xmlns:xsp="http://www.apache.org/1999/XSP/Core">

    <xsp:structure>
        <xsp:include>org.apache.xerces.parsers.DOMParser</xsp:include>
        <xsp:include>org.apache.xml.serialize.*</xsp:include>
        <xsp:include>org.w3c.dom.*</xsp:include>

```

```

<xsp:include>org.xml.sax.SAXException</xsp:include>
<xsp:include>java.io.*</xsp:include>
<xsp:include>java.util.*</xsp:include>
</xsp:structure>

<xsp:logic><![CDATA[

    /**
     * Return 'null' if first person in not found
     */
    private Node getFirstPerson(Node node) {
        // returns 'null' if first person in not found
        if (node == null || node.getNodeName().equals("person"))
            // return 'null' or 'person' node
            return node;
        else {
            Node child = node.getFirstChild();
            if (child == null)
                return getFirstPerson(node.getNextSibling());
            else return getFirstPerson(child);
        }
    }

    /**
     * Get the node with the specified person id.
     * Return the person node or 'null' if the person is not found.
     */
    private Node getPersonNode(String pID, Node node) {
        Node personNode = getFirstPerson(node);
        // NOTE: it is important to FIRST check for the validity of
        'personNode'
        while (personNode != null
            && !personNode
            .getAttributes().getNamedItem("id").getNodeValue().equals(pID)) {
            personNode = getFirstPerson(personNode.getNextSibling());
        }
        // Now 'personNode' is either null or the node for the searched
        person
        return personNode;
    }

    private Node makePersonNode(Document doc, String id, String name,
        String statval, Hashtable days){
        Node newPerson = null;
        try {
            /**
             * Create element 'week' and all its child elements (days)
             */
            Element week = doc.createElement("week");
            for (Enumeration e = days.keys(); e.hasMoreElements(); ) {
                String dayName = (String) e.nextElement();
                Element day = doc.createElement("day");
                day.setAttribute("name", dayName);
            }
        }
    }

```

```

day.appendChild(doc.createTextNode((String)days.get(dayName)));
    week.appendChild(day);
}

/*
 * Create 'status' element
 */
Element status_elem = doc.createElement("status");
status_elem.setAttribute("value", statval);

/*
 * Create 'person' element
 */
Element person = doc.createElement("person");
person.setAttribute("name", name);
person.setAttribute("id", id);

/*
 * Append 'status' and 'week' to 'person'
 */
person.appendChild(status_elem);
person.appendChild(week);

newPerson = person;
}
catch (Exception e) {
    e.printStackTrace();
}
return newPerson;
}

synchronized private String update(HttpServletRequest request,
HttpServletResponse response) {

    /*
     * Read some properties from the configuration file
     */
    String propertyFile = XSPU-
til.pathComponent(request.getPathTranslated()) +
        System.getProperty("file.separator") +
        "board.properties";
    Properties properties = new Properties();
    try {
        FileInputStream fiStream = new FileInputStream(propertyFile);
        properties.load(fiStream);
        fiStream.close();
    } catch (IOException e) {
        return e.toString();
    }

    /*
     * Setup parameters from the request

```

```

        */
        String[] weekdays = {"monday", "tuesday", "wednesday", "thurs-
day", "friday"};
        String xmlFile = properties.getProperty("xml_file",
"board.xml");
        String pID      = re-
quest.getParameter(properties.getProperty("rp_person_id", "per-
sonID"));
        String status   = re-
quest.getParameter(properties.getProperty("rp_status", "status"));
        Hashtable days = new Hashtable();
        String dayParam;
        for (int i=0; i < weekdays.length; i++) {
            dayParam = request.getParameter(
                properties.getProperty("rp_"+weekdays[i], week-
days[i])).trim();
            if (!dayParam.equals("")) {
                days.put(weekdays[i], dayParam);
            }
        }

        /*
        * All fine so far, proceed the update operation
        */
        try {
            /*
            * Try to read XML document
            */
            DOMParser parser = new DOMParser();
            parser.parse(xmlFile);
            Document doc = parser.getDocument();
            /*
            * Document is well-formed
            */

            /*
            * Search the required person node.
            */
            Node personNode = getPersonNode(pID,
doc.getDocumentElement());
            if (personNode == null)
                return "Person not found";
            else {

                /*
                * Person node is found; make a new person node with the
provided data.
                */
                String name = person-
Node.getAttributes().getNamedItem("name").getNodeValue();
                Node newPerson = makePersonNode(doc, pID, name, status,
days);

```

```

        parentNode.getParentNode().replaceChild(newPerson, person-
Node);

        /*
         * Now that the document is created we need to *serialize*
it to file.
        */
        OutputFormat format = new OutputFormat(doc,
            properties.getProperty("output_encoding", "iso-8859-1"),
            true);
        format.setLineSeparator("\r\n");
        format.setLineWidth(72);
        format.setIndent(2);
        format.setPreserveSpace(true);

    /*
     * System.out is for debugging purposes
     * XMLSerializer serializer = new XMLSerializer(System.out,
format);
    */

        FileOutputStream file = new FileOutputStream(xmlFile);
        XMLSerializer serializer = new XMLSerializer(file, format);
        serializer.serialize(doc);
        file.flush();
        file.close();
    }
}
catch (Exception e) {
    return e.toString();
}
/*
 * update successful
 */

    return "success";

}

/*
 * NOTE: redirect has a hard coded URL ("board.xml")
 *
 * Trying to get the URL dynamically, e.g. using 're-
quest.getHeader("Referer")' does not
 * seem to work with a WML request (at least not in the Nokia WAP
Toolkit)
 *
 * It may help if the transformation to WML is made before redirect-
ing.
 */
private String redirect(HttpServletRequest request, HttpServletResponse response) {
    String newURL = null;

```

```
newURL = "board.xml";
try {
    response.sendRedirect(newURL);
    update(request, response);
} catch (java.io.IOException e) {
    return e.toString();
}
response.setStatus(HttpServletResponse.SC_MOVED_PERMANENTLY);
response.setHeader("Location", newURL);
return newURL;
}
]]>
</xsp:logic>

<page title="Redirect">
    Let's redirect to...
    <xsp:expr>redirect(request, response)</xsp:expr >
</page>

</xsp:page>
```