FlexXML:Engineering a More Flexible and Adaptable Web

Alan Kaplan and Jack Lunn
Department of Computer Science
Clemson University
Clemson, SC 29634-0974 USA
{kaplan,jlunn}@cs.clemson.edu

Abstract

Engineering applications for the World Wide Web is becoming increasingly difficult. Web engineers are forced to contend with a heterogeneous collection of Web services, which differ in terms of available bandwidth, display quality and connection type. Faced with these sources of heterogeneity, Web engineers often publish separate, typically redundant Web sites, each of which is tailored to a specific Web device.

This paper describes an XML-based approach, called FlexXML, that enables a more flexible and adaptable Web. Using FlexXML, a Web engineer publishes a single Web site, using XML to describe the site's content and XSL to specify a collection of style sheets. FlexXML allows a Web user to specify a document quality (e.g., text- or graphics-based) that they desire from a Web site. Based upon the Web user's preferences and browser environment, the FlexXML framework automatically selects a suitable XSL style sheet, creates the document and delivers the appropriate content.

1. Introduction

The World Wide Web (WWW) has evolved quite dramatically since its inception. In its early years, the WWW was a relatively simple and homogeneous environment. Browsers were generally confined to desktop platforms, while Web site information consisted primarily of plain text. Much of the initial power and popularity of the WWW came from its simple and efficient HTML (Hypertext Markup Language)-based model of site design, delivery and access. Using HTML and an appropriate HTTP (Hypertext Transfer Protocol) WWW server, a Web engineer could create and publish information that could be accessed efficiently by a Web user using almost any browser running on virtually any platform.

Nearly a decade later, the WWW has been transformed into an extremely sophisticated and heterogeneous environment, consisting of a diverse collection of browsers, devices and network connections. Although browsers generally support the HTML standard, individual browsers often have idiosyncrasies that can influence the design of a Web site. Moreover, while browsers continue to be developed for desktop platforms, browser technology is making its way into a variety of other Internet devices, such as handheld organizers and cellular phones. Web users today are also more likely to utilize a variety of network connections, ranging from high-speed networks in the workplace to low-speed modems on the road.

Despite these significant changes, the publication and delivery model that was central to the early WWW has largely remained the same. This situation is extremely unfortunate since it has had an enormous impact on how Web engineers create and maintain Web sites and how Web users access Web sites. More specifically, although the information content provided by a Web site is essentially the same, how the site is delivered efficiently and displayed can be significantly different for different browsers, devices and network connections.

The emergence of the XML-(eXtensible Markup Language) and XSL (eXtensible Stylesheet Language)-based model of Web site design, delivery and access [1, 4] represents an important opportunity for both Web engineers and users. XML is used to specify information content and structure, while XSL is used to describe translations from XML to appropriate representations.

This paper presents an extension to the current XML/XSL model, called FlexXML, whose overall goal is to hide the various sources of heterogeneity that are inherent in the WWW. FlexXML compliments XML/XSL by allowing Web engineers to create a single WWW site that can be accessed by Web users operating in heterogeneous environment. The approach also allows users to easily customize their browser environments to adapt to different Web site representations without sacrificing the quality of information.

The remainder of this paper is organized as follows.

0-7695-1062-0/01 $10.00 © 2001 IEEE

To motivate the FlexXML framework, Section 2 provides a characterization of the sources of heterogeneity in the WWW and their impact on both Web engineers and users. The FlexXML framework in presented in Section 3 and a FlexXML prototype is described in Section 4. An illustration of FlexXML is given in Section 5. A comparison of the FlexXML approach with related approaches is given in Section 6. The paper concludes with a summary in Section 7.

2. Heterogeneous Browser Environments

Today's browser environments have become extremely sophisticated and diverse. In particular, three primary sources of complexity and heterogeneity include network bandwidth, client browser applications and hardware devices. Due to variations and differences in the browser environment, both developers and users encounter numerous difficulties in their respective interactions with the WWW In the remainder of this section, we outline the problems associated with each of these sources of browser environment heterogeneity. In Section 3, we present the FlexXML framework and show how it addresses these problems.

The Multiple Bandwidth Problem It is common for a Web user to utilize a range of connections to the WWW, each with different bandwidth capacities. For example, a user might have a T1 connection in the workplace, a 56K modem at home and a 14.4K cellular modem on the road. Designing a Web site that accounts for bandwidth differences is a complex task, requiring Web engineers to balance the need for fast downloads with the attractive content options (e.g., video, audio, graphics, etc.) that high speed connections allow

When faced with the bandwidth problem, Web engineers generally offer two alternatives. At best, they provide a redundant, low-bandwidth version of a Web site and, at worst, they simply ignore the multiple bandwidth problem and provide only a single version of a site. While the former approach is sometimes effective, it forces Web engineers to maintain (at least) two different source trees, one for high-bandwidth access and another for low-bandwidth access. Maintaining two source trees is not only time consuming, but often inconsistencies arise between the low-bandwidth and high-bandwidth versions of a Web site. Frequently only a subset of information is presented in a low- bandwidth format, so only high-bandwidth users are able to access the full content of the Web site. In addition, Web sites generally require explicit and manual navigation to the low- and high-bandwidth versions of the site. Low-bandwidth users often must wait for a graphics-intensive page to load before they are presented with a link allowing them to navigate to the low-bandwidth version of the site.

The single site alternative does not take bandwidth into consideration. This "lowest common denominator" approach unfortunately fails to leverage the potential of high-bandwidth connections. Other Web sites, though similarly unappealing, are designed specifically for high-bandwidth users. Under this option, low-bandwidth users must either wait a considerable amount of time for the site to download or are must go elsewhere to find the information they require.

Based on the capacity of their bandwidth, users often maintain separate URLs for particular versions of a site. They must record both the address of the (default) graphics-based site, when they expect full graphics, and the address of the text-based site, when they are willing to forgo graphics at the expense of faster access times. Other common practices employed by users include using text-only browsers or manually toggling between graphics and text display capabilities in a browser. None of these practices is very appealing to users. Managing multiple names for a single site is extremely confusing, while turning off graphics capabilities generally results in a loss of document quality and information.

The Multiple Browser Application Problem Another problem faced by Web engineers is the proliferation of different Web browser applications, such as Lynx, Netscape Communicator and Microsoft Internet Explorer. Web browsers differ in their capabilities as well as their support for current (so called) standards. For example, the Lynx browser only downloads the text portion of a Web site. Because it ignores graphics, Lynx is very fast, which makes it a viable solution for users with slow connections to the Internet. In order to work effectively with Lynx, Web engineers must provide an alternative textual description of every graphic using *alt-text* tags (i.e., alternative text tags). The use of alt-text tags is important because many hypertext links use for navigation are hidden behind images. Thus, if an alt-text tag is missing, navigation with Lynx is nearly impossible. While this problem can be eliminated through proper HTML coding style (i.e., always using alt-text tags in conjuction with default graphic images), Lynx must still reformat the presentation of a Web page dynamically. The drawback of Lynx is that even with alt-text tags specifications for every image, many Web sites still look poor and are difficult to navigate.

Many browsers also include proprietary extensions, which naturally are not supported by other browsers. To account for differences among browsers, many Web engineers (again) resort to the "lowest common denominator" approach and suffer the associated consequences. Alternatively, Web sites are often designed to work only for a specific type of browser (e.g., Internet Explorer or Netscape). Web users suffer since they must change browsers in order to access the browser-specific site.

The Multiple Device Problem The rise of portable Internet devices, such as cellular phones and portable digital assistants (PDAs), presents yet another problem for Web engineers and users. These new portable devices are problematic not only because they utilize low-bandwidth connections, but also because their displays and memory capacities are limited. Typical low-bandwidth or text-only solutions are not sufficient because they are still designed with large displays in mind. Such sites will not display properly on devices with limited display sizes. An additional problem is that most portable devices support only two-bit graphics, so desktop browser images must be converted before they can be viewed.

Web engineers have responded to the portable device problem in three ways. The first is to create a separate, duplicate site made strictly for portable devices. The problems associated with maintaining duplicate sites are detailed above. The second solution involves transforming existing Web sites in real time into a format viewable on the portable device. This transformation can occur on the server side, client side, or on a middleware proxy. The problem with transformation-related solutions is that the transformed document often looks distorted on a portable device. This problem occurs because the Web engineer has sacrificed control over the layout of the page to the transformation engine. Recognizing the difficulties associated with presenting Web pages on portable devices, Web engineers have turned to a third solution, namely utilizing Wireless Markup Language (WML) [10]. WML is optimized for devices with small screens and low memory capacities. Because existing Web sites are coded with HTML, they must be translated to WML in order to be viewed on WML devices.

3. The FlexXML Framework

One of the primary benefits of XML is that separates information content and structure from information presentation. Accompanying XSL specifications allow different presentations to be produced using a single XML specification. Unfortunately, the XML/XSL standard by itself does not provide any means to automate the selection of particular XSL style sheets for a specific XML specification in a given context.

The FlexXML framework is an extension to XML/XSL. Using XML, a Web engineer develops a single specification of the content for a Web site and creates various XSL style sheets, each of which are tailored for different bandwidths, browser applications and devices. For example, one XSL specification might be used to create an HTML presentation for a high-bandwidth connection on graphics-intensive

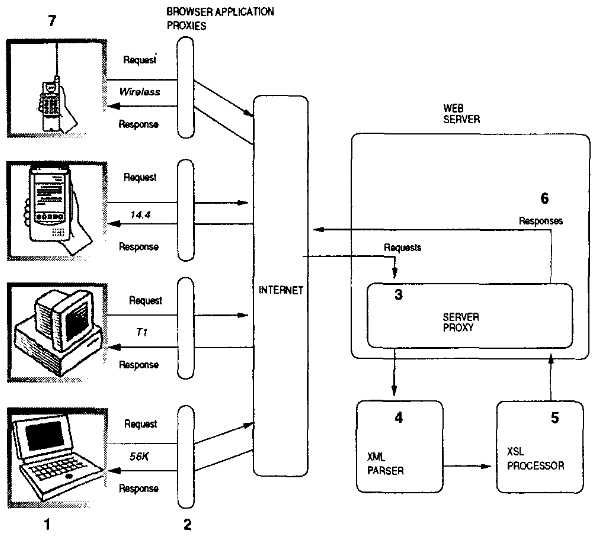

Figure 1. The FlexXML Framework

display using Internet Explorer. Another XSL style sheet (for the same XML specification) might create a WML presentation for a low-bandwidth connection on a text-based display using a PDA browser application. Based on the particular browser environment, the FlexXML framework automatically selects the required XSL style sheet, applies it to the XML specification and transmits the appropriate content. Thus, Web engineers do not have to maintain multiple versions of a particular Web site.

FlexXML also offers Web users a unified view of a Web site. A particular Web site has a single URL, independent of a user's, potentially changing, browser environment. When accessing a site, the content most appropriate to their browser environment is automatically produced. The FlexXML framework also allows users to customize their environment. Users can indicate the kind of documents they expect from a site (e.g., text-based or graphics-intensive) as well as indicate their current network connection. FlexXML uses this information to transmit information that best fits a user's browser environment.

Figure 1 gives a conceptual overview of the FlexXML framework and its relationship to WWW services. The FlexXML framework augments existing WWW services with following four components: a *browser application proxy*, a *Web server proxy*, an *XML parser*, and an *XSL processor*. Each of these components, and their relationship to one another, is described in detail below.

The *browser application proxy* is a transparent extension to existing bowser applications. Its purpose is to communicate browser environment features directly to a Web server proxy (see below). There are two primary categories of browser environment features. Browser customization features include an indication of the current bandwidth and a specification of text-only or graphics-rich content, while browser application features include the browser type and the browser device. These features can be specified by a user and/or automatically supplied by the browser. Browser environment features can be used to account for other important characteristics of an individual's browser environment, such as time zone and language information.

The browser application proxy component is transparent since it does not require modifications to existing browser applications. Once configured, the browser application proxy sits between a user's Web browser application and a Web server. The browser application proxy intercepts all HTTP requests coming from a browser application. It then packages each browser environment feature in the form of a request header and appends it to the original HTTP request[1]. The proxy server then forwards the amended request to the Web server.

As shown in Figure 1, the FlexXML *Web server proxy* sits between the browser application proxy and an actual Web server. Its main purpose is to intercept and process the HTTP headers that are added by a FlexXML browser application proxy. The Web server proxy uses this information to select an XSL stylesheet that best satisfies the user's browser environment features.

For example, an HTTP header created by the FlexXML framework might indicate a request for "text-only" documents, while another header might indicate a request for "graphics-intensive" documents. Using this information, a Web server proxy automatically selects an XSL style sheet best suited to transform an XML document into the desired representation.

The server proxy forwards an XML document to the *XML parser*. The parser generates a tree representation of the XML document and then passes the tree to the *XSL processor*. Using the XSL style sheet selected by the server proxy, the XSL processor traverses the tree and generates the appropriate representation for the Web browser by following the translation rules given in the XSL stylesheet.

4. Prototype

This section describes a prototype implementation of the FlexXML framework. This prototype makes extensive use of existing Java-based software components. One of the benefits of the FlexXML framework is that it does not require modifications to existing web browser applications and web servers.

The browser application proxy is implemented as a pure Java application. The proxy includes a graphical user interface (GUI) that allows users to select the type of representation (text or graphics) that they expect to receive. Because it is written in Java, the browser application proxy is able to run on a variety of operating systems without recompilation, including Linux, Solaris and Windows, and devices, including desktop workstations, portable digital assistants and cellular phones.

A user must configure their browser application to direct HTTP traffic to the port where the browser application proxy is listening. The proxy may be accessed locally or remotely over a network. In the case of a local proxy, the user directs the browser application to a local host, while for remote proxies, the user points their browser to the machine running the proxy on the network. Any browser application that can redirect HTTP connections to an HTTP proxy is capable of using the browser application proxy. Specifically, our browser application proxy has been tested using a variety of browser applications. Included among these are Netscape Communicator, Lynx, Internet Explorer and Palmscape.

As described in Section 3, all HTTP requests are redirected to the browser application proxy. In our prototype implementation, the browser application proxy adds a single header called `representation` to every request. The `representation` header can have one of two values, either `text` or `graphics`. The user can toggle this value via the browser application proxy's GUI. Additionally, the web server proxy reads the value of the HTTP standard `user-agent` header in order to determine what type of browser is making the request. Based upon these two headers, the web server proxy selects an XSL style sheet and creates an instance of an XSL processor with that specification. If the web server receiving the HTTP request does not support the FlexXML framework, then the `representation` header is simply ignored and the request is processed as an ordinary HTTP request. If the web server supports FlexXML, then the request is passed to the web server proxy.

The FlexXML prototype web server proxy is currently implemented as a Java servlet [5]. The servlet is installed on a Linux host running an Apache Web Server with servlet support. This server is configured to redirect any requests for XML documents to the Web server proxy.

The FlexXML prototype uses the Saxon XSL processor [9] to transform XML documents. Saxon provides an implementation of XSL transformations [11] and includes a Java API that allows the processor to operate inside a servlet. Once an instance of an XSL style sheet is created in memory, the servlet loads the XML source document and sends it to an instance of the XML parser. The prototype uses a modified version of the Aelfred XML parser[2] that is packaged with the Saxon distribution. The servlet passes the parsed XML document to the XSL processor which transforms the document. Finally, the servlet returns

[1]Section 5.3 of the HTTP specification allows the addition of experimental headers to HTTP requests

the transformed document to the Web server which in turn sends it back to the user's browser application for display.

5. Example

In order to better understand the FlexXML framework and prototype, this section describes a FlexXML-based designed Web site from the perspective of both the Web engineer and the Web user. Consider the problem of designing a personnel directory service to be published on the WWW. The service should include an entry for each employee giving their address as well as an accompanying photograph. Furthermore, it should be accessible to employees on the company intranet as well as to off-site personnel. While building such a Web site seems straight forward, it actually presents a number of difficulties for the Web engineer, as described in Section2.

In order to build the personnel directory using the FlexXML framework, the Web engineer must first install the FlexXML Web server proxy on the Web server, create data files in an XML format, and create XSL stylesheets that transform the data for presentation on a Web browser. Assuming that the Web server proxy has been installed, the Web engineer must encode the personnel directory in an XML format.

After encoding the XML directory, the Web engineer creates XSL stylesheets, each corresponding to three types of end-users in the company. One type of user works on-site, has a direct connection to the company's intranet, and uses a high powered graphics workstation. A second type of user works from home, has a 56K modem connection to the internet, and uses a high powered personal computer. A third type of user travels extensively and uses a low bandwidth WML-enabled cellular phone to connect to the office. In this scenario, the Web engineer creates three separate style sheets. For high bandwidth clients with graphics capability, the Web engineer creates an XSL stylesheet that generates a graphical representation of the personnel directory. For employees with full graphics capability but who are hindered by low bandwidth the Web engineer creates an XSL stylesheet that generates a text-only representation of the personnel directory. Finally, for mobile employees with WML clients and low bandwidth connections, the Web engineer creates a style sheet that generates a text-only WML representation of the directory.

From the Web users perspective, users wanting to access the personnel directory must install a browser application proxy, select their browser environment features and redirect their Web browsers to the browser application proxy. At this point, personnel directory is ready to be accessed.

To help understand the various steps that occur when FlexXML is employed, we return to Figure 1, which illustrates the FlexXML framework. In step one, the user enters the URL of the phone directory into their Web browser application. The Web browser could be running on a personal computer, a cellular phone, a PDA, or any other Web device. The Web browser applications creates an HTTP request and forwards it to the browser application proxy.

In step two, the proxy server receives the request from the Web browser and adds a `representation` header to the request. The value of the `representation` header is either `text` or `graphics`, depending on the user's preferences. After adding the header, the browser application proxy forwards the request to the Web server that hosts the phone directory.

In step three, the Web server receives the request from the browser application proxy. Because the request is for an XML file, the Web server forwards the request to the Web server proxy. The Web server proxy must then decide which stylesheet to apply to XML specification of the phone directory. In order to make this decision, the proxy relies on headers that accompany the request. All HTTP requests include a user-agent header. The user-agent header tells the Web proxy server what type of client (e.g., Netscape, Internet Explorer, Nokia, etc) made the request. The proxy uses this header to determine what type of markup (i.e., WML or HTML) that the client requires. Recall that in step two, the browser application proxy adds a representation header to the request. The Web server proxy uses the representation header to determine whether the user wants text or graphics. Using the user-agent and the representation headers, the Web server proxy is able to select an XSL stylesheet and create an instance of an XSL processor.

In step four, the Web server proxy creates an instance of an XML parser. The XML parser ensures that the XML file is well-formed and creates a tree representation of the file in memory. The Web server proxy passes this tree representation to the XSL processor instance.

In step five, the XSL processor traverses the tree and generates the appropriate representation based upon the XSL stylesheet. Control then passes back to the Web server proxy.

In step six, the Web server proxy transmits a response for display on the Web browser. The browser application proxy then forwards the response to the Web browser (without further modification) where it is then displayed, in step seven, for the user.

This simple, but representative example, is intended to illustrate the benefits of FlexXML. In contrast to traditional approaches, Web engineers no longer are required to create separate Web sites to handle variations in bandwidth, browsers or devices. A single source document is shared by all users regardless of their particular environment. Web sites are easier to maintain since changes to the source document are automatically propagated to all users. Variations in client configurations (bandwidth, browsers, or hardware)

are easily handled by creating appropriate XSL stylesheets. Web users have a single point of entry into a Web site regardless of their environment, which greatly simplifies navigation. Furthermore, users are able to customize their viewing experience by adjusting their browser application proxy settings.

6. Related Work

Various approaches have been developed to address the problems detailed in Section 2. Both the Apache project's Cocoon system [3] and IBM's XML Enabler [7] use XML, XSL, and servlets to provide device-specific content to Web users. IBM provides an alternative solution through the use of their proxy-based transcoding technology [8]. Transcoding represents an attempt to modify or "transcode" Web content on the fly in order to tailor a presentation to a specific device. Taking a similar approach, [6] describes a proxy based system that modifies graphical content for display on heterogeneous devices.

The servlet-based technologies (i.e., Cocoon and XML Enabler) both allow the Web engineer to seperate the XML content from the XSL presentation, which reduces the amount of duplication that was previously necessary to service heterogeneous clients. Furthermore, a Web engineer is able to control the way the content will display on the client. The drawback of both of these systems is that they rely strictly on the user-agent header when selecting a stylesheet for use with each client. Thus, the Web user is given no choice as to what type of representation will be displayed.

The proxy-based technologies [6, 8] require little intervention on the part of the Web engineer because they are designed to work with existing Internet content. There is no need to write stylesheets or convert data into XML. Furthermore, both proxy approaches allow the user to specify what type of representation he or she desires. The drawback of these approaches is that the Web engineer sacrifices control over the presentation of the data.

The FlexXML system shares the advantages of both Cocoon and XML Enabler while allowing the user to specify preferences like the proxy based approaches. Thus, the Web developer is able to tailor more than one style sheet for each user-agent type and the Web user is able to select between them. The FlexXML approach maximizes the control of both the Web engineer and the Web user.

7. Conclusion

In this paper, we have presented the FlexXML framework and a prototype implementation of the framework. The FlexXML approach augments XML and XSL by allowing a suitable XSL style sheet for an XML document to be automatically selected for a given browser environment. We have also illustrated the use of FlexXML by applying it to a simple, yet representative problem.

The primary benefits of FlexXML is that it presents a unified view of WWW services by allowing a Web site to transparently adapt to changing user browser environments. In addition, Web users view a particular Web site using a single URL, independent of their current browser environment. Moreover, they have the ability to customize their browser environment. This allows the most appropriate content and presentation to be delivered based on their current operating environment. FlexXML requires almost no modifications to existing Web browser applications and Web servers. Any Web browser application that has the ability to redirect its HTTP requests to a general proxy can be extended with the FlexXML browser application proxy. Web servers that can be incorporate Java servlets can similary be extended with a FlexXML Web server proxy, XML parser and XSL processor.

References

[1] S. Adler, A. Berglund, J. Caruso, S. Deach, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, and S. Zilles. Extensible Stylesheet Language (XSL), Version 1.0. World Wide Web Consortium Working Draft, Oct. 2000.

[2] Opentext corporation home page. http://www.opentext.com.

[3] Cocoon users guide. http://xml.apache.org/cocoon/guide.html.

[4] T. Bray, J. Paoli, C. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML), Version 1.0. W3C Recommendation, Oct. 2000.

[5] J. Davidson and D. Coward. *Java Servlet Specification 2.2*. Sun Microsystems, Inc., Cupertino, CA, Dec. 1999.

[6] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. Adapting to network and client variability via on-demand dynamic distillation. *ACM SIGPLAN Notices*, 31(9):160–170, Sept. 1996.

[7] IBM XML Enabler. http://www.alphaworks.ibm.com/tech.

[8] R. M. J. R. Smith and C.-S. Li. Transcoding internet content for heterogenous client devices. In *Proceedings of the IEEE Inter. Symp. on Circuits, Syst. (ISCAS)*, June 1998.

[9] Saxon home page. http://users.iclway.co.uk/ mhkay/saxon.

[10] Wireless Application Protocol Forum, Ltd. *Wireless Application Protocol, Wireless Markup Language Specification, Version 1.3*, wap-191-wmp edition, Feb. 2000. http://www.wapforum.org.

[11] XSL Transformations (XSLT), Version 1.0. W3C Recommendation. http://www.w3.org/TR/xslt, Nov 16, 1999.