SIBLINGS

A Server Framework for the Platform-Adaptive Delivery of Site Content

Joseph Sant

A dissertation submitted to the University of Dublin,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

September, 2000

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: ____________________

Joseph Sant

September 15, 2000

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed:

____

Joseph Sant

September 15, 2000

Summary

The increasing use of devices such as connected palmtops and internet appliances has led to a need for web site designers to accommodate a wider range of client platforms and capabilities. Several approaches for transparently supporting this task have been proposed and implemented. Most assume that one site structure and navigation scheme can be appropriate for all devices and concentrate on adapting the presentation of content. This dissertation presents a server infrastructure that will support transparent adaptation of presentation while allowing for different site structures and navigation schemes.

This dissertation describes the design and implementation of a server framework based on Jigsaw's object-oriented server architecture that enables the adaptive delivery of site content to diverse client platform types. The server can support the use of parallel platform-specific static HTML page collections in web site authoring by dynamically selecting the pages most appropriate for the client platform. The framework was designed to be able to use current protocols and work effectively on small and medium-powered servers. Most of the server enhancements are pluggable components that would facilitate the accommodation of new standards by encoding and plugging new components. It can work with traditional HTML as well as more advanced document base structures (i.e. XML for content representation and XSL style sheets to provide formatting).

The server enhancements implemented include the characterisation of user requests according to their platform type, the automatic provision of client platform information to dynamic page creation tools such as servlets, and options in server configuration to allow web masters to specify the adaptive redirection of user requests to more suitable directories, adaptive re-mapping of user request document names and scheduled regeneration of documents from their respective XML and XSL source.

A sample web site involving different navigation and presentation approaches for desktop and palmtop clients is implemented using the server.

Table of Contents

Table of Figures

1 INTRODUCTION

1.1 Introduction

Most web sites have been designed with the assumption that they would be viewed using a desktop with relatively good graphics capabilities, a large screen (14” or greater) and a keyboard. With the advent of inexpensive, powerful and connected Personal Digital Assistants (PDAs) and a variety of other types of internet appliances, the assumption that pages will be viewed only on a desktop is no longer valid. This has led to a need for web site designers to accommodate a wider range of client platforms and capabilities.

This need has led to research and development of server and client architectures that transparently present the information requested in a manner most suitable for the client device. Several different architectures have been proposed for the adaptation of output to accommodate client platform features [MA2000, ABRAMS1998, ORACLE2000]. Many of these assume that a single site structure and navigation scheme can be suitable for every platform and therefore focus on providing alternate presentations of the same documents. An example of this approach is maintaining the one document set but eliding or converting images to lower resolution or grey-scale for palmtop platforms.

This dissertation presents the Siblings Adaptive Server Framework, a server architecture and implementation that would provide for adaptive output but accommodate different site structures and navigation schemes for different platforms. A novel feature of this framework is that it will support the use of static pages in the design and implementation of adaptive web sites. It allows the web master to use separate hierarchies of static pages for different platforms. Until recently, the suggestion of having a separate site structure for every platform would have been considered difficult to manage [ABRAMS1998]. The use of the Extensible Markup Language (XML) to represent site content and style sheets to produce the necessary web pages makes it feasible.

The server architecture presented assumes that parallel document bases with possibly different structures may be necessary to accommodate different platforms. The Jigsaw Server from the World Wide Web Consortium was used as a base for the implementation. This server architecture was chosen because its object-oriented design and the availability of source code facilitated customisation and extension of web server function. The enhancements to the server include the ability to detect the Platform type from the User-Agent String, the provision of platform information to servlets launched by the server, and the ability to redirect requests and re-map document names to more suitable directories or documents based on the request's platform type.

An adaptive web site was designed and implemented using the Siblings architecture. This was used to evaluate the framework for both system function and practicality of use. This sample site was designed for viewing with either a desktop or palmtop and involved the use of both intelligent retrieval of static pages and the implementation of platform-sensitive servlets. Document name re-mapping was used and some of the files were automatically regenerated by the server from their source XML and XSL (Extensible Stylesheet Language) stylesheets. All static pages on the site were created from XML documents and XSL stylesheets. Some of the base XML docments were complex.

1.2 Heterogeneity and Clients, Servers and Users.

Media Independent Access to Web document bases is a growing area of interest due to an increasing diversity of input, output and processing features of Internet client devices and the increasing use of alternate transmission media for Internet communications (e.g. wireless lans, GSM). A recent Price-Waterhouse [PW1999] report stated, "On the client side, it is perhaps inevitable that the highly uniform environment defined by ubiquitous Personal Computers (PC's) built around x86 processors and running various versions of Microsoft Windows can only give way to greater platform diversity". A broad range of Internet appliances such as set-top boxes, smart phones, and connected personal digital assistants are already being used to communicate over the Internet. International Data Corporation [IDG1999] has predicted that there will be more than 151 million Internet appliances deployed by 2002 and that purchases of Internet appliances will eventually surpass those of PC's. Of particular interest to this thesis is the role of connected Personal Digital Assistants (PDA's) and PC's in the future panorama of clients accessing Internet Document Bases.

1.3 Objectives of Dissertation

The primary objective of this dissertation is to develop an infrastructure that enables platform-adaptive delivery of site content through the implementation of intelligent retrieval of the documents most suitable for the clients platform and provision of platform information to dynamic page programs. A secondary objective of the infrastructure is to accommodate next generation document bases that will likely entail greater use of XML source documents and XSL style sheets. The dissertation will focus on accommodating the most common desktop platform, a Windows-based PC and the most common palmtop platform, and a PalmOs based PDA. The dissertation also discusses current research in the adaptation of content for heterogeneous clients.

The infrastructure will be developed attempting the best compromise between the following issues:

- Efficiency in Processing of Requests.
- Ease of Use for Webmasters.
- Flexibility of Adaptivity.
- Protection of Intellectual Property and Valued Data.
- Author Control of Output.

- Ease of Deployment.

The dissertation involved reviewing pertinent literature in web-based document bases and adaptive architectures, then using that background to design and implement Siblings as an adaptive web server framework. The implementation was evaluated using a case study and some test suites.

1.4 Overview of Dissertation.

This thesis starts with an introduction (Chapter 1) that gives an overview of the thesis. This is followed by a discussion of the necessary background in web-based document bases and adaptive implementations necessary to design and implement an adaptive framework (Chapter 2 and 3). The design and implementation of Siblings is presented in Chapters 4 and 5. The final chapter, Chapter 6, presents an evaluation of the Siblings implementation and the conclusions that may be drawn from the thesis. A chapter-by-chapter breakdown of the thesis follows:

Chapter 1

Introduction. This chapter presents an overview of the dissertation and describes the motivation behind the research and the primary objectives.

Chapter 2

Web-Based Document Bases. This chapter introduces the background to understanding Web document bases, including major protocols, common document types, and user-interface considerations.

Chapter 3

Media Independent Access to Web Documents. This chapter introduces the major issues in Device-Independent Access to Web Document Bases, and discusses current research, standards and implementations in the field.

Chapter 4

System Architecture. This chapter introduces the system design and discusses the design goals, fundamental assumptions, design strategy. It also discusses the Jigsaw Web Server architecture that will be used a base architecture and the design changes proposed for that architecture. Design strategies are discussed in reference to existing implementation of adaptive content delivery.

Chapter 5

Implementation. This chapter discusses the implementation of changes to the Jigsaw server made to effect platform-base adaptation of server responses.

Chapter 6

Evaluation and Conclusion. This chapter evaluates and critiques the server design and implementation, lists some suggestions for future enhancements and provides a conclusion for the dissertation. An adaptive web site that was designed to support either desktops or palmtops transparently implemented using the Siblings architecture is also discussed.

2 Web-Based Document Bases

2.1 Introduction

The Siblings Server Framework is intended to serve documents adapted to the client's platform using Hypertext Transport Protocol (HTTP). To properly design and discuss a web server architecture, consideration must be made for the nature of current and proposed web document bases, the mechanics of document retrieval using HTTP, and the user view of retrieved documents.

A typical usage pattern on the web entails a user via a browser submitting a request for a pre-formatted document to a server, the server performing a lookup to find the document, the server returning the document to the client, and the browser on the client rendering the document. Most commonly, HTTP (Hypertext Transport Protocol) would be used as the request protocol and HTML (Hypertext Mark-up Language) would be used to encode the documents. Increasingly, documents are also being dynamically created in response to a request. This is done by launching a new process or thread on the server that uses the information in the request to create HTML output specifically for that request.

The increasing use of XML (a mark-up language for describing content) enables alternate modes of interaction between clients and web-based document bases. With XML, it is feasible for the client to request XML-encoded content from a web site and perform both the formatting and rendering of the downloaded content on the client.

2.1.1 Static Documents

Static Documents are persistent files that are stored on the server. These files may obey any one of a number of commonly used formats. The dominant formats used are HTML, simple text, JPEG, GIF, PNG and multi-media file formats such as MPEG and MP3. Static documents can be served very efficiently by web-servers because most of the mechanics for looking up and retrieving the file has already been implemented efficiently within the underlying operating system. Another advantage of static documents is that they can be cached very easily.

Static document bases also have major disadvantages. Using static documents for responses to HTML form queries is not practical. The number of possible responses to most form queries requires that a response be created specifically for that query. Another problem with large static document bases is that they can be very difficult to maintain. Large static document bases may entail thousands of files with complex inter-relationships due to links between files. The addition and deletion of files from the document base causes problems with maintaining consistency and avoiding broken links (links that target a document that has been deleted or moved). If a site structure is altered, very often a large number of files must be changed to reflect the change in site structure (i.e. changing link targets). If documents are generated by using cascading style sheets or programmatically this problem can be ameliorated since one or mores scripts can be run to update all files that are impacted by the changes.

2.1.2 Dynamic Documents.

A document can be dynamically created in response to a request by launching a program or thread to deal specifically with that request. This is especially useful if the request entails access to a database on the server or if the generation of the response to the query requires complex logic or calculations. Most modern servers support one or more ways of delivering dynamic documents. The most common techniques to generate dynamic documents are CGI applications, Servlets, and server pages. CGI applications are any application that is designed to accept and return information conforming to the Common Gateway interface. CGI applications are loaded as processes with every CGI request and the process is destroyed after a reply is returned.

Servlets are programs that are loaded only once and therefore tend to be more efficient. Every request will cause a new thread to be launched instead of a new process. Server pages allow you to embed executable instructions inside some type of mark-up. Server pages invert the standard dynamic page creation approach of using executable code to generate mark-up. With server pages you embed executable instructions inside the mark-up and the file will be automatically compiled. These might be compiled into servlets or CGI applications. [WBP2000] The major disadvantage to all dynamic document creation techniques is the extra computational load that results from having to launch a new operating system process or thread to service each request.

2.1.3 Document Formats

2.1.3.1 HTML

The main features of HTML are that it is easily learned and that there is a large set of pre-defined tags for the formatting and presentation of content. Importantly, HTML is the most widely supported mark-up language with respect to browser and tools. It is also the most widely known. HTML can be used to describe compound documents, such as documents that comprise text information and references to graphics files. HTML was originally designed as a simple markup language to exchange documents between scientists at the CERN research facility. Since then it has evolved with a long series of ad-hoc changes making it unwieldy. As a result there will not be another revision of HTML. Instead of revising HTML, a new standard XHTML, is a restatement of HTML 4.0 as an application of XML. As such it provides compatibility with current HTML 4.0 browsers while allowing for extensions through the addition of new tags [RICHMOND2000].

A major problem with HTML is that it necessarily intersperses information content with presentation-oriented mark-up. Since the document is structured according to its presentation it is very difficult to programmatically extract the information content from an HTML page.

2.1.3.2 XML

"XML is a framework for developing an unlimited number of special-purpose data languages" [Bosak 2000]. Whereas HTML's prime focus was to provide an easy-to-use mark-up language for the presentation of information, XML, or Extensible Markup Language, focuses on the representation of information. It will allow industry groups to collaborate in the creation and use of industry-specific communication, and specify that language using an XML Document Type Definition (DTD). Once defined, the tags and structure specified in the DTD can be used for communication and information storage.

The Organization for the Advancement of Structured Information Standards (OASIS) estimates that well in excess of 100 industry trade groups are working to develop their own industry-specific syntax for communication using XML [GASKIN2000]. The United Nations and OASIS are co-operating to help standardise XML business specifications under the moniker of ebXML [EBXML] . Possible uses of XML include electronic commerce applications, enhancing Internet searches, enabling self-describing BLOBs (binary large objects) and content personalization [HOGAN].

There are many web sites, articles and textbooks on XML and its related technologies. Two excellent collections of links and categorised resources can be found at xml.org [XMLORG] and the Cetus Links page [CETUSLINKS]. Both include many links to XML reference sites, white papers, link collections and tutorials. There are also links to similar materials on the Extensible Style Language (XSL) and the Extensible Link Language (XLL). Jon Bosak and Tim Bray, two technologists who played crucial roles in the development of XML, have written an excellent overview article for the Scientific American titled "XML and the Second-Generation Web" [BOSAK1999]. A variety of introductory and advanced tutorials are available from individuals and companies including those from SUN [SUNXML], Microsoft [MSFTXML] and IBM [IBMXML].

A purported advantage of the use of XML is that it is ideal for "open" solutions to data exchange problems since the documents are text-based and the exchange format could be agreed upon and published. An open XML data exchange schema will circumvent interference from third parties and eliminate dependence on software vendors or bindings to specific tools [Bosak 2000].

Another of the key benefits that XML provides is the increase in the granularity of information retrieval on the web. Any investigation of the design of browsers, servers and proxies elucidates the fact that much of the effort on the web revolves around the request and retrieval of files (most commonly HTML files and the graphic files they reference). XML will allow the creation and use of tags and attributes to describe any class of objects. An XML document base could simply be properly tagged descriptions of a company's products or a university's courses. If the items in the document are tagged appropriately it enables users to selectively retrieve only those parts of the document that are of interest to them.

2.1.3.3 XSL - Converting XML to Other Formats

The fundamental benefit of XML is that it allows information content to be represented separately from its presentation mark-up. This does not eliminate the need for a consistent way of formatting and presenting the information in XML documents. XSL is used to prescribe formatting instruction and rules for XML documents. It can be implemented in a manner similar to that of CSS, Cascading Style Sheets. XSL is a rich formatting/style language that allows the user to prescribe formatting instructions taking into account the structure of the XML document. It has special keywords that allow the Style Sheet author to associate formatting activity with certain parts of the document structure tree. XSL also includes many built-in functions and the ability to embed JavaScript for highly customised output.

XSL is not without its detractors. Mike Leventhal in a series of articles for XML.com [LEVENTHAL1999a,b,c] makes the point that the language is difficult to learn, is subject to inconsistent implementation by vendors, and for all its complexity does not represent any significant advance over existing technologies. He supports his arguments by pointing to complex XSL code used to perform straightforward tasks, and shows a comparison between formatting of XML using XSL and another using both DOM (Document Object Model) and CSS (Cascading Style sheets).

2.1.4 RDF

Resource Description Format (RDF) is used for the specification of metadata for digital resources. It allows applications on the web to discover properties about the web resources they are processing. These properties may include cataloguing information, relationships to other web resources, or intellectual property issues. RDF files must be valid XML files. RDF provides the advantage over XML in that it provides information on the interpretation of data, rather than simply specifying a structure for data representation.

In order for RDF to be useful in describing metadata, it must be able to express a variety of relationships, it must be extensible, and it must be modular.

2.2 HTTP and Document Retrieval

The most commonly used protocol for document retrieval on the web is the HTTP (Hypertext Tranport Protocol). Versions of this protocol in common use are HTTP 0.9, 1.0 and 1.1. Garshol has written an excellent discussion of how HTTP works [Garshol]. A good discussion of the main features and main differences with respect to HTTP 1.0 and 1.1 is found in Krishnamurthy et al. [KRISH99]. Using this protocol, a client (e.g. browser) sends a text-based 'request' message containing the requested document name and some information about the client to a server. The server then interprets the request, attempts to look for the document on disk or creates it dynamically, and sends the document back with header information that can be used by the client browser. The combination of the header plus any data is called a response. If the server cannot retrieve a requested document, it will typically return a response containing an error code in the header and a document indicating that an error has occurred. A simplified schematic of the HTTP protocol in action is shown in Figure 2-1.

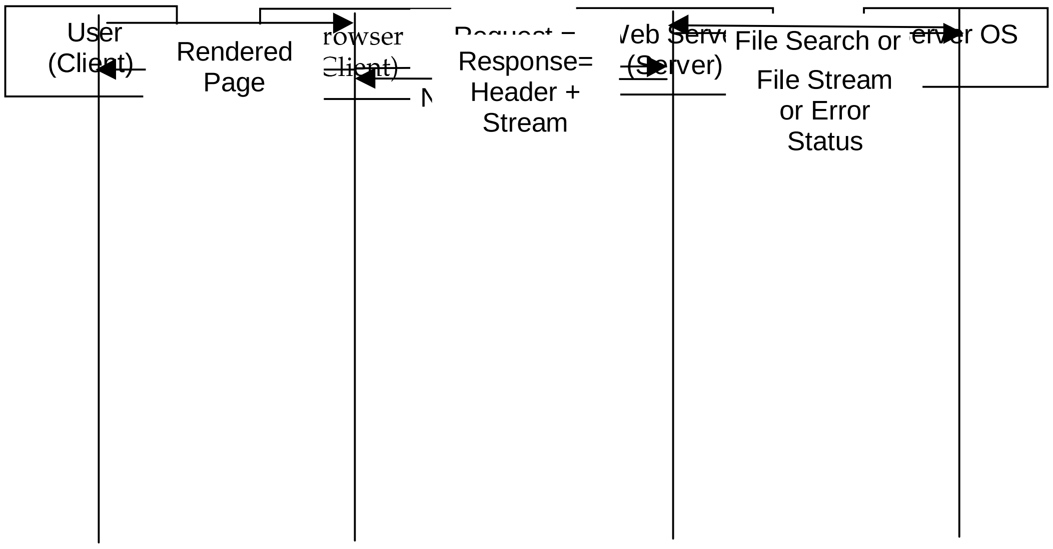


Figure 2-1 HTTP Request Processing

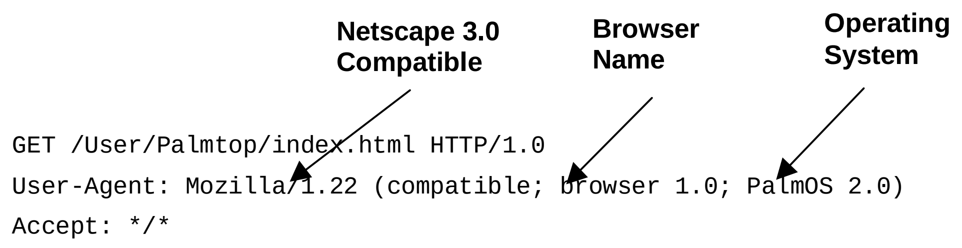
The most common used functions of an HTTP web server is to retrieve files (typically HTML and graphic files) for remote clients or to launch processes or threads which would dynamic create a document at the users request. Another function of a web server is to redirect requests for a specific document in a specific directory to other directories or other files. This may be necessary when a path becomes obsolete (e.g. the file or directory has been deleted or is empty). Other functions of web servers include authorization and security.

2.2.1 HTTP Requests

HTTP document requests must be formatted to include information about the request method and the document name. Optionally information about the client can be included. In the simplest case, a browser sends the request method, the document name and the protocol followed by a blank line.

```
GET /User/Palmtop/index.html HTTP/1.0
[blank line here]
```

A request can include information about the client such as Authorization, Content-Encoding, Content-Length, User-Agent and several other parameters. A full list of the request header fields for both HTTP 1.0 and 1.1 is available on the World Wide Consortium site [RFC1945, RFC2068]. Of special significance to this thesis is the User-Agent header field. This field is commonly used to report the name or compatibility of the browser and perhaps the operating system name. An example of a request with an User-Agent header field and an accept header field is shown below:



```
GET /User/Palmtop/index.html HTTP/1.0
User-Agent: Mozilla/1.22 (compatible; browser 1.0; PalmOS 2.0)
Accept: */*
```

If one could inventory the names of browsers and operating systems associated with palmtops the User-Agent could be used to differentiate between palmtops and desktops.

2.2.2 HTTP Responses

Once the server has processed the request it sends an appropriately formatted response to the remote client. The server will attempt to locate the requested document then form a response composed of a specially formatted header followed by the document content. Typically the header will include the HTTP version, a numeric status code for the request, a short text code, the date, the date the file was last modified, the length of the document and the MIME type of the document. A detailed explanation of both the request and response formats is available at the World Wide Web Consortium web site

[RFC1945, RFC2068]. There is a defined list of status codes which indicate a variety of error conditions or success. When redesigning server function or implementing proxies, these status codes could be used to trigger special actions such as redirection to an HTML page relating to the error condition.

2.3 User Interfaces for Web Document Bases

A full discussion of User Interface Design for the Web would be out of the scope of this dissertation. There are several issues in User Interface Design that might impact the design of a server intended to serve information to diverse platforms. The central issue considered is whether the different platforms require different site structures, tool usage, and navigation schemes. If this is the case then any approaches that focus on customising the presentation of a single document base with a single structure for each platform must be called into question.

There are very few empirical studies of the differences between browsing behaviour on diverse platforms such as desktops and palmtops. Jones et al. [JONES1999] performed a study of 20 participants in performing specific tasks using a desktop browser and a palmtop browser. The study results indicated significant differences in task completion on the different platforms. It was also found that small screen users selected search facilities twice as often as large screen users, with as many as 80% of small screen users beginning by using the search options of the site. The most commonly viewed pages differed significantly between users of the two platforms. Users of palmtops scrolled to find information more often than the desktop group. Jones et al. suggests that the metaphors used with larger screens are not the most appropriate for the new devices [JONES2000].

2.4 Summary

This chapter presented an overview of the types and formats of documents that may be found in a web-based document base. An overview of the HTTP protocol and a brief discussion of how the features of the client device might necessitate different document base structures was also presented. In the next chapter, we will look at the issues, models and existing implementations that relate to media-independent access to web-based document bases.

3 Media-Independent Access to Web Documents

3.1 Introduction

In chapter 2, we discussed the features of web-based document bases and their retrieval via HTTP. This chapter discusses the main issues associated with media-independent access, proposed models for device-independent interaction, and current research and implementations in the field of media-independent access to web document bases. The chapter provides further background for the design of the Siblings Adaptive Server Framework.

3.2 Issues in Media-Independent Access

3.2.1 Platform Hardware Characteristics

The standard Desktop environment for accessing the internet has been a PC-ased on a 32-bit Intel-compatible chip running a version of Microsoft Windows Operating environment. This platform represented over 90% of Desktop sales in 1999 [LEM1999].

As of 1999 the dominant browsers for the PC platform were Internet Explorer and Netscape Navigator, together having a market share of almost 100% [ITANALYSIS1999]. Each browser is feature-rich and require several megabytes of memory and disk space to run properly.

By contrast, the dominant Personal Digital Assistants were the PalmPilot and their work-alikes (e.g. Handspring). PalmOS-compatible PDA's had 78.6% of the U.S. market-share for the high-end professional PDAs in 1998 [SJMT1999]. A representative example of the features of a connected PDA could be those of the PalmPilot V. This palmtop had a 160 by 160 grey scale display with no keyboard and a 6cm by 6cm touchscreen. It uses a 16 MHz Risc Chip and came standard with 2 Megs of RAM. The choice of chip is largely constrained by power consumption since one of the advantages of the Palm platform is their ability to run for extended periods of time on commonly available AA batteries.

3.2.2 User Interface Capabilities

Typically Desktop PC's would provide the widest range of user interface capabilities due to their large full-featured display, extensive RAM and disk space, fast processor and full-sized keyboard. However, the standard user interface for access to the web in a PC is that constrained by the typical browser. The typical interface is therefore graphic and text-oriented with heavy reliance on the mouse and keyboard for input. The most commonly used interface tools that would be provided are those that are available in an HTML form; command buttons, text boxes, text areas, text labels, drop-down lists, check-boxes and radio buttons. All of these except for text boxes and text areas depend on the mouse for input. Since the screens are comparatively large it is possible to display reasonable amounts of textual or graphics on the screen.

The dominant OS in the PDA/Palmtop market is the PalmOs with approximately 75% of the market. The form factor and function of most of the PalmOs compatible palmtops are similar. Although there are colour versions of the Palm, most Palm machines are characterised by their small grey-scale touch screen suitable for pen-based data-entry and their lack of a keyboard. The PalmPilot V has a small backlit LCD screen (160x160) that restricts the amount of text and the type of graphics that can be displayed at one time. It uses a grey-scale display. In order to compensate for the lack of a keyboard, the PalmOS supports hand-writing recognition using a special alphabet set called Graffiti. This script is very easily learned and it is estimated that after 5 minutes of practice character input accuracy of 97% is typical [MACKENZIE97]. It has also been stated that text input speeds as high as 30 wpm are possible using the Graffiti script [PALM1995]. Alternatively, the image of a keyboard (known as a "soft keyboard") can be displayed on the touchscreen and can be used to enter character-type data. In either case the speed of character input is not as fast as with a full-sized keyboard.

3.2.3 Communications Characteristics

There is a wide variation in latency, bandwidth and connection stability between the various communication techniques available on desktops and connected PDAs. Typically desktops will use communication techniques that have higher bandwidth, lower latency and more connection stability. The wireless networks that would be used to connect PDAs are typified by low bandwidth, high latency and poor connection stability [APION1999]. The bandwidth of the faster communications techniques available for desktops (ASDL, Cable Modems, ISDN, T1) is orders of magnitude faster than those expected for connected PDAs (wireless lans, cellular communications). High performance lans using guided media would have bandwidths in the 100 Mbps range whereas high performance wireless lans would be in the 10 Mbps.

3.2.4 User Interface Design for Media-Independent Access

The most common web-oriented user interfaces were designed for a standard desktop/laptop configuration. This was sufficient since the overwhelming majority of web clients in the mid and late 1990's were likely to be desktops. This is changing now with the increasing market share of PDA's, set-top boxes and smart-phones. Since user interface design depends on the devices available and users' device configurations are becoming more diverse, the question "What is the device" is now problematic [WINOGRAD1999].

The platform characteristics of desktops and palmtops have already been discussed. Several critical differences between a desktop and a PDA are:

- PDA's have a smaller screen size
- PDA's may have only grey-scale.

- PDA's may be restricted to pen-based touchscreens.
- Connectivity with PDA's is likely to be slower.
- PDA's will most likely have slower processors and less available memory.

Although we may expect technological advances and market forces to result in faster processors, more memory, and greater availability of colour screens, PDA's will remain constrained by their smaller screen size and lack of a keyboard.

3.2.4.1 Desktops, PDA's and User Interaction Models.

The dramatic differences between the PC and PDA platforms forces us to consider whether the user interaction model for PDA's differs from that of desktops. If the user interaction model for PDA's is different, it could mean that users of an XML or HTML document base on PDAs might intrinsically access the document base differently in both the information they request and the way they navigate through the document base.

An indication of the differences between accessing Web pages via PDA or desktop can be gleaned from suggested best practices for designing web pages for handheld devices. Kacin provides a series of tips for designing web pages for handheld devices [KACIN1999]. The article suggests only including the most critical information on the site to try to avoid extravagant effects or colours. There is a strong recommendation to minimize any use of graphics. Where possible, graphics should be designed for smaller screen sizes (e.g. 150x140 pixels). Of particular interest is that the article suggests page lengths be kept small, and that drill down navigation of multiple pages be used in preference to scrolling a single large page. The article suggests that dynamic generation of HTML be done on the server. Form input should be kept to a minimum because of slow speed and higher error rates of input using Graffiti (the Palm symbolic handwriting script) and soft keyboards. One suggestion was to pre-populate form fields where possible.

3.3 Models for Device-Independent Interaction

Several models have been developed to accommodate device-independent interaction with users. Some are strictly theoretical, whereas others have implementations. There is also a commercial package, Digital Plastic, which promises adaptive display of content [ECLIPTIC2000].

Thevenin and Coutaz [THEVENIN1999] present a model and demonstration implementation for Adaptive and Plastic interfaces. They differentiate between "Adaptability", the capacity of a system to allow users to customize their system from a predefined set of parameters and "Adaptivity", which is the capacity of the system to perform adaptation automatically without deliberate action from the user's part. They also define the term plasticity as the capacity of a user interface to withstand variations of the system physical characteristics and the environment while preserving usability. It should be possible to define plastic interfaces once for a variety of configurations thereby reducing development and maintenance costs. The article discusses some of the critical issues of design for adaptation and describes a framework of models for developing plastic interfaces. The framework includes information derived from a user task model, a definition for an abstract user interface, descriptions for the physical characteristics of the target platforms, potential environmental affects on the system or user behaviour and interactors available for communicating with the user.

Winograd [WINOGRAD1999] discusses several different models for Interaction Architectures. Early interaction models required programs to directly interaction with sensors/actuators (i.e. devices). In the model used currently, user applications interact with a timesharing/window manager that that uses device drivers to communicate with the actual sensors/actuators (devices). A significant advantage was that device drivers abstracted the interface to classes of devices.

Winograd takes the abstraction one step further by re-interpreting the role of the manager as responding to phenomenon (instead of a device-oriented interaction). Phenomena are things and events that are relevant to a program. The manager would communicate with one or more "observers". This layer of observers replaces rather than adds to the layer of device drivers. An observer may fulfil the same role as a device driver or may interpret the results from other observers (e.g. simple device oriented observers) and report the complex phenomenon to the manager. It is possible that a hierarchy of observers may be required to implement a system.

3.4 Media-Independent Access - Current Research and Standards.

The issues surrounding media-Independent access are dealt with in a variety of different research and standardisation efforts including; Composite Capability/Personal Profile [CCPP1999], Adaptive Content Delivery [MA2000], and Appliance-Independent User-Interface Languages [ABRAMS1998]. Some of the major issues in Media-Independent Access include:

- Mechanisms for detection of the software and hardware capabilities of the client device.
- Standardizing methods for describing user preferences.
- Developing content adaptation algorithms that will optimally render and retrieve data according to the needs and capabilities of the client. This includes policies on when to use one algorithm over another.
- Where to do the adaptation - server, client or proxy.
- Efficiency of methods
- Flexibility of methods.

3.4.1 Composite Capability/Personal Preference

Composite Capability/Personal Profile or CC/PP is a developing standard being led by the W3C to describe and communicate the capabilities and preferences of web-enabled appliances. An overview and discussion of this standard is available from the W3C [CCPP1999]. CC/PP will allow the discovery of a client's capabilities and preferences to enable more appropriate content negotiation. The standard proposes the use of XML and RDF to describe the client's capabilities and preferences. The features that are considered important to describe are those for hardware and any software set-ups for the user agent. The standard is being developed with the goals of ensuring reasonable content negotiation speeds, minimizing content negotiation transactions, allowing assembly of a profile from multiple sources, allowing for some user control over agent information, enabling the use of compact data formats and allowing for the possibility of multiple network elements between the user agent and origin server. The document gives examples of some of the hardware, software and preference information that might be represented as metadata. These include hardware features such as screen size, colours, vendor and class of device and a variety of software features and preferences.

A profile may change over the duration of one session. CC/PP focuses on the critical issue of reliably describing and propagating the "current" status of the profile for any single network transaction. Although there may be advantages to keeping a persistent view of a users original profile and any changes effected over the session it is not addressed by the standard.

A profile may be composed of default settings provided by a software or hardware vendor, persistent local changes and temporary changes. The default settings could be the typical configuration associated with a standard product. Persistent local changes would be changes to the standard configuration resulting from customisation. A user may have added memory or a printer to their hardware configuration or added a new capability to a software product via a plug-in. Temporary changes are those that may occur during a session such as turning sound or cookies on or off.

Another important consideration is that a profile might best be "assembled" from several distinct documents or profiles. This can be done with an "inline" definition that contains all information necessary for the profiling or by using indirect referencing. Indirect referencing uses remote sites to retrieve default configurations, possibly directly from the hardware and software vendor's sites. This indirection to other remote sites could expose the profiling process to new security attacks. It was suggested that a solution to this attack could be the use of digital signatures to verify the URL's for the profile.

Other issues discussed included the efficiency considerations of propagation temporary changes to a composite capability/personal profile. It was suggested that although the entire profile could be transmitted with each change a more efficient means would be to only propagate the features that have been changed.

3.5 Architectures for Media Independent Access

Several architectures have been suggested for media-independent access. Typically the research and techniques strive for generality; that the architecture work for a wide variety of client-devices. These architectures may be server-based, client-based (e.g. browser enhancements), proxy-based, or some combination thereof.

Ma et al [MA2000] discusses some of the benefits and disadvantages of server-based versus proxy-based content adaptation. Server-based solutions allow both static (offline) and dynamic (on-the-fly) content adaptation. Static adaptation allows the creation of multiple versions of the authored content (offline if necessary). Dynamic adaptation adapts content on a request-by-request basis. The study suggests that server-based approaches allow for more author control since it would be easier to specify and view the adapted output under different preference and capability profiles. Other advantages include the ability to encrypt all output and more control of the document base with respect to copyright and business implications.

Some of the disadvantages of the server-based approach is the increased computational load and resource use on the server, the difficulty of geographically distributing the adapted content to widely dispersed clients, and the management problems associated with the alternate document sets generated through server-based static adaptation.

Ma et al. also discusses the advantages and disadvantages of proxy-based architectures. With proxy-based content adaptation, a proxy would intermediate between client and server passing on requests from the client to the server and adapting the server response for the client. Some suggested advantages of proxy-based content adaptation include ease of situating content adaptation geographically close to clients, efficiencies relating to allowing proxies to adapt content from several different servers.

Ma et al. mention several disadvantages to a proxy-based approach. Since the proxy typically must be able to adapt output from several different servers the adaptation would have to be generic. There would be less control of the adaptation of an author's content and the presentation on some platforms might not be acceptable to the author. Another issue is the potential of a proxy to be used to alter the content so that its business value to the original source is decreased (e.g. blocking advertisements).

Ma et al. have categorized content adaptation techniques into Information Abstraction, Modality transformation, Data transcoding, Data priorisation, and purpose classification. Information abstraction involves providing digests of text or thumbnails of images. Only the most important information is retained. Modality transforms involve changing the format of the data so that it can be accommodated by client device. An example of this might be sending sets of images instead of a video to devices that are not capable of playing videos. Other modality transforms are text-to-speech and speech-to-text. Data transcoding is converting media to the format most appropriate for the client device. Converting colour images to grey-scale is an example of data transcoding. Data priorisation priorises the data being sent so that the appropriate level of quality of service can be associated with the data. Purpose classification exposes the purpose of a media or interface item allowing redundant items to be ignored on devices with constrained interfaces.

Ma et al mentioned the advantage server-based approaches have over proxy-based approaches in maintaining control of copyright and dealing with the business implications of the document base. Although not discussed in the article, this has special significance for XML document bases. The information in the XML document bases may have been costly to acquire, assemble, or create and may represent great business value to clients. The XML document would be of greater value to someone wishing to "steal" the information since the tags describe the meaning of the data and would allow the creation of programs to parse this information and if beneficial reuse, reformat, or reassemble it. Converting XML to HTML or WML documents at the server,therefore, provides better protection for the intellectual property of the authors. The authors developed a framework for adaptive content delivery which used an intermediate XML format to represent document structure and provided support for HTML re-authoring and transcoding graphics and multi-media files.

Bickmore and Schilit [DIGESTOR] developed a proxy, Digestor, which would accept user requests and automatically transform the document to better suit the client device. Digestor would convert the retrieved HTML document into an intermediate Abstract Syntax Tree before going through a convolution of transformations to adapt output to the client device.

Abrams et al [ABRAMS1998] suggested using a combination of XML-compliant appliance-independent user interface language [UIML] and device or server specific compilers that would compile the UIML into an interface appropriate for the specific device. This allows the adaptive rendering to be performed on the server or on the client (perhaps as a plug-in to a browser). Through the use of this mark-up language it is suggested that only one interface document base need be maintained and that development of multi-platform interfaces could be facilitated by having previewers for the different platform targets. The UIML could be compiled into anything from native user-interface API's (e.g. ActiveX) to JavaBeans or mark-up languages.

The steps in defining a UIML interface are:

- listing the interface elements and the abstract class they belong to.

- specifying which elements would be used for a given appliance and how they would be used,
- providing the data to be used by the interface,
- specifying a style by associating actual device-specific interface elements with the abstract classes that were assigned to the interface elements and finally listing any event actions.

Portal-to-Go [ORACLE2000], a commercial product available from Oracle also implements a media-independent solution using an intermediate user interface language defined in XML. The user interface language is known as SimpleResult. Although both use a device-independent user interface language, Abrams et al. suggests specifying the web interface in their device-independent language whereas Portal-to-Go can "scrape" existing HTML documents to discover user interface features and create the intermediate document using tags from SimpleResult DTD automatically. The adaptation is accomplished through the creation and use of 'adaptors' and 'transformers'. Adaptors convert information from source documents to intermediate SimpleResult documents. Transformer convert the SimpleResult document into the appropriate mark-up for the target device. The major advantage to this approach is that by using an intermediate document it avoids the potential combinatorial explosion associated with having many-to-many relation between document types and target device types. Another benefit is that the allowance for multiple transformers avoids the least common denominator approach to specifying user interfaces.

Another commercially available is the IBM WebSphere Transcoding Publisher [IBM2000]. The system uses an image transcoding engine and a text transcoding to adapt documents for a client. The image transcoder can change various features of the image including the image format. The text transcoder can eliminate or alter features in the document that are not appropriate for the client. Transcoders are implemented as JavaBeans (Java classes that have the capability of reporting their capabilities to a client). An interesting feature of the Transcoding Publisher is that the transcoding can be distributed on different machines and different types of transcoding can be chained.

The table below, Table 3-1, summarises some major features of the various architectures discussed above.

Architecture	Locus of Work	Intermediate Mark-up or Representation	Automated Re-Authoring/ Transforms	Single Document Base	Transcoding
Abrams et al.	Server,Proxy Browser	Yes	No	Yes	No
Digestor	Proxy	No	Yes	Yes	Yes
Ma. et al.	Server/Proxy	No	Yes	Yes	Yes
Portal-to-Go	Server	Yes	Yes	Yes	No
Websphere Transcoder	Server/Proxy	No	Yes	Yes	Yes

Table 3-1 Adaptive Architecture Comparison

3.5.1 Discussion of Architectures

The architectures discussed above all have the advantage that they can support platform-adaptivity and require only one document base. This would reduce the level of effort needed to create and maintain a site. Another advantage of all the architectures, excepting that of Abrams et al. ,is that they perform automated re-authoring of existing documents. This means that the existing investment in the desktop-oriented HTML web site is preserved. These systems will retrieve the existing pages and intelligently reformat them for the client device. There are several problems with re-authoring. It is very difficult to extract the information content out of HTML mark-up. Since it is difficult to assess the purpose of the elements of a page, the architectures depend on simple heuristics related to presentation features of the HTML page to re-author the document. Assessing whether the content is appropriate is often impossible.

Several of the implementations could require extensive programmer support to optimise a site for a platform. Portal-to-Go and WebSphere

Transcoding Publisher depended on user-written programs or scripts for any customisation of the adaptivity over and above that provided by the respective packages. This would be a serious drawback for smaller and medium-sized sites.

3.6 Summary

This chapter discussed the issues involved in media-independent access to web document bases and provided background into the models and architectures that have been developed to provide media-independent access. This chapter and Chapter 2, combined, provided the background from which to embark on the design of a platform-adaptive web server framework; the Siblings Framework. In the next chapter we will discuss the design of the Siblings Framework.

4 System Design

4.1 Introduction

With the background on document base structures, retrieval mechanisms and existing adaptive architectures presented in the previous chapters we can now embark on the design of the Sibling Adaptive Server Framework. This chapter presents the design for the Siblings Web Server Framework in the context of the design assumptions and goals. This framework involves a set of enhancements to the Jigsaw web server and therefore uses its architecture as its starting point. A discussion of the Jigsaw architecture is presented as background for understanding the design of the enhancements.

4.2 Design: Fundamental Assumptions

Any technology must be both effective and practical in order to be widely adopted. For a web site intended to be adaptive to be effective it must deliver the most appropriate site content, presented in the most suitable manner for a client platform. It should also provide a navigation scheme that is appropriate for the platform. For an adaptive server to be practical it should be efficient in operation so that it will not overload current hardware. More importantly, it must be practical for the web masters and web-site architects to create and manage a site that uses the technology. In light of these considerations, the following assumptions have been the driving force behind the design.

- *Different Content, Presentation and Navigation Schemes Might Be Required for Different Platforms.*

By looking at the suggested best practices for the desktop and palmtop platforms and empirical studies it is reasonable to assume that a superior web site on a palmtop might require a different site structure and perhaps even different content than that for a Desktop [KACIN1999,JONES2000].

- *Maintaining a Small Number of Device-Class Specific Page Hierarchies is Practicable.*

  It can be practical to maintain a small number of device-class specific Page Collections through the use of XML, XSL and dynamic page creation programs. Less volatile site content could be stored in a single XML document base, and different sets of XSL style sheets developed for the various platforms. For more volatile information, dynamic page creation programs such as servlets can be used to access a database. By storing site content in XML and using XSL style sheets to produce HTML pages, changes in the XML document can be used to trigger the automatic recreation of the affected HTML pages using the appropriate XSL style sheet and an XSLT processor. XSLT, or XSL Tranformations, is part of the XSL stylesheet language that deals with transformations and methods of accessing parts of an XSL style sheet. This can be highly automated by writing operating system scripts that run the XSLT processor with the appropriate files when required.

- *Adaptation based solely on Dynamic Page Creation is CPU-intensive.*

  Dynamic pages could consume several orders of magnitude more CPU time than that required to serve a static page [IYENGAR]. It is therefore thought that this approach would not be suitable for small and medium-sized server installations.

- *Dynamic Page Creation is a Critical Part of Any Adaptive Framework.*

  Dynamic Page creation is the most reasonable way for web-masters to deal with volatile information. Any adaptive system should be able to provide these dynamic page creation programs with access to information about the client's platform.

4.3 Design Goals

The design goals for the Siblings framework are to:

- provide for platform-adaptivity via intelligent retrieval.
- provide for platform-adaptivity via forwarding Platform information to servlets.
- accommodate next generation document bases such as those that use XML and XSL to create HTML or XHTML documents.
- support current protocols and standards;
- minimise re-architecting of existing Jigsaw Framework.
- Use best practices in Object-Oriented Design in design of framework.
- Make as many of enhancements pluggable.
- Ensure enhancements integrate with Jigsaw.

4.4 Candidate Strategies

Five major strategies were considered to accomplish the adaptation of site content for diverse platforms:

- *Dynamic Pages*. Creating dynamic pages on the server that are customised for the client platform.
- *Intelligent Retrieval*. Using the server to select the most appropriate static page for a particular platform from multiple sets of static pages.
- *Client Proxy.* Server serves XML source and style sheets, client proxy formats locally.
- *Intermediate Mark-up Languages.* Develop or use a platform-independent user interface mark-up language either on server or client. This intermediate user interface language is used to define an 'abstract interface' for the site which is rendered differently for different platforms.

These strategies were subjectively evaluated against the following criteria

- *Server Loading*. The CPU loading on the server could affect scalability and restrict the applicability of the architecture to more powerful server platforms.
- *Author Control*. Author control of the final presentation can sometimes be lost when using generic formatting for a platform.
- *Data Protection*. Site content is often highly valued by the site owners. Returning XML to a client instead of HTML distributes the site content in a much more usable form.
- *Presentation Quality*. Generic approaches to interface design often result in less attractive designs.
- *Ease of Management*. Using a single document base for all platforms generally would be more manageable than multiple document bases for each platform.
- *Flexibility in Navigation/Structure*. Different platforms might necessitate different navigation schemes and site structures.

Each design strategy has several advantages and disadvantages. Because Server-based Dynamic page creation programs are executed with every request it is possible to create output closely tailored to the features of the device. This approach can accommodate the features of individual client configurations. Dynamic page creation programs are also suitable for situations where the information is volatile or involves database access. The disadvantages of the use of dynamic page creation programs for platform-adaptivity are that they do consume considerable CPU time. Another problem with this approach is that if a different user experience (different navigation, different content, and different site structure) were desired for different platforms the necessary logic to encode this would likely be very complex. As with any server-based approach, the critical site content is partially protected from unauthorised re-use since the information is maintained in pure form on the server and distributed in the HTML pages interspersed with extra mark-up commands. The extra mark-up in the HTML makes it very difficult to extract the information from a page. This advantage is shared by any server-based approach to adaptation (e.g. intelligent retrieval).

Server-based intelligent retrieval has the advantage of efficient use of the CPU, the ability to effectively use caching, and the enabling of the use of different site structures and navigational schemes for different sites. Intelligent retrieval uses the static page retrieval mechanism of the server that is typically highly optimised and very efficient. Intelligent retrieval as implemented in this thesis would require different directory trees for each different platform supported. This puts limits on the number of different platforms that can practicably be supported and the level of customisation that is possible for the client. Server-based intelligent retrieval would only be useful to adapt output for client device *classes* and not for individual client *configurations*.

Another approach to platform-adaptation is to use XML to store site content and using a proxy on the client to render the XML into appropriate HTML for the client configuration. This enables highly customised output and very efficient use of server resources. Transformation of the XML to HTML is done on the client. This offloads most of the work involved in adaptation from the server to the client. One disadvantage of using a proxy on the client is that there is a loss of author control of output. The proxy must be able to transform a wide variety of XML files for a wide variety of configurations. The author would not necessarily be able to predict how the output would be rendered on these different configurations.

Intermediate user-interface mark-up languages are currently being used in commercially available adaptive solutions (e.g. Harmonia, Portal-to-Go). This approach promises ease of management since only one document base would be necessary. The abstract interface defined using the intermediate mark-up language would be rendered differently for different platforms. This necessarily forces the authors to use a 'one size fits all' approach to choosing site content to display and navigation. This approach provides multiple presentations for the same user experience, but does not accommodate multiple user experiences for diverse platforms. Table 4-1, below, summarises the advantages and disadvantages of the various strategies is shown below.

	Server Load.	Author Control	Data Protection	Present-ation Quality	Ease of Manage-ment	Navigatory Strucutural Flexibility
Dynamic Pages (S)	Med-High	Med-High	High	Med - High	High	Med
Intelligent Retrieval (S)	Low	High	High	Med - High	Med	High
XML+Proxy (Client)	Low	Low	Low	Med - High	High	Low
Transcoding	Med	Med	High	Med	Med	Low
UIML (Client)	Low	Low	High	Low - Med	High	Low

Table 4-1 Comparison of Adaptation Strategies

4.5 Siblings – Strategy

The strategy chosen was a hybrid server-based strategy that provides for intelligent retrieval of pages based on the client platform and enhanced support for platform-adaptive dynamic page creation programs by forwarding platform information to these programs. It was decided that this strategy provided the best compromise of advantages and disadvantages to run on low-end server platforms using current technology and standards to produce suitable output and allow different site navigational schemes.

The advantages of a server-based approach include:

- PDAs and SmartPhones might not have the processor or memory capacity to efficiently perform adaptive rendering.

- Converting XML to HTML at the server provides some protection for the intellectual property and business value of the information in the XML document base.
- Better author control of output.
- Good Use of Bandwidth. The server could have profiles for low bandwidth systems that would retrieve and render the outputted HTML so that it was better optimised for low bandwidth.
- More flexibility in navigation and retrieval.

Since the intelligent retrieval of documents proposed here is based on using parallel directories to store the HTML for each major platform, this could add to site management problems. A major problem with multiple hierarchies is maintaining consistency between the hierarchies. If all site content is encoded in XML and the HTML is generated from XSL or CSS style sheets, much of the updating can be automated. If an HTML file is dependent on an XML file, it can be recreated every time the XML file is changed. Programs such as MAKE are designed to automatically run procedures whenever a file is older than the files which depend on it.

Intelligent retrieval typically involves a table lookup and optionally a redirection to a different directory. Neither is as CPU intensive as launching a new process or thread. A major advantage of intelligent retrieval is that it allows site architects to use different navigational structures and prescribe different content for different platforms.

It was decided to use the Jigsaw Web Server from the W3C as a base architecture because the:

- architecture is fully object-oriented and allows alteration of the request/retrieval process in several ways;
- source is available;
- server is Java-based making it easy to extend;
- performance is acceptable when compared to other servers.

The main concern with choosing a Java-based server is speed. Java compiles to an intermediate byte-code that must be interpreted into machine code at run time before execution. C and C++ based web servers compile directly into machine code (i.e. object code) avoiding any run-time interpretation. The authors of Jigsaw state that the server can achieve performance comparable with C and C++ based web servers. This is done primarily through very sophisticated use of caching. The most compelling reason to use Jigsaw is that it has a clean, fully object-oriented design oriented toward graceful extension of the server's function.

4.6 The Jigsaw Web-Server.

4.6.1 Overview.

The Jigsaw web-server is a Java-based Object-Oriented web server developed by the World Wide Web Consortium. The key design advantage of the server is that it facilitates customisation and extension of the web-server's function and that it has a clean object-oriented design. Other notable features include support for the HTTP 1.0 and HTTP 1.1 protocols and the fact that the server has been open-sourced.

The server's design represents a departure from more conventional servers. Most conventional servers interpret the document path in the user request simply as a file or directory name to be retrieved. Jigsaw, by contrast, treats the path as a sequence of separate lookup and processing steps. Each component of the path acts as an anchor to attached objects that can perform lookup, retrieval and filtering tasks. These objects are instances of either a frame or a filter. Frames are used to provide specialised lookup and retrieval techniques. Filters are typically used to modify a request or reply either before or after passing through a frame although they are capable of lookup functions. The diagram below illustrates how the document path "/archives/index.html" would be interpreted.

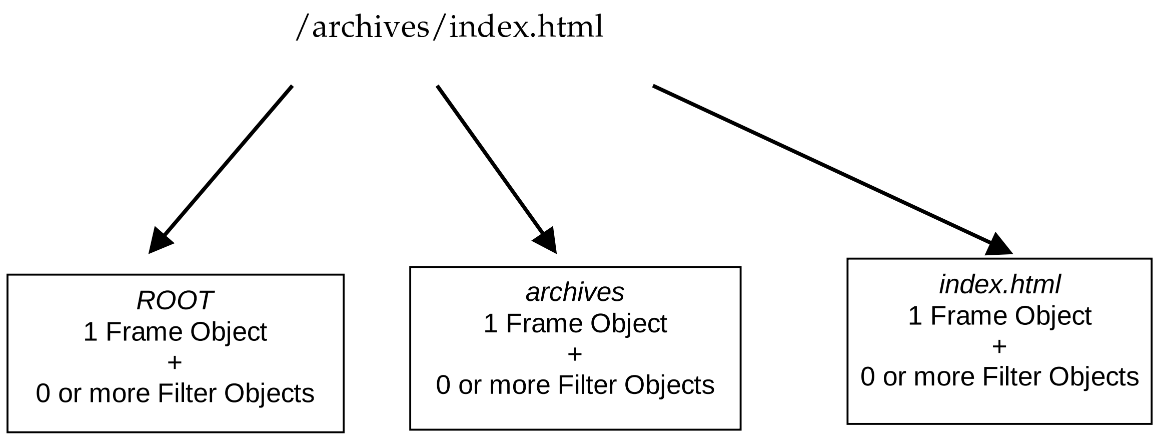
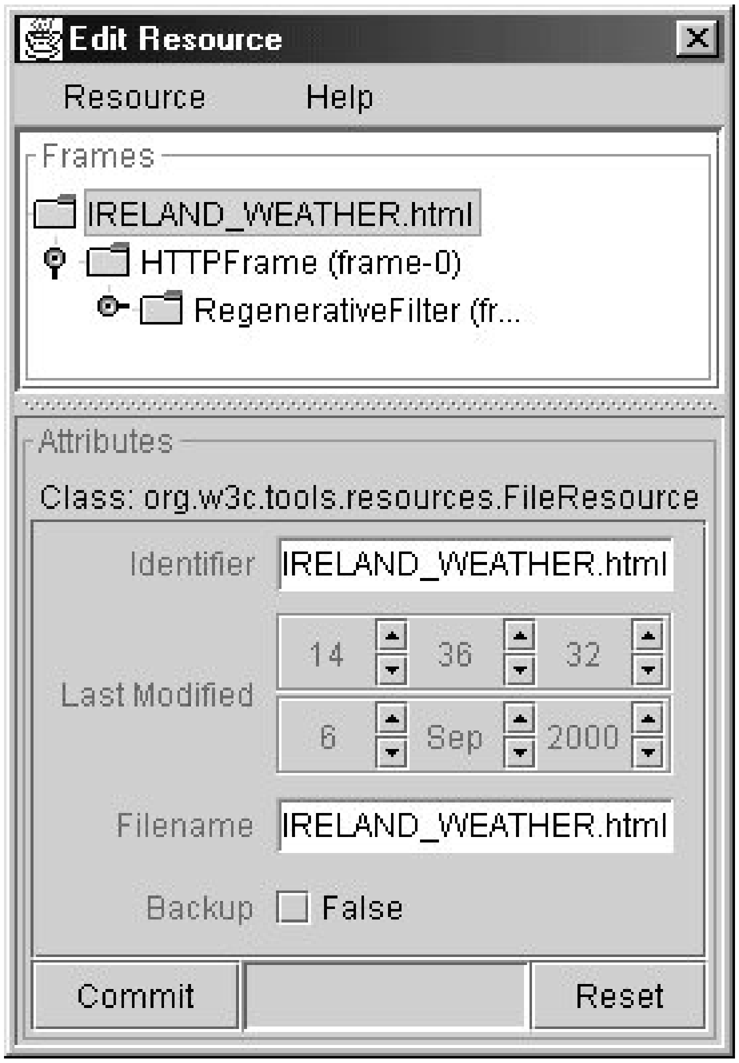


Figure 4-1 Document Path Components

As Jigsaw follows through the document path, it calls the appropriate routines of any components associated with a particular directory or file. Jigsaw's Lookup and Perform Algorithm is discussed in greater detail in Section 4.6.4. An administrator can customise request processing by attaching different Java frame or filter classes anywhere in the web server document hierarchy. This 'per directory' or 'per file' configuration allows for fine-tuned customisation of server function. The diagram below shows the Edit Resource Window of the Jigsaw Administration Program, with an HTTPFrame attached to the CANADA_WEATHER.html file and a RegenerativeFilter attached to the HTTPFrame. The HTTPFrame class encodes lookup and retrieval functions that are associated with standard HTTP requests. The HTTPFrame class is included in the Jigsaw distribution. The *RegenerativeFilter* could be any Jigsaw or programmer-supplied class that derives from a standard Jigsaw ResourceFilter class.



Class: org.w3c.tools.resources.FileResource

Identifier	IRELAND_WEATHER.html
Last Modified	14 36 32
	6 Sep 2000
Filename	IRELAND_WEATHER.html
Backup	False

Figure 4-2 Attaching Filters to File Resources

4.6.2 The Daemon

Jigsaw uses a daemon, *httpd.java*, which waits on a server port, servicing incoming requests by forwarding the request to a pool of client objects. The client object is capable of many of the tasks essential to communicating with a client. These tasks include starting, interrupting and closing connections; processing requests, emitting replies, and maintaining critical state to enable the determination of whether a connection can be retained after request processing. Client objects run in their own thread. The pool of clients will attempt to associate a new request with an idle client before creating a new client object for that request. This improves efficiency because it reduces the need for the expensive task of creating and destroying threaded objects. If there are no idle client objects and the maximum number of connections has been reached, a connection is refused. Figure 4-3 lists simplified pseucode for the Jigsaw daemon logic.

```
while server not interrupted
        Accept a client connection.
        Forward connection request to client pool
        if maximum connections reached then
            reject connection
            continue to next iteration
        else if idle-client-exists then
            assign connection to available client
        else
            create new client and add to pool
            assign connection to newly created client.
        end-if
        Use Resource Module to locate Resource.
        Fire the 'ingoing' method of all filters.
        Call the 'perform' of the target Resource.
        Fire the 'outgoing' method of all filters.
        Log reply.
end-while.
```

Figure 4-3 Jigsaw - Simplified Server Logic

The daemon also manages a set of classes to perform tasks such as logging, authentication and resource management. The Java class used to perform logging can easily be reset using a Server Administration program. The daemon does load a catalogue of 'authentication realms' but the mechanics of implementing security is primarily associated with resources themselves.

4.6.3 Resources, Frames and Filters

Jigsaw treats every server resource (e.g. directories, files) as a full Java object. These are lightweight objects that usually only store an identifier and some last-modified attributes. Resources can be classified as a simple or container resource (resources which contain other resources). If the resource is a container (e.g. a directory) it will also store a reference to an indexer object to enable the resource to index its contents.

The methods used to 'serve' the resource are encoded in a 'Frame' that is associated with the resource. For instance, an HTTPFrame would provide all the methods necessary to serve a resource using the HTTP protocol. Two key functions of any frame are the 'Lookup' and 'Perform' functions. You can also use the frame to associate more state information with the basic resource. These frames are a powerful tool for configuring and extending the operations of the server. The Jigsaw administration tool allows you to attach one of several frames to directories and other resources thereby defining the behaviour of that directory or resource. There are several specialized frames included with the server that provide extra functionality over the HttpFrame such as content negotiation, request redirection, invoking CGI scripts and servlets, and emitting ZIP files.

For the programmer wishing to extend the capabilities of the server, the most common method would be to inherit from one of the existing frames or filters and add functionality as needed. Filters are similar to frames but are used to intercept requests before being processed by a frame and/or intercept the emitted reply from the frame before the reply is returned to the client. Filters can be attached to any frame. Examples of filters available with the server includes filters to compress output, display the content of requests and replies to the console, limit concurrent access to a resource, and many others.

4.6.4 Jigsaw's Lookup and Perform Algorithm.

As mentioned before, Jigsaw treats a document path as a series of components to be used in the search and retrieval of a document. As Jigsaw processes the document path it will call the appropriate functions from the components and maintain a lookup state. Amongst other things, the lookup state stores a representation of the document path still to be processed. Amongst the objects that can be associated with a document path component is one and only one frame, and zero or more filters. Each frame and filter has a lookup method that determines how the next component in the document path can be located. Each frame also has a perform function, which encodes how to create a response if the frame is associated with the terminal component of the document path. Filters are used to do pre-processing of the request, or post-processing of the response. For example, a URL, "/archives/index.html", has 3 components; "/" or the root, "archives" which is a sub-directory, and "index.html" which is the actual target file. Let's assume that during the server configuration process, the web-site administrator associated an HTTPFrame with the root, the archives subdirectory, and the contents of the archives subdirectory. The HTTPFrame has the routines for the lookup and retrieval of documents using the HTTP protocol. Let's also assume that the administrator has attached a filter, F1, to the root directory, F2 to 'archives' subdirectory and F3 to the "index.html" file. Please note that attaching filters to frames such as the HTTPFrame is optional and typically would not be necessary.

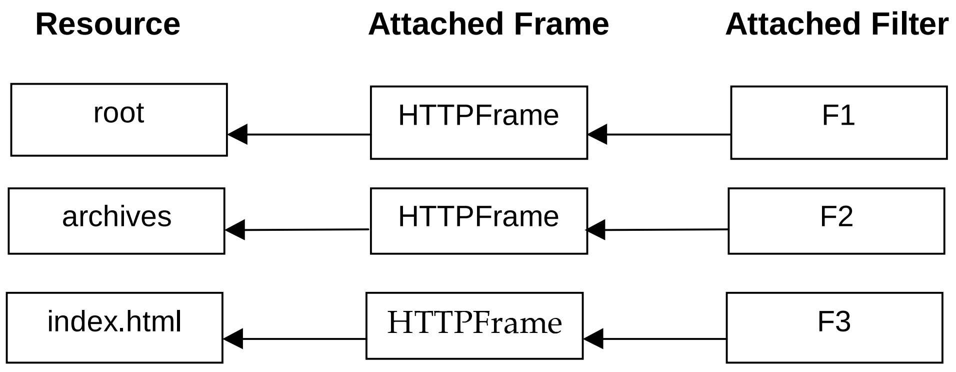


Figure 4-4 Set-up for */archives/index.html*

In this scenario, Jigsaw would start from the root directory, calling the *lookup()* function of the HTTPFrame attached to the root directory. This *lookup()* function would cause the filter attached to the frame to be added to its growing 'list of filters'. Jigsaw will attempt to call all the lookup routines of each filter in its list of filters. Since the root is a directory, the *lookupDirectory()* routine is called. If the resource were a File, the *lookupFile()* routine would be called. The *LookupState* object will have been updated appropriately. This same sequence is repeated for all components of the path until the components are exhausted. Using this scheme, by the time we have cascaded to the target resource, we will have accumulated a list of filters; F1, F2, F3. The 'ingoing' functions of F1, F2, and F3, will be fired, in that order. A retrieval of the resource is then executed by calling a 'perform' routine and a reply created, after which the *outgoing()* functions of F3, F2, and F1 are called, in that order.

Figure 4-5, taken from the W3C site [LOOKUP], illustrates the lookup described above. Notice how the lookup cascades down from the root directory calling each frames lookup then the filter lookups. The diagram is a simplification in that HTTPFrame2 would call the lookup of F1 and F2, and HTTPFrame3 would call the lookup of F1, F2, and F3.

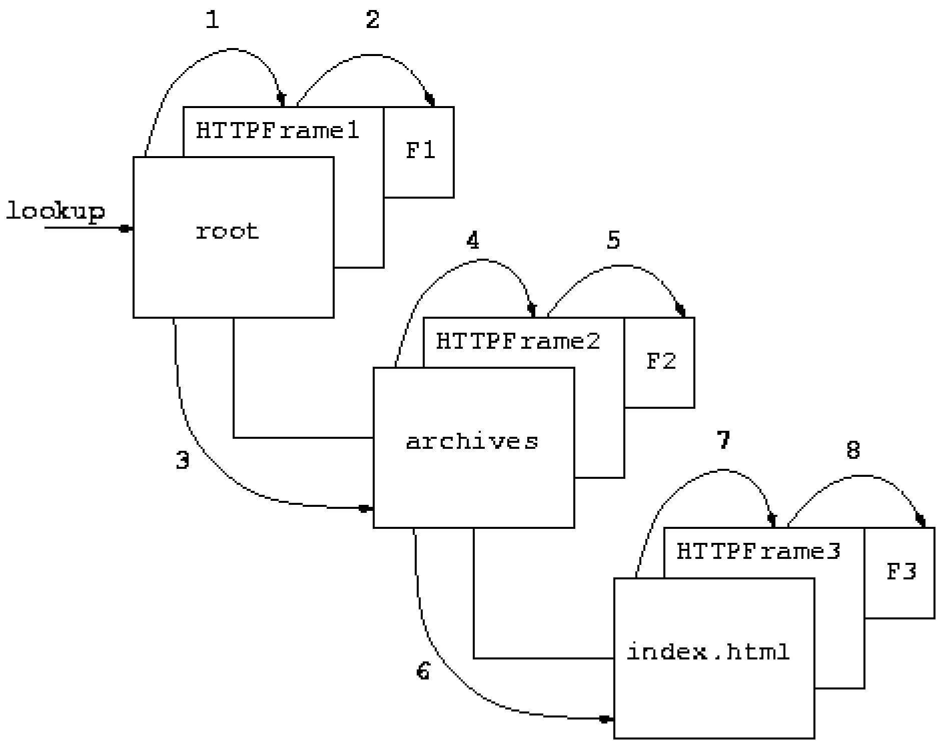


Figure 4-5 Jigsaw - Look-up Logic [LOOKUP]

Figure 4-6, also taken from the W3C consortium site [LOOKUP], shows the perform algorithm. Although all frames have a perform function, *only the perform method of the frame attached to the target resource is called.* The perform method actually creates the reply once a document is found. Only one reply is necessary for one request. All *ingoingFilter* functions will be called before the perform method of the resources, and all *outgoingFilter* functions will be called after.

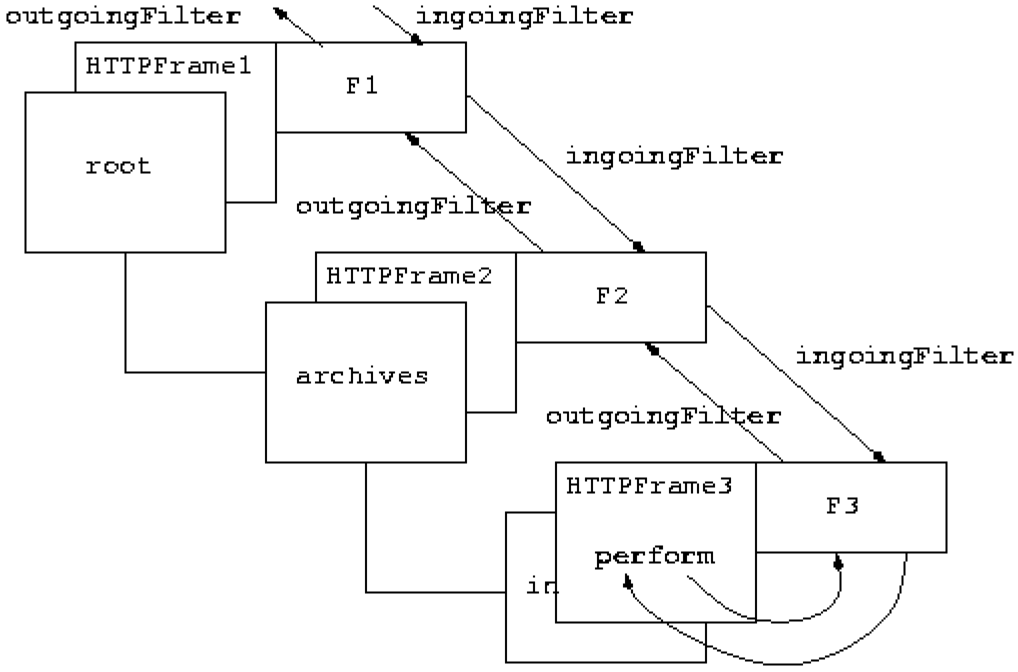


Figure 4-6 Jigsaw - Perform Logic [LOOKUP]

4.7 Architecture.

The creation of Siblings involved a set of structural and operational alterations to the Jigsaw distribution. The main design changes made to the Jigsaw architecture are:

- Adding a new subsystem to characterise the platform of the request's client.
- Changing appropriate classes in the Jigsaw framework to enable dissemination of platform information to both static retrieval and dynamic page generation programs.
- Changing the control flow of request processing when intelligent retrieval is required. This required the definition of two new classes, *AdaptorFrame* and *ReMapper*.

- Adding a new class, *RegenerativeFilter*, to the Jigsaw class hierarchy, which can automatically regenerate HTML files from specified XML and XSL files on a specified time schedule.

4.7.1 Characterisation Subsystem

The purpose of this subsystem is to identify the platform of the client. A critical issue in the design of this subsystem is that the method used to identify the platform be easily changed. This was achieved by establishing an abstract interface, *PlatformFactory*, which must be implemented by all concrete Platform Factories. The Factory design pattern is a common design pattern used to instantiate objects based on parameters or information supplied at run-time [GAMMA95]. Using a Factory pattern for platform discovery made it easier to allow the encoding and plugging of new platform discovery algorithms.

Subclasses of *PlatformFactory* could use any technique for platform discovery including interpreting user-agent strings in the request or Composite Capability/Personal Profile. It was decided to use the information in the User-Agent string of the request to determine the platform type of the client. This User-Agent string typically contains information on the client browser and operating system and therefore is an indicator of the type of the client device. A *UserAgentFactory* class was defined which implemented the *PlatformFactory* interface and used the User-Agent string to search through a catalogue of platforms stored in a *PlatformCatalog* object to find the most suitable platform for that User-Agent String. A new class *Platform* was defined which was able to store pertinent information about a platform such as name, screen dimensions, and colour capability. Figure 4-7, below, shows a UML static structure chart of the subsystem.

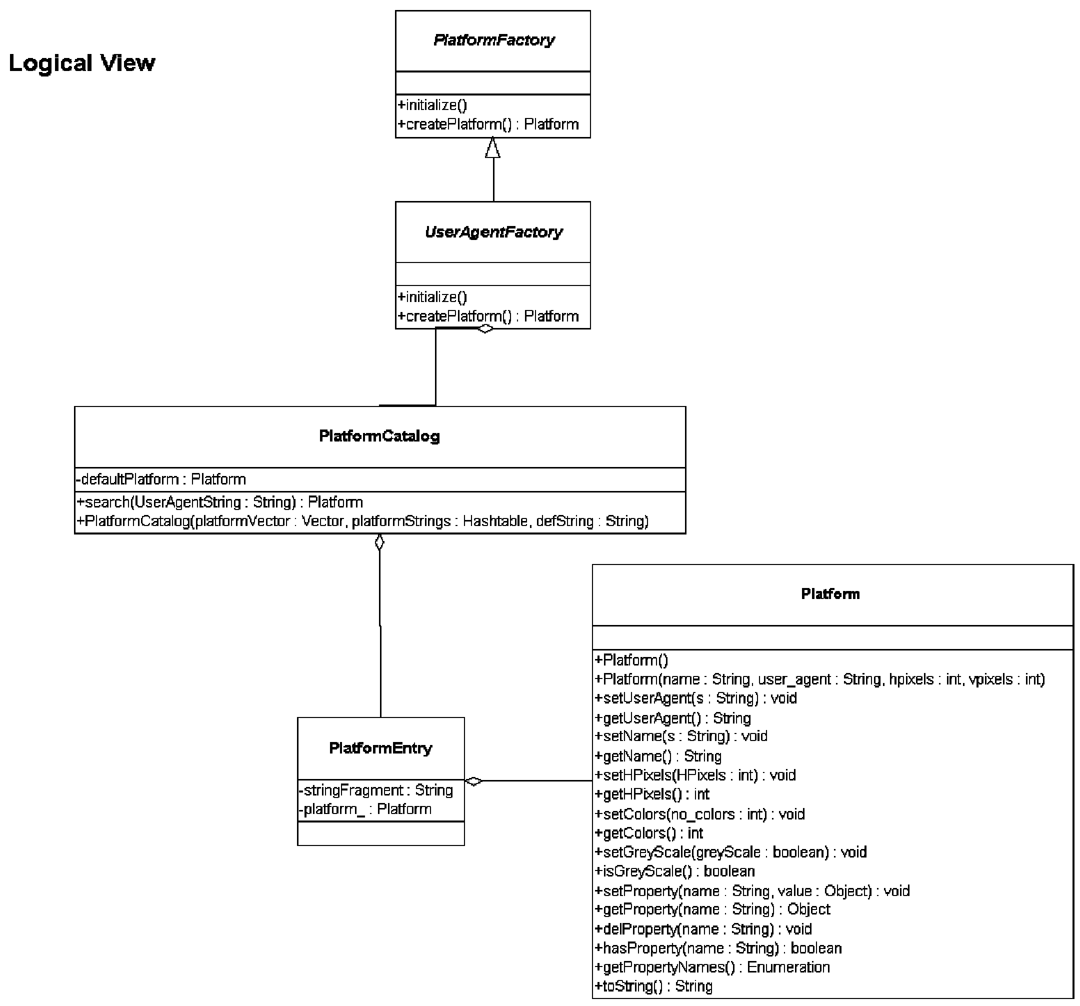

Logical View
PlatformFactory
+initialize()
+createPlatform() : Platform
UserAgentFactory
+initialize()
+createPlatform() : Platform
PlatformCatalog
-defaultPlatform : Platform
+search(UserAgentString : String) : Platform
+PlatformCatalog(platformVector : Vector, platformStrings : Hashtable, defString : String)
PlatformEntry
-stringFragment : String
-platform_ : Platform
Platform
+Platform()
+Platform(name : String, user_agent : String, hpixels : int, vpixels : int)
+setUserAgent(s : String) : void
+getUserAgent() : String
+setName(s : String) : void
+getName() : String
+setHPixels(HPixels : int) : void
+getHPixels() : int
+setColors(no_colors : int) : void
+getColors() : int
+setGreyScale(greyScale : boolean) : void
+isGreyScale() : boolean
+setProperty(name : String, value : Object) : void
+getProperty(name : String) : Object
+delProperty(name : String) : void
+hasProperty(name : String) : boolean
+getPropertyNames() : Enumeration
+toString() : String

Figure 4-7 Characterisation Subsystem - Static Structure

4.7.2 Dissemination of Client Platform Information

It was important that the information derived about the client platform during characterisation was made available to as much of the Jigsaw infrastructure as possible. Since the Jigsaw *Request* class is used by all request processing routines in Jigsaw, this was an ideal class to provide an accessor to retrieve the *Platform* object associated with the request. It was decided that during characterisation the *Platform* object would be bound to the *Client* object and not the *Request* object. Jigsaw's *Client* class definition was changed to store a reference to a *Platform* object and an accessor that could return the reference. All *Request*s have a reference to their *Client* object and therefore would have access to the *Client*'s *Platform* object. This was done for efficiency reasons. Jigsaw supports the connection-oriented request processing that is possible with HTTP 1.1 [RFC1945,RFC2068]. This means that the socket connection to the client may be retained and reused after a request has been processed, unlike HTTP 1.0. By associating the *Platform* with the *Client* it was possible to run the characterisation routine only when a client connects rather than with every new request. A simplified static structure diagram showing the relationships between the Request, Client and Platform classes is shown in Figure 4-8.

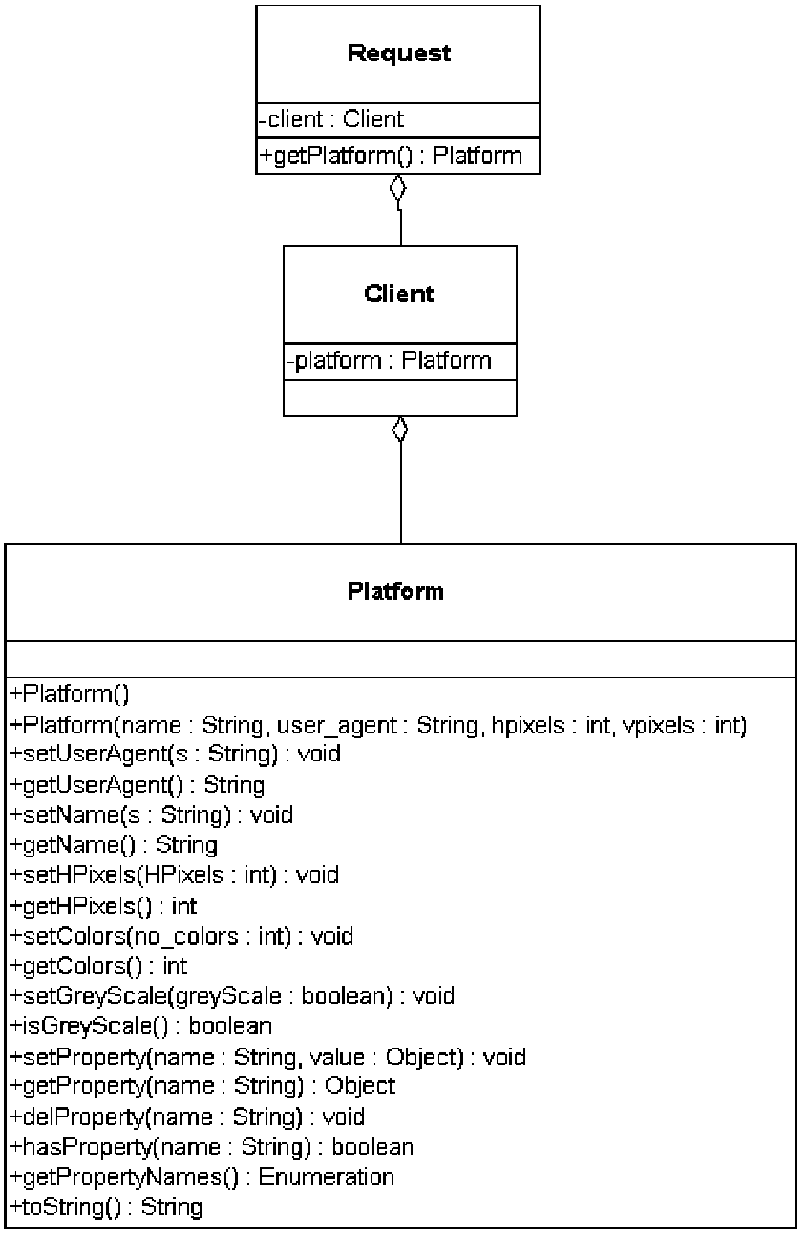

Request
-client : Client
+getPlatform() : Platform
Client
-platform : Platform
Platform
+Platform()
+Platform(name : String, user_agent : String, hpixels : int, vpixels : int)
+setUserAgent(s : String) : void
+getUserAgent() : String
+setName(s : String) : void
+getName() : String
+setHPixels(HPixels : int) : void
+getHPixels() : int
+setColors(no_colors : int) : void
+getColors() : int
+setGreyScale(greyScale : boolean) : void
+isGreyScale() : boolean
+setProperty(name : String, value : Object) : void
+getProperty(name : String) : Object
+delProperty(name : String) : void
+hasProperty(name : String) : boolean
+getPropertyNames() : Enumeration
+toString() : String

Figure 4-8 Request Static Structure-Simplified

4.7.3 Request Processing

HTTP request processing was altered for the case where intelligent retrieval was required. Siblings uses an advanced form of redirection to accomplish intelligent retrieval. The Siblings framework allows a web-master to associate a platform type with a directory, and every directory could have a set of sibling directories that deal with the same content but with a different presentation so that requests could be redirected to more appropriate directories by comparing the platform type of the request and that of the directory and its various sibling directories. The document name could also be re-mapped to a different document name during the redirection process. The altered flow of control for request processing with intelligent retrieval is shown in Figure 4-9 on the following page.

It was decided to enable intelligent retrieval by deriving a new class, *AdaptorFrame,* from *HTTPFrame* and defining a new class, *ReMapper,* as a base class that can be subclassed to encode re-mapping logic. *HTTPFrame* was chosen to be subclassed because *HTTPFrame* already contains the logic to do a look-up and perform for a standard HTTP request. *AdaptorFrame* will encode the altered flow of control in Figure 4-9 through changes to the look-up and perform routines. It was necessary to also add a reference to a *ReMapper* object to the *AdaptorFrame* so that it could re-map file names during the redirection process. *ReMapper* was designed as a very simple base class with a narrow interface so that it could be easily subclassed when new re-mapping logic was required.

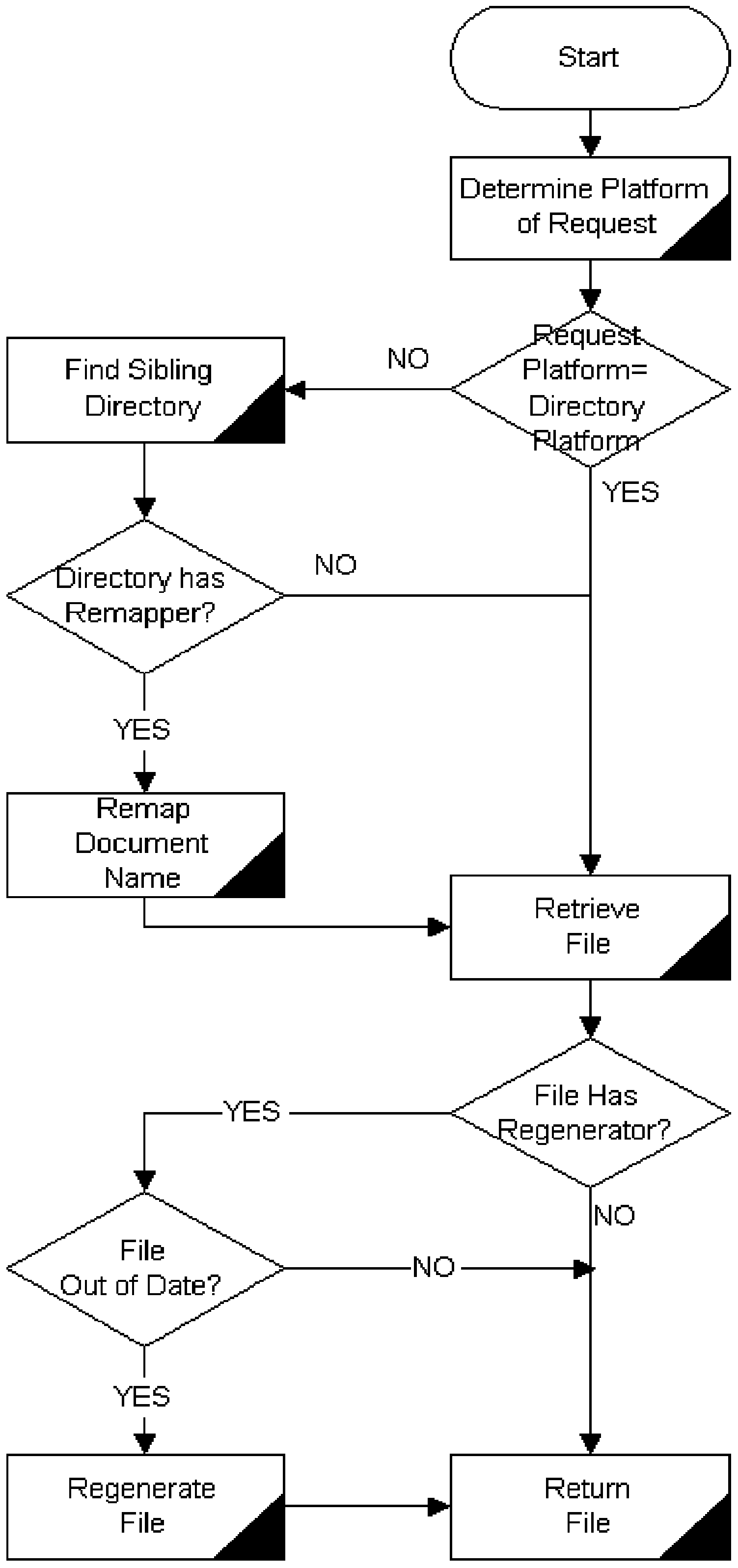


Figure 4-9 Sibings: Intelligent Retrieval Request Processing

4.7.4 Automated Document Regeneration

A new class, *RegenerativeFilter*, derived from Jigsaw's *ResourceFilter*, was required which would store attributes for an XML source file name, an XSL style sheet name, a string list of XSL parameters, and an integer for the age in seconds of the target file that triggers a regeneration. It was decided that a filter would be used for document regeneration instead of a frame because it is typically easier for a user to configure filters and it meant that only one method had to be implemented, the *outgoing()* method.

The *RegenerativeFilter* would compare the returned file's date stamp to determine if it is older than the maximum age. If it is older than the maximum allowed age, the file will be recreated by using an XSLT processor to transform an XML file into an HTML file using the specified XSL style sheet.

4.8 Summary

This chapter presented the design for the Siblings Web Server Framework. The assumptions and goals motivating the design decisions were also listed. Since the Siblings Framework was intended from the outset to be a set of enhancements to the Jigsaw Web Server, the design and operation of the Jigsaw Web Server was also discussed. The next chapter will discuss how the design was implemented.

5 Implementation

5.1 Introduction

Chapter 4 discussed the changes to the design of Jigsaw necessary to achieve the goals of the framework. This chapter discusses how those design changes were implemented in the context of a set of implementation goals.

5.2 Implementation Goals

The implementation goals of this thesis are:

- Implement server enhancements specified in Design for characterisation, dissemination of platform information, adaptive redirection, adaptive re-mapping, and automated document regeneration.
- Enhancements are easily extended.
- Enhancements are pluggable at run-time where possible.
- Enhancements are configurable through the JIGADMIN GUI server administration program where possible.
- Minimize changes of the interfaces of Jigsaw's core classes.

5.3 Functional Changes

5.3.1 Platform Characterisation

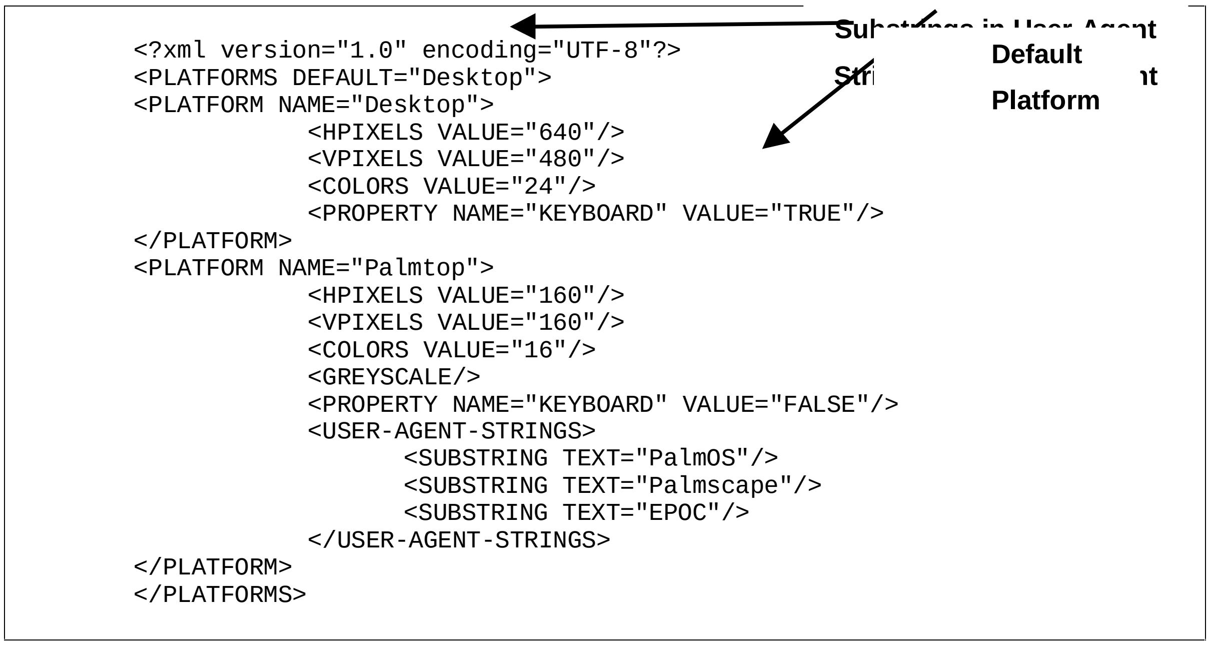
Characterisation, with respect to the Siblings framework, is the determination of the platform type and platform features to be associated with a client request. This step is critical since the platform name is used to redirect requests and platform features can be used by dynamic page routines to produce customised output.

The functionality associated with characterisation was implemented through the definition of an abstract *PlatformFactory* interface. Concrete platform factories would be required to implement this interface. The Factory design pattern is a common design pattern used to instantiate objects based on parameters or information supplied at run-time [GAMMA95].

The Jigsaw daemon*, httpd.java*, has been changed so that it will dynamically load a subclass of a *PlatformFactory*, initialise that factory, and then during request processing use the *PlatformFactory* to associate a *Platform* object with the *Client* object associated with the incoming request. To improve efficiency the association of a platform with a client is done when the client connects.

Any subclass of *PlatformFactory* can be used to perform client characterisation. This means that different approaches can be used to characterise incoming requests (e.g. user-agent strings, Composite Capability/Personal Profile). The implementation developed in this thesis used the information in the *User-Agent* string of the HTTP request to determine the platform type. The *User-Agent* string is sent with every request from a browser and usually contains the name of the browser and optionally the operating system of the client. A subclass of *PlatformFactory*, *UserAgentFactory*, was implemented which loaded a *PlatformCatalog* object with the platform information contained in an XML file.

An example of a such simple platform file is listed in Figure 5-1. In the example file below there is a list of substrings that may be used to identify a client as a Palmtop: "PalmOS","Palmscape","EPOC". The User-Agent string will be searched to determine whether it contains any of these strings. Also note that the file specifies a default platform, "Desktop". The file layout is not RDF-compliant. Making this file RDF-compliant might be considered for future work.


Default Platform

```
<?xml version="1.0" encoding="UTF-8"?>
<PLATFORMS DEFAULT="Desktop">
<PLATFORM NAME="Desktop">
            <HPIXELS VALUE="640"/>
            <VPIXELS VALUE="480"/>
            <COLORS VALUE="24"/>
            <PROPERTY NAME="KEYBOARD" VALUE="TRUE"/>
</PLATFORM>
<PLATFORM NAME="Palmtop">
            <HPIXELS VALUE="160"/>
            <VPIXELS VALUE="160"/>
            <COLORS VALUE="16"/>
            <GREYSCALE/>
            <PROPERTY NAME="KEYBOARD" VALUE="FALSE"/>
            <USER-AGENT-STRINGS>
                  <SUBSTRING TEXT="PalmOS"/>
                  <SUBSTRING TEXT="Palmscape"/>
                  <SUBSTRING TEXT="EPOC"/>
            </USER-AGENT-STRINGS>
</PLATFORM>
</PLATFORMS>
```

Figure 5-1 platforms.xml Source

The *PlatformCatalog* class was designed and implemented to be immutable. Immutable objects cannot be modified once they are instantiated since they have no member functions that modify instance or class variables. This means that they can be made thread-safe without employing synchronization (which is CPU-intensive). This will make calls to the search routine both thread-safe and efficient. *UserAgentFactory* returns references to existing platform objects rather than creating a new *Platform* object for every *Client* object. This was done to improve efficiency. If detailed information is required about the individual client's device a new Platform object would have to be associated with every client.

A *PlatformParser* class was implemented to parse the XML file and create a *PlatformCatalog* object. The *PlatformParser* class uses the standard SAX parser that comes with the Jigsaw distribution. The *UserAgentFactory* uses this *PlatformParser* to create a *PlatformCatalog*.

5.3.2 Disseminating Platform Information

Platform information determined during Platform characterisation must be disseminated to the static page retrieval mechanisms and servlets.

This involved the definition of a new class *Platform*, and fundamental changes to some of Jigsaw's core classes. The Jigsaw *Client* class was altered to store a reference to a *Platform* object. The appropriate Platform object reference is assigned to a Client object when a client connects to the server. Jigsaw *Request* objects have references to their *Client* object so that Platform information is accessible to a *Request* indirectly through its referenced *Client* object. It was decided to associate *Platforms* with the *Client* instead of directly with a *Request* since this meant that for connection-oriented protocols such as HTTP 1.1, the platform need only be determined when the client connects instead of with every new request. The *Request* class was changed to include a method, *getPlatform()*, which returns the request's platform information as a *Platform* object.

5.3.3 Adaptive Redirection

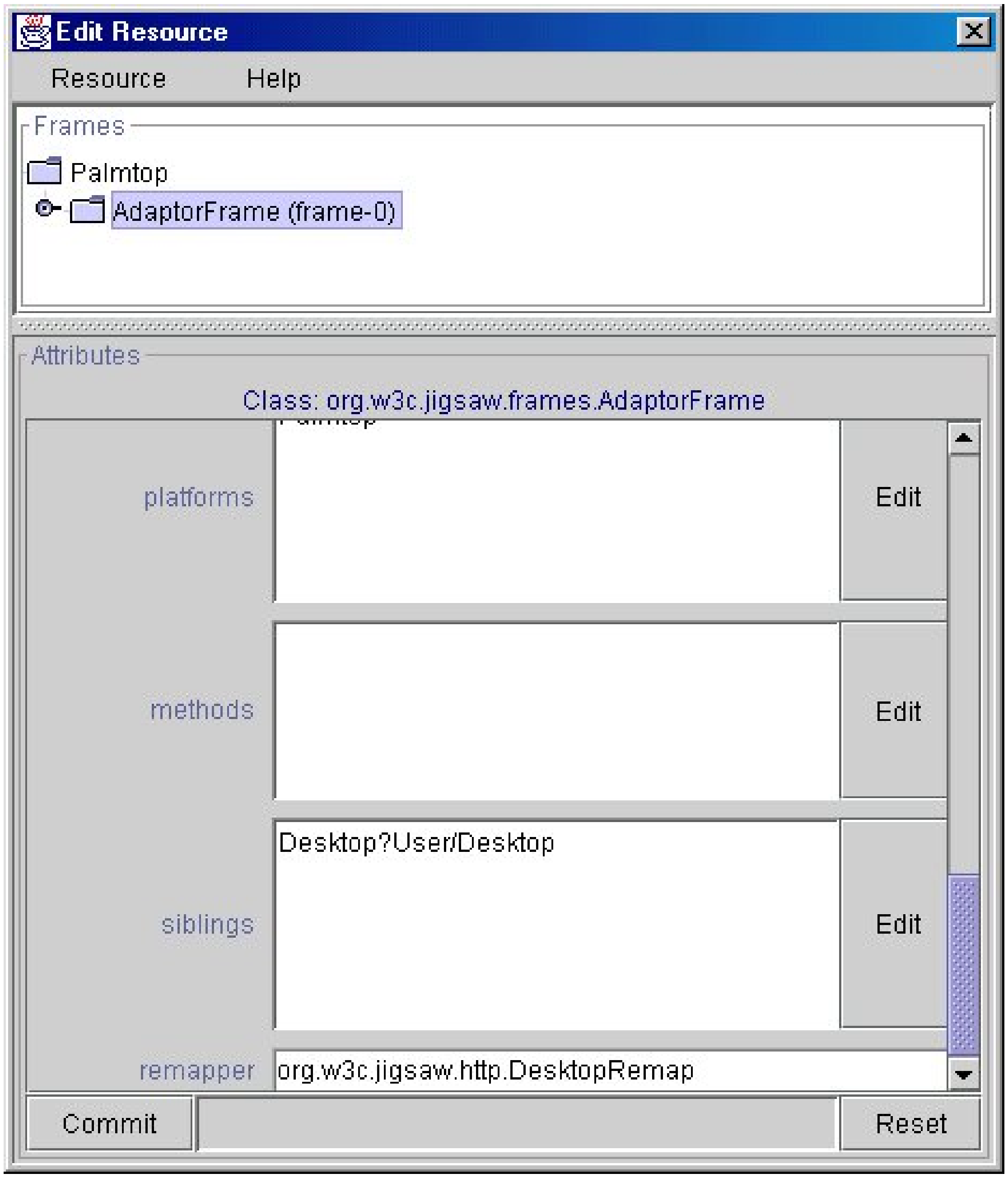
Adaptive redirection was achieved by implementing an AdaptorFrame class. This class subclasses the standard *HTTPFrame* class that comes with the Jigsaw distribution and changes the way a standard HTTP look-up and perform is executed.

AdaptorFrames can be attached to any directory. They differ from standard *HTTPFrames* in that they:

- Store a list of Platform names permitted to access the directory.
- Store a list of Sibling strings with a format of

  <PLATFORM_NAME> + "?" + <DIRECTORY_NAME>
- During lookup and perform, compares the Request Platform Name to the list of allowed Platform Names to determine whether to retrieve the file from the directory or redirect the request to a sibling directory.
- Store a reference to a *ReMapper* subclass that can be used to remap the request document name during redirection.
- Invoke the *ReMapper* subclass's *remapFileName* during redirection.

Adaptive redirection is accomplished by a simple table search. The platform name associated with the request is compared against the platform names in the list of allowed platforms. If there is a match the file is retrieved. If there is no match then the platform name associated with the request is compared against the platform name component of the Sibling strings. If there is a match then the request is redirected to that sibling directory, and if a *ReMapper* class has been specified, the *remapFileName* method of the *ReMapper* will be invoked to change the document name.



Class: org.w3c.jigsaw.frames.AdaptorFrame

platforms

methods

siblings: Desktop?User/Desktop

remapper: org.w3c.jigsaw.http.DesktopRemap

Figure 5-2 Configuring an AdaptorFrame

5.3.4 Adaptive Re-mapping

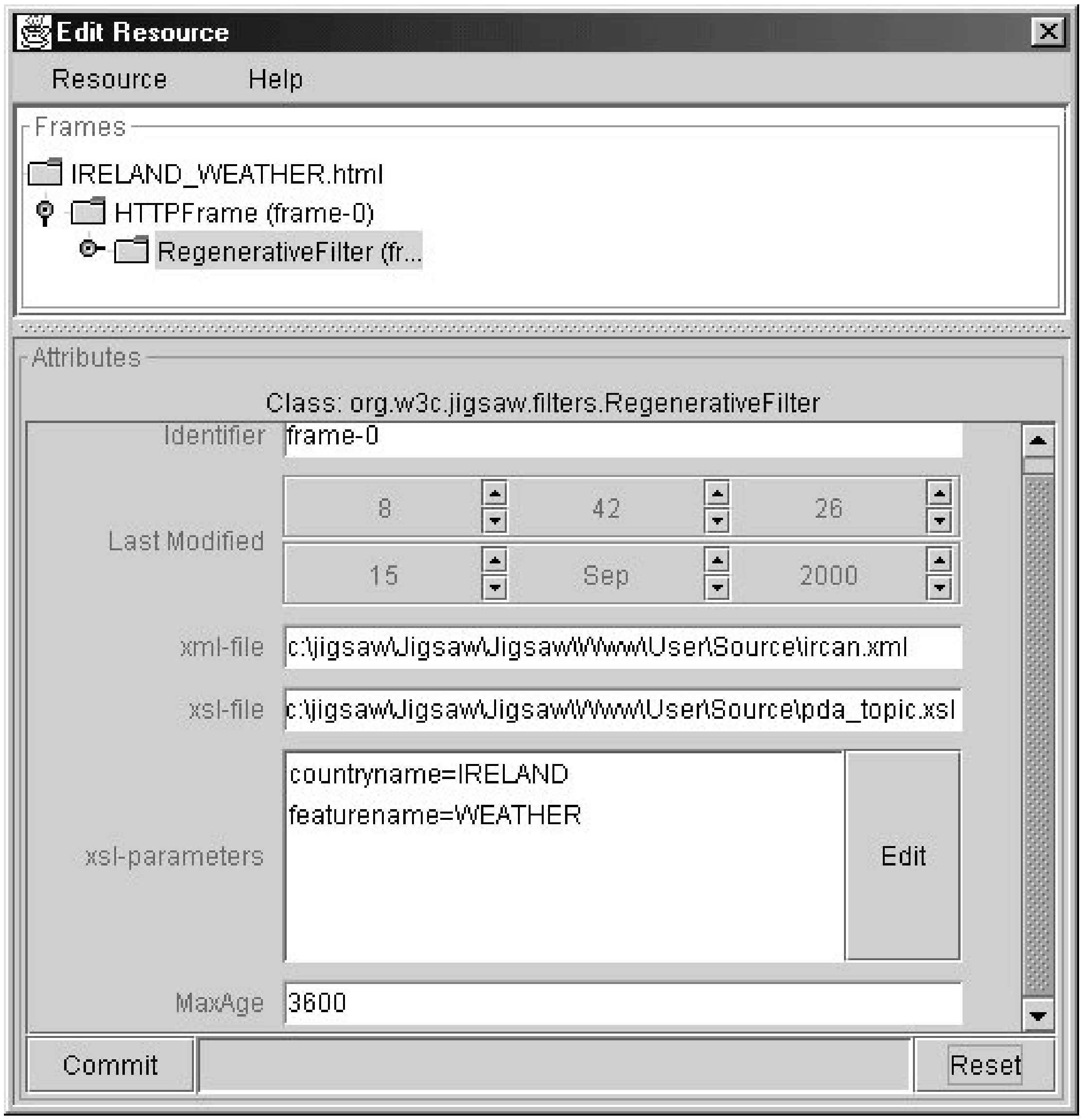
Adaptive re-mapping was accomplished by encoding the *AdaptorFrame* so that it could dynamically load a subclass of the *ReMapper* class, instantiate an object of the class, then call the *remapFileName* method of that class during redirection. If no *ReMapper* class was specified in the configuration of the *AdaptorFrame*, then re-mapping would not be attempted. The *AdaptorFrame* will provide a string representing the redirect path name, a string for the original platform name and a string for the redirected platform name. The information in these parameters allows the use of one class for an entire set of sibling directories. The method signature for *remapFileName* is shown below:

```
public String remapFileName(String path,String CurrentPlatform,String OriginalPlatform )
```

Web site authors can provide re-mapping capabilities by subclassing *ReMapper* and encoding re-mapping logic into the *remapFileName* function.

5.3.5 Automated Regeneration.

The Siblings architecture was designed to automatically regenerate documents from their source XML and XSL files. This was done by implementing a *RegenerativeFilter* as a subclass of a Jigsaw *ResourceFilter*. The RegenerativeFilter takes as arguments the source XML file, the source XSL file, a list of parameters for the XSL style sheet, and a maximum age before the file is regenerated. The *RegenerativeFilter* uses these parameters to invoke an XSLT processor using these files and parameters. The XSLT processor used was LotusXSL Release 1.0.1 from IBM.



Class: org.w3c.jigsaw.filters.RegenerativeFilter

Identifier: frame-0

Last Modified: 8 42 26 / 15 Sep 2000

xml-file: c:\jigsaw\Jigsaw\Jigsaw\Www\User\Source\ircan.xml

xsl-file: c:\jigsaw\Jigsaw\Jigsaw\Www\User\Source\pda_topic.xsl

xsl-parameters:
countryname=IRELAND
featurename=WEATHER

MaxAge: 3600

Figure 5-3 Configuring Automated Regeneration

5.4 Implementation Features and Discussion

Section 5.3, Functional Changes, presented how the functional changes were implemented. The evaluation of how well these functional changes perform is discussed in Chapter 6. The implementation satisfied most of the implementation goals. The enhancements in this thesis are easily extended. The newly defined classes typically have very narrow interfaces making them easy to use and extend. The use of a Factory pattern for the Platform characterisation means that the method of characterising clients can be changed easily. The classes used in characterisation, adaptive redirection, adaptive re-mapping and automated document regeneration are pluggable at run-time since these classes are dynamically loaded. Most of the enhancements can be configured from the JIGADMIN GUI. It is not possible to specify the name of the *PlatformFactory* subclass from the JIGADMIN GUI.

This could be a future enhancement. Unfortunately, it was not considered practical to implement the functional enhancements to the server without minor changes to two of Jigsaw's core classes, the *Request* class and the *Client* class. Only minor changes were required to these classes; the *Client* class now had to reference and provide accessors to a *Platform* object, and an accessor has been added to the *Request* class that returns its *Platform* type.

5.5 Summary

Chapters 5 presented a discussion of how the design described in Chapter 4 was implemented. This was approached from the viewpoint of major functional changes to the Jigsaw server. It was found that it was possible to implement the design changes specified in Chapter 4 in accordance with most of the implementation goals specified at the beginning of this chapter. Minor changes were required to some of Jigsaw's core classes, however. The following chapter discusses how this implementation was evaluated and presents a conclusion for the thesis.

6 Evaluation and Conclusions

6.1 Introduction

The preceding chapters presented a discussion of the design and implementation of the Siblings Server Framework. This chapter provides an evaluation of that implementation and conclusions for the thesis as a whole. Suggestions for future work are also discussed.

The Siblings Server Framework was evaluated on the basis of the design goals, the practicality of its use and its benefits or disadvantages with respect to other approaches to adaptive delivery of content. Special consideration will be given to the level of effort and level of skill needed to implement a web site using the infrastructure, support for XML and XSL and support for alternate site structures for different platforms. A proper discussion of platform-adaptive architectures must also deal with the issue of the interacting effects of device characteristics and user behaviour when using the Web, since this could be the prime determinant of the effectiveness of any adaptive web site.

6.2 Evaluation.

The main questions that must be answered in the evaluation are as follows:

- Do the server enhancements work as intended?
- Is it practicable to implement web sites using this technology?
- What are the performance impacts of the different enhancements?

The approach used in the evaluation was as follows:

- Design and implement a non-trivial Web Site designed for viewing from both Desktop and Palmtop client devices employing XML/XSL technology for the generation of the HTML documents.
- Develop and run test suites to evaluate each server enhancement.

- Develop and run a test suite to compare speed of server function with and without Siblings framework enhancements.

6.2.1 EireCan – A Case Study

EireCan, a sample web site was developed to demonstrate the use of the major features of the server framework and investigate the level of effort needed to implement a small web site so that it produces appropriate output for both desktops and palmtops. EireCan is a mock web site devoted to providing information about Ireland and Canada, with respect to travel, geography, and news links. All content on the site is stored in 3 XML files and all HTML on the site is generated from 4 XSL styles sheets. By using parameters with two of the style sheets it is possible to create several target HTML files by providing different parameters to the XSLT processor. There is one XML file for the main site information, and one XML file that stores statistics on each country; Ireland and Canada. The site also uses one servlet to allow palmtops to retrieve detailed statistics about individual countries. Another feature of the site is the use of automatic regeneration of HTML files from XML data files and XSL style sheets. HTML files for weather in Ireland and Canada are automatically regenerated if the server detects that the files are beyond their maximum age (1 hour).

6.2.2 EireCan - Site Structure.

The EireCan site was implemented using a total of 4 directories; a Palmtop directory to store palmtop-related HTML files, a Desktop directory to store desktop-related HTML files, a Source directory to store the XML and XSL files, and a Servlet directory to servlets. Jigsaw defaults to using the 'Www' subdirectory as the document root for its web server so that all these directories must be descendants of the 'Www' directory.

6.2.3 EireCan – Source Directory

This directory stores the XML files for the encoding of site content and XSL files to provide for the automated formatting of content for the specific platforms. These files were used to create all the html files in the Desktop and Palmtop directories. The major source files required for the site were *ircan.xml*, *ei.xml*, *ca.xml*, *dsk_country.xsl*, *pda_country.xsl*, *pda_topic.xsl*, *dsk_topic.xsl* and *pda_links.xsl*. These files are discussed below

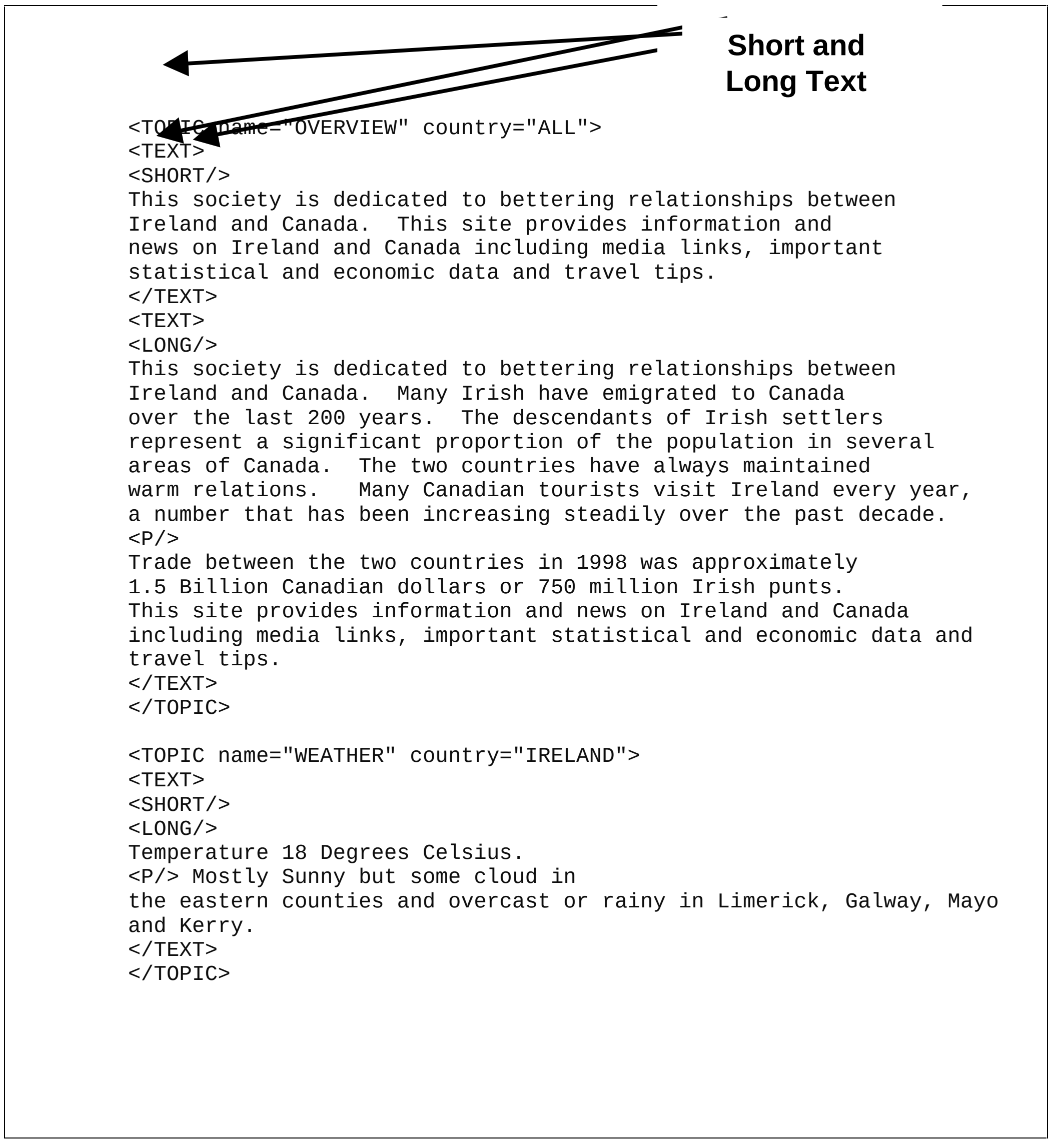
ircan.xml

This is an XML file that stores site content in both long and short text forms for most of the pages on the site. The file stores site content according to topic, not page, and often has both long and short text on the topic. This allows the XSL authors to selectively choose either short or long text when creating an HTML page. *ircan.xml* also stores a collection of links identified by country, area, and topic. A DTD for the file is shown in Figure 6-1.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT LINK (#PCDATA)>
<!ATTLIST LINK
 type (NEWSPAPER | TOURISM) #REQUIRED
 country (CANADA | IRELAND) #REQUIRED
 area CDATA #REQUIRED
 platform CDATA #REQUIRED
 href CDATA #REQUIRED
 name CDATA #IMPLIED
>
<!ELEMENT LINKS (LINK+)>
<!ELEMENT LONG EMPTY>
<!ELEMENT P EMPTY>
<!ELEMENT SHORT EMPTY>
<!ELEMENT TEXT (#PCDATA | SHORT | LONG | P)*>
<!ELEMENT TOPIC (TEXT+)>
<!ATTLIST TOPIC
 name (OVERVIEW | TRAVEL) #REQUIRED
 country (ALL | CANADA | IRELAND) #REQUIRED
>
<!ELEMENT WEBSITE (TOPIC+, LINKS)>
<!ATTLIST WEBSITE
 name CDATA #REQUIRED
>
```

Figure 6-1 IRCAN DTD

Example content taken from *ircan.xml* is shown in Figure 6-2. Please note that the "OVERVIEW" topic has two text sections, one labeled with a <*SHORT*/> tag and another with *a* <*LONG*/> tag. The "WEATHER" topic has only one text section and this section has been labeled with both a <*LONG*/> and <*SHORT*/> tag.


Short and Long Text

```
<TOPIC name="OVERVIEW" country="ALL">
<TEXT>
<SHORT/>
This society is dedicated to bettering relationships between
Ireland and Canada.  This site provides information and
news on Ireland and Canada including media links, important
statistical and economic data and travel tips.
</TEXT>
<TEXT>
<LONG/>
This society is dedicated to bettering relationships between
Ireland and Canada.  Many Irish have emigrated to Canada
over the last 200 years.  The descendants of Irish settlers
represent a significant proportion of the population in several
areas of Canada.  The two countries have always maintained
warm relations.   Many Canadian tourists visit Ireland every year,
a number that has been increasing steadily over the past decade.
<P/>
Trade between the two countries in 1998 was approximately
1.5 Billion Canadian dollars or 750 million Irish punts.
This site provides information and news on Ireland and Canada
including media links, important statistical and economic data and
travel tips.
</TEXT>
</TOPIC>

<TOPIC name="WEATHER" country="IRELAND">
<TEXT>
<SHORT/>
<LONG/>
Temperature 18 Degrees Celsius.
<P/> Mostly Sunny but some cloud in
the eastern counties and overcast or rainy in Limerick, Galway, Mayo
and Kerry.
</TEXT>
</TOPIC>
```

Figure 6-2 IRCAN.XML Sample Mark-up

ei.xml,ca.xml

These files contain extensive statistics on the geography, economy, and people of Ireland and Canada. The data was derived from the CIA World Fact Book and is relatively current. Each file contains over 100 separate elements with elements nested up to 4 levels deep.

_dsk_country.xsl, pda_country.xsl_

These XSL files provide formatting instructions for the presentation of country data from either *ei.xml* or *ca.xml* for the desktop and palmtop environments. Although *ei.xml* and *ca.xml* contain many elements with deep nesting, the actual style sheets involved only elementary use of XSL.

_pda_topic.xsl, dsk_topic.xsl_

These parameterised XSL files provide formatting instructions for individual topics in the *ircan.xml* file. It involved the use of more advanced XSL formatting instructions. The use of parameters, such as "countryname=IRELAND" and "topicname=TRAVEL", allow the web master to use the same style sheet and XML file to create a large number of HTML files (e.g. potentially one HTML file for each topic in "ircan.xml").

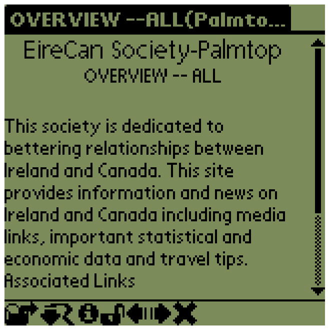
_pda_links.xsl_

This file was used to create a page of links for the palmtop platform.

6.2.4 EireCan – Look-and-Feel and Navigation

The EireCan site was designed with a different look and feel for the palmtop-oriented and the desktop-oriented site. The desktop-oriented documents generally have more text and typically have a sidebar with a list of important links. The HTML pages with detailed information about each country include map and flag graphics, and over 90 statistics grouped into 3 major categories. The palmtop-oriented HTML pages generally have shorter text and no graphics. The pages displaying country statistical information display only 12 statistics, but have a pop-up to query on another 10 statistics. The palmtop-oriented pages link to a page of hot-links, *links.html*. The desktop-oriented directory does not contain a *links.html* file. This is the only structural difference between the directories.

The index page for the Palmtop is shown in Figure 6-3 below. It displays brief text and a limited number of links to other pages in the Palmtop directory, including a page designed solely to display the hot-links that are shown in the side-bar of the Desktop index page. The user must scroll to view the navigation links.

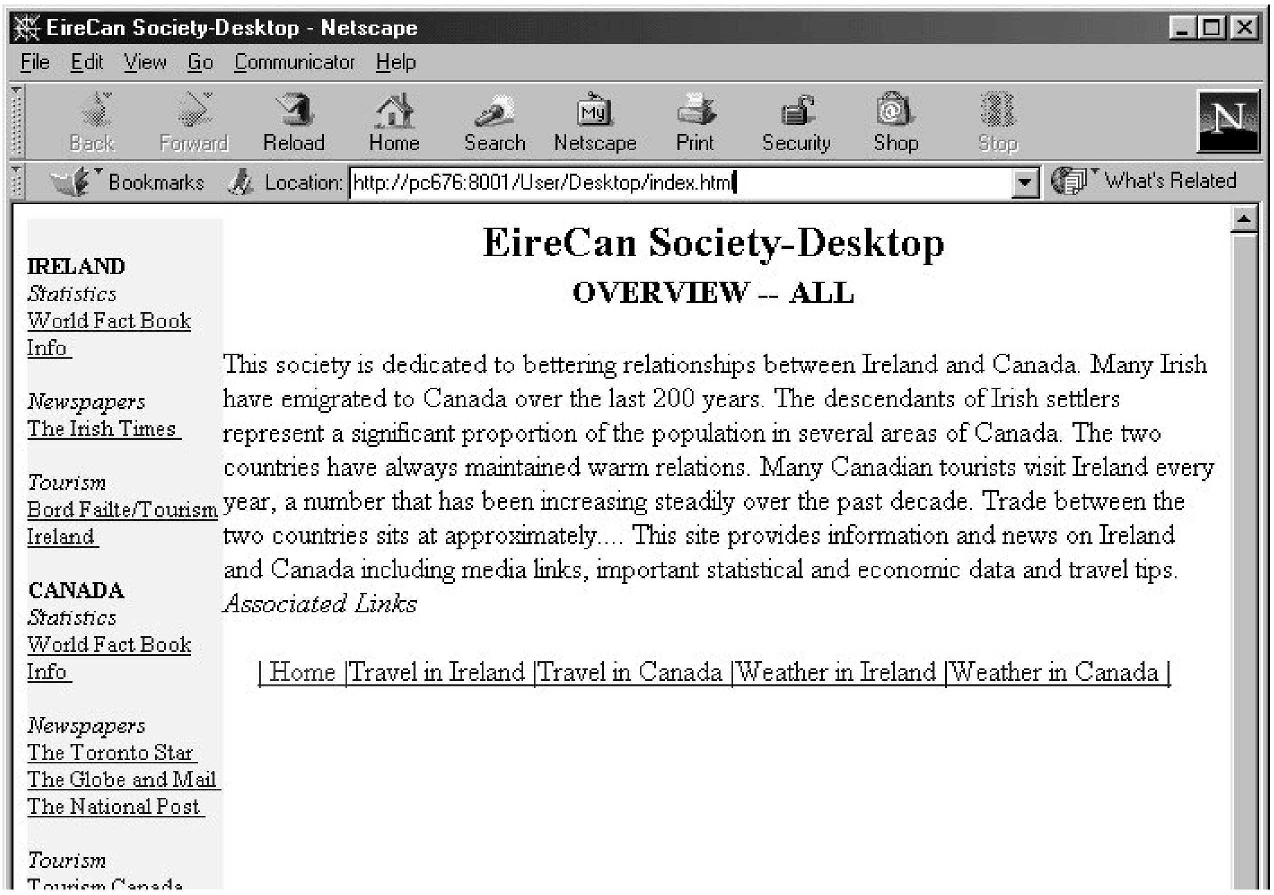


EireCan Society-Palmtop
OVERVIEW -- ALL

This society is dedicated to bettering relationships between Ireland and Canada. This site provides information and news on Ireland and Canada including media links, important statistical and economic data and travel tips.
Associated Links

Figure 6-3 EireCan Palmtop Screen Shot

The Desktop index page shown in Figure 6-4 has a sidebar of links and more complete text.


EireCan Society-Desktop - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://pc676:8001/User/Desktop/index.htm
What's Related

IRELAND
Statistics
World Fact Book Info

Newspapers
The Irish Times

Tourism
Bord Failte/Tourism Ireland

CANADA
Statistics
World Fact Book Info

Newspapers
The Toronto Star
The Globe and Mail
The National Post

Tourism
Tourism Canada

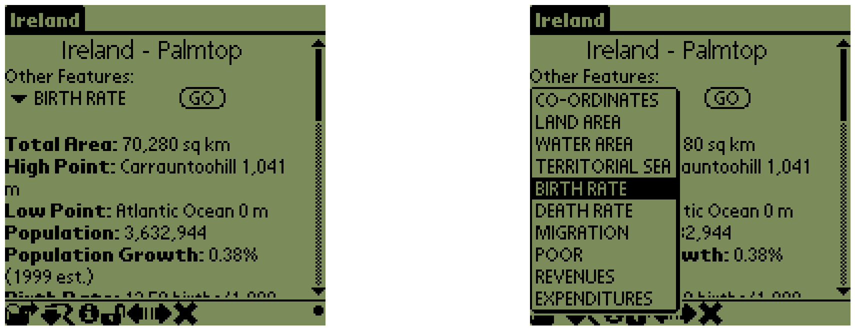
EireCan Society-Desktop
OVERVIEW -- ALL

This society is dedicated to bettering relationships between Ireland and Canada. Many Irish have emigrated to Canada over the last 200 years. The descendants of Irish settlers represent a significant proportion of the population in several areas of Canada. The two countries have always maintained warm relations. Many Canadian tourists visit Ireland every year, a number that has been increasing steadily over the past decade. Trade between the two countries sits at approximately.... This site provides information and news on Ireland and Canada including media links, important statistical and economic data and travel tips.
Associated Links

| Home |Travel in Ireland |Travel in Canada |Weather in Ireland |Weather in Canada |

6.2.5 EireCan – Servlets

The EireCan site uses one servlet, *CountryInfo.class*. This servlet was intended to provide palmtop clients extra statistical data items about Ireland and Canada primarily for the palmtop platform. The figure below shows a palmtop screen that displays statistics on IRELAND with and without a pop-up for extra features activated. Tapping the GO button would call the *CountryInfo* servlet sending a form parameter for country name (IRELAND) and feature name (BIRTH RATE).


Ireland
Ireland - Palmtop
Other Features:
BIRTH RATE GO

Total Area: 70,280 sq km
High Point: Carrauntoohill 1,041 m
Low Point: Atlantic Ocean 0 m
Population: 3,632,944
Population Growth: 0.38% (1999 est.)

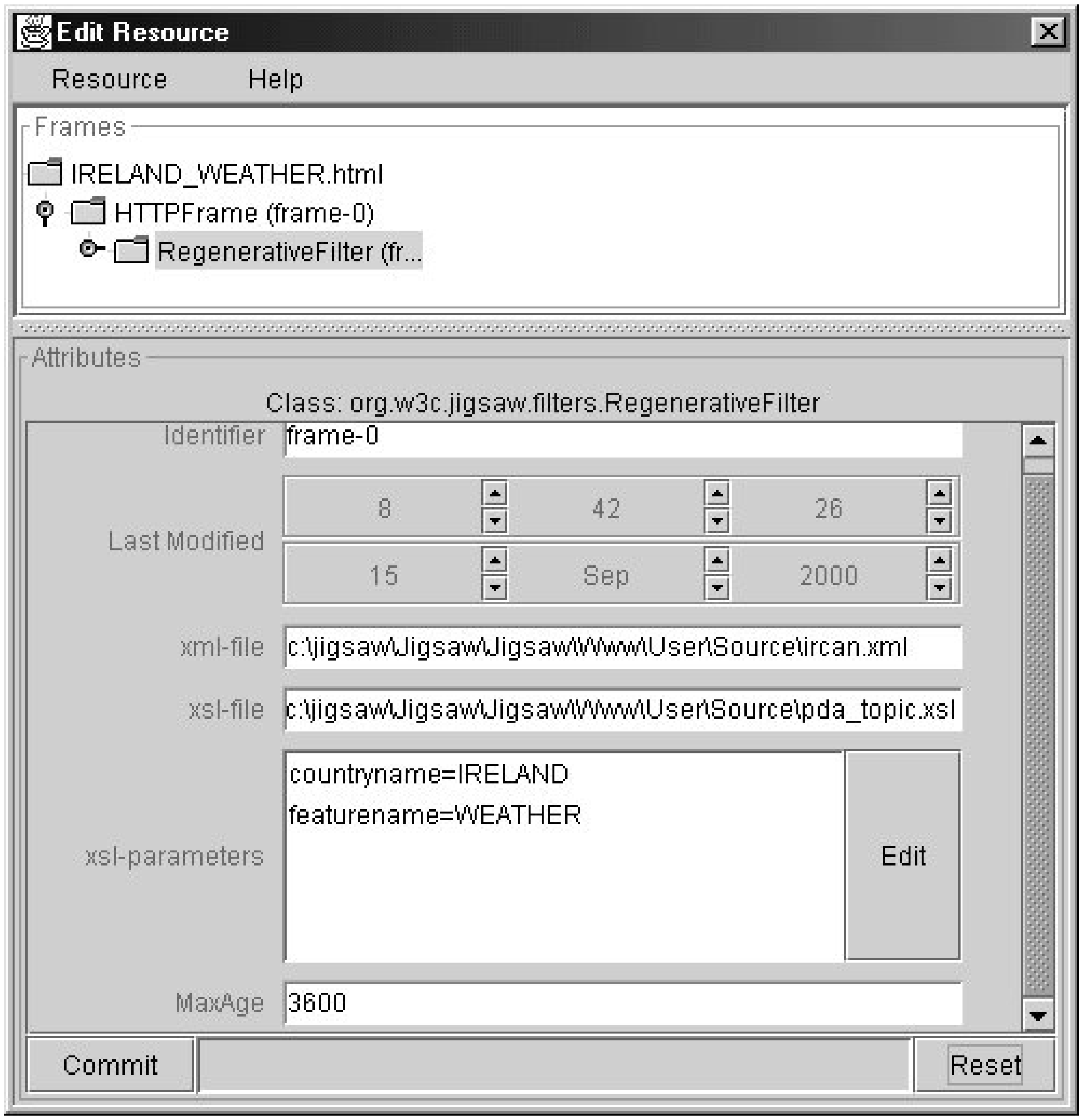
Ireland
Ireland - Palmtop
Other Features:
CO-ORDINATES
LAND AREA
WATER AREA
TERRITORIAL SEA
BIRTH RATE
DEATH RATE
MIGRATION
POOR
REVENUES
EXPENDITURES

Figure 6-5 EireCan Palmtop Query Form

If the servlet is called from a palmtop browser it will reply with a simple screen containing only the information for Ireland's birth rate. If the servlet is called from a desktop it will simply redirect the request to the HTML file in the Desktop directory that contains statistics on Ireland, *ei.html.* The servlet uses information about the platform that is contained in the *Request* object to determine whether to create a palmtop-oriented response or to redirect the request..

6.2.6 EireCan – Automated File Regeneration.

The EireCan site has pages on both the Desktop and Palmtop directories that display the current weather in Ireland and Canada. The information in these files is derived from elements in the *ircan.xml* file that are identified by having a TOPIC name of "WEATHER". It is reasonable to assume that weather data might be updated on a regular schedule, perhaps every hour or 24 hours. A *RegenerativeFilter* has been attached to the HTTPFrame associated with the each of these weather-oriented files. The *RegenerativeFilter* specifies the base XML file to use *(ircan.xml)*, the XSL file for formatting (*pda_topic.xsl* in the Palmtop directory and *dsk_topic.xsl* in the Desktop directory), the necessary parameters for the XSL formatter, and a *MaxAge* specification. The *MaxAge* specification indicates that when the HTML file is older than a specified number of seconds it should be regenerated. In our case, we have set the weather-related HTML files to be regenerated if they are older than 3600 seconds (1 hour). The figure below shows how the *IRELAND_WEATHER.html* file in the Palmtop directory is configured in the JIGADMIN program.



Class: org.w3c.jigsaw.filters.RegenerativeFilter

Identifier: frame-0

Last Modified: 8 42 26 / 15 Sep 2000

xml-file: c:\jigsaw\Jigsaw\Jigsaw\Www\User\Source\ircan.xml

xsl-file: c:\jigsaw\Jigsaw\Jigsaw\Www\User\Source\pda_topic.xsl

xsl-parameters:
countryname=IRELAND
featurename=WEATHER

MaxAge: 3600

Figure 6-6 Configuring Regenerated Files

6.2.7 EireCan – Adaptive Re-mapping.

The Siblings server framework allows web masters to use parallel document hierarchies to customize the presentation of content for specific device classes. Siblings allows for different site structures and navigation through document name re-mapping. This is necessary because there might not be a one to one correspondence between the documents in two sibling directories. In EireCan, *links.html* in Palmtop directory has no analogue in the Desktop directory. *links.html* simply displays a list of important links that relate to Ireland and Canada. Each major topic page in the Desktop directory has a sidebar with important links, however. It would therefore be appropriate to redirect Desktop-based requests for *links.html* in the Palmtop directory to the *index.html* in the Desktop directory, since that file displays a list of links. A Java class, *DesktopRemap*, was written that would intercept any redirection of a request for *links.html* in the Palmtop directory by a Desktop machine and re-map the name to *index.html*. The re-mapping method for the *DesktopRemap* Java class was very simple to write since it involved encoding one major decision in one method. A sample of the re-mapping code is shown below:

```
public String remapFileName(String path,String CurrentPlatform,
        String OriginalPlatform )
{
        String newPath = null;
        String filename  = extractFileName( path );
if(CurrentPlatform.equals("Desktop") && OriginalPlatform.equals("Palmtop") &&
                                        filename.equals("links.html") ){
                int lastSlash = path.lastIndexOf(  '/');
                newPath = path.substring( 0, lastSlash + 1 ) + "index.html";
        }
        if( newPath != null )
                return newPath;
        return path;
}
```

Figure 6-7 remapFileName example

More complex sites would probably need more complex re-mapping logic but only necessitate the proper implementation of one function; *remapFileName* in a class derived from the *ReMapper* class.

6.2.8 EireCan – Set-Up

The JigAdmin program was used to set-up EireCan as an adaptive web site. This program is a GUI-based administration program. The steps in the set-up are as follows.

- Attach *AdaptorFrames* to the Palmtop and Desktop directories and specify the platforms allowed to access the directory, the sibling directories and their respective platforms, and the name of the re-mapping class.
- Attach a *RegenerativeFilter* to *CANADA_WEATHER.html* and *IRELAND_WEATHER.html* in both the Palmtop and Desktop directories. Provide the names of XML file (*ircan.xml*) and the appropriate parameters and XSL style sheet name to the appropriate text boxes of the Edit Resource form.
- Place the *CountryInfo.class* servlet in the *servlet* directory

6.2.9 EireCan – Discussion

The EireCan web site demonstrated the use of all the major features of the Sibling Framework. Although the web site was not very complex, it did require slightly different navigation for the palmtop and desktop client and there were significant differences in the content provided to clients on different platforms. The site required the creation of two document sets, one for palmtops and one for desktops, a simple Java class to handle file-name re-mapping, and a servlet to handle country statistics queries from palmtops. The parallel document sets were not difficult to create since all HTML pages were developed from 3 XML files and 5 XSL style sheets. The creation of content for EireCan would be more time-intensive than a non-adaptive site since long and short text was often stored for the various topics. This drawback would be more pronounced for larger sites. The level of effort to create the presentation documents from content stored in XML should not increase dramatically for larger sites if all pages are created from XSL style sheets. The weather related HTML files were successfully recreated from their respective XML and XSL files on the schedule prescribed. Once the document sets, servlet and necessary *ReMapper* subclass were created the set-up was quickly done.

6.3 A Critical Analysis

The Siblings Architecture was designed to provide platform-adaptive delivery of site content through the implementation of intelligent retrieval and the provision of client platform information to dynamic page routines. Features of the architecture include the ability to allow different site structures for different platforms, the ability to work with current standards and the ability to work efficiently with small and medium-sized servers. The success of the Siblings architecture and implementation, like any other approach to adaptation, will depend on the efficacy of the adaptation, the practicality of using the implementation, whether the elements of the architecture are properly implemented, and the performance characteristics of the implementation. These will be discussed in turn.

6.3.1 Efficacy of Adaptation

One of the main features of Siblings is the support for intelligent retrieval of static pages based on the characterisation of the client platform. The support for intelligent retrieval will allow for different site structures for different platforms and for web masters to design page sets explicitly for specific client device classes. The entire user interaction, including navigational design, can be customised for the client device class instead of simply making alterations to how the content is rendered. This contrasts with the approaches that use an intermediate user-interface mark-up language. Intermediate user-interface mark-up allow the web master to define a single site structure with pages represented by an 'abstract' interface. The abstract interface will be rendered differently on different platforms either at the server, client or using a proxy.

A web master might reasonably decide that different content displayed in a different order and format is appropriate for different platforms. The EireCan site contains XML data files with very detailed statistics about the economy, geography, and people of Ireland and Canada. A map, a flag, and approximately 90 different statistics are displayed under 3 categories

(Geography, People, Economy) for desktop clients. The page did not invoke any querying facility because all content is available on the page. This format has been used quite effectively at the CIA World Fact Book and its mirrors. A web master designing for palmtop interaction might dispense with the categories and design a page with only a small subset of the statistics that would be considered most important and a pop-up that could access some of the other statistics via a servlet on the server. This is not a case of differentiated *rendering* but differentiated *interaction and content*. The Siblings implementation was able to accommodate this differentiated interaction successfully.

The use of intelligent retrieval and static pages will necessarily limit the number of platform configurations that can be dealt with. The Siblings framework achieves intelligent retrieval by characterising the request according to its client device class and directing the request to the most appropriate directory. Redirection based on client characterisation would not be practical to fine-tune the output for the specific capabilities of an individual client's device if they differ from the standard device of that class. It would not be possible to produce a page set for every possible configuration of every platform.

Is the accommodation of a limited number of well-defined device classes sufficient to achieve most of the benefits of platform adaptation? For the two platforms discussed in this thesis there is already considerable convergence in operating systems used, form factors, and device capabilities. The desktop generally will have high resolution colour screens with good audio and video capabilities. There is more variation in the capabilities of palmtops, but convergence seems to be occurring in the palmtop platform as well. The popularity of the palmtop platform is a relatively recent phenomenon. There is no reason to believe that the form factors and capabilities of this platform with respect to video and input would not converge even further over time as it did with the desktop. Even allowing for the current divergence in capabilities of palmtops, web masters could accommodate most palmtops by maintaining a text-oriented palmtop directory and a directory that allows for palmtops having colour graphics. This would eliminate any need for CPU-intensive dynamic adaptation using servlets or processes.

6.3.2 Practicality of Use

The practicality of using Siblings to create platform-adaptive web sites will depend on both the level of effort and level of skill needed to create and maintain the web site. Siblings supports the use of either dynamic page creation (servlets) or static pages in creating platform-adaptive web sites.

The use of servlets will differ very little from standard servlet programming except that the servlets are now provided with information about the client's platform. It will be the responsibility of the servlet to use the platform information appropriately to create pages suitable for the client platform. Java programmers writing servlets can access platform information with a simple method call to the *Request* object.

The intelligent retrieval supported by Siblings assumes that parallel document sets are used for the presentation of content for different platforms types. Creating and maintaining a document set for each platform class would be prohibitively time-consuming if it is not possible to automate the generation and update of these parallel document sets. The use of XML and XSL to create and maintain these documents sets makes this task practicable. XSL style sheets allow for very sophisticated formatting of XML files. If the XML file is designed carefully and we use parameterised style sheets we can use the same XML file and XSL style sheet to create many HTML pages. In the EireCan case study the XML file, *ircan.xml*, contained several *TOPIC* elements, each with a *name* and *country* attribute. By applying different parameters to the same XSL file, *dsk_topic.xsl*, it was possible to create all the desktop-oriented HTML files except for the two files containing the detailed statistics about Canada and Ireland. The number of different files that could be created using *ircan.xml* and *dsk_topic.xsl* is only limited by the number of

TOPIC elements in *ircan.xml*. XSL allows us to select TOPIC elements for formatting based on the values of the element's attributes. For instance, we could generate a file using the information in any topic that has a name of "WEATHER" and a country of "IRELAND" by providing these attributes as parameters to the "*dsk_topic.xsl*" style sheet. Theoretically one could generate an entire web site from one XML file, one parameterised XSL style sheet, and one script file to run the parameterised translations.

Maintaining consistency between the document sets in the face of changes to the XML content files is also important. This can be automated by using the standard MAKE utility available on a variety of operating systems. MAKE is designed to detect files that are older than the files upon which they depend and execute some type of command. It can be used to detect when an HTML file is older than an XML or XSL file upon which it depends, and run the XSLT processor to regenerate the HTML from the appropriate XSL and XML files.

User-written Java classes derived from a base *ReMapper* class perform the re-mapping of file names between sibling directories. Re-mapping in the EireCan case study involved very simple decision-making. More complex re-mapping schemes between sibling directories might be required for more complex site structures. One possible approach to simplifying re-mapping is to identify a limited number of 'anchors' in the various document hierarchies. These act as targets for re-mapping and represent superior entry points into the document hierarchy from the point of view of navigational choices.

In order to implement a non-trivial web site using Siblings, staff will be required to be familiar with the JIGADMIN administration program and understand the operation of the major classes added by the Siblings architecture, the *AdaptorFrame* class, the *ReMapper* class, and the *RegenerativeFilter*. JIGADMIN uses a graphical user interface and was found to be very easy to learn and use. Ideally all HTML will be generated through the use of XML and XSL, thereby requiring expertise in these two areas. Elementary Java skills will be required to encode the logic to re-map document names between sibling directories. The use of Siblings would not require ‘expert’ level skills in JAVA, XML or XSL.

6.3.3 System Testing

A suite of tests was run against the Siblings implementation to ensure the major features executed as intended. The charts below summarize these results. Assume that all request and response document names are pre-pended by “User/”.

Request	Client Device	Re-Map	Expected Response	Status
Palmtop/index.html	Palmtop	No	Palmtop/index.html	OK
Palmtop/index.html	Desktop	No	Desktop/index.html	OK
Palmtop/links.html	Palmtop	No	Palmtop/links.html	OK
Palmtop/links.html	Desktop	Yes	Desktop/index.html	OK
Desktop/index.html	Desktop	No	Desktop/index.html	OK
Desktop/index.html	Palmtop	No	Palmtop/index.html	OK
Palmtop/ggg.html	Palmtop	No	File Not Found	OK
Desktop/ggg.html	Palmtop	No	File Not Found	OK
Palmtop/ggg.html	Desktop	No	File Not Found	OK
Desktop/ggg.html	Desktop	No	File Not Found	OK

Table 6-1 Siblings Function Test Results

Automated regeneration was tested by requesting a file, *Desktop/CANADA_WEATHER.html,* that had an attached *RegenerativeFilter*, and comparing the file date stamp before and after the request was processed. In Figure 6-2, notice how the first access forces a regeneration of the *CANADA_WEATHER.html* file whereas the second one does not. The RegenerativeFilter has been configured to regenerate files that are older than 1 hour (3600 seconds) old.

File Name	Max Age	Time Of Test	File Date (Before)	File Date (After)
Desktop/CANADA_WEATHER.html	3600	06/09/00 11:33	03/09/00 16:42	06/09/00 11:33
Desktop/CANADA_WEATHER.html	3600	06/09/00 11:37	06/09/00 11:33	06/09/00 11:33

Table 6-2 Regeneration Test Results

6.3.4 Performance Testing

Preliminary performance tests were done on the Siblings architecture under light load conditions. Further work is required for an accurate assessment of the performance impacts of the Siblings enhancements under a wide variety of conditions. These preliminary tests indicate that once characterisation is performed, there are no significant performance differences between a standard HTTP request and one that is processed using an *AdaptorFrame*. This may be due to Jigsaw's heavy use of caching. Attaching a *RegenerativeFilter* to a file caused a significant decrease in performance even when the file wasn't regenerated. Performance with an attached *RegenerativeFilter* was still significantly faster than invoking a servlet.

6.4 Future Work

This thesis was primarily a proof-of-concept for an adaptive web server that would enable the use of intelligent retrieval in addition to dynamic approaches to adaptivity. There are many features that can be added or improved in the framework. Suggestions for future work are listed below:

- Implement a Platform Factory based on Composite Capability/Personal Profile and/or Content Negotiation.
- Intensive testing of performance with and without Siblings enhancements.

- Investigate the level of effort required to maintain as well as create a platform-adaptive web site using the Siblings architecture, XML and XSL.
- Investigate the optimisation of characterisation, redirection, and regeneration.
- Compare performance against other popular web servers and adaptive implementations.
- Determine the performance costs of the characterisation process.
- Investigate implementing a larger, more complex web site using the Siblings framework.
- Change the Platforms configuration file so that it is RDF compliant.

6.5 Conclusions

The major conclusions can be drawn from the work in this thesis:

- It is possible to implement a framework that meets the goals laid out in the introduction. These goals included the; implementation of intelligent retrieval; provision of platform information to dynamic page creation programs and support for next generation document bases comprised of XML and XSL.
- It was practicable to implement smaller web sites that provide different navigation and content for different client platforms by using the Siblings framework with XML and XSL.
- The framework provides a scalable approach to web site design by enabling the choice of a dynamic approach to content delivery or through pre-compiling the documents from XML and XSL and using static retrieval.
- The Jigsaw Server Architecture is easily extended and modified. Its object-oriented design and unique way of handling the look-up of documents allows enhancements to be implemented in a number of ways.

Bibliography

[ABRAMS1998] UIML:An Appliance-Independent XML User Interface Language, Abrams, Phanouriou, Batongbacal, Willams, Shuster
http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html

[APION1999] BUILDING THE MOBILE INTERNET
http://www.apion-tss.com/CM/WhitePapers/wap_wp.html

[BOSAK1999] XML and the Second-Generation Web, Jon Bosak, Tim Bray
http://www.sciam.com/1999/0599issue/0599bosak.html

[BOSAK2000] Grass-roots XML, Xtech 2000 Presentation, San Jose
http://www.oasis-open.org/xtec/sld00000.htm

[CCPP1999] Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation. W3C Note 27 July 1999
http://www.w3.org/TR/NOTE-CCPP/

[CETUSLINKS] Internet & Intranets: XML, DOM, XSL, RDF and related technologies
http://www.cetus-links.org/oo_xml.html

[DIGESTOR] Digestor: Device-Independent Access to the World Wide Web. T. W. Bickmore and B. N. Schilit., Computer Networks And ISDN Systems, Volume 29, Issue 8-13, pp. 1075-1082, September 1997

[EBXML] ebXML General Information
http://www.ebxml.org/geninfo.htm

[ECLIPTYC2000] Digital Plastic Product Page, Ecliptyc Systems.
http://www.ecliptycsys.com/esimain/products.html

[GAMMA1995] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns, Addison-Wesley, Reading, MA, 1995

[GARSHOL] How the web works: HTTP and CGI explained, Lars Marius Garshol
http://www.stud.ifi.uio.no/~lmariusg/download/artikler/HTTP_tut.html

[GASKIN2000] XML Comes Of Age, James E. Gaskin ,

http://www.planetit.com/techcenters/docs/enterprise_apps/
product/PIT20000406S0006

[HOGAN] XML:The Foundation for the Future, Mike Hogan, OASIS
http://www.oasis-open.org/html/xml_foundation_future.html

[IBMXML] Introduction to XML (A Tutorial)
http://www-4.ibm.com/software/developer/education/xmlintro/xmlintro.html

[IDG1999] Be Inc. plans IPO, eyes Internet device market
http://www.idgnet.com/idgns/1999/05/07/BeIncPlansIPOEyesNet.shtml

[INSTONE] Usable Web Website
http://usableweb.com

[ITANALYSIS1999] http://www.it-analysis.com/99-11-10-3.html

[IYENGAR] Techniques for Designing High-Performance Web Sites,
Arun Iyengar, Jim Challenger, Daniel Dias, and Paul Dantzig
http://www.research.ibm.com/people/i/iyengar/ieeeic/ieeeic.html

[JIGSAW] Jigsaw – The W3C's Server
http://www.w3.org

[KACIN1999] Optimizing Web Pages for handheld devices,
PalmPower Magazine, February 1999
http://www.palmpower.com/issuesprint/issue199902/avantgotips.html

[KRISH1999] Key Differences between HTTP/1.0 and HTTP/1.1
Balachander Krishnamurthy Jeffrey C. Mogul David M. Kristol
http://www.research.att.com/~bala/papers/h0vh1.html

[LEM1999] Low End Mac,
http://lowendmac.net/musings/g4vp3.shtml

[LEVENTHAL1999a] Semantic Information Threatened by XSL Mike Leventhal
http://www.xml.com/pub/1999/05/xsl/xslconsidered_4.html

[LEVENTHAL1999b] XSL is an Ugly Difficult Language, Mike Leventhal
http://www.xml.com/pub/1999/05/xsl/xslconsidered_5.html

[LEVENTHAL1999c] XSL has Set Back the Web at least 2 Years, Mike Leventhal
http://www.xml.com/pub/1999/05/xsl/xslconsidered_6.html

[LOOKUP] Jigsaw Internal design of Jigsaw 2.0
http://www.w3.org/Jigsaw/Doc/Programmer/design.html#lookup-algo

[MACKENZIE97] http://foxglove.ccs.yorku.ca/faculty/academic/mack/GI97a.html

[MA2000] A Framework for Adaptive Content Delivery in Heterogeneous
Network Environments. Ma, Bedner, Chang, Kuchinsky, Zhang
http://www.cooltown.hp.com/papers/MMCN2000.htm

[MSFTXML] XML Tutorial
http://msdn.microsoft.com/xml/tutorial/default.asp

[ORACLE2000] http://technet.oracle.com/technetportaltogo/ptgpaper/ptgquote.htm

[PALM1995] Palm Computing. (1995, January). Suddenly Newton understands
everything you write.
Pen Computing Magazine, p.9.

[PW1999] Technology Forecast: 1999 Canada,Price Waterhouse Coopers
http://www.pwcglobal.com/extweb/ncsurvres.nsf/DocID/8701704A6347F6498525671A006DFA2E#greater

[RFC1945] rfc1945, Request for Comments: 1945,Hypertext Transfer Protocol -- HTTP/1.0
http://www.w3.org/Protocols/rfc1945/rfc1945

[RFC2068] rfc2068, Request for Comments: 2068 ,Hypertext Transfer Protocol -- HTTP/1.1
http://www.w3.org/Protocols/rfc2616/rfc2068.html

[RICHMOND2000] Introduction to XHTML, with eXamples Alan Richmond
http://wdvl.com/Authoring/Languages/XML/XHTML/

[SJMT1999] http://www.mercurycenter.com/svtech/news/indepth/april/hand041999.htm

[SUNXML] http://java.sun.com/xml/docs/tutorial/index.html

[THEVENIN1999] Adaptation and Plasticity of User Interfaces,
David Thevenin, Joelle Coutaz
http://iihm.imag.fr/thevenin/papiers/I3Workshop99/I3Workshop99.html

[USEIT] Jakob Nielsen's Website

	http://www.useit.com
[WBP2000]	Webopedia
	http://www.webopedia.com/
[WINOGRAD1999]	Towards a Human-Centred Interaction Architecture.
	http://graphics.stanford.edu/projects/iwork/papers/humcent/
[XMLORG]	XML.ORG The XML Industry Portal
	http://www.xml.org/
[YALE]	Web Style Guide
	http://INFO.MED.YALE.EDU/caim/manual/contents.html