BROWSER EVOLUTION: DOCUMENT ACCESS ON THE WORLD WIDE WEB

A Thesis Presented To

The Faculty of the

Fritz J. and Dolores H. Russ
College of Engineering and Technology

Ohio University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Dethe Elza

March, 1998

THIS THESIS ENTITLED

"BROWSER EVOLUTION: DOCUMENT ACCESS ON THE WORLD WIDE WEB"

by Dethe Elza

has been approved

for the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology

Shawn D. Ostermann
Assistant Professor of School of Electrical Engineering and Computer Science

---

Warren K. Wray, Dean
Fritz J. and Dolores H. Russ
College of Engineering and Technology

ACKNOWLEDGMENTS

I would like to thank Dr. Shawn Ostermann for teaching me to love networking technology and for helping me to survive the rigors of Academia. I also would like to thank Tom Reid and Jeffery Hirzel for providing me with such a wonderful work and learning environment. I am grateful to the other members (past and present) of the Internetworking Research Group for many interesting conversations, great ideas, and pointers in the right direction.

Most of all, I thank my lovely wife, Daniela, and our equally lovely daughter, Mina, for their vast patience and unyielding support, and for giving my life purpose and meaning. *Mnogo vi obicham, moi sekrovishta.*

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

1. INTRODUCTION

This thesis will address the problem of document access. There are many ways in which a document can be inaccessible to users. A document may be stored on media, such as a cassette tape, which the computer is no longer able to read. The document may be stored in a format of a word processor which is not installed. A document may be stored on one computer, but access may be required from another. All of these problems, and some related concerns, will be addressed in this thesis.

Word processors today are often designed to be used on documents that are only accessed by one person at a time on a single machine. This poses some problems in today's world of multiple machines, platforms, and collaborative work. Additionally, there are compatibility problems when upgrading or changing word processors or machines, where either the format of the document may no longer be supported, or the media the document lives on may no longer be supported.

Some of these problems can be addressed by building a word processor that operates over the Internet, storing its documents on a server. This thesis will explore several of the technologies and standards that exist to address to problem of doument access. The emerging Extensible Markup Language standard for document storage will be especially useful to help ease transitions by preserving document structure and content between different computer platforms, applications, and application revisions.

Word processors can be extended to work across the network, allowing documents to be worked on from a variety of sites, shared, collaborated on, and preserved across changes of platform, application, or application version. We will see how such a word processor can extend the power of the Internet and the World Wide Web by operating

within the major web browsers. Additional benefits of this approach include an integrated approach to security, the ability to easily share documents on the World Wide Web, and the ability to collaborate on shared documents. The client/server design proposed is suited to either a small application which is loaded over the Internet and run within a web browser (called an applet) or a standalone application.

1.1 Pieces of the Problem

The general problem is one of word processing documents becoming inaccessible, whether because of the user moving to another physical computer, upgrading to a new computer or word processing application, or several users needing access to the document simultaneously for purposes of data sharing or collaboration. There are several pieces to this problem, and the solution proposed in this thesis has some additional consequences which must be addressed.

1.1.1 Format Decay

When a word processing document is stored on disk in a proprietary format, the document's format must be periodically updated as the user upgrades his or her software. For instance, the user may change from Microsoft[1] Word 4.0 to Microsoft Word 5.0, necessitating that all the stored documents be updated. This process becomes more complicated as the change becomes more drastic, for instance changing from Microsoft Word 2.0 to Microsoft Word 6.0, or from Claris[2] Works to WordPerfect[3], or from a Macintosh[4] application to a Windows application. In some cases, the document must be opened in the old application and saved in a format which is common to both applications, then opened in the newer application and saved in the new

[1]Microsoft is a registered trademark of Microsoft Corporation
[2]Claris is a registered trademark of the Claris Corporation
[3]WordPerfect is a registered trademark of Corel, Inc.
[4]Macintosh is a registered trademark of Apple Computer, Inc.

proprietary format. At any point in this process, some of the formatting or content of the document may not be transferred correctly. A shared common format may not support all of the features of one or both application's proprietary formats. The newer application may not support a feature of the previous application. The translation mechanism between the two programs may inadvertently introduce errors. Additionally, it is far more complicated to transfer a document from the format of a newer version to that of an older one, for instance if a colleague uses Word 6.0 for a document and wants to share the document with you when you are using Word 4.0.

1.1.2 Media Decay

As computers are replaced with newer, faster models and some types of computers fade into obscurity, documents created on these older computers and stored in the media supported by these computers must be transferred to the new system or risk becoming inaccessible when the old machines are removed and the new ones cannot read the older media. Earlier systems used audio and video tape cartridges, 8 inch floppies, 5 1/4 inch floppies, low-density floppies, even reel-to-reel tapes. When new systems that do not support these media replace the old systems, there has often been no convenient way to move the documents to the new system This transition is also complicated by the likelihood of format decay described above. Even now, 1.44 megabyte floppies are being replaced by 100 megabyte Zip[5] disks, rewriteable optical disks, and other media.

1.1.3 Mobility and Accessibility

The user of a word processing document should be able to access that document from wherever he or she happens to be working. It is not uncommon to have a personal computer in the office, another at home, and a portable computer for travel. Keeping copies of working documents on all these machines can be difficult because

[5]Zip is a registered trademark of Iomega Corporation

often the user will not be able to foresee all of the documents he or she will require when moving between these three computing environments. Also, as the documents are revised, the various copies will diverge from one another. There needs to be a way to access the same document from many places, possibly even from many places simultaneously. Finally, the ability to access a document only from the machine it is stored on is completely insufficient for purposes of sharing documents or collaborating.

1.1.4 Security

Computers are not failsafe. Malicious users or simple mechanical or software failure can cause data to be lost. The best prevention is also one of the least practiced: make frequent backups. It is easy to automate data backup if the documents are stored in a central repository that can be accessed remotely. Security measures, such as access control, can help prevent the malicious or unauthorized used of data. The use of shared repositories for data can make these security measures much easier to administrate.

1.1.5 Style

A word processor that was designed to produce text in a given style, such as APA [Ame83], should abstract that style from the text of the document. By abstracting the style away from the text, a user could choose from many styles in a menu, and could easily change a document from one style to another. Many people are not good at creating their own styles. This may be a problem with word processors that display text as it will be printed, a technique known as WYSIWYG, or "what you see is what you get." Having a selection of standard styles to choose from would make these users' documents much more presentable. The problem with WYSIWYG is that it puts the burden on the writer for the entire design of a document, rather than just the writing, whereas a word processor that enforced a certain style could allow the writer the freedom to concentrate entirely on the writing, not even displaying the

headers, footers, page numbers, margins, etc. unless the writer specifically switched into that mode.

Additionally, many organizations like to control the style of their documents, which would be possible if documents in a system automatically conformed to a chosen style. Document styles can be chosen from a common set of templates or stylesheets. In this way styles can easily be changed throughout an organization simply by changing the templates.

1.1.6 Feature Creep

The features in a program are requested by many different people and organizations and for many different reasons. Many standard commercial applications have far more features than any given user needs. Additionally, because these features are often written into the code of the program itself, the program is larger, more complex, and slower, than it would be if it contained only the features each user needed [Bro75].

Once a program, such as a word processor, is written and released to the public, the feedback cycle begins. Problems with the program are found and fixed, and new features are requested. As a program is rereleased in successive versions, these features are often added and the program grows larger and slower. This is a common pattern. Instead of retaining efficiency without sacrificing the features people want, many software development companies rely on the continuing increase of speed and processing power of new computers and the increasing size of new hard drives to make up for the inefficiencies of their programs.

1.2 History of the Problem

Computers have been evolving rapidly since the 1950s [SG94]. As computers have grown and matured, the applications for them have become more feature-laden. Since

many applications used a proprietary format to store documents, it was difficult for the applications to exchange data, and whenever a new application was developed it would add features or change formats, eliminating the possibility of sharing data even with older versions of the same application.

Different machines had different ways of storing elementary data types, such as characters. The American Standard Code for Information Interchange (ASCII), ISO Latin 1 (ISO 8859) and IBM's Extended Binary Coded Decimal Interchange Code (EBCDIC), of which there were at least six mutually incompatible versions, are three methods for encoding simple alpha-numeric characters [TM98]. Unicode is a more recent, more inclusive attempt to encode not only Latin characters, but the characters of most modern and some historical written languages [Uni96].

There have also been attempts to standardize document encoding. Rich Text Format is a method of preserving the format of a page when transporting a document between two otherwise incompatible programs. The Standard Generalized Markup Language (SGML) is a way to define markup languages so that they are self-describing, contain separate content and structure, and can be transferred between applications in a vendor-neutral way with no loss of information or formatting.

1.3 The Solution as a Moving Target

At one time, software design was considered a linear, three-step process: Specification, Design, and Implementation. Major applications, such as web browsers and word processors, have a complex life cycle consisting of at least nine distinct phases: Assignment, Specification, Code Design, Coding, Testing, Debugging, Release, Maintenance, Revision, and Review [Oua95]. In short, a program is not developed into a final product, but is continually revised and updated, whether to fix problems, add features, or keep up with changing technologies.

A typical software application today is composed of pieces drawn from many complementary technologies and standards. As each of these technologies is being developed and defined separately, it makes sense to isolate the different technologies from each other as much as possible. By creating clearly-defined interfaces between components each component can be modified at will, as long as the interface is maintained [GHJV95, Boo94, Mey94].

Given that software is composed of interlocking and separately updated standards, and that creating well-define interfaces between them is a good thing, where does this lead? If we can update underlying components of a program without changing the other components, we have software that can continuously evolve, rather than be discontinuously updated. Further, the components may be used for more than one application, and all of the applications which use that component will maintain the same level of functionality and inter-application compatibility if they are based on the same external, updatable component.

2. STANDARDS AND TECHNOLOGIES

Just as there are many parts to the problem, there are many parts to the solution working together to create a system which addresses the overall problem of document access. Recent standards and technologies have created open, non-proprietary, solutions to these classic problems. In this thesis descriptions of these standards and technologies are grouped into two loose categories, Programming for the Network, and Document Formats. In the following section, the pieces discussed here will be pulled together into a proposed solution. The work done to create useful programs which address the problem of document access will also be described.

2.1 Programming for the Network

The popularity of the World Wide Web has demonstrated the usefulness of client-server applications on a wide scale. By providing the formerly arcane Internet with an easy and intuitive user interface, the role of client-server technology has expanded dramatically. The new popularity of the Internet has increased accessibility to the network, with home access now common, access while travelling becoming more common, and business access a necessity.

2.1.1 Transmission Control Protocol/Internet Protocol

The Transmission Control Protocol and Internet Protocol suite (TCP/IP) is a collection of standards that describe a way to transmit and receive data. Most of the rest of the Internet is built on top of TCP/IP [Com95]. TCP/IP is a set of related protocols and standards whose importance lies in its success as an open system, as

opposed to proprietary systems as AppleTalk and NetWare. Because TCP/IP is an open standard, anyone can implement it for any system without license fees, and all necessary information for implementing TCP/IP is publicly available. By accepting TCP/IP as the common denominator, any system in the world can interconnect with any other system over the Internet.

2.1.2 Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) standard is the foundation of the World Wide Web. HTTP is a protocol that allows web clients to communicate with web servers when requesting and being served web pages, where a web page is the basic unit of the World Wide Web. By creating HTTP as an open standard, its designers ensured that it would be relatively easy to create these web clients and servers, and any type of client could connect to any type of server. The extensibility of HTTP also allows these clients and servers to be easily customized and extended for particular specialized uses as will be discussed below in the description of Java.

2.1.2.1 Common Gateway Interface

The Common Gateway Interface (CGI) is a part of an HTTP server that can communicate with other programs running on the server [Gun96]. The Common Gateway Interface greatly expands what web pages can do by allowing a web page to interact with programs, display information from programs, perform calculations, access databases, etc. The Common Gateway Interface is relatively easy to use, but it is also easy to inadvertently open up huge security holes on the server by accessing programs through the Common Gateway Interface. Using the Common Gateway Interface also can put a tremendous load on the web server, causing access to be very slow if a web server is popular and serves thousands of hits per day [Gre97].

2.1.2.2 Server Side Includes

Another method to extend the information that web pages can access is through the use of Server Side Includes. Server Side Includes are similar to the Common Gateway Interface, but are part of the HTPP server program which provide communication with the server's operating system and files. Unlike the Common Gateway Interface, Server Side Includes are not usually used to access programs, but files, information about files, and other information such as the current time and date. A Server Side Include could insert the current time in a web page, or a file's size or last modification date. An important use of Server Side Includes is the ability to include external files within the web page, so that a template can be built that includes all the information that is common to multiple pages on a site and files that contain only what is unique to the page can be created. Creating templates like this can greatly ease the effort involved in maintaining a large site, with thousands of individual web pages, since changes which need to be made throughout the site can be made to the template files and will then effect every page that uses the template.

2.1.3 Remote Procedure Call

Various methods have been devised to allow computer programs to have their operations distributed across the network, which I am grouping together under the term Remote Procedure Call, of which Sun RPC [Mic88], from Sun Microsystems, is an example. For ease of understanding and maintenance, computer programs are broken into smaller pieces. One way to do this is to create functions, each of which may take some arguments, perform operations, and return data as a result. Using Remote Procedure Calls a program has a well-defined method to call functions which are not necessarily on the local machine, but instead can be located and run anywhere on the network, allowing a program to spread its operations to many systems for better performance, better access to distributed data, or other purposes. The abstraction of

the Remote Procedure Call allows programmers to use a mechanism (the function call) that they understand well and are familiar with to create network-enabled programs (which are inherently more complicated).

2.1.4 Common Object Request Broker Architecture

Building on the ideas that began with Remote Procedure Calls, the Common Object Request Broker Architecture (CORBA) extends the idea of network programming from function-based programming to object-oriented programming. CORBA handles all of the behind-the-scenes details of finding servers and services, translating data between systems with different representations of that data, and other common tasks that programmers have had to deal with in networked applications, thus removing of the difficulty and the sources of common bugs. CORBA allows programs written in C and Pascal to seamlessly integrate with programs written in Java or Cobol. CORBA also allows programs running under Unix or Apple's System 7 to easily communicate with programs running under Windows or VMS. CORBA transparently translates data formats so that programs running on DEC Alpha chips to exchange data with programs running on Intel Pentium[1] or IBM/Motorola PowerPC[2] chips. Finally, CORBA provides a means for new network programs to access data from old legacy databases in a standard and extensible manner.

2.1.5 OpenDoc

The OpenDoc Consortium attempted to address many of the issues discussed in this thesis. OpenDoc was an attempt to create a component technology that would allow users to easily build their own custom applications with only the features they needed, paying only for the specific functionality they required, and updating only the portions of programs that benefitted their needs. Further, OpenDoc was designed to

[1]Intel is a registered trademark and Pentium is a trademark of Intel Corporation
[2]PowerPC is a trademark of IBM

work by downloading the components over the Internet, working well with CORBA's integrated model of network services [OHE96].

But OpenDoc was subject to both overambition and fierce competition. Its designers attempted to put too many features into the OpenDoc framework, causing it to suffer from the same feature creep it was designed to relieve, and Microsoft proposed a simpler solution called Object Linking and Embedding (OLE). Object Linking and Embedding was far less powerful than OpenDoc was intended to be, but OLE shipped to customers years before OpenDoc did, and was well established in the marketplace by the time OpenDoc was finally released.

2.1.6 Java

Java has been the subject of a great deal of publicity for a programming language. Java's promise of "Write once, run anywhere" sounds like a programmer's Holy Grail, and in many ways it is. On the other hand, Java is simply an interpreted language, like many scripting languages (Perl, for instance) which is nothing new, except that Java has the advantages of machine speed and ubiquity over earlier interpreted languages. Machines are fast enough now that the speed penalties of an interpreted language are not as severe as they once were. Java is being built into the major web browsers from Netscape and Microsoft, and both browsers or standalone Java runtime engines may be downloaded for free, allowing Java to proliferate across networked computers.

Java is an object-oriented language, which reduces the effort of a programming team when developing or maintaining applications, especially applications with graphic user interfaces. Standard items, such as windowed displays with scroll bars, can be derived from existing objects and inherit common default behavior from these parent objects. Since much of traditional programming consists of reimplementing common features, the use of object-oriented programming helps programmers a great deal by providing these common features in object libraries and application frameworks [Zuk97].

Java integrates well into the CORBA model of network services [OH97], as well as providing its own lightweight networking model, Remote Method Invocation, a Java-specific form of Remote Procedure Call. Because Remote Method Invocation and Java's other networking models (including low-level TCP/IP functionality) are very easy to integrate into a program, network applications can be prototyped and experimented with, leading to new classes of applications [Har97].

As an example, a very simple web server can be created in Java in an afternoon, then readily extended to serve new datatypes [Har97]. Java's object model and libraries allow the programmer to easily integrate very advanced features in terms of security and services [Fla97b]. Further, Java applications can be shipped to the user automatically via web pages as applets: safe, simple, and usually small programs designed to provide unique features for a web page or serve as a fully-functional program.

Java has other features that benefit the programmer and the end-user. Java's security model protects the user's machine from arbitrary code downloaded across the Internet [Har97, Gre97]. Java's built-in memory management system helps to eliminate many of the most common sources of programming errors by automatically freeing unused memory and not allowing unused memory to be accessed by accident [GK97, Gra97].

Additionally, Java is both a programming language and a platform (in the sense that Windows and Macintosh are platforms), so transferring data between a Java program running under Windows and one running on the Macintosh should be trivial, since both programs are really running under Java [Fla97a]. Also, Java integrates many of the elements discussed in this previous section, including easy handling of TCP/IP and HTTP, and Remote Method Invocation. Java can also be used with CORBA [OH97]. Java uses Unicode internally, and has other features which allow Java programs to be customized for various countries and languages [Fla97b].

2.1.6.1 Java Beans

The idea of component-oriented software has been around for a while, and OpenDoc, discussed previously in section 2.1.5, was one of the most ambitious attempts at component-oriented software development. Java Beans is a more modest attempt at a component-oriented software architecture. Java Beans is a standard method of creating components that a programmer can easily use to build an entire application [GK97, Fla97b]. This still involves some programming, albeit at a very high level, whereas OpenDoc was intended for non-programmers. Java Beans allow the application programmer to utilize well-tested components to build applications and well-designed programs to only load the components the user needs, in order to reduce feature creep.

2.1.6.2 Remote Method Invocation

Remote Method Invocation is the Java-native form of Remote Procedure Calls [Har97] and a lightweight alternative to CORBA, with less overhead because it is specific to Java and doesn't have to deal with as many details of translation between different machines, operating systems, and languages as CORBA handles. Also, Remote Method Invocation does not include most of the advanced forms of CORBA networking such as transaction processing. These limitations make Remote Method Invocation very easy to use and much more readily available than CORBA, because not everyone who has Java installed on their machine has CORBA services accessible [OH97].

2.1.7 Scripting Languages

The scripting language formerly known as JavaScript by Netscape and JScript by Microsoft, has recently been accepted by the European Computer Manufacturers Association and the standardized form of JavaScript is now known as ECMAScript.

ECMAScript is an important scripting language because, like Java, it is built into the web browsers from both Netscape and Microsoft, allowing elements of a web page to be manipulated programmatically [Fla97c]. As the Document Object Model becomes a standard within the World Wide Web Consortium, web pages and other document types which support the Document Object Model will be exposed to all scripting languages, allowing people to create custom applications and tools using Perl, Python, AppleScript, or whatever their favorite scripting tool is. Currently, Netscape and Microsoft have incompatible versions of the Document Object Model, and both are exposed only to built-in scripting languages (ECMAScript on both browsers and VBScript on Microsoft's browser). When the standard for the Document Object Model is completed by the World Wide Web Consortium, and supported by browsers, the popularity of scripting languages will take a huge leap forward.

2.2 Document Formats

Tools to create network applications have been evolving to seamlessly integrate data from multiple sources on the Internet. In parallel with this evolution, the formats that the data is stored in have also been evolving. Programs which store their documents in proprietary formats are slowly being replaced by or upgraded to programs which store their documents in formats defined by open standards.

2.2.1 LaTeX

LaTeX is a typesetting system for formatting technical documents for printing. LaTeX is available for nearly any computer system in use and provides excellent cross-platform capabilities. LaTeX is able to create highly stylized documents from plain text and simple commands, but unlike a WYSIWYG system, it cannot be learned by simply running the program and typing your document (although there is a WYSIWYG editing system which uses LaTeX). LaTeX is a document processing

system [Lam94] that resembles a programming language more than it resembles a word processor. LaTeX is extremely powerful, but the document files used as input to LaTeX are not readily legible unless the reader is familiar with LaTeX. LaTeX files are typically created in a text editor, inserting formatting commands by hand or through a customized program called a macro. The files are then run through one or more post-processing programs to produce the result.

2.2.2 Rich Text Format

An early attempt to create an intermediate format between various proprietary formats, Rich Text Format is widely supported in commercial word processors. A document created in the proprietary format of one word processor can be opened in that application and saved in Rich Text Format, then opened in another word processor and saved in that program's native, proprietary format with most formatting intact. This is an improvement over raw, ASCII text which allowed the textual content of a document, stripped of all formatting, to be transferred between programs. Rich Text Format still does not provide an adequate solution, however. Also, Rich Text Format is not a native format for any program. Because documents are not saved as Rich Text Format by default, transitioning a document from one format to another can be a tedious process. Because Rich Text Format files are not human-readable, problems that arise in the translation process are harder to correct. Problems arise because Rich Text Format does not support many of the formatting commands used by modern word processors, or supports them in a slightly different way.

Finally, Rich Text Format is not extensible. Word processing programs cannot encode their own specific, non-standard features into Rich Text Format files to be preserved, even if not acted upon, when transitioning a document from one program to another. This means that if a file is converted to Rich Text Format and then read in by the same program, some information may be lost.

2.2.3 Standard Generalized Markup Language

Standard Generalized Markup Language (SGML) was first published as ISO 8879 in 1986 [CKR97]. SGML grew out of two separate efforts in the 1960s to create structured documents that could be exchanged and manipulated. One was Graphic Communications Association's GenCode and the other was IBM's Generalized Markup Language [KR98]. SGML provides an interchange language for text processing that has found application in disciplines ranging from chemistry to music, from publishing to national defence.

SGML is an important standard because it bring three concepts to document storage: Reuse, Portability, and Interchange. SGML encourages reuse because you can create templates and fragments which can be used in multiple documents. SGML is portable because it can be stored in, and read from, a wide variety of character encodings. Any SGML parser can parse a given SGML file, so interchange of SGML files between different programs easily without format conversion in the process.

SGML separates content data and structure so that each can be manipulated independently. A Document Type Definition defines the relationship between the data and the structure of that data. Once an SGML document is parsed into an application, it takes on a database-like structure, that can then be searched, printed, or modified by programs or scripts [vH94]. While SGML offers a rich and extensible format for storing and exchanging data, it is extremely complex. By trying to serve all of the publishing and data storage needs of a vast array of industries and sectors, including the Department of Defense, SGML has included a number of little-used, but powerful features which are difficult to learn, use, and/or implement. The difficulty of SGML has hindered wide-spread adoption of SGML as a data format.

2.2.4 Hypertext Markup Language

HyperText Markup Language (HTML) is a well known SGML document type, because HTML is the language of the World Wide Web [MK97]. Unfortunately, HTML violates some of the SGML tenets by including tags to manipulate presentation, rather than layout (for instance bold, italic, and blink). SGML documents should maintain strict separation between content data and presentation for several reasons, including the preservation of meta-data, information about the content. For instance, by keeping all of my headings in a document in the HTML `<Hn>` tags I can write a program to compile a table of contents by searching for heading tags, and this program will be useful for anyone who uses these tags. But if I use `<H1>` for a first-level heading and you use `<font size="+2" weight="bold">` for the same thing, then the system breaks down very quickly. This problem is being addressed in the HTML world by the Cascading Style Sheets standard [LB96], which gives web page designers control over presentation of the standard tags without having to modify the content data. Unfortunately, support for Cascading Style Sheets in the leading browsers remains inconsistent at this time.

Because of the growth and usefulness of the World Wide Web, there has been a great deal of interest in converting legacy documents into web pages, but HTML has many limitations. Unlike SGML (which is fairly complex), HTML is relatively easy to learn and documents can be converted quickly. Maintaining a large group of HTML documents is a formidable task, however, and the presentation of data in HTML still leaves a lot to be desired. A common problem is the creation of headline titles outside the narrow range of font modifications allowed in HTML. Many sites bypass the HTML font problem by creating graphics for their title, giving them more control over the display. However, these titles are no longer able to be indexed by search engines such as AltaVista and Yahoo. Also, such graphics often print poorly and do not show up in the page at all if the user has graphics turned off in the browser. HTML has no concept of custom tags, which would be useful in, for instance, an

inventory document, where you could mark items with <PARTNO> and <COST> tags to track the meaning (metadata) of the fields. This is just not possible in HTML. "In short, while HTML provides rich facilities for display, it does not provide any standards-based way to manage [a document] as data" [PSL+97, page 18].

2.2.5 Dynamic Hypertext Markup Language

A great deal of attention has been paid to Dynamic HTML (DHTML). DHTML is not really a language or a specification, but a group of interrelated standards, including HTML, Cascading Style Sheets, the Document Object Model and ECMAScript [Dav97]. By combining these technologies, documents can be created that can actively change within the browser, without reloading a new page from the server.

2.2.6 Extensible Markup Language

A word processing document has at least three elements. A document has content, the actual text the end-user will read. A document also has a structure, implied or implicit, that gives context to the text, such as who wrote the text, when it was written, how many pages it is, etc. A document also has a presentation, which is how it is laid out in terms of visual formatting. The Extensible Markup Language (XML) allows these three elements to be abstracted from each other, so that one element can be changed independently of the other elements.

This abstraction allows, for instance, a table of order numbers and prices to be viewed as a graph by changing the presentation of the document. Because XML includes explicit structure with tags surrounding and defining text, a computer can process the content more intelligently, for instance a program might know that anything within <Price> tags is a price that should be compared with prices of similar items for automated comparison shopping. Once a structure and presentation are created, the content can be changed to suit the user: a table of prices and descriptions can be used for computer peripherals or fruits and vegetables.

XML is a subset of SGML designed to replace HTML as the language of the World Wide Web [Lan97]. Like SGML, XML is a meta-language in that it allows users to create their own language tags, but it is vastly simplified over SGML and optimized for transfer over the Internet. In many ways, XML simply codifies the way people were already using SGML, using a subset of the standard to create self-describing documents that could be manipulated and transported easily [Den97]. As John Bosak [Bos97, page 220] writes,

> XML differs from HTML in three major respects:
>
> 1. Information providers can define new tag and attribute names at will
> 2. Document structures can be nested to any level of complexity
> 3. Any XML document can contain an optional description of its grammar for use by applications that need to perform structural validation.

With XML, the document type definition is optional, allowing languages to be declared on the fly, the only requirement being that documents are well-formed.

Well-formedness allows XML documents to be extremely flexible, since tags can be used without defining them first, but it has two requirements. First, elements must be properly nested within each other with no overlap (strictly hierarchical). Second, each element must be paired with a closing element after the content of the element. Elements which contain no content (like the IMG tag in HTML) may be their own closing element by marking the element with a trailing slash. Well-formedness is a radical departure from HTML, because HTML allows many closing tags to be optional and implementations of HTML have permitted elements to overlap.

Some examples may help clarify this point. An example of XML opening and closing tags would be: `<MyTag>MyContent</MyTag>`. An example of XML element with no content: `<MyContentFreeTag/>`. Notice that the trailing slash in the content-free element signals the parse engine not to look for a closing tag. This allows arbitrary

XML documents to be parsed without an explicit Document Type Definition to define the language.

XML allows documents to be self-describing in that their content can be labelled in such a way that the structure of the data is preserved. There are several related standards that interlock with XML that are still being worked out: Extensible Link Language [BD97], Extensible Style Language [ABC+97], and the Document Object Model [WHW+97]. The Extensible Link Language standard provides a vastly richer linking model than HTML, including bidirectional links, one-to-many links, transclusion links (enabling content from several documents to be included in one "virtual" document), and links that are preserved when documents are moved. The Extensible Style Sheet (XSL) standard corresponds to the presentation element of a document. XSL will probably be based on SGML style sheets, Document Style Semantics and Specification Language [Int96], but will also support the current HTML cascading style sheets [CLM97, LB96]. The Document Object Model standard will provide a unified way to approach the XML parse tree with any scripting language.

According to the specification of XML [BPSM97, pages 29–30],

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

9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

2.2.7 Document Object Model

Most of the document formats discussed have been methods of encoding documents in files, but the Document Object Model is a way to expose the document while it is actively being edited or displayed by a program. The reason for exposing the elements of a document is to allow scripts and other programs to access them. In HTML the Document Object Model allows elements of the document to be positioned during viewing, which creates simple animation effects with a minimum of bandwidth.

Every program has a method of maintaining the state of a document while the program is manipulating it. On some systems the document can be manipulated by script commands. For instance, on the Macintosh, which defines an Open Scripting Architecture, most programs support AppleScript to a greater or lesser degree and allow documents to be manipulated by AppleScript or other Open Scripting Architecture-compatible languages [Goo94]. In the web browsers from Microsoft and Netscape the Document Object Model has only been open to scripting languages built into the browser: ECMAScript in browsers from both companies and VBScript in the Microsoft browsers. Browsers from both companies have conflicting Document Object Models, which create a lot of problems for web page designers who want to create scripts which will work with all browsers.

The World Wide Web Consortium is working on standardizing the Document Object Model for XML and HTML so that the model will be standard on all platforms and accessible from all scripting and programming languages [WHW+97].

3. EXAMPLES OF WEB-BASED APPLICATIONS

This chapter discusses two prototype applications developed as part of this thesis to test the ideas addressed in this thesis. The first, Interactive Hotlist, is a modest program using a Common Gateway Interface and some templates to create and maintain an outline-based hotlist of web links on the World Wide Web, rather than on an individual machine. The second, FreeWord: An Extensible Outline Tool, is a more ambitious program, designed to evolve into a full-fledged web based word processor, which addresses most of the topics discussed in this thesis.

3.1 Interactive Hotlist

The Interactive Hotlist grew out of frustration with the way the Bookmark feature was implemented in Netscape Navigator. Favorite links can be shared on a web page for other people or accessed from another machine, but the Bookmark command would only store links locally, forcing manual editing of the HTML to update the links. What was needed was a way to add links from within the hotlist web page itself, so a page could be "bookmarked" at work and that link could be accessed from home or in the field, working on someone else's computer. This system has been used for a year now and works well, although there are some weaknesses which will be discussed below.

3.1.1 Design

The design of the Interactive Hotlist is simple. The hotlist is to be a hierarchical list, with subjects which can contain other subjects, links, or both (Figure 3.1). The

hotlist needs to be able to run on any web browser for maximum access, and be very easy to use, providing essentially the same functionality as the Netscape Navigator "Bookmark" command in a network-accessible way.

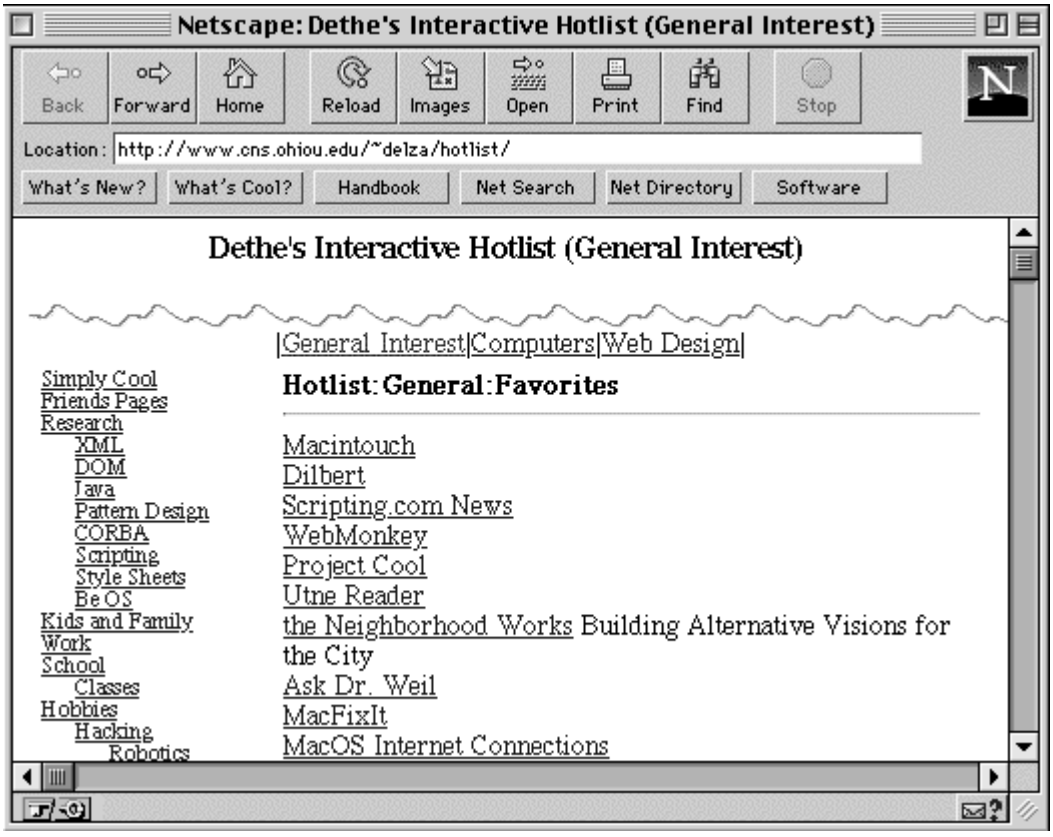

Figure 3.1 Interactive Hotlist at Top of Page

While simple, this program addresses the problem of document access quite well. It is not a general solution to the problems raised in this thesis, since the program is designed to fulfill a very narrow task, but it provides a good testbed for some of the issues.

3.1.1.1 Accessibility

Accessibility is the primary focus of this thesis, and it is the primary motivation behind the Interactive Hotlist. People who want or need to make their hotlists more readily accessible have not had a tool like the Interactive Hotlist, which solves the access problem very well.

3.1.1.2 Security

It can be very tricky to do Common Gateway Interface programming without opening up major security holes. The design encompassed three approaches to security:

1. The Common Gateway Interface script strips out Unix metacharacters that could conceivably be used to write destructive data to the system;

2. Data can only be appended to existing files, new files cannot be created through the user interface; and

3. New additions to the hotlist are e-mailed to the owner of the list. This final check is to ensure not only the security, but also to make sure users do not load my list with their own advertisements or other undesirable links.

3.1.1.3 Maintenance

Maintenance is a weak point in this design. Because of the limitations of the system chosen (Common Gateway Interface and templates using Server Side Includes) it would be very difficulty to create a system that is truly maintainable, that is, which allows links to be edited, moved and deleted, and allows new categories to be created, renamed, moved and deleted. The only maintenance allowed through the user interface is to add new links (Figure 3.2), all other maintenance happens by logging into a Unix account and working on the command line. This has proven

to be sufficient, however, because the program has such a narrow focus that further maintenance is rarely necessary.

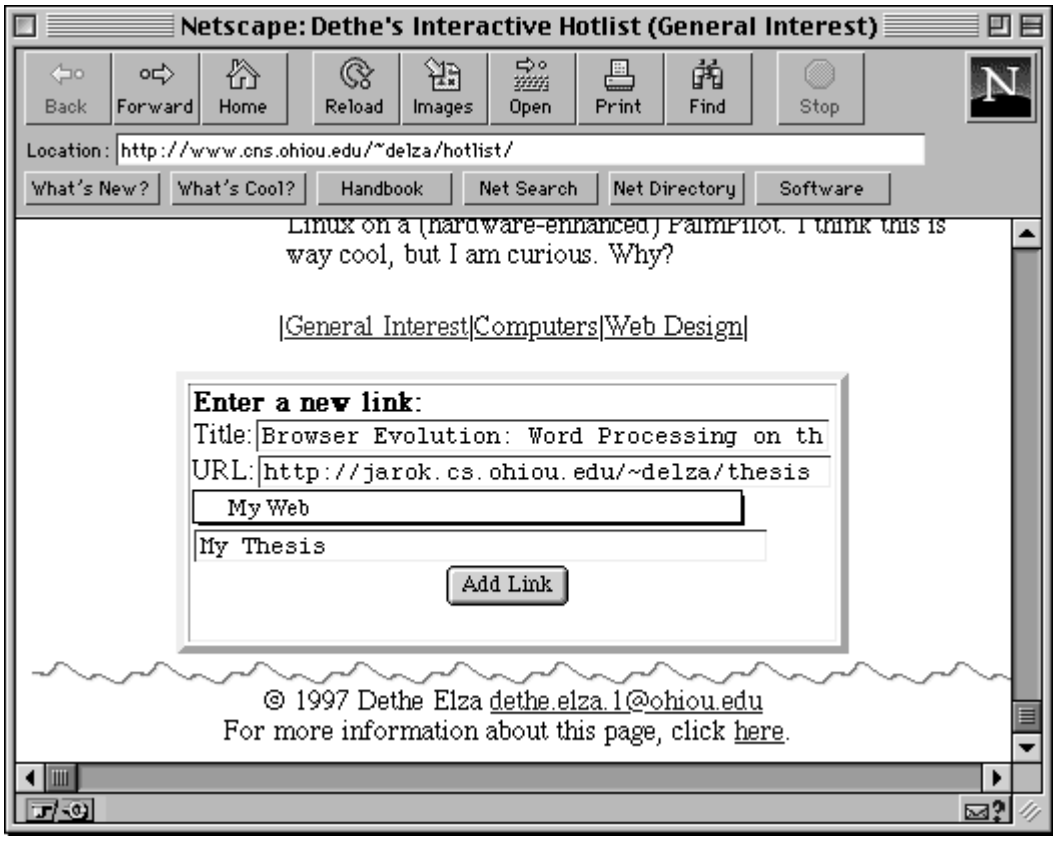

Figure 3.2 Interactive Hotlist at Bottom of Page

3.1.2 Implementation

The implementation of the Interactive Hotlist was fairly simple (A). The hotlist web pages use templates for the format of a given page using Server Side Includes, and each page is composed of the template and its included files. The template includes on each page an outline list of all links to all related pages in the Hotlist, a list of

links which can be modified from within the page, and a form which allows new links to be specified. The form sends its data to a script, written in the awk scripting language [Dou90], which checks for required information, updates the link list, and sends a message to the user to confirm that the update was successful or request more information if a required field is blank.

The simplicity of its design gives the Interactive Hotlist several advantages. Universal access is the first, the program can be accessed from any browser on any platform, requiring no script or program to run on the client machine, and working adequately in text mode. Another is that the program is easy to understand and maintain. Finally, it does its one task reasonably well.

There are also several disadvantages to the design of the Interactive Hotlist. The inability to manage lists is a problem, and creating a solution to it without introducing new security risks would be challenging. Also, there are many files involved, which can be confusing when doing maintenance and generally does not provide a clean implementation. Even though security issues have been addressed, it is not clear that the system is entirely secure;, the Common Gateway Interface script could provide an unknown security breach. The format of the web pages is fixed and inflexible. The interface for adding items to the list is integrated into the display of the list, taking considerable room on the page, which would be unacceptable if further maintenance functionality were added. The outline portion of the page, which lists related pages, is not a functional outline, that is, it cannot be expanded or collapsed to focus on detail. Because the outline cannot be collapsed, the entire hotlist is divided into three relatively independent sections, each with its own outline so none of the outlines are too long. The division into sections is fairly arbitrary. Finally, pages must be reloaded when switching subject or adding links.

3.2 FreeWord: An Extensible Outline Tool

The second application developed for this thesis was FreeWord. FreeWord addresses the deficiencies of the Interactive Hotlist, but with features that are more extensible. FreeWord is intended to be a simple outliner, but one that encapsulated its functionality well enough to be extended easily to provide other functions and styles. An outliner is used for this proof of concept project because it is simple enough to tackle yet still powerful enough to be useful, and because there are some excellent examples of outliners to use as models, especially Symantec's More 3.0 [Sym90] and Userland's Frontier 5.0 [Win98].

3.2.1 Design

A document must be maintainable with simple menu commands, just as in a regular word processor (Figure 3.3). Editing changes should be immediately visible in the document, including outline collapse and expand (Figure 3.4, Figure 3.5). An outline should be capable of being zoomed into, so that a sub-outline can be viewed as a stand-alone outline. All of the functionality of the Interactive Hotlist should be maintained, but several issues need to be addressed further: Concurrency, Cache Coherency, and Security.

3.2.1.1 Concurrency

The ability to share documents is useful, and the ability to collaborate on documents is even better than simple sharing. The concurrency model for FreeWord allows users to simultaneously work on different parts of the same outline, down to the level of a single item in the outline, and allows users to simultaneously add items to the outline. However, since the outline can be worked on while off-line, it is not possible for multiple users to modify the same item of the outline simultaneously. Attempting to do so will raise a dialog box asking the user which version to save, and allowing

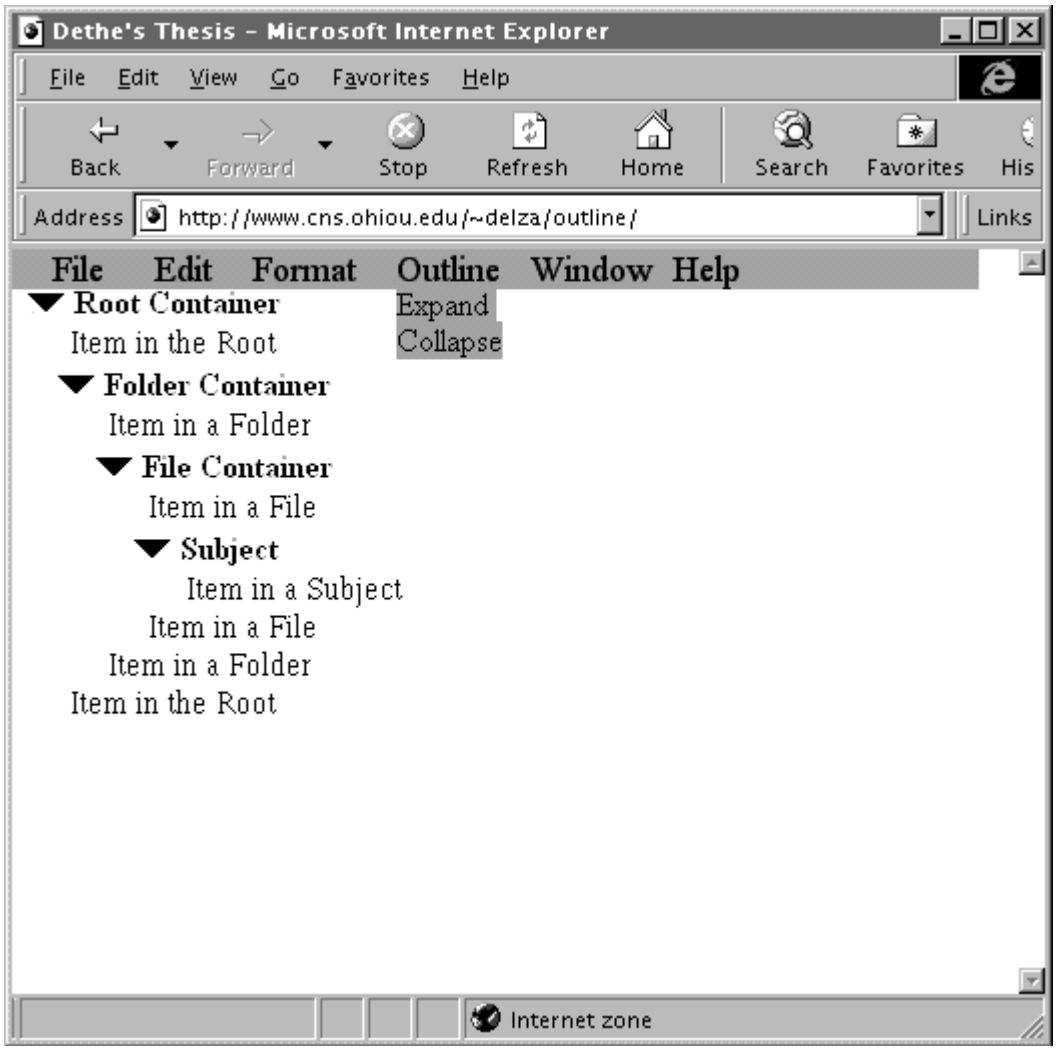

Figure 3.3 FreeWord Menu Example

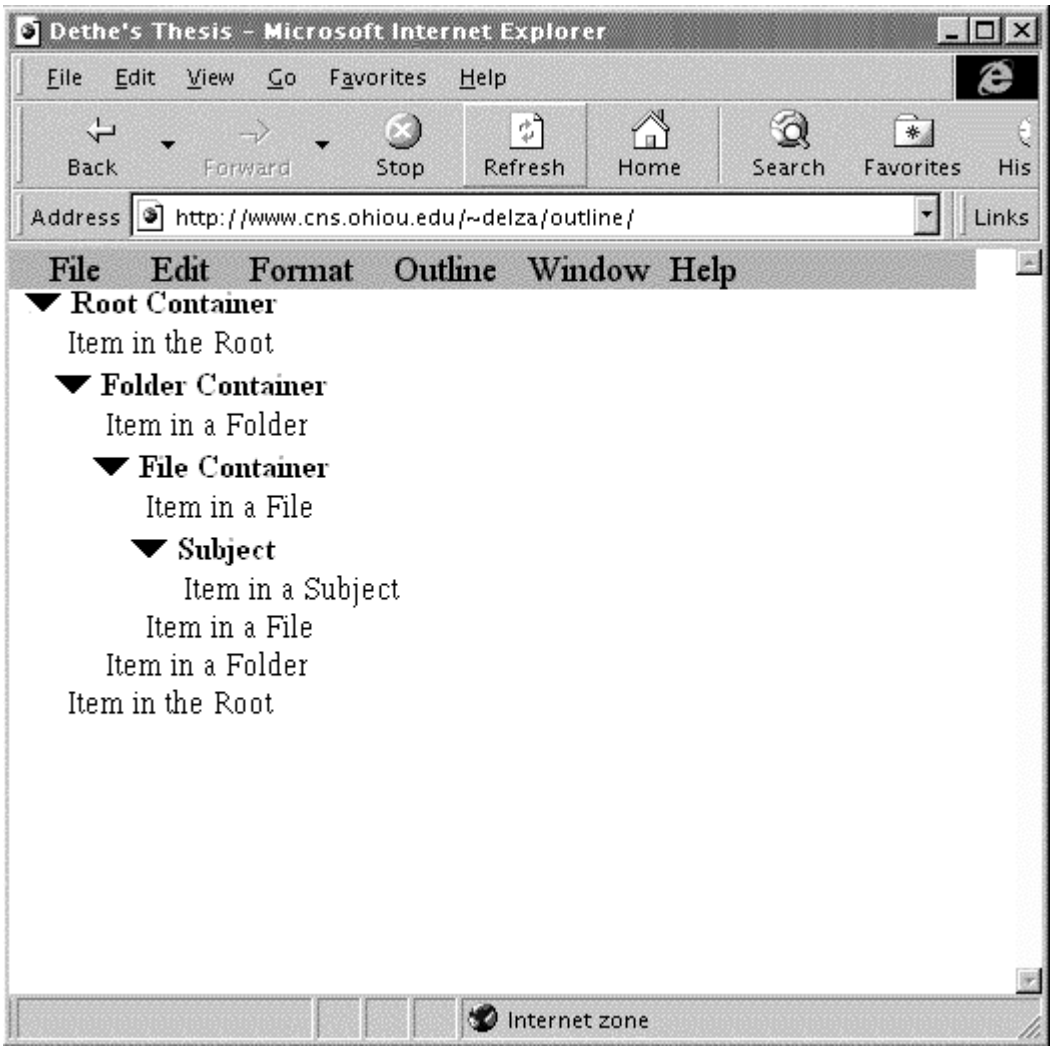

Figure 3.4 FreeWord Outline Fully Expanded

Figure 3.5 FreeWord Outline Partially Collapsed

both versions to be compared and modified. All concurrency checks are done when the files are saved back to the server.

3.2.1.2 Cache Coherency

There are several issues involved in keeping documents up to date. A user can be working on a document on their local machine, make changes, and save them back to the server. Meanwhile, another user may be working on the same document and have saved changes earlier, causing the two versions of the document to be out of synchronization with respect to each other. Since the level of change is the individual entry (a point in the outline which contains no children), different entries in a document can be modified without conflict, it is only when the same entry is modified by multiple users that a problem arises. This is dealt with in several ways.

First, if a user is on-line, he or she can request to be notified of changes to a document to keep his or her version up to date. Second, each entry of a document contains the date and time of its last modification. When a document is saved back to the server, the modification date for each entry (from when it was opened) is compared with the modification dates on the server–if the server dates are newer there is a conflict. The user is then prompted to resolve the discrepency in one of the following ways:

1. Save the latest version;
2. Keep the older version; or
3. View both versions and edit one to merge the changes into a single document.

3.2.1.3 Security

Security issues in FreeWord are much more complex than in the Interactive Hotlist. When FreeWord is installed, some decisions must be made. Access to FreeWord is made on the basis of a username and password, but this can be set up in several ways.

Usernames may be allocated by a system administrator, or by allowing each user to provide a username and password when they first use the system. The latter system would only be workable on a secure intranet, so that the list of users is somewhat controlled without requiring a great deal of maintenance. Also, read-only access to a file can be provided without a username or password, giving the user access to only a few commands (outline expand and collapse, and edit, which allows them to log into edit mode and make changes). In this way, documents may be provided to the general public for viewing, but can only be maintained by those with valid accounts. Each sub-outline may maintain its own access list of owners and permitted editors and viewers, and this list is maintained by the server and not accessible to clients.

3.2.2 Implementation

The implementation of FreeWord (B) uses XML as the storage format, since XML provides many of the desired attributes of the program, it is extensible and tools exist to parse it. Since no general-purpose XML browsers/editors exist, one was simulated by converting the XML into HTML to display, and using ECMAScript to create editing functions. The differences between the Microsoft and Netscape implementations of the Document Object Model (DOM) limit this program to running under Microsoft Internet Explorer 4.0 on Windows 95.

Outlines in FreeWord can be expanded and collapsed by double-clicking the triangle controls (Figure 3.4, Figure 3.5). Outline items can be edited by double-clicking on them (Figure 3.6). The design can easily be extended. The document format is self-describing. Potentially, the ECMAScript functions could be included in the document as well. The design is very flexible. Security uses a sandbox model–there is really one big outline which is viewed as a filesystem, all documents are really just sub-outlines, so only one file is accessed by the server and no new files can be written to the system. Most commands in FreeWord are menu driven, with some keyboard and mouse shortcuts. Unlike earlier outline-based web pages, reloading is unnecessary

when modifying the outline. FreeWord outlines can be nested indefinitely, making the program powerful enough for useful work to be done (Figure 3.7).

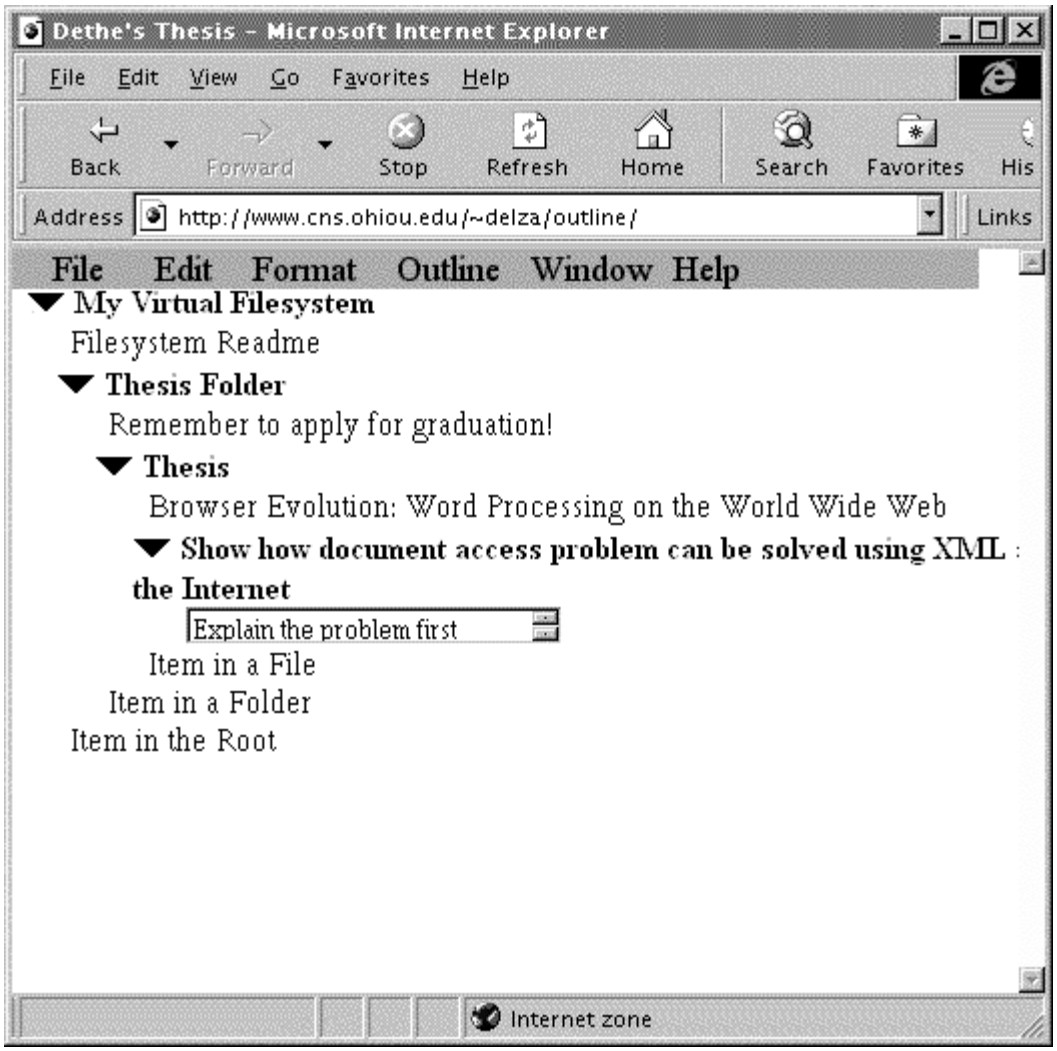

Figure 3.6 FreeWord Editing an Outline Item

Currently FreeWord works only in Internet Explorer 4.0 on Windows 95. The program must translate between XML and HTML, requiring two different Document Object Models. Because the files are stored in XML and displayed using HTML, they

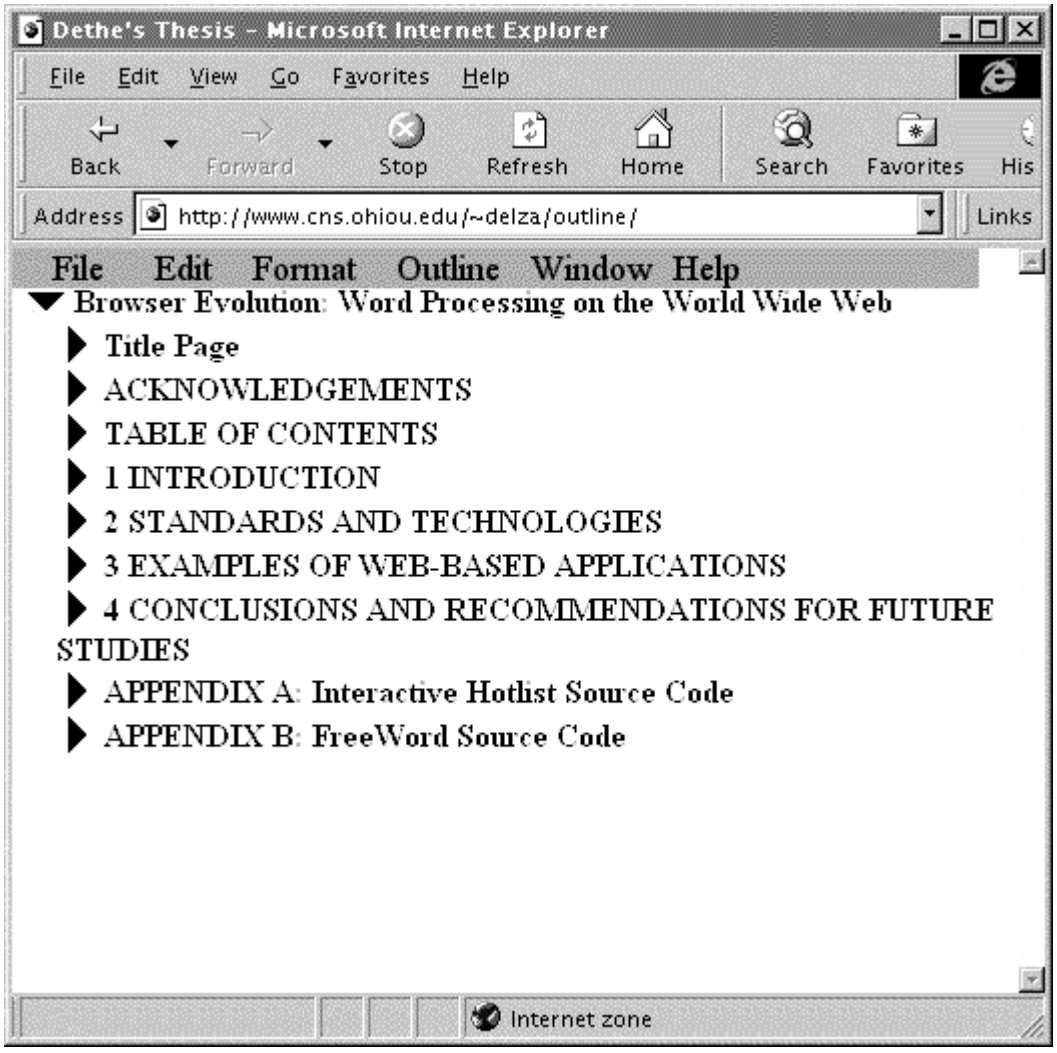

Figure 3.7 FreeWord Thesis Outline

must be parsed back and forth more than once, the exact number depends on the implementation used to save files. Because Java has been intentionally crippled by Microsoft in Internet Explorer, Remote Method Invocation cannot be used to pass data back and forth. Having two Document Object Models requires two different identifiers to be used for each subtree: a 32-bit integer is used to identify each element and its parent, for replacing the element properly in the tree structure, and a path-based string identifier, for opening a particular level of the outline tree with URL syntax.

4. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE STUDIES

For the goal of access to documents, both the Interactive Hotlist and FreeWord are steps in the right direction. The Interactive Hotlist is a useable program with a very specific function. FreeWord is a more ambitious project, but currently has very limited capabilities and the ability of the program to evolve into a real word processor has not been proven. A full-fledged XML browser would go a long way towards fulfilling the needs discussed in this thesis. Since most, if not all, of the interlocking technologies involved are changing, creating well-defined interfaces between them is highly desirable and would allow the various pieces to change separately and the browser application to evolve continuously rather than experience discontinuous upgrade leaps.

Both OpenDoc and Microsoft's Component Object Model (COM) are attempts to create a working document-centered component technology. The current trend of XML-based technologies appears offer a better solution because they are simpler, configurable by users (not just programmers), flexible, and independent of programming language or platform. The combination of these technologies may allow document-oriented and component-based programs to evolve towards the goals of OpenDoc, without the bureaucracy that project involved.

One good place for further study would be in different models for transmitting the XML data to the browser and displaying that data. The model used for FreeWord uses HTML for display and XML for data storage. Because both of these languages have a different Document Object Model, one of the concerns in FreeWord is keeping reparsing to a minimum. Obviously, a pure XML browser would be very beneficial here.

An XML document forms a hierarchical database in linear form. Using object databases as opposed to potentially massive files may make using XML more efficient and scalable. Object and relational databases are already helping to create and maintain large websites of HTML code [Win98, Gre97]. Combining databases with scripting languages it will be possible to create very flexible and efficient next-generation XML servers.

This research has found that an XML-based browser/editor can offer a feasible solution to the problem of document access. In the course of the research it was also found that many of the neccessary standards have not been fully implemented yet by the major vendors. The lack of a standardized Document Object Model was a particular hinderance in developing FreeWord. On the other hand, the original plan was to use HTML with custom extensions as an underlying format for FreeWord, but while researching for the design the newly proposed XML was discovered. While designing and building FreeWord, XML went through three major revisions, becoming an accepted standard of the World Wide Web Consortium during final editing of this thesis. XML now appears to be a much better and more long-term approach to the underlying format of a document than HTML.

BIBLIOGRAPHY

BIBLIOGRAPHY

[ABC+97] Sharon Adler, Anders Berglund, James Clark, Istvan Cseri, Paul Grosso, Jonathan Marsh, Gavin Nicol, Jean Paoli, David Schach, Henry S. Thompson, and Chris Wilson. A proposal for xsl (extensible style language), 1997. URL: http://www.w3.org/TR/NOTE-XSL.html.

[Ame83] American Psychological Association. Publication Manual of the American Psychological Association, 3rd edition, 1983.

[BD97] Tim Bray and Steve DeRose. Extensible markup language (xml): Part 2: Linking, 1997. URL: http://www.w3.org/TR/WD-xml-link-970731.

[Boo94] Grady Booch. Object-Oriented Analysis and Design with Applications. Addison-Wesley, 2nd edition, 1994.

[Bos97] Jon Bosak. Xml, java, and the future of the web. *XML Principles, Tools, and Techniques: World Wide Web Journal*, 2(4):219–227, 1997.

[BPSM97] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible markup language (xml), 1997. URL: http://www.w3.org/TR/PR-xml-971208.

[Bro75] Fred Brooks. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, 1975.

[CKR97] Dan Connolly, Rohit Khare, and Adam Rifkin. The evolution of web documents: The ascent of xml. *XML Principles, Tools, and Techniques: World Wide Web Journal*, 2(4):119–128, 1997.

[CLM97] Stuart Culshaw, Michael Leventhal, and Murray Maloney. Xml and css. *XML Principles, Tools, and Techniques: World Wide Web Journal*, 2(4):109–118, 1997.

[Com95] Douglas E. Comer. Internetworking with TCP/IP, volume I: Principles, Protocols, and Architecture. Prentice Hall, 3rd edition, 1995.

[Dav97] Angus Davis. Positioning html elements: Cascading style sheet positioning, 1997. URL: http://developer.netscape.com/news/viewsource/angus_css.html.

[Den97] D. C. Denison. The road to xml: Adapting sgml to the web. *XML Principles, Tools, and Techniques: World Wide Web Journal*, 2(4):5–12, 1997.

[Dou90] D. Dougherty. sed and awk. O'Reilly, 1990.

[Fla97a] David Flanagan. Java Examples in a Nutshell. O'Reilly, 1997.

[Fla97b] David Flanagan. Java in a Nutshell. O'Reilly, 2nd edition, 1997.

[Fla97c] David Flanagan. JavaScript: The Definitive Guide. O'Reilly, 2nd edition, 1997.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[GK97] Mark Grand and Jonathan Knudsen. Java Fundamental Classes Reference. O'Reilly, 1997.

[Goo94] Danny Goodman. Danny Goodman's AppleScript Handbook. Random House, 2nd edition, 1994.

[Gra97] Mark Grand. Java Language Reference. O'Reilly, 2nd edition, 1997.

[Gre97] Philip Greenspun. Database Backed Web Sites: The Thinking Person's Guide to Web Publishing. Ziff-Davis, 1997.

[Gun96] Shishir Gundavaram. CGI Programming on the World Wide Web. O'Reilly, 1996.

[Har97] Elliote Rusty Harold. Java Network Programming. O'Reilly, 1997.

[Int96] International Standards Organization. DSSSL (Document Style Semantics and Specification Language)[ISO/IEC 10179:1996], 1996. URL: ftp://ftp.ornl.gov/pub/sgml/WG8/DSSSL/dsssl96b.pdf.

[KR98] Rohit Khare and Adam Rifkin. X marks the spot: eXtensible markup language opens the door to a motherlode of automated web applications, 1998. URL: http://www.cs.caltech.edu/ adam/papers/xml/x-marks-the-spot.html.

[Lam94] Leslie Lamport. LaTeX: A Document Preparation System User's Guide and Reference Manual. Addison-Wesley, 2nd edition, 1994.

[Lan97] Richard Lander. Xml: The new markup wave, 1997. URL: http://www.csclub.uwaterloo.ca/u/relander/XML/xml_mw.html.

[LB96] Hakon Wium Lie and Bert Bos. Cascading style sheets, level 1, 1996. URL: http://www.w3.org/TR/REC-CSS1.

[Mey94] Bertrand Meyer. Reusable Software: the Base Object-Oriented Component Libraries. Prentice-Hall International (UK), 1994.

[Mic88] Sun Microsystems. Rpc: Remote procedure call protocol specification version 2, 1988. RFC: 1057. URL: http://sunsite.cnlab-switch.ch/ftp/doc/standard/rfc/10xx/1057.

[MK97] Chuck Musciano and Bill Kennedy. HTML: The Definitive Guide. O'Reilly, 2nd edition, 1997.

[OH97] Robert Orfali and Dan Harkey. Client/Server Programming with Java and CORBA. Wiley, 1997.

[OHE96] Robert Orfali, Dan Harkey, and Jeri Edwards. The Essential Client/Server Survival Guide. Wiley, 2nd edition, 1996.

[Oua95] Steve Oualline. Practical C++ Programming. O'Reilly, 1995.

[PSL+97] Jean Paoli, David Schach, Chris Lovett, Andrew Layman, and Istvan Cseri. Building xml parsers for microsoft ie4. *XML Principles, Tools, and Techniques: World Wide Web Journal*, 2(4):187–195, 1997.

[SG94] Abraham Silberschatz and Peter Baer Galvin. Operating System Concepts. Addison-Wesley, 4th edition, 1994.

[Sym90] Symantec. More 3.0 Reference, 1990.

[TM98] Gabor J. Toth and Paul Mayer. The free on-line dictionary of computing, 1998. URL: http://wfn-shop.princeton.edu/cgi-bin/foldoc.

[Uni96] The Unicode Consortium. The Unicode Standard, Version 2.0, 1996.

[vH94] Eric van Herwijnen. Practical SGML. Kluwer Academic, 2nd edition, 1994.

[WHW+97] Lauren Wood, Arnaud Le Hors, Andrew Watson, Bill Smith, Chris Lovett, David Singer, Gavin Nicol, James Clark, Jared Sorensen, Jonathan Robie, Mike Champion, Paul Grosso, Peter Sharpe, Phil Karlton, Rick Gessner, Robert Sutor, Scott Isaacs, Sharon Adler, Steve Byrne, Tim Bray, and Vidur Apparao. Document object model specification, 1997. URL: http://www.w3.org/TR/WD-DOM-971209.

[Win98] Dave Winer. Frontier 5.0 home page, 1998. URL: http://www.scripting.com/frontier5.

[Zuk97] John Zukowski. Java AWT Reference. O'Reilly, 1997.

APPENDIX

A. INTERACTIVE HOTLIST SOURCE CODE

Each page in the interactive hotlist is composed of several files organized by a template with the extension .html. Many of these files are pieces of the template that are common to all the pages in a given section. There are three sections, organized somewhat by content: General Interest, Web Page Design, and Computers. The example given here is for the main page of General Interest, which is the home page for the Interactive Hotlist. Some of the files consist of only one html element, or only one line of code. This is to provide consistency and expandability—the sections are logical divisions and can be added to without breaking the existing design.

A.1 File One: general.html

General.html is the main template file that simply includes all of the other files that make up the page. In order to change this to a page in the Computers section of Interactive Hotlist, the following changes would be made: title_general.tpl becomes title_computers.tpl; general.list becomes computers.list; gen_form.list becomes comp_form.list; and form_list_general.tpl becomes form_list_comp.tpl. Modifying this template for another page in the same section is even easier: favorites.list becomes subject.list, where subject.list is the specific contents for that page. The entire file follows:

```
<!--#include file="templates/head1.tpl"-- >
<!--#include file="templates/title_general.tpl"-- >
<!--#include file="templates/head2.tpl"-- >
<!--#include file="templates/head_script.tpl"-- >
```

```
<!--#include file="templates/head3.tpl"-- >
<!--#include file="templates/body1.tpl"-- >
<!--#include file="templates/title_general.tpl"-- >
<!--#include file="templates/body2.tpl"-- >
<!--#include file="lists/general.list"-- >
<!--#include file="templates/body3.tpl"-- >
<!--#include file="lists/favorites.list"-- >
<!--#include file="templates/body4.tpl"-- >
<!--#include file="lists/gen_form.list"-- >
<!--#include file="templates/body5.tpl"-- >
<!--#include file="templates/form_list_general.tpl"-- >
<!--#include file="templates/body6.tpl"-- >
<!--#include file="templates/copyright.tpl"-- >
<!--#include file="templates/body7.tpl"-- >
```

A.2 File Two: head1.tpl

Head1.tpl is a simple file, just three HTML opening elements common to every page:

```
<html><head><title>
```

A.3 File Three: title_general.tpl

The title_general.tpl file contains the title of the page, as it will be displayed in the window title bar, in the Go menu of Netscape Navigator, and in the user's hotlist if bookmarked. Currently, there is only one title for each general section, putting a title for each page would be a big improvement. The following text is used twice in the template so that the same title is used as a heading on the page, as well as in the places mentioned above:

```
Dethe's Interactive Hotlist (General Interest)
```

File Four: head2.tpl

The entire head2.tpl file consists of only one HTML closing element, to close the title section of the code. We are still in the header section and can add material which is appropriate there:

```
</title>
```

A.4 File Five: head_script.tpl

Head_script.tpl is a file that does nothing at this time. It is included in the template to make it easier to add ECMAScript functions to the template. The functions would be defined here in the page header, hidden from browsers which do not handle ECMAScript, and called from the body of the page:

```
<script> <!-- Cloak script from old browsers function
doNothing() { if(false) break; // De-cloak --> </script>
```

A.5 File Six: head3.tpl

Head3.tpl is another small file, which serves only to close the header section of the page. It could easily have been included in the body1.tpl file, but is kept as a separate file for easier understanding of the template:

```
</head>
```

A.6 File Seven: body1.tpl

The body1.tpl file begins the body portion of the page, defines the background, centers and opens the heading tag for the title to be placed in:

```
<body background="/~delza/hotlist/gifs/skull.jpeg"><center><h3>
```

A.7 File Eight: body2.tpl

The body2.tpl file closes the heading, adds an image as a horizontal rule, and adds a text line of hotlinks to the various sections of the Interactive Hotlist. Then the table which contains most of the page body is defined and opened:

```
</h3>
<img alt="" src="/~delza/hotlist/gifs/wavy.gif">
<br> |
<a href="/~delza/hotlist/">General Interest</a> |
<a href="/~delza/hotlist/computer/">Computers</a> |
<a href="/~delza/hotlist/webdesign">Web Design</a>|
<br>
</center>
<table cellspacing=5 border=0><tr><td valign=top>
<font size=-1>
```

A.8 File Nine: general.list

The general.list file contains the menu for the General section of the Interactive Hotlist. From here the user can jump to any page in the section. This text is included on every page, so they are all cross-linked together:

```
<a href="cool.html">Simply Cool </a><br>
<a href="friends.html">Friends Pages </a><br>
<a href="research.html"> Research </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<ahref="xml.html">XML </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="dom.html">DOM </a><br>
```

```
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="java.html">Java</a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="pattern.html">Pattern Design </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="corba.html">CORBA </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="scripting.html">Scripting </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="style.html">Style Sheets </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="be.html">Be OS </a><br>
<a href="family.html"> Kids and Family </a><br>
<a href="work.html">Work </a><br>
<a href="school.html">School </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="class.html"> Classes </a><br>
<a href="hobbies.html">Hobbies</a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="hacking.html">Hacking </a><br>
    <img src="gifs/tab.gif" height=1 width=34 alt="">
<a href="robot.html"> Robotics </a><br>
    <img src="gifs/tab.gif" height=1 width=34 alt="">
<a href="hack_soft.html">Software </a><br>
    <img src="gifs/tab.gif" height=1 width=34 alt="">
<a href="hack_hard.html">Hardware </a><br>
    <img src="gifs/tab.gif" height=1 width=34 alt="">
<a href="electronics.html">Electronics Catalogs </a><br>
```

```
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="food.html">Cooking </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="beer.html">Homebrew </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="comics.html">Comics </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="satire.html">Satire </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="games.html">Games</a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="kitch.html">Kitch Chic </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="books.html">Books </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="madscience.html">Mad Science </a><br>
    <img src="gifs/tab.gif" height=1 width=34 alt="">
<a href="orgone.html">Wilhelm Reich </a><br>
    <img src="gifs/tab.gif" height=1 width=34 alt="">
<a href="synergetics.html">Synergetics </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="paranoia.html">Paranoia </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="costume.html">Costume &amp;amp; Clothing </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="origami.html">Origami</a><br>
<a href="myweb.html">My Web </a><br>
<a href="athens.html">Athens, Ohio &amp;amp; Environs </a><br>
```

```
<a href="reference.html">Reference </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="news.html">News</a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="bookstores.html">Bookstores </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="libraries.html">Libraries </a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="health.html">Medical</a><br>
    <img src="gifs/tab.gif" height=1 width=17 alt="">
<a href="bulgaria.html">Bulgaria</a><br>
```

A.9 File Ten: body3.tpl

The entire list above is contained within a single table cell. The body3.tpl file closes the cell and opens another one:

```
</font></td><td valign=top>
```

A.10 File Eleven: favorites.list

The favorites.list file is different for each page and defines the main body of the page. Since this is the default page when entering the Interactive Hotlist, it contains my favorite and most-travelled links:

```
<strong>Hotlist:General:Favorites</strong><hr>
<a href="http://www.macintouch.com">Macintouch</a><br>
<a href="http://www.unitedmedia.com/comics/dilbert">Dilbert</a><br>
<a href="http://www.scripting.com">Scripting.com News</a><br>
<a href="http://www.webmonkey.com">WebMonkey</a><br>
```

```
<a href="http://www.projectcool.com/">Project Cool</a><br>
<a href="http://www.utne.com/">Utne Reader</a><br>
<a href="http://www.cnt.org/tnw/tnwhome.htm">
    The Neighborhood Works</a>
    Building Alternative Visions for the City<br>
<a href="http://www.drweil.com">Ask Dr. Weil</a><br>
<a href="http://www.macfixit.com/">MacFixIt</a><br>
<a href="http://www.i-55.com/mac2">MacOS Internet Connections</a><br>
<a href="http://www.freepress.com:80/myee/ultimate_mac.html">
    The Ultimate Macintosh</a><br>
<a href="http://www.webintosh.com/">Webintosh</a><br>
<a href="http://www.versiontracker.com/">Version Tracker</a><br>
<a href="http://macfixit.pair.com/">MacFixIt</a><br>
<a href="http://til.info.apple.com/til/til.html">
    Apple Tech Info Library</a><br>
<a href="http://hyperarchive.lcs.mit.edu/HyperArchive.html">Info-Mac
    HyperArchive</a><br>
<a href="http://www.infohiway.com/faster/">The Bandwidth
    Conservation Society</a><br>
<a href="http://www.mbmdesigns.com/macjava/">Apple
    Flavored Java</a><br>
<a href="http://www.eskimo.com/~pristine/">Pure Mac </a>
    Carefully selected software<br>
<a href="http://mac-web-tips.clearway.com/">Top 20 Tips for
    Macintosh Webmasters </a><br>
<a href="http://ds.dial.pipex.com/pixelp/wpdesign/wpdintro.shtml">
    Web Page Design for Designers</a><br>
<a href="http://www.AmbrosiaSW.com/DEF/">Mac Pruning Page</a> Home
```

```
    of the InformINIT, find out what your extensions really do!<br>
<a href="http://www.ora.com">O'Reilly Publishing</a>
    Publishers of the best computer books.<br>
<a href="http://www.newton.apple.com">Newton Home Page</a><br>
<a href="http://www.ardi.com">Executor Home Page</a>
    Mac Emulation for PC<br>
<a href="http://leb.net/vmac/">vMac</a> The Virtual Macintosh<br>
<a href="http://www.amcoex.com">American Computer Exchange</a>
    Used computer prices<br>
<a href="http://www.macshare.com">MacShare</a>
    Macintosh shareware galore<br>
<a href="http://www.javaman.com">JavaMan</a> Looking for java job
    listings?<br>
<a href="http://www.useit.com/alertbox/">The AlertBox</a> Current
    Issues in Web Usability<br>
<a href="http://www.npr.org/news/tech/971106..net.html">Esther Dyson
    Interview</a><br>
<a href="http://www.macwindows.com/">MacWindows</a>
    Cross-platform solutions and tips<br>
<a href="http://emporium.turnpike.net/P/ProRev/">
    Progressive Review</a>
    Includes Sam Smith's Great American Political Repair Manual<br>
<a href="http://wfn-shop.princeton.edu/cgi-bin/foldoc">
    Free On-Line Dictionary of Computing</a><br>
```

A.11 File Twelve: body4.tpl

The body4.tpl file closes the table containing the menus and links above and defines the menu of different sections again so the user can reach it easily whether they are at the top or bottom of a page. It would have been convenient to remove this menu to a separate file, but the web server I am using wouldn't allow a Server Side Include within an Included file. I also open a table and a form here, and begin defining the form content:

```
</td></tr></table><br> <center> |
<a href="/~delza/hotlist/">General Interest</a> |
<a href="/~delza/hotlist/computer/">Computers</a> |
<a href="/~delza/hotlist/webdesign">Web Design</a>|<p>
<table border=4><tr><td>
<form method="POST" name="AddLink" action=
    "http://voyager2.cns.ohiou.edu/~delza/cgi-bin/hotlist.cgi">
<strong>Enter a new link:</strong><br>
Title:<input type="text" name="link_name" value="" size=40><br>
URL:<input type="text" name="link_url" value="" size=40><br>
<select name="link_list">
```

A.12 File Thirteen: gen_form.list

The gen_form.list file contains a list of the pages in this section of the Interactive Hotlist. The user selects which page to add a link to when they contribute to the Hotlist:

```
<option selected value="">Choose a sublist of General Interest
<option value="">=======================
<option value="favorites.list">Favorites
```

```
<option value="cool.list">Simply Cool
<option value="friends.list">Friends Pages
<option value="research.list">Research
<option value="xml.list"> XML
<option value="dom.list"> DOM
<option value="java.list"> Java
<option value="pattern.list"> Pattern Design
<option value="corba.list"> CORBA
<option value="scripting.list"> Scripting
<option value="be.list"> Be OS
<option value="style.list"> Style Sheets
<option value="family.list">Kids &amp; Family
<option value="work.list">Work
<option value="school.list">School
<option value="class.list"> Classes
<option value="hobbies.list">Hobbies
<option value="hacking.list"> Hacking
<option value="robot.list">  Robotics
<option value="hack_soft.list">  Software (Hacking)
<option value="hack_hard.list">  Hardware (Hacking)
<option value="electronics.list">  Electronics Catalogs
<option value="food.list"> Cooking
<option value="beer.list"> Homebrew
<option value="comics.list"> Comics
<option value="satire.list"> Satire
<option value="games.list"> Games
<option value="kitch.list"> Kitch Chic
<option value="books.list"> Books
```

```
<option value="madscience.list"> Mad Science
<option value="orgone.list">  Wilhelm Reich
<option value="synergetics.list">  Synergetics
<option value="paranoia.list"> Paranoia
<option value="costume.list"> Costume and Clothing
<option value="origami.list"> Origami
<option value="myweb.list">My Web
<option value="athens.list">Athens, OH &amp; Environs
<option value="reference.list">Reference
<option value="news.list"> News
<option value="bookstores.list"> Bookstores
<option value="libraries.list"> Libraries
<option value="health.list"> Health
<option value="bulgaria.list"> Bulgaria
```

A.13 File Fourteen: body5.tpl

The body5.tpl file adds the rest of the body of the form begun above, allowing the user to add some descriptive text for their new link and defining a button for them to press to create the link:

```
</select><br>
<input type="text" name="link_desc" value="Short Description" size=40>
<br><center>
<input type="submit" name="submit" value="Add\Link">
</center>
```

A.14 File Fifteen: form_list_general.tpl

The form_list_general.tpl file creates a part of the form which the user should not see, but which tells the Common Gateway Interface script where to find the file it will be appending to. This is an example of security through obscurity and could certainly be improved:

```
<input type=hidden name=list value="/home/delza/www/hotlist/lists/">
```

A.15 File Sixteen: body6.tpl

The body6.tpl file closes the form and the table and adds another graphic horizontal rule, but does not close the body of the page yet:

```
</form></table> <img alt="" src="/~delza/hotlist/gifs/wavy.gif"><br>
```

A.16 File Seventeen: copyright.tpl

The copyright.tpl file contains my copyright and contact information, added to the bottom of every page:

```
&copy; 1997 Dethe Elza <a href="mailto:dethe.elza.1@ohiou.edu">
dethe.elza.1@ohiou.edu</a><br> For moreinformation about this page,
click <a href="/~delza/hotlist/about.html">here</a>.
```

A.17 File Eighteen: body7.tpl

Finally, we close the body of the page and the page itself with the body7.tpl file:

```
</center></body></html>
```

A.18 File Nineteen: hotlist.cgi

The hotlist.cgi file contains the Common Gateway Interface program that I use to append new entries to the Interactive Hotlist. It consists of an awk script embedded in a shell script, and it includes a call to formview, which is an external program that I did not write, but which strips off many of the unix metacharacters which could otherwise create security risks.

```
#!/bin/sh

PATH=/bin:/usr/bin:/usr/local/bin:/usr/contrib/bin export PATH

formview | awk -F"=" '
BEGIN{
    PATH = "/home/delza/www/hotlist/lists/";
    NAME = "My Home Page";
    URL = "http://voyager.cns.ohiou.edu/~delza";
    LIST = "category.list";
    DESC = "Indescribable";
}
{
    if ($1 == "link_name") NAME = $2;
    if ($1 =="link_url") URL = $2;
    if ($1 == "link_list") LIST = $2;
    if ($1 == "link_desc") DESC = $2;
    if ($1 == "list") PATH = $2;
}
END{
    FILE = PATH LIST;
```

```
MESSAGE = NAME "(" URL ") added to " LIST;
if ((URL == "URL") || (URL == ""))
{
    print "Content-type: text/html";
    print "";
    print "You must include a URL!";
}
if ((NAME == "Name") || (NAME == ""))
{
    print "Content-type: text/html";
    print "";
    print "You must include a Name!";
}
if (LIST == "")
{
    print "Content-type: text/html";
    print "";
    print "You must choose a sub-list!";
}
if ((DESC != "") && (DESC != "Short Description"))
{
    print "<a href=\"" URL "\">" NAME "</a> " DESC "<br>" >> FILE;
    system("mail -s " MESSAGE "delza@voyager.cns.ohiou.edu");
} else
{
    print "<a href=\"" URL "\">" NAME "</a><br>" >> FILE;
    system("mail -s " MESSAGE " delza");
}
```

```
    print "Content-type:text/html";
    print "";
    print NAME " successfully added to " LIST "!<p>";
    print "Press \"Back\" button to return to hotlist"; }
'
```

B. FREEWORD SOURCE CODE

My implementation of FreeWord is still evolving towards my design goals. The complete source code for the project is listed below. I have noted some areas which would be good areas for future research and further development, as well as problems with implementing FreeWord using Microsoft Internet Explorer 4.0.

B.1 File One: Outline.html

The Outline.html file is the web page a user would bring up in their browser in order to use FreeWord. It contains definitions of the menu, pointers to the script files, a pointer to the stylesheet, a pointer to the Microsoft Java XML parser. There is a lot of room here for further research. One option would be to remove the menu definitions and put them in their own XML file. Another possibilty would be for structures within XML files to contain pointers to their own scripting functions. A fixed stylesheet could be replaced by dynamic ECMAScript styles. Given these changes, all the HTML file will need to include is a pointer to the ECMAScript XML parser and the names of the XML files to load:

```
<html> <head> <title> Outline Example </title>

<link rel="stylesheet" type="text/css" href="outline.css"
    title="OutlineStyles">
<script language="JavaScript1.2" src="utility.js"></script>
<script language="JavaScript1.2" src="outline.js"></script>
<script language="JavaScript1.2" src="debug.js"></script>
```

```
<script language="JavaScript1.2" src="menu.js"></script>
<script language="JavaScript1.2" src="working.js"></script>
<script language="JavaScript1.2" src="keyboard.js"></script>
<script FOR=window EVENT=onload language="JScript"> initialize();
</script >
</head>
<body>
<applet code="com.ms.xml.dso.XMLDSO.class" width="0" height="0"
    MAYSCRIPT="true" id="xmldso"> <PARAM NAME="url" VALUE="outline.xml">
</applet>
<span class="MenuBar">
<span class="Menu" id="menu" onmousedown="toggleShow();"
    onmouseover="showMenu();" onmouseout="hideMenu();">File
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseout="hideMenu(event.srcElement.parentElement);"
    onmouseup="fileNew();">New</span>
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseout="hideMenu(event.srcElement.parentElement);"
    onmouseup="fileOpen();">Open</span>
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseout="hideMenu(event.srcElement.parentElement);"
    onmouseup="fileSave();">Save</span>
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseout="hideMenu(event.srcElement.parentElement);"
```

```
    onmouseup="fileClose();">Close</span>
</span>
<span class="Menu" id="menu" onmousedown="toggleShow();"
    onmouseover="showMenu();"
    onmouseout="hideMenu();">Edit
<span class="MenuItem" id="menuitem"style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseup="editCut();">Cut</span>
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseup="editCopy();">Copy</span>
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseup="EditPaste();">Paste</span>
</span>
<span class="Menu" id="menu"
    onmousedown="toggleShow();" onmouseover="showMenu();"
    onmouseout="hideMenu();">Format
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseup="formatStyle();">Style</span>
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseup="formatDocument();">Document</span>
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseup="formatText();">Text</span>
</span>
```

```
<span class="Menu" id="menu"
    onmousedown="toggleShow();" onmouseover="showMenu();"
    onmouseout="hideMenu();">Outline
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseup="outlineExpand();">Expand</span>
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseup="outlineCollapse();">Collapse</span>
</span>
<span class="Menu" id="menu" onmousedown="toggleShow();"
    onmouseover="showMenu();"
    onmouseout="hideMenu();">Window
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseup="windowSwitch(1);">Window1</span>
</span>
<span class="Menu" id="menu"
    onmousedown="toggleShow();" onmouseover="showMenu();"
    onmouseout="hideMenu();">Help
<span class="MenuItem" id="menuitem"style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseup="helpAbout();">About</span>
<span class="MenuItem" id="menuitem"style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
    onmouseup="HelpUsing();">Using</span>
<span class="MenuItem" id="menuitem" style="display: none;"
    onmouseover="showMenu(event.srcElement.parentElement);"
```

```
    onmouseup="helpIndex();">Index</span>
</span>
</span>
<span id="PutTextHere"></span>
<textarea class="DEBUG" id="debug" name="Debug" cols="80" rows="2"
    readonly wrap="virtual" style="display: none;
    position: absolute; top: 0; left: 0;">
</textarea>
<textarea id="edit" class="EDIT" name="Edit" cols="80"
    rows="1" wrap="virtual" style="display: none;
    position: absolute; top: 0; left: 0; font-family: serif;">
</textarea><p>
<input type=button value="Show HTML Data"
    onclick='traceAllChildren();'> <p>
<input type=button value="Show XML Data"
    onclick='testXMLtoHTML(xmldso.getDocument());'>
</body>
</html>
```

B.2 File Two: outline.css

The outline.css file is the stylesheet for FreeWord. Because Microsoft's implementation of ECMAScript does not permit stylesheet styles to be manipulated by the program, the styles have been migrating into the script files. An area for further research would be to remove this file entirely and replace it with dynamic styles written in ECMAScript:

```
BODY {
    border: 0; z-index: 10;
```

```
}
TABLE {
    margin-left: 0.1in; z-index: 10;
}
TABLE.ITEM {
    margin-left: 0.35in; z-index: 10;
}
.MenuBar {
    background-color: #778899;  position: absolute;
    left: 0; top: 0; z-index: 100;
}
.Menu {
    position: relative; top: 0; left: 15pt;
    font-size: larger; z-index: 100;
}
.MenuItem{
    position: absolute; background-color: #778899;
    width: 50; font-size: smaller; z-index: 100;
}
.CONTAINER{
    position: relative; left: 0in; z-index: 10;
}
.TITLE{
    position: relative; left: -0.08in;
    font-weight: bold; font-family: serif;
    z-index: 10;
}
.CONTENTS{
```

```
    position: relative;left: .2in; z-index: 10;
}
.ITEM{
    position: relative; left: 0in;
    font-family: serif; font-weight: normal;
    z-index: 10;
}
.WEDGE{
    position: relative; left: -0.08in;
}
```

B.3 File Three: utility.js

The utility.js contains functions which are used by the other functions to reduce repetitive coding. There are functions to detect mouse movements and functions to determine position of an element for editing. WalkCollection() is a function listed here which is used throughout the program. In ECMAScript a routine often may return either a single item or a collection of items. The CollectionWalker tests whether a collection or a single item is returned and applies a given function to the item (if one) or each item in the collection (if multiple):

```
function overSubject(){
    event.srcElement.style.color ="#B8860B";
}
function offSubject(){
    event.srcElement.style.color = "#000000";
}
function getOffsets(obj){
    if (obj != null) var myObj = obj;
```

```
    else var myObj = event.srcElement;
    var mySize = new Object();
    mySize.left = myObj.offsetLeft;
    mySize.top = myObj.offsetTop;
    mySize.height = myObj.offsetHeight;
    mySize.width = myObj.offsetWidth;
    return mySize;
}
// General collection walking function
//
// To use pass an object which may or may not be a collection to
// walkCollection, along with an item which will be used with
// the collection and a function to actually do the work
//
function walkCollection(coll, func){
    var returnValue = "";

    if (coll != null){
        if (coll.length != null){
            for (var i = 0; i < coll.length; i++){
                returnValue = returnValue + func(coll[i]);
            }
        }
        else {
            returnValue = returnValue + func(coll);
        }
    }
    return returnValue;
```

```
}
```

B.4 File Four: outline.js

The outline.js contains the script functions for managing an outline in FreeWord. A user can double-click an outline control (a small triangle to the left of a container) to expand or contract a container item. Also, menu functions are provided to expand or contract an entire outline:

```
function toggleSubOutline(obj){
    if (obj != null)
        var myImg = obj;
    else
        var myImg = event.srcElement;
    var myState = myImg.parentElement.STATUS;
    if (EDITMODE == "edit") switchEditMode();
    if (myState == "Open"){
        hideSubOutline(myImg);
    }
    else{
        showSubOutline(myImg);
    }
}
function hideSubOutline(theImage){
    if (theImage != null)
        var myImage = theImage;
    else
        var myImage = event.srcElement;
    var myParent = myImage.parentElement;
```

```
    var myChild = myParent.children;
    var myState = myParent.STATUS;
    if (EDITMODE == "edit") switchEditMode();
    for (var i = 0; i < myChild.length; i++){
        var myName = myChild[i].className;
        if (myName == "CONTENTS")
            hideItem(myChild[i]);
    }
    myImage.src = hideImg;
    myParent.STATUS = "Closed";
}
function hideItem(item){
    item.style.display = "none";
}
function showSubOutline(theImage){
    if (theImage != null)
        var myImage = theImage;
    else
        var myImage = event.srcElement;
    var myParent = myImage.parentElement;
    var myChild = myParent.children;
    var myState = myParent.STATUS;
    if (EDITMODE == "edit") switchEditMode();
    for (var i = 0; i < myChild.length; i++){
        myName = myChild[i].className;
        if (myName == "CONTENTS")
            showItem(myChild[i]);
    }
```

```
    myImage.src = showImg;
    myParent.STATUS = "Open";
}
function showItem(item){
    item.style.display = "";
}
```

B.5 File Five: menu.js

The menu.js file contains handlers for the FreeWord menus. Menu setup and placement is actually in the working.js file and actual functions which the menus call are in the appropriate files: Editing commands are in edit.js, file commands are in file.js, etc.:

```
function arrangeMenus(){
    walkCollection(document.all.menu, doArrangeMenu);
}
function doArrangeMenu(theMenu){
    theMenuItem = theMenu.children.menuitem;
    if (theMenuItem != null){
        if (theMenuItem.length != null){
            for (var i = 0; i < theMenuItem.length; i++){
                theMenuItem[i].style.pixelLeft =
                    theMenu.style.offsetLeft;
                theMenuItem[i].style.pixelTop = LINEHEIGHT * (i + 1);
            }
        }
        else{
            theMenuItem.style.pixelLeft = theMenu.style.offsetLeft;
```

```
            theMenuItem.style.pixelTop = LINEHEIGHT;
        }
    }
}
function toggleMenu(){
    if (MENUVISIBLE == "none")
            MENUVISIBLE = "";
        else
            MENUVISIBLE = "none";
}
function toggleShow(){
    toggleMenu();
    showMenu(event.srcElement);
}
function doShowMenu(theMenuItem){
    theMenuItem.style.display = "";
}
function showMenu(theMenu){
    if (MENUVISIBLE != "")
        return;
    if (theMenu != null)
        var myMenu = theMenu;
    else
        var myMenu = event.srcElement;
    var myMenuItems = myMenu.children.menuitem;
    walkCollection(myMenuItems, doShowMenu);
}
function doHideMenu(theMenuItem){
```

```
    theMenuItem.style.display = "none";
}
function hideMenu(theMenu){
    if (theMenu != null)
        var myMenu = theMenu;
    else
        var myMenu = event.srcElement;
    var myMenuItems = myMenu.children.menuitem;
    walkCollection(myMenuItems, doHideMenu);
}
function menuPlaceholder(theMenu, theString){
    HideMenu(theMenu); ToggleMenu();
    alert("Command " + theString + " is not yet implemented");
    return false;
}
// File functions
function fileNew(){
    MenuPlaceholder(event.srcElement.parentElement, "File:New ");
}
function fileOpen(){
    MenuPlaceholder(event.srcElement.parentElement, "File:Open");
}
function fileSave(){
    MenuPlaceholder(event.srcElement.parentElement, "File:Save");
}
function fileClose(){
    MenuPlaceholder(event.srcElement.parentElement, "File:Close");
}
```

```
// Edit Functions
function editCut(){
    MenuPlaceholder(event.srcElement.parentElement, "Edit:Cut");
}
function editCopy(){
    MenuPlaceholder(event.srcElement.parentElement, "Edit:Copy");
}
function editPaste(){
    MenuPlaceholder(event.srcElement.parentElement, "Edit:Paste");
}
// Format Functions
function formatStyle(){
    MenuPlaceholder(event.srcElement.parentElement, "Format:Style");
}
function formatDocument(){
    MenuPlaceholder(event.srcElement.parentElement,
        "Format:Document");
}
function formatText(){
    MenuPlaceholder(event.srcElement.parentElement, "Format:Text");
}
//Outline Functions
function outlineCollapse(){
    hideMenu(event.srcElement.parentElement);
    myGroup = document.all.tags("IMG");
    if (myGroup != null){
        for (var i = 0; i < myGroup.length; i++){
            hideSubOutline(myGroup[i]);
```

```
        }
    }
}
function outlineExpand(){
    hideMenu(event.srcElement.parentElement);
    myGroup = document.all.tags("IMG");
    if (myGroup != null){
        for (var i = 0; i < myGroup.length; i++){
            showSubOutline(myGroup[i]);
        }
    }
}
// Window Functions
function windowSwitch(winNum){
    MenuPlaceholder(event.srcElement.parentElement,
        "Window:Switch");
}
// Help Functions
function helpAbout(){
    MenuPlaceholder(event.srcElement.parentElement, "Help:About");
}
function HelpUsing(){
    MenuPlaceholder(event.srcElement.parentElement, "Help:Using");
}
function helpIndex(){
    MenuPlaceholder(event.srcElement.parentElement, "Help:Index");
}
```

B.6 File Six: working.js

The working.js file contains the heart of the FreeWord application. Initialization and setup are done here, as well as editing, parsing from XML to HTML and back, and display:

```
// Global variables
var rootURL = "";
var showImg = rootURL + "down.gif";
var hideImg = rootURL + "right.gif";
var show = new Image();
var hide = new Image();
var LINEHEIGHT = 18;
var LINEWIDTH = 150;
var MENUVISIBLE = "none";
var EDITMODE = "";
var EDITFIELD;
show.src = showImg;
hide.src = hideImg;
function initialize(){
    arrangeMenus();
    loadXML(xmldso.getDocument());
    initializeState();
    document.onkeypress = handleKeyPress;
}
function newElement(evt){
    if (EDITMODE != "edit")
        return;
    var thisElem = EDITFIELD; // <Span>
```

```
    var parElem = thisElem.parentElement.parentElement.parentElement;
        // <Span> -> <TD> -> <TR> -> <TBODY>
    switchEditMode();
    var myString = '<tr><td>';
    var myString = '<img id="LevelControl" align="absmiddle"';
    var myString += ' src="down.gif"';
    myString += ' border="0" alt="" ondblclick=hideChildren()';
    myString += ' style="display: none;">';
    myString += '<span id="type" style="display: none;">';
    myString += ' Item</span>';
    myString += '<span id="status" style="display: none;">';
    myString += 'Open</span>';
    myString += '<span id="Subject" onmouseover="overSubject();"';
    myString += ' onmouseout="offSubject();" ';
    myString += 'ondblclick="switchEditMode();"';
    myString += ' style="position: relative;"></span></td></tr>';
    parElem.insertAdjacentHTML("BeforeEnd", myString);
    var newElem =
        parElem.children("SPAN")[parElem.children("SPAN").length - 1];
    switchEditMode(newElem);
}
function nextElement(thisElem){
    if (thisElem.className != "SUBJECT")
        return;
    if ((thisElem.Status == "Open") && (thisElem.children != null))
        return nextChild(thisElem);
    if (thisElem.Status == "Closed")
    return nextSibling(thisElem);
```

```
}
function nextChild(thisElem){ // placeholder function
}
function nextSibling(thisElem){ // placeholder function
}
function showAll(){
    document.all.Level[0].style.display = "";
}
// Edit Functions
function switchEditMode(theText){
    var myEditPane = document.all.edit;
    if (EDITMODE != "edit"){
        if (theText != null)
            var myText = theText;
        else
            var myText = event.srcElement;
        EDITMODE = "edit";
        EDITFIELD = myText;
        var mySize = getOffsets(myText);
        myEditPane.style.pixelLeft = mySize.left;
        myEditPane.style.pixelTop = mySize.top;
        if (mySize.height < LINEHEIGHT)
            myEditPane.styl.pixelHeight = LINEHEIGHT;
        else
            myEditPane.style.pixelHeight = mySize.height;
        if (mySize.width < LINEWIDTH)
            myEditPane.style.pixelWidth = LINEWIDTH;
        else
```

```
            myEditPane.style.pixelWidth = mySize.width + 30;
        myEditPane.value = myText.innerHTML;
        myText.style.visibility = "hidden";
        myEditPane.style.display = "";
    }
    else{
        EDITMODE = "";
        var myText = EDITFIELD;
        myText.innerHTML = myEditPane.value;
        myEditPane.style.display = "none";
        myText.style.visibility = "visible";
    } return false;
}
// setup functions
function initializeState(){
    walkCollection(document.all.tags("DIV"), doInitState);
}
function doInitState(theLevel){
    if (theLevel.className != "SUBJECT")
        return;
    var theStatus = theLevel.STATUS;
    var theImage = theLevel.children.tags("IMG");
    if (theStatus != null){
        if (theStatus == "Closed"){
            walkCollection(theImage, hideSubOutline);
        }
    }
    else{
```

```
        message("Holy mole, Batman, we don't have any STATUS ids!");
    }
}
function XMLtoHTML(theNode){
    var myString = "";
    // Check for comments and deal with them appropriately
    if (theNode.type == 2) {
        myString += '<div class="COMMENT"><!--' + theNode.text;
        myString += ' --></div >\n';
        return myString;
    } // PCDATA makes up the bulk of text, simply return it
    if (theNode.type == 1){
        myString = theNode.text + "<br> \n";
        return myString;
    }
    var tagname = theNode.tagName;
    myString = '<span class="' + tagname + '" ';
    var attrs = theNode.attributes;
    while (attrs != null && attrs.hasMoreElements()){
        var a = attrs.nextElement();
        var myName = a.getName();
        myString += a.getName();
        var myValue = a.getValue();
        if ((myValue != null) || (myValue != ""))
            myString += '="' + myValue + '" ';
        else
            myString += '=""';
    }
```

```
if (tagname == "CONTAINER"){
    //Insert special container attributes here
}
else if (tagname == "TITLE"){
    // Insert special title attributes here
    myString += ' ondblclick="switchEditMode();" ';
}
else if (tagname == "CONTENTS"){
    // Insert special contents attributes here
}
else if (tagname == "ITEM"){
    // Insert special item attributes here
    myString += ' ondblclick="switchEditMode();" ';
}
myString += ">\n";
if (tagname == "CONTAINER"){
    // Put in the wedge outline controller image
    myString += '\n<img class="WEDGE" src="down.gif"';
    myString += ' border="0" alt="" align="absmiddle"';
    // Insert special image attributes here
    myString += ' ondblclick="toggleSubOutline();" ';
    myString += '>\n';
}
var numElements = theNode.numElements();
if (numElements == 0)
    myString += "</span>\n";

if (numElements > 0) {
```

```
        var j;
        for (j = 0; j < numElements; j++) {
            myString += "\n";
            var child = theNode.getChild(j);
            if (child.type != 12)
                myString += XMLtoHTML(child);
        }
           myString += "\n</span>\n";
    }
       return myString;
}
function loadXML(doc) {
    document.title = "Dethe's Thesis";
    var xml = XMLtoHTML(doc.root);
    var myPlaceholder = document.all.PutTextHere;
    if (myPlaceholder == null)
        message("Hmmm, there seems to be a communication failure.");
    else
        myPlaceholder.innerHTML = xml;
}
function testXMLtoHTML(doc){
    var w = window.open("", "", "");
    w.document.title = "Dethe's Thesis";
    w.document.write("<html><head>\n");
    w.document.write('<link rel="stylesheet" type="text/css"');
    w.document.write(' href="outline.css" title="OutlineStyles">\n');
    w.document.write("</head><body>\n");
    var xml = XMLtoHTML(doc.root);
```

```
    w.document.write(xml);
    w.document.write("\n</body></html>");
}
// Turn the html back into xml function
doWriteXML(theObj){
    var myString = "";
    if (theObj == null)
        return myString;
    if (theObj.id == "LevelControl")
        return myString;
    if (theObj.id == "type"){
        if (theObj.innerText == "")
            myString = "<TYPE/>\n";
        else
            myString = "<TYPE>" + theObj.innerText + "</TYPE>\n";
    }
    else if (theObj.id == "status"){
        if (theObj.innerText == "")
            myString = "<STATUS/>\n"
        else
            myString = "<STATUS>" + theObj.innerText;
            myString += "</STATUS>\n";
    }
    else if (theObj.id == "Subject"){
        if (theObj.innerText == "")
            myString = "<SUBJECT/>";
        else
            myString = "<SUBJECT>" + theObj.innerText;
```

```
        myString += "</SUBJECT>\n";
}
else if (theObj.id == "Level"){
    if (theObj.className == "level2") {
        if (theObj.children == null)
            myString = "<LEVEL2/>\n";
        else {
            myString = "<LEVEL2>" + writeXML(theObj.children);
            myString += "</LEVEL2>\n";
        }
    }
    else if (theObj.className == "level3"){
        if (theObj.children == null)
            myString = "<LEVEL3/>\n";
        else {
            myString = "<LEVEL3>" + writeXML(theObj.children);
            myString += "</LEVEL3>\n";
        }
    }
    else if (theObj.className == "level4"){
        if (theObj.children == null)
            myString = "<LEVEL4/>\n";
        else {
            myString = "<LEVEL4>" + writeXML(theObj.children);
            myString += "</LEVEL4>\n";
        }
    }
    else{
```

```
            if (theObj.children == null)
                myString = "<LEVEL1/>\n";
            else {
                myString = "<LEVEL1>" + writeXML(theObj.children);
                myString += "</LEVEL1>\n";
            }
        }
    }
    else{
        if (theObj.children != null)
        myString = writeXML(theObj.children); else return "\nc\n";
    }
    return myString;
}
function writeXML(theObj){
    var myString = "";
    if (theObj == null)
        return myString;
    myString = walkCollection (theObj, doWriteXML);
    return myString;
}
function writeDocument(){
    var myString = "<?XML VERSION='1.0' RMD='NONE'>\n";
    myString += "<OUTLINE>\n";
    myString += writeXML(document.all.Level[0]);
    myString += "</OUTLINE>\n";
    return myString;
}
```

```
function showNewXML(){
    var w = window.open("", "", "resizeable,scrollbars,
        width=640,height=480");
    w.title = "XML DATA";
    w.document.open();
    w.document.write("<plaintext>");
    var xml = writeDocument();
    w.document.write(xml);
    w.document.close();
}
```

B.7 File Seven: keyboard.js

The purpose of the keyboard.js file is exclusively to handle keypresses and route them to the appropriate functions. This can be used for keyboard equivalents to menu commands, or for keyboard-only commands. Unfortunately, Internet Explorer traps most of the standard key combinations (Cntrl-N for new, Cntrl-O for open, etc.) and doesn't let the program access them:

```
function handleKeyPress(){
    if (event.altKey == true){
        handleAltKeyPress(event);
    }
    else if (event.ctrlKey == true){
        handleCtrlKeyPress(event);
    }
    else if (event.shiftKey == true){
        handleShiftKeyPress(event);
    }
```

```
    else
        handleNormalKeyPress(event);
    event.returnValue = true;
    event.cancelBubble = true;
}
function handleAltKeyPress(evt){
    switch(evt.keyCode){
        default:
            handleNormalKeyPress(evt);
            break;
        // If we're not interested in Alt-Key combination,
        // treat as normal keypress
    }
}
function handleCtrlKeyPress(evt){
    switch(evt.keyCode){
        // CTRL-Enter, make new field as lower sibling
        // of current field
        case 10:
            newElement(evt);
            break;
        default:
            handleNormalKeyPress(evt);
            break;// do nothing
    }
} // handleCtrlKeyPress()

function handleShiftKeyPress(evt){
```

```
    switch(evt.keyCode){
        default:
            handleNormalKeyPress(evt);
            break;
    }
}
function handleNormalKeyPress(evt){
    switch(evt.keyCode){
        default:
            break;
            // do nothing
    }
}
```

B.8 File Eight: outline.xml

The outline.xml file contains some sample xml code to use with the FreeWord program. This is where I test possible tags and attributes of those tags, such as creation date and modification date. An area for further research is in the addition of tags which are exclusive to the client or to the server, for security purposes, for instance:

```
<?XML VERSION="1.0" RMD="NONE"?>
<CONTAINER Type="Root" Status="Closed" DateCreated="887380642"
    DateModified="887380899" EditMode="" ID="7130812">
<TITLE> Root Container </TITLE>
<CONTENTS>
<ITEM DateCreated="887380676" DateModified="887380915" EditMode=""
    ID="73850368">Item in the Root </ITEM>
```

```
<CONTAINER Type="Folder" Status="Closed" DateCreated="887380768"
    DateModified="887381377" EditMode="" ID="20316116">
<TITLE> Folder Container </TITLE>
<CONTENTS>
<ITEM DateCreated="887380779" DateModified="887381393" EditMode=""
    ID="61675857"> Item in a Folder </ITEM>
<CONTAINER Type="File" Status="Open" DateCreated="887380790"
    DateModified="887381411" EditMode="" ID="56286423">
<TITLE> File Container</TITLE>
<CONTENTS>
<ITEM DateCreated="887380803" DateModified="887381423" EditMode=""
    ID="71906788"> Item in a File </ITEM>
<CONTAINER Type="Subject" Status="Open" DateCreated="887380816"
    DateModified="887381433" EditMode="" ID="04185241">
<TITLE> Subject </TITLE>
<CONTENTS>
<ITEM DateCreated="887380837" DateModified="887381445" EditMode=""
    ID="62033500"> Item in a Subject </ITEM>
</CONTENTS>
</CONTAINER>
<ITEM DateCreated="887380848" DateModified="887381659" EditMode=""
    ID="32806354"> Item in a File </ITEM>
</CONTENTS>
</CONTAINER>
<ITEM DateCreated="887380863" DateModified="887381456" EditMode=""
    ID="66624426"> Item in a Folder </ITEM>
</CONTENTS>
</CONTAINER>
```

```
<ITEM DateCreated="887380879" DateModified="887381471" EditMode=""
    ID="65914030"> Item in the Root </ITEM>
</CONTENTS>
</CONTAINER>
```