

THE DESIGN AND IMPLEMENTATION OF A WORLD
WIDE WEB NAVIGATION HISTORY TOOL

By

RALPH ANTHONY GRAYSON

Bachelor of Science

Langston University

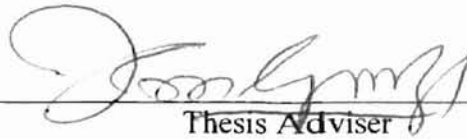
Langston, Oklahoma

1995

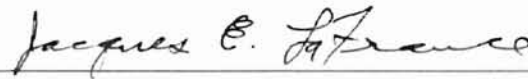
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2000

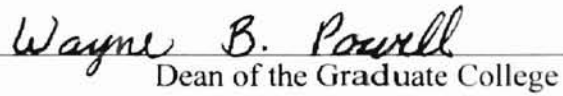
THE DESIGN AND IMPLEMENTATION OF A WORLD
WIDE WEB NAVIGATION HISTORY TOOL

Thesis Approved:


Thesis Adviser






Dean of the Graduate College

ACKNOWLEDGEMENTS

I want express my sincere gratitude to Dr. K. M. George, my principal adviser, for giving me invaluable advice throughout my graduate study. His guidance and generous aid helped make this work possible.

I am grateful to Dr. G. E. Hedrick and Dr. Jacques LaFrance who gave me support and advice to guide me through the thesis writing process. They helped me shape and organize my work.

Last, but certainly not least, I want to thank God for giving his only begotten Son, Jesus Christ, who died for my sin, so that I might have abundant life.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.....	1
Background.....	1
The Problem.....	2
Objective.....	2
Organgization.....	3
II. RELATED WORK.....	4
III. DEVELOPMENT TOOLS.....	11
3.1 HyperTextMarkup Language (HTML).....	11
3.2 JavaScript.....	12
IV. DESIGN AND IMPLEMENTATION.....	14
4.1 The Framework of Persistent History Navigation Assistant.....	14
4.2 Interface.....	16
4.3 Implementation Scheme.....	18
4.4 Events and States.....	19
4.5 PHNA Algorithms.....	21
4.6 Event Loop Algorithm.....	21
4.7 PHNA's ADT.....	22
4.8 PHNA's SAVE_URL Algorithm.....	26
V. CONCUSION.....	31
VI. BIBLIOGRAPHY.....	33
VII. APPENDIX I IMPLEMENTATINON SOURCE CODE.....	35
VIII. APPINDIX II GLOSSARY.....	51

LIST OF FIGURES

Figure	Page
1. User View Model.....	15
2. PHNA Interface.....	17
3. PHNA Interface With Graphical History View.....	18
4. PHNA Event State Diagram	19
5. Event Loop.....	20
6. PHNA's ADT.....	23
7. SAVE_URL Algorithm	24
8a. PHNA Visits Site A.....	29
8b. PHNA Visits Site B.....	29
8c. PHNA Visits Site C.....	30

Chapter I

INTRODUCTION

1.1 Background

The Internet began as ARPANET(Advanced Research Projects Agency Network) a computer network that was originally developed to link research institutions for the U.S. department of defense. This network is no longer limited to research institutions and has grown to millions of sites. Now the Internet is more than a distributed internal network of computers sharing information via a text-based browser [SS97]. It has grown to become a massive loosely configured web of several networks of computers located at sites all around the world including, but not limited to, schools, organizations, corporations, and individual homes [Cap98]. The Internet has grown in span largely due to the advent of the World Wide Web (WWW) authored by Tim Bernes-Lee [BF99]. The World Wide Web gives information a place to persist [BF99]. All of these sites on the WWW contain information that can be viewed in electronic form. In order to view contents of a site visitors use a web browser. Mosaic was the first widely used browser that could read information on the Internet as well as display graphics in a variety of formats. Amaya, Arachne, Opera, Internet Explorer and Netscape Navigator are the names of browsers that have followed Mosaic. One of the most widely used is Netscape Navigator browser. It allows its users to view publicly available information on the Internet using protocols such as HyperText Transfer Protocol (HTTP). The browser also facilitates its user's movement on the WWW by providing a history list, Forward and Back buttons that allow the user to view previously visited sites.

1.2 The Problem

There is a problem with the actual use of the Forward and Back buttons. The browser keeps a list of the sites visited during a particular session. This list is intended to be a complete list of the sites visited by a browser since it began executing. However; often times this is not the case. The Forward and Back buttons are intended to take the user back and forward through the list of sites kept by the browser. In many cases it does not do this. The actions of the Forward and Back buttons on browsers can cause users to experience bewilderment at some point in their sessions by displaying pages that are not expected by the user [HAY99]. A solution to this problem is important since studies have shown that, even though other means of access to history is available, in most cases users use the Back button to access a previously visited site [AHY99]. In addition to the frequent use of the Back button the average American Internet user spends 6.7 hours per week using the Internet [EOR00]. Browser users would greatly benefit from having navigation features that are modeled after the user's view of navigation sessions.

1.3 Objective

Our research aims to provide the users with a new tool to make navigation easier than before and provide history that is well organized. In this research we define a user-view model and present a Persistent History Navigation Assistant (PHNA) based on the model. This tool provides all of the basic features needed in a navigation session modeled after the user's view. This tool maintains a complete list of the user's history during and between sessions. Furthermore, the relative order of visited URLs is also kept. This will give users the freedom to move back and forth between previously visited sites whether

the sites were encountered during the current use of the browser or not. The PHNA gives users a set of buttons for navigational support using the mouse.

1.4 Organization

This thesis is organized in the following way: Chapter II provides a review of related work. Chapter III discusses the use of HTML and JavaScript languages. Chapter IV presents the design and implementation of the world wide web navigational tool. Chapter V concludes the thesis.

Chapter II

RELATED WORK

In the previous chapter we described the problem that is present in existing browsers and a solution to the problem presented in this research. In this chapter we will discuss work that is related to the research undertaken in this thesis. The work of several researchers is considered.

The explosive growth of the World Wide Web makes it difficult for users to locate information that is relevant to her/his'' interest [LEA95]. There are many servers to access and pages to browse [LEA95]. Keeping track of new information as it becomes available online is a consuming task [LEA95]. As such efforts to make technology more manageable are highly in demand [LEA95]. Using advanced information retrieval techniques is one approach to such efforts [LEA95]. Regardless of the potential benefit of these techniques in reducing users' information overload and improving the effectiveness of access to online information, little research has been done on applying them to the World Wide Web [LEA95].

Several WWW resource discovery applications have been built to address the current need to make technology more manageable. These applications while differing in size and effectiveness in varying degrees share one commonality the use of key word searching. This method of searching involves the applications receiving a key word and

using that key word to search their internal indexes for occurrences of that word. When occurrences of the word are found the application shows a list of URLs (Uniform Resource Locator) or page addresses that correspond to the key word used in the search. This method complements browsing or hypertext navigation, which is the dominant access method of WWW users, by giving users of the applications potentially relevant starting points [LEA95].

A major concern with a keyword-based search tool is the design [LEA95]. The design of the search tool should yield an effective tool that will meet the user's information requirements. This involves the choice of the search algorithm and the user-system interaction component [LEA95]. Both of these components are present in successful commercial keyword-based search tools such as Yahoo, Lycos, Excite, and Webcrawler. The components of a keyword-based search tool include the indexer robot, the search engine, and the user interface [LEA95]. The indexer robot contains the indexes that the search tool uses. The search engine calculates the scores of WWW pages in the index given a key-word.

The user interface to the search engine is an HTML form, which can be invoked by standard WWW client programs such as Mosaic, Internet Explorer, or Netscape [LEA95]. The form permits the user to type in a query, execute a search, set the maximum number of hits, access documentation pages, access/run sample queries or saved queries, and invoke other HTML forms for registering a URL or writing comments [LEA95]. Most user interface mechanisms are implemented using the standard Common

Gateway Interface (CGI). After the user of the key-word based tool types in the keyword(s), the query can be sent to the search engine by clicking the designated submit query button. Upon receiving the result form the search engine, the user interface displays a list of URL's and their respective titles ordered in descending relevance scores. At this point the user of the tool can physically access the URL's by clicking on the titles. The user also has access to other information and other facilities.

Organizing a large collection of hypermedia documents is one of many important issues for effective and efficient use of information [CLV99]. Several Studies have proposed various clustering techniques [CLV99]. Clustering hypermedia documents dynamically based on similarity is one proposed solution; however, it has been met with some difficulty. The classification accuracy is highly dependent on the number of documents being classified [CLV99]. Also, finding good labels for selected categories generated based on clustering has proven a problem [CLV99]. Because of the shortcomings the clustering approach has been deemed inferior to the manual classification and labeling approach for navigation [CLV99].

Classification and navigation has become a dominant approach to access information [CLV99]. Several techniques can be applied to the classification and navigation approach. A technique which utilizes external classifiers for classifying and navigating hypermedia documents has been proven to be adaptable and have many desirable properties [CLV99]. Two such desirable properties are breath and depth of classification trees. One application the technique has been applied to is a personal URL

bookmark organizer [CLV99]. In this application the user's bookmarked URLs are classified based on keywords extracted from documents [CLV99]. These bookmarked URLs are organized as a hierarchical structure for efficient access and effective navigation [CLV99].

Another application is image classification [CLV99]. The application extracts keywords around images from HTML documents to query the user for their possible categories [CLV99]. Given this information the application organizes images in a tree structure [CLV99]. This facilitates effective navigation and avoids information clutter. The application further performs image clustering by visual characteristics, such as color and shape. After clustering one representative image is chosen for each cluster [CLV99]. The system displays only the representative images of clusters for each category [CLV99].

Given the potential access to hundreds of millions of pages on the Web, most users have difficulty finding information they require. It is also easy to become entangled in a large and complex web of decentralized, unstructured, and potentially unreliable information [HAY99]. Once the information is found, all of the pages containing relevant information are not readily accessible. Though it is important to find pages that contain useful information, it is also important to support the retrieval of previously accessed information [HAY99]. Tools that manage user histories are powerful. Given that half of accessed Web pages are revisits, according to a paper presented in the 8th international World Wide Web Conference entitled "World Wide Web Information

Retrieval Support Through User Histories" written by Milena Head, Norm Archer, and Yufei Yuan [HAY99].

Most Web browsers have some type of history support within or between navigation sessions that allow users to backtrack. This history support is implemented as a push-down stack [HAY99]. The history stack is not a true trace of the user's navigation pattern [HAY99]. Also, depending on how the page is loaded, it may pop several pages from the Bookmarking and History Lists [HAY99]. A history tool called the Memory Extender Mechanism for Online Searching (MEMOS) for Netscape Navigator developed by Milena Head, Norm Archer and Yuffei Yuan, considers this problem [HAY99]. This tool supports a history list during Navigator sessions [HAY99]. Research that was conducted using MEMOS has shown that users agree that bookmarks are useful history aids for sites that are frequently visited [HAY99]. However, bookmarks do not adequately support less popular sites [HAY99]. It would be unrealistic to suggest that users should bookmark all potentially relevant pages. This approach would fail since the bookmark list would soon become unwieldy and the relevance of a page is often not known until after the page has been shown in the browsing session [HAY99].

The explosive growth of the WWW makes navigating the web to obtain useful information an important issue [CLV99] [HAY99] [GW99]. Navigating the web is mostly a matter of using techniques such as searching for information using keyword and subject based searches [HAY99] [GW99]. However, tools that search are not designed to find the geographic location of information sources [GW99]. With some exceptions most

search tools search the web and return a list of documents that match the keyword or subject used for the search [HAY99] [GW99]. The current capabilities of search tools do not fulfill all the needs of a search tool user. There is no way for a search tool user to find relationships among several results of a keyword search [GW99]. Searches of search tool users were analyzed in a study conducted by Jansen et.al. [BEA98]. This research revealed that search tool users are interested in categories dealing with a geographical location [GW99] [BEA98].

Jayesh Govindarajan and Matthew Ward present a new search tool called GeoViser [GW99]. GeoViser differs from other search tools (e.g. Excite, Yahoo, Infoseek) only in that it provides a map of the United States [GW99]. Points are plotted on this map that correspond to the location of the URL that is given for a particular search [GW99]. This search tool gives valuable information to users who are looking for answers to questions that are location specific.

Another problem presented by the users and providers of the internet is that of consistency [RY99]. When interrelated documents of information are provided over the internet frequent updates of the information becomes an issue [RY99]. Very often users acquire inconsistent information, or they are unable to acquire any information whatsoever [RY99]. One solution proposed for this problem is found in a technique presented by Sampath Rangarajan and Shalini Yajnik of Lucent Technologies and Bell Laboratories respectively [RY99] [RY99]. This technique involves using client side state Hypertext Transfer Protocol (HTTP) cookies [RY99] [RY99] [CSS99]. The histories of

a client's access are kept using cookies using a technique that is based on the Netscape cookie proposal [RY99] [RY99] [CSS99].

CHAPTER III

DEVELOPMENT TOOLS

3.1 HyperText Markup Language (HTML)

The browser is a client-side software application that allows the user to navigate the World Wide Web [BF99]. The browser interprets HTML commands to format documents for the viewer. The browser also gives users the ability to follow links in the documents. The browser interacting with the server accomplishes this. The user initiates a request for information or action and the server interprets the request and takes some action. Among the most popular graphical browsers is Netscape Navigator [BF99].

There are several tools that can be used to create browser enhancements that make the users navigation session more useful including Hypertext Markup Language (HTML) and JavaScript. HTML code uses a set of tags that tell the browser how to format, load and align text and graphics. Tags are commands that define the overall form of the HTML document and give basic structure to the way a page appears. Tags are not visible on the browser, but their effects are. A tag might note that a line should be a title or a heading, for example.

Each tag is enclosed in angled brackets. Paired tags are different in that the last tag has a forward slash just before the command. Commands are not case sensitive, but are usually written in uppercase to promote clarity by making commands easier to spot

when reading an HTML file. Hypertext links are special tags that link one page to another page or resource. When a mouse is placed over a link and clicked, the browser jumps to the link's destination.

3.2 JavaScript

JavaScript is a new technology that was developed initially by Netscape under the name LiveScript. It is intended to extend the capabilities of basic HTML. JavaScript usually resides between the `<SCRIPT>...</SCRIPT>` tags in ordinary HTML documents. It gives developers the ability to write scripts that interact with objects within a web page, such as forms, frames, and background color. In its current state it is more closely linked to Java which is why the name was eventually changed. It is designed to allow logic to exist on the client side to perform tasks such as data validation.

JavaScript is different from Java in that it is not as strict or sophisticated as Java. Java is an object-oriented programming language and JavaScript is object-based. Java has "strong typing (all variable data types must be declared), static binding (object references must exist at compile time), and is compiled into bytecode. The bytecode is then interpreted. In contrast, JavaScript has loose-typing and dynamic binding [FS98]. JavaScript is strictly interpreted not compiled even though the term "JavaScript compiler" is commonly used to refer to the built-in browser mechanism that reads the code and executes it or produces an error message. Both JavaScript and Java can be used to make web pages more sophisticated and exciting by executing the 'local code' [Way97]. The biggest difference is that JavaScript will *only* run on a browser. It is

tightly integrated into HTML whereas Java is simply connected to an HTML document through the <APPLET> tag and is stored in another file.

JavaScript is interpreted [Way97]. Variables and functions can be defined dynamically and used several lines later. There is no compiler or preprocessor [Way97]. The disadvantage of being interpreted is that it takes longer for the code to execute because the browser translates the instructions at runtime just before executing them [WEA97]. The advantage is that it is easier to update the source code. When the script changes in the source HTML file the new code is executed the next time the user accesses the document [Way97].

Other characteristics of JavaScript include its being event-driven [WEA97]. Most JavaScript code is written to respond to events generated by the user or the system. HTML objects, such as buttons, or text fields are enhanced to support event handlers [WEA97].

Finally JavaScript is a good multipurpose tool that allows developers to accomplish many goals. For example, it helps enhance static HTML pages, through special effects, animation, and banners [WEA97]. It permits validation of data without passing everything to the server and is a building block for client/server Web applications. JavaScript serves as a bond between HTML objects, Java applets, and Netscape Plug-ins while providing connectivity without using a Common Gateway Interface [WEA97].

CHAPTER IV

DESIGN AND IMPLEMENTATION

As mentioned in the introduction, the goal of this research is the implementation of a World Wide Web tool to assist browser users. In order to develop the tool, a user view model is introduced. Various aspects in the design and implementation of the PHNA are described in this chapter.

4.1 The Framework of Persistent History Navigation Assistant

In this section we present the basic framework of the PHNA by describing its GUI, algorithms, ADT, and underlying model. The sites visited by web users can be viewed as the nodes of a network. Two relations, front and back, can be defined between two nodes of this graph (shown in figure 1). These relations map to the BACK and FORWARD buttons. Figure 1 illustrates these concepts. We call this a user view model

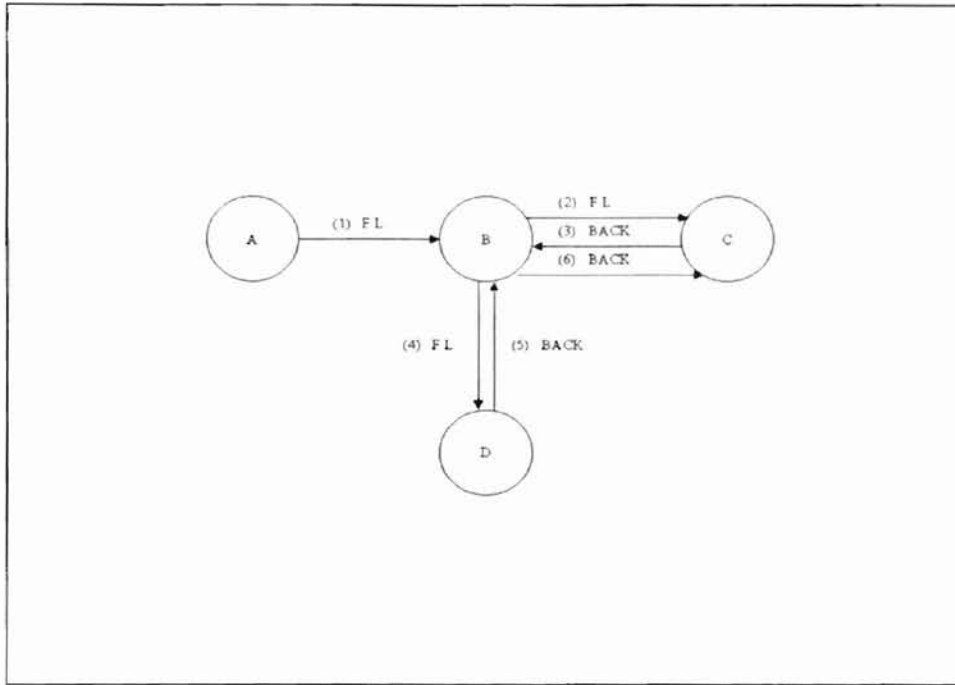


Figure 1. User View Model.

of history as opposed to the current stack model. In this model (illustrated in figure 1) site A is the site that is loaded when the PHNA is started. The user follows a link (F.L.) from site A to site B. The user then follows a link from site B to site C. The BACK button (B.B.) is used to visit site B again. From site B a link is followed to site D. When the user uses the BACK button for the second time he/she is taken to site B. This time when the BACK Button is used the user is taken to site A in most browsers. However, in the PHNA the user is taken to site C. This is the correct action taken by a BACK button modeled after the user's view of history. Current browsers contain BACK and FORWARD buttons, that often times do not match a user's view of the navigation history. The confusion caused by BACK and FORWARD buttons that are not modeled after the user's view is prevented in the PHNA. The BACK and FORWARD buttons

have access to the complete history list. The user can also view the complete navigational history.

The PHNA implements BACK and FORWARD button histories on an inter-sessional basis using *persistent client state HTTP cookies* as well as on an intra-sessional basis using a tree structure implemented using arrays. Previously visited sites can be visited by clicking the BACK or FORWARD button, or by clicking the VIEW button on the PHNA interface which produces a visual model of the tree structure, and allows the user to see how pages relate to one another. Thus, enabling them to make decisions about the path they want to follow. This is in contrast to the stack model of current browsers which do not provide a sense of space or proximity modeled after the user's view. The PHNA utilizes multiple arrays to maintain proper relationships between elements of the tree. A parent array is used to store Uniform Resource Locators (URL). If links have been followed the element stored in the parent array is a reference to a separate child array which in turn contains a reference to the URL. Complete implementation details can be found in section 4.8. We describe the interface in the following section.

4.2 Interface

The interface of the PHNA facilitates its use. It consists of two frames; an upper frame for tool use and a lower frame for browsing. Figure 2 illustrates the PHNA interface. A user may access sites by using the location bar within the upper frame, or by following links. Either method produces an entry into the URL_array. If the user revisits sites by using the BACK and FORWARD buttons, then the site is not added. In this case the user is logically moved backward or forward to the requested site in the tree.

The PHNA considers only the sites actually visited and links followed by the user when preparing the graph in figure 3. It does not make assumptions about issues of relevance. This is appropriate since a large number of users report that finding pages already visited as a problem [BEA99]. This is significant since 88% of individuals pages are revisits [BEA99].

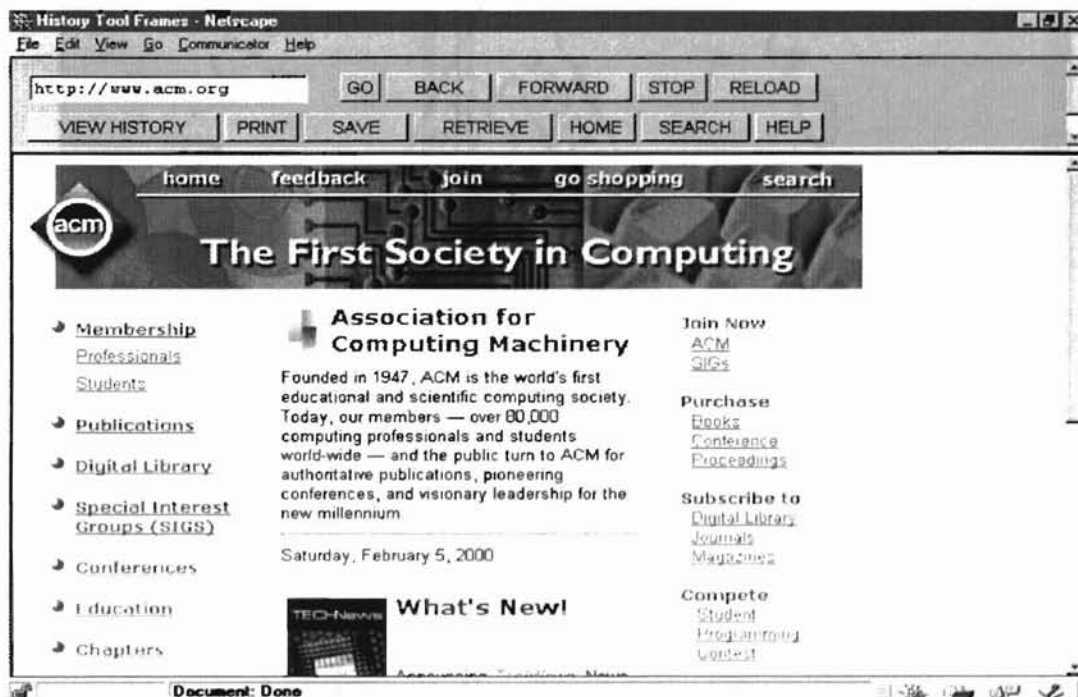


Figure 2. PHNA Interface.

The VIEW button allows a user to view the history list. The list is presented in a tree structure as shown in Figure 3. The tree is generated from the information in the ADT arrays (refer to section 5.2) and is displayed in the lower frame.

Inter-sessional support is provided using cookies when the user clicks the SAVE button. The tree is stored on the users computer for two weeks. To retrieve the tree the user simply has to click the RETRIEVE button and the list will be loaded.

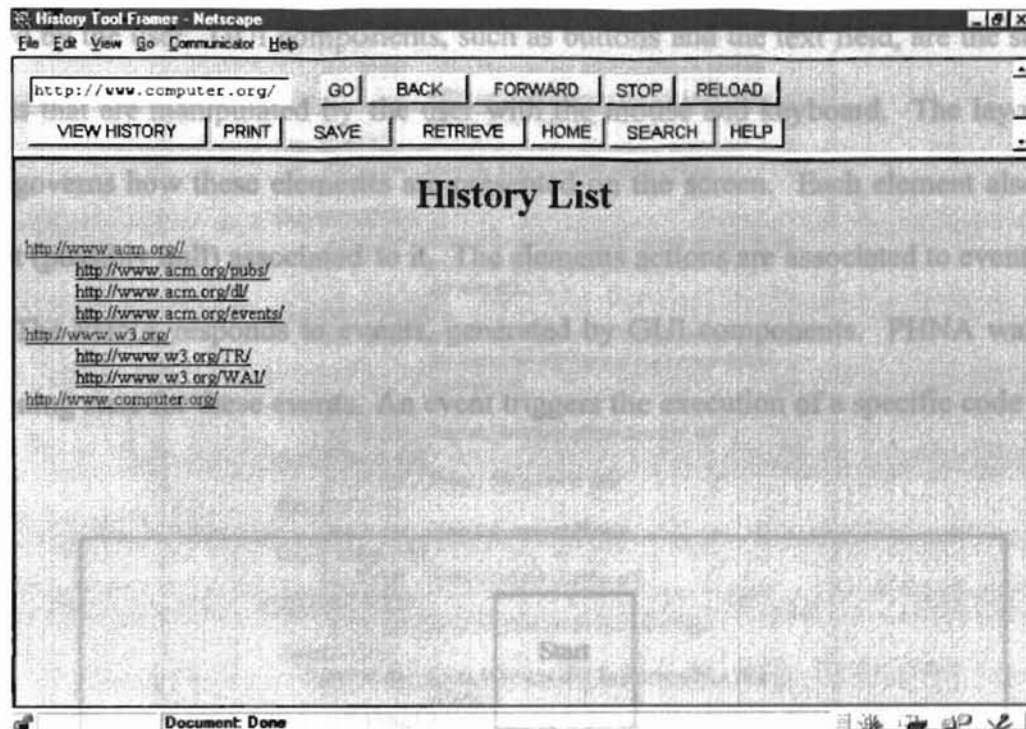


Figure 3. PHNA Interface with Graphical History View.

4.3 Implementation Scheme

In this section we describe the details associated with the implementation of the PHNA. Models and concepts used in designing and implementing PHNA – state transition diagram, event-driven programming, ADT, etc are provided in this section. The state diagram, event-loop, and the ADT and algorithms characterize the implementation. They are described in the following subsections.

4.4 Events and States

The key elements of the PHNA are GUI components, Layout, Listening state, and Event processing. Events signal important user actions like pressing enter key or a mouse click. The listening state is the state in which the program waits for events to be triggered by the user. GUI components, such as buttons and the text field, are the screen elements that are manipulated by the user with the mouse and keyboard. The layout of PHNA governs how these elements are presented on the screen. Each element also has an event (possibly null) associated to it. The elements actions are associated to events.

The PHNA responds to events, generated by GUI components. PHNA waits in the listening state for these events. An event triggers the execution of a specific code

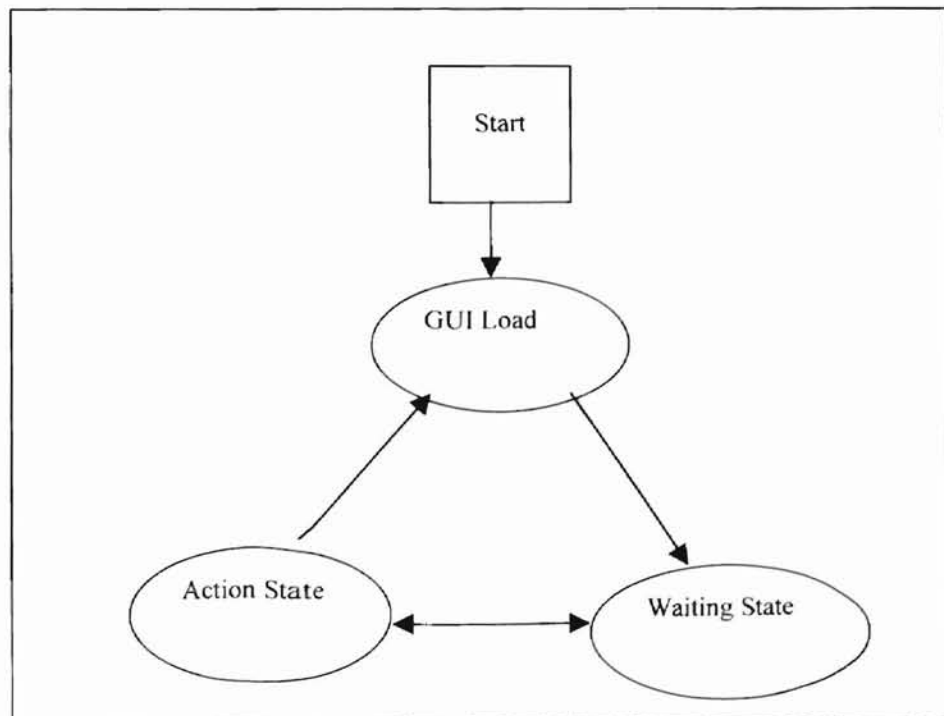


Figure 4. PHNA Event State Diagram


```

While(Event != Quit){
    Case (GO):
        ACTION - Save the current URL and appropriate
                 information and make the viewing frame
                 show the correct uri
    Case (LINK):
        ACTION - Save the links uri and appropriate
                 information
    Case (BACK):
        ACTION - Access the array BACK_array to obtain the
                 index for the logical predecessor of the
                 current uri
    Case (FORWARD):
        ACTION - Access the array FORWARD_array to obtain
                 the index for the logical successor of the
                 current uri
    Case (STOP):
        ACTION - Stop the loading of the current uri
    Case (RELOAD):
        ACTION - Reload the current uri
    Case (PRINT):
        ACTION - Print the current Frame
    Case (SEARCH):
        ACTION - Load a search engine uri
    Case (HOME):
        ACTION - Load the first uri in the uri array
    Case (SAVE):
        ACTION - Save all arrays and index variables to a
                 cookie
    Case (RETRIEVE):
        ACTION - Retrieve arrays and indices from the cookie
    Case (HELP):
        ACTION - Show help file contents
} //End Event Loop

```

Figure 5. Event Loop

segment. When PHNA starts it immediately enters the listening state. When an event occurs, PHNA enters the action state in which events are handled. After the event is handled the program modifies the GUI, if necessary, and returns to the listening state. This process is described in the following section. The event state diagram is shown in Figure 4.

4.5 PHNA Algorithms

The actions of PHNA may be explained via the major algorithms. They are the event loop algorithm and the SAVE_URL algorithm. Events are the result of user actions (e.g. button click or keystroke). They are handled by Javascript built in event handlers which invoke application specific event methods. The event loop algorithm defines the process PHNA uses to handle specific events including the distinct event methods that are called.

The SAVE_URL algorithm defines how URLs are saved. The definition covers all possible instances in which a URL should be saved in order to maintain complete history of all the sites visited during a particular session. This includes URLs that are typed in, site link clicks, and history link and button clicks.

4.6 Event Loop Algorithm

The event loop algorithm (given in Figure 5) calls several application specific methods. The GO application method calls the SAVE_URL application method explained in section 4.8. It passes as an argument the URL that was typed into the text box in PHNA by the user. The GO application method also changes the view frame (refer to Figure 2) in PHNA to the URL that the user entered.

In contrast to the GO method, the LINK application method gets the URL from the view frame. It then calls the SAVE_URL application method and passes this URL as

an argument. The BACK application method changes the site in the view screen by assigning it the predecessor of the current URL. It gets the predecessor of the current URL by calling PHNA's MOVE_BACK method described in section 4.7. The application method named FORWARD gets a successor to the current URL from the MOVE_BACK method. The successor URL is then assigned to the view screen. The STOP application method stops the loading of the current URL. The RELOAD application method loads the current URL in the view screen. It gets the current URL from the GET_CURRENT_URL method explained in section 4.7. The Print application method prints the site shown in the view screen. The SEARCH application method loads search engine site www.netscape.com. Home is an application method that loads the first URL in the history list in the view screen. The SAVE application method saves the PHNA data structure in a cookie by using the SAVE_AS_COOKIE discussed in section 4.7. RETRIEVE is the application method that retrieves and restores the data structure saved by the SAVE_AS_COOKIE. The HELP application shows a series of help screens to assist the user in the use of the PHNA.

4.7 PHNA's ADT

The essence of the PHNA implementation is encapsulated in its ADT. The PHNA Abstract data type (shown in figure 6) is composed of nine private data members and five service methods. The program uses this data type to perform functions that are vital to keeping a complete history list and providing user access to this list in a way that is consistent with the users-view model based view of history. One of the private data members is the dynamic string array URL_array. This array is used to store all of the URLs that are visited. The BACK_array is a dynamic integer array that is used to store

```

Class History{
  Private:
    String URL_array [];
    Int BACK_array[];
    Int FORWARD[];
    int CHILD_Count[];
    String CHILD [[]];
    Int URL_Index;
    int BACK_Index;
    Int FORWARD_Index;
    Int CURRENT_Index;

  Public:
    //Save data members as a cookie
    void SAVE_AS_COOKIE()
    //Restore data members from the cookie
    void RETRIEVE_FROM_COOKIE()
    //Returns logical predecessor of the current url
    String MOVE_BACK()
    //Returns logical successor of the current url
    String MOVE_FORWARD()
    //Saves a url argument and appropriate
    information
    //returns true if successful and false otherwise
    SAVE_URL(String FORMAL_URL, Boolean
    CHILD_URL)
} //End Class History

```

Figure 6. PHNA's ADT

```

//The following method saves the appropriate indices,
//URL and values in arrays
SAVE_URL (String URL, Boolean CHILD_URL){
    //Save the URL to the proper element in URL_array
    URL_array[URL_index] = URL
    If (URL_index > 0){
        //Save the index of the current URL in the proper
        //element of the BACK_array
        BACK_array[BACK_index] = Current
        ++BACK_index
        Current_index = URL_index
        //Save the index of the current URL in the proper
        //element of the FORWARD_array
        FORWARD_array[FORWARD_index] = Current_index
        ++FORWARD_index
    } // End If
    Else
        Current_index = URL_index
    If (CHILD_URL == FALSE){
        //initialize element of CHILD_Count array
        CHILD_Count[URL_index] = 0
        //Create a reference to an array that stores the current
        //URL's children
        CHILD[URL_index] = new array()
        //Initialize the first element of the first element of the new array
        CHILD[URL_index][CHILD_Count[URL_index]] = ""
    } // End If
    Else {
        //Increment element of CHILD_Count array
        ++CHILD_Count[URL_index]
        //Assign the Current URL to the proper location in the CHILD array
        CHILD[URL_index][CHILD_Count[URL_index]] = URL
    } // End Else
    ++URL_index
} // End SAVE_URL

```

Figure 7. SAVE_URL Algorithm

the indices. The indices that are stored in the array are the indices of predecessor URLs. The FORWARD_array is also a dynamic integer array that stores indexes. However; the indices that are stored in this array are indices of successor URLs. The CHILD_Count array is an array that is used to store the number of children of each URL stored in the URL array (or the number of URLs that are visited as a result of following a link from that URL). The CHILD array is a dynamic two-dimensional array that stores the actual child URLs. The first dimension corresponds to the URL in the URL array the second dimension contains the child URL. The URL_index, BACK_index, and FORWARD_index are integer variables used to access the URL_array, BACK_array, and FORWARD_array arrays respectively. However; the CURRENT_index integer variable holds the index of the URL that is currently seen in the view frame.

The MOVE_BACK method uses CURRENT_index to access the BACK_array. This element in the BACK_array is the index of the element in the URL_array that contains the predecessor of the current URL. The predecessor URL is then returned by the MOVE_BACK method.

The MOVE_FORWARD method uses the CURRENT_index to access the FORWARD_array. This access yields the index of the successor to the current URL that is saved in the URL_array. The index is used to obtain the successor URL. The MOVE_FORWARD array then returns this successor URL.

The SAVE_AS_COOKIE method concatenates all of the contents of the data members into a string object and saves the contents of this string object in a cookie. The RETRIEVE_FROM_COOKIE method retrieves the cookie and assigns the contents to a

string object. It then reassigns the values to their appropriate data members. The SAVE_URL method saves URLs according to the user view model.

4.8 PHNA's SAVE_URL Algorithm

The SAVE_URL algorithm (shown in Figure 7) is implemented in the SAVE_URL method. This method facilitates the saving of all URLs visited by the user and ensures that the resulting history preserves the user view model. It accomplishes this by utilizing all the private data members. Figures 8a-8c illustrate the actions performed by the algorithm using three snap shots.

Each visited URL is saved in the first available element in the URL_array. If this is the first URL visited, there is no successor, predecessor, or child URL. We assign the integer variable CURRENT_index the value of the index of the recently saved URL. However, if this is not the first URL visited, then we save the value of CURRENT_index in BACK_array using BACK_index as an index and increment the BACK_index variable. Then assign the index of the element in the URL_array that contains the recently saved URL to CURRENT_index. CURRENT_index now contains the index of the predecessor to the previous URL. Therefore, we assign the value of CURRENT_index to FORWARD_array using the FORWARD_index variable as an index.

If the visited URL is not a child URL then the corresponding element in the CHILD_array is initialized to zero. However, if this URL is a child URL then the value in the corresponding CHILD_Count array is incremented by one. We instantiate an array

object and make the first element in the array reference this array object. Then save the URL at the proper location in the CHILD_URL by using the values in URL_index and the value in the CHILD_COUNT array. We increment the URL_index, BACK_index, and FORWARD_index variables so it may be used to access the next empty slot in the URL_array.

Figures 8.a, 8.b, and 8.c illustrate the effect of the algorithm via an example. We assume that a user visits three different sites. The URLs of the sites are represented by sites A, B, and C respectively. Site A is visited when PHNA starts. The user clicks a link in site A connecting him/her to site B. The user then enters the URL for site C in PHNA's location box and clicks the GO button. In this session URL B is a child of A. A and C do not have any parent URLs. A is the URL of the first site visited. It is saved as the first element of the array URL_array. All elements of the BACK_array, FORWARD_array, CHILD_Count, and the CHILD array remain empty since no URL other than A has been visited. Two private data members of PHNA's ADT (URL_index and BACK_index) have values of one. This is the index of the next empty in the URL_array. Since the first slot in the URL_array contains the URL of the site that is viewed when the PHNA is started. The array BACK_array will never have a value in its first element since there is no predecessor for the first URL. Therefore, the first value stored in Back_array is stored in its second element element BACK_array[1]. CURRENT_index is zero because the URL of the site being viewed (in the view screen) is in the first element in the URL_array (URL_array[0]). FORWARD_index contains the value zero because URL A has no successor.

When the second site is visited its URL (B) is saved in the second element of the URL_array. This site was visited by clicking on a link in the previous site (A). This makes B a child of A. B is saved in the second location in the URL_array. The integer zero is saved in the second location of the BACK_array. Because the predecessor to the URL B is A and the index of the element that contains A in the URL_array is zero. One is saved in the first element of the URL_array since B is the successor of A. The index of the element containing B in the URL_array is one. The first element in the CHILD_Count array is assigned the value of one. B is a child of A so URL A has one child. We instantiate an array and make the first element in the array reference this array object. We then store the URL B and all subsequent child URLs of A in this array object. The second element in the CHILD array remains empty and the second element in CHILD_Count array is assigned the value of zero because B has no children. URL_index is incremented by one so that it can index the next empty element in the URL_array. One is assigned to CURRENT_index since the index needed to access B in the URL_array is one. The variables BACK_index and FORWARD_index are assigned the value of two and one respectively so that they can save the appropriate indices when the next site is visited.

The URL C is saved in the third element of the URL_array when C is entered into the location box of the PHNA and the go button is pressed. The corresponding third element in the BACK_array is assigned the integer one (the index of C's predecessor in the URL_array). The second element in the FORWARD_array is assigned the value of two (the index of B's successor in the URL_array). The third element in Child_count is

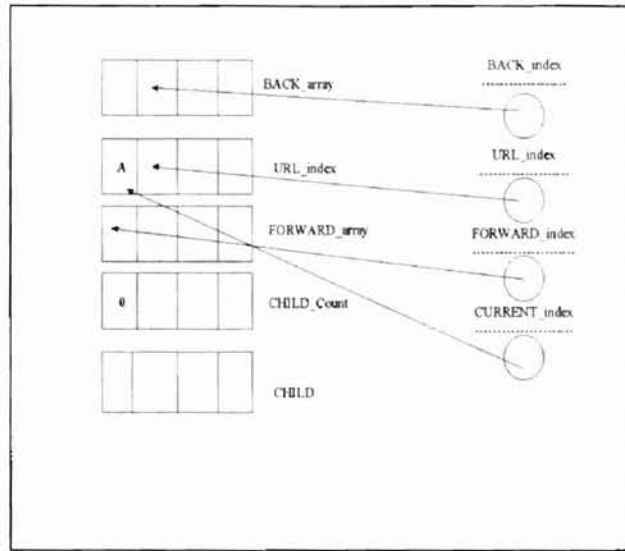


Figure 8a . PHNA state after the initial visit (to site A)

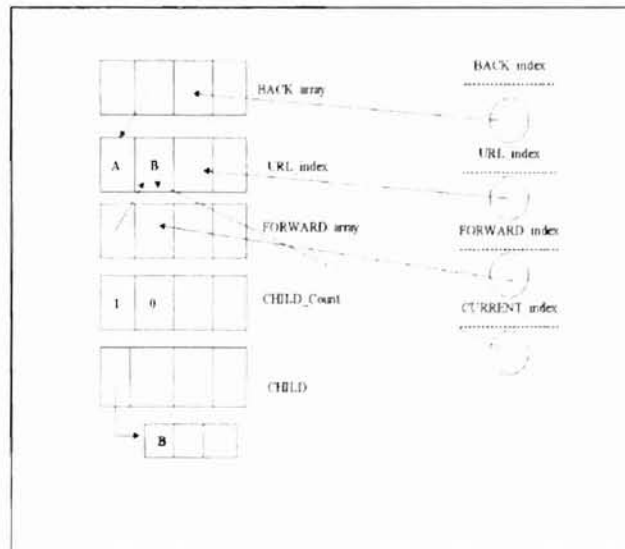


Figure 8b. State of PHNA after viewing another site (Site B)

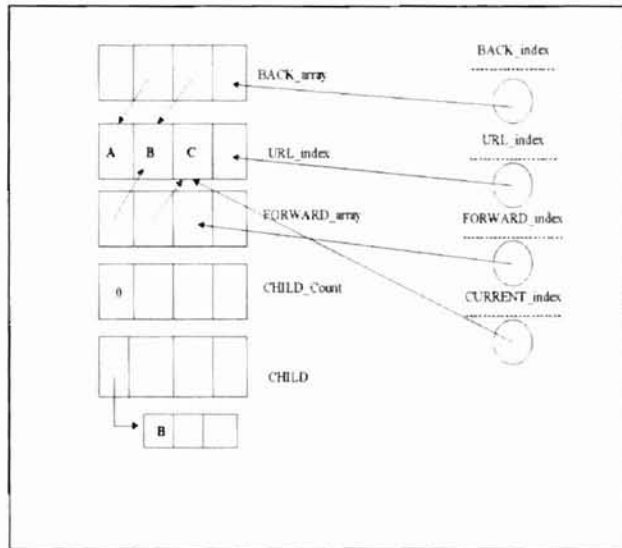


Figure 8c . PHNA state after visiting site C

CHAPTER V

Conclusion

Netscape Navigator™ and other browsers have features that may cause for users confusion. Features such as the BACK and FORWARD buttons on these browsers are intended to help the user navigate the World Wide Web. However, since these features use a history that is not modeled after the user's logical view of their navigational pattern they can cause confusion instead of providing clarity. This confusion is due in part because the history is implemented using a pushdown stack. Depending on how some sites are loaded the browser may pop sites of the stack and never push them back onto the stack. This effectively loses sites by making them makes inaccessible to users through the BACK and FORWARD buttons.

This research presents the PHNA as a solution to the problem of improperly modeled history lists found in browsers. The PHNA provides the user with BACK and FORWARD buttons that are modeled after the user's view of history. Its history is implemented using dynamic arrays. This saves space and allows all visited sites to be saved in a way that corresponds with the user's view of history. Since the PHNA uses arrays to store its history no sites are lost as result of the user going back several sites.

The PHNA can be used with any Javascript™ enabled Netscape browser. Persistent client state HTTP cookies are used to save the complete history list. Which

can be retrieved or overwritten in a later browser session. The PHNA eliminates confusion caused by incomplete history lists that are not modeled after the user's view.

REFERENCES

- [AA99] The Anatomy of an Applet. Java Tutorial,
http://www.gip.jipdec.or.jp/~kato/java_tutorial/applet/antomy/index.html
- [BEA98] Bateman, J., et al., The subjects they search, and sufficiency: A study of a large sample of EXCITE searches, In Proceedings WebNet '98, November, 1998.
- [BF99] Berners-Lee, Tim, Fischetti, Weaving The Web, HaperCollins, 1999
- [CHI96] Chan, P., Hopson, K. C., Ingram S., E., "Developing Professional Java Applets", Sam Publishing, 1996
- [CSS99] "Client Side State - HTTP Cookies",
http://home.netscape.com/newref/std/cookie_spec.html.
- [CLV99] Chang, Edward, Li, Wen-Syan and Quoc Vu,, On Constructing Personalized Navigation Trees for Web Documents, 8th International World Wide Web Conference 1999
- [DB96] Davis, O. McGinn, T., Bhatiani, A., "Instant Java Applets" Ziff-Davis press, 1996
- [DD97] Deitel, H.M., Deitel, P., J. Java How To Program, Prentice Hall, 1997
- [EEA99] Eades, Peter, et. al, Visual Web Browser - mapping and browsing the web with a sense of "space" 8th International World Wide Web Conference 1999, pp. 86-87
- [FEA97] Feather, S., "Java Script by Example". Joseph B. Wilcert and Co., 1997
- [GR96] Gulbransen, D., K. Rawlings, J. December, J.
"Creating Web Applets With Java ", 1996
- [GW99] Govindarajan, J., Ward, M., "GeoViser: Geo-Spatial Clustering and Visualization of Search Engine Results" 8th International World Wide Web Conference 1999
- [HAY99] Archer, N., Head, M., Yuan, Y., Wold Wide Information Retrieval Support Through User Histories, 8th International World Wide Web Conference 1999

- [KM99] Kopetzky, Teodorich, Max Muhlhauser, Visual Preview for Link Traversal on the WWW, 8th International World Wide Web Conference 1999
[<http://www8.org/w8-papers/4b-links/visual/visual.html>]
- [Mcc96] McComb, Gordon. "JavaScript Source Book" John Wiley & Sons, Inc. New York, 1996
- [JVJ97] Java Versus Java Script. Special Edition Using Java, Second Edition. 1996-97
- [LEA95] Lam, Savio L. Y., et al., A World Wide Web Resource Discovery System, WWW 5 1996
- [LL98] Lewis, John & Loftus, Willam Java Software Solutions, Addison Wesley, 1998
- [OA97] Overview of Applets. Java Tutorial. 1996-97
- [PM97] Purell, Lee, Mary Jane Mara, "The ABC's of Java Script" Sybex, Inc. Alameda, CA. 1997
- [Rey96] Reynolds, M. C., Wooldridge, A., "Using JavaScript Special Edition" QUE, 1996
- [RY99] Rangarajan, S., Yajnik, S. WCS: Consistent Update and Retrieval of Documents in a WWW Server, 8th International World Wide Web Conference 1999
- [SS97] Sachs, David and Henry Stair, The 7 Keys to Effective Web Sites, Prentice Hall, 1997.
- [Way97] Wayner, Peter. Java and JavaScript Programming, Academic Press Inc., 1997
- [WEA97] Wagoner, Richard, et al., JavaScript Unleashed, 2nd ed. Sams.net, 1997
- [WJS97] What Is Java Script? Java Script Handbook. 1996-97.

Appendix I

IMPLEMENTATION SOURCE CODE

Start_Frame

```
<html>
<head>
<title>History Tool Start Page</title>
</head>

<body bgColor=Silver>
<SCRIPT LANGUAGE="JavaScript">
//<!--
var newWindow
//Open a new window with no location box
newWindow =
window.open("HistoryFrames_1.html","", "location=0,toolbar=0,resizable=1,scrollbars=1
,status=1,menubar=1")
//close the current window
window.close()
// -->
</SCRIPT>
</body>
</html>
```

Frame_Set

```
<html>
<head>
<title>History Tool Frames</title>
</head>

<Frameset Rows="15%,85%">

<Frame SRC="Hist_tool_1.html" Name = "bridge" >
<Frame SRC="http://a.cs.okstate.edu" Name = "screen">
</Frameset>

</html>
```

Driver_Frame


```

<html>
<head>
<title>History Tool Control Frame</title>
</head>
<body bgcolor=silver>
<SCRIPT LANGUAGE="JavaScript">
<!--
    //Variable to contain the selected text
    var selected_Text= ""
    //Variable that designates a selected text event
    var select = 0
    var part = ""
    var add = "http://"
    var temp = 0
    //Designates the URL change as the result of a link click
    var link = 0
    //Syncohnizes the link_click function with other functions
    var start = 0
    //Array to store URLs
    var URL = new Array()
    //Array to store appropriate BACK url indices
    var B_URL =new Array()
    //Array to store appropriate Forward url indices
    var F_URL = new Array()
    //Array used to store an array of Urls that are result of following a link from
another Url
    var Child =new Array()
    //Array used to store the number of children a Url has
    var Child_Count = new Array()
    //Index for Child array
    var Child_index = 0
    //The URL array will always fill up one element ahead of the F_URL array
    var index = 0
    //The first array element of the B_URL will always be empty
    var B_index = 1
    //The F_URL array will always fill up one element behind the B_URL array
    var F_index = 0
    var temp = 0
    var Back = 1

```

```

var Forward = 0
var Current = 0
// The name of the cookie that is saved and retrieved
var cookie_Name = "PHT_info"

//variable used to store cookie information
var Cookie_comp = " "
var   Cookie_arc = ""
var       urlString = ""

//Assign the first element of the URL array an initial value
URL[index]= "file:///A:\paper_graphic.htm"
//Make a child array for this URL
    Child[0] = new Array()
    Child[0][0] = " "

//Set the number of children this URL has to zero
Child_Count[0] = 0
++index
//Call to get_priv
get_priv()

/*
    This function gains security access and uses a timer to
    call the function load in four seconds.
*/
function get_priv(){

//Obtain Universal Browser Access so the Url of the "Screen" frame can
be viewed

netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserAccess")
//Call the Load function in four seconds
timerID = setInterval('Load()', 4000)

```

```

    }
    /*
        This function clears the previous timer and sets a new one that
        calls the link_click method every two seconds.
    */
    function Load(){

        clearInterval(timerID)
        timerID = setInterval('link_click()', 1000)

    }

    /*
        This function changes the "Screen" frame to a URL by accessing the
        the appropriate element of the Back array and using its value
        as an index to the URL array. The function then changes the
        appropriate arrays and variables.
    */

    function goPrev() {

        //Obtain Universal Browser Access so the Url of the "Screen" frame can
        be viewed

        netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserAccess")

        //Disable the link method
        start = 1
        if (B_index == 1){
            alert("Please visit another site before using the Back button")
        }
        else{
            if (Current == 0){
                alert("You are at the beginning of your history list.")
            }
            else{
                //Assign the current index teh correctindex to the URL
                array
                Back = B_URL[Current]
            }
        }
    }
}

```

```

        //change value of the current variable to represent the
        //appropriate position in the array
        Current = Back
        //Change the "viewing" screen (screen frame)
        //show the correct URL
        parent.frames[1].location.href = URL[Back]
    }//End Second Else
} //End First Else
//Wait for the URL to load into the "screen" frame
    while (parent.frames[1].location.href != URL[Current])
        start = 1
//Enable the link method
start = 0

} //End goPrev

/*
    This function changes the "Screen" frame to a URL by accessing the
    the appropriate element of the Forward array and using its value
    as an index to the URL array. The function then changes the
    appropriate arrays and variables.
*/

function goNext() {

    //Obtain Universal Browser Access so the Url of the "Screen" frame can be
viewed
    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserAccess")

    //Disable the link method
        start = 1

    if (F_index == -1) {
        alert("Please visit a site before you use the FORWARD button.")
    }
    else {
        if (Current >= F_index){

```

```

        alert("You are at the end of your history.")
    }
    else{
        //Assign the current index to the variable Forward
        Forward = F_URL[Current]
        //change value of the current variable to represent
        //appropriate position in the array
        Current = Forward
        //Make the view frame "screen" show the correct
        parent.frames[1].location.href = URL[Forward]
    }//End Second Else
} //End First Else

//Wait for the URL to load into the "screen" frame
while (parent.frames[1].location.href != URL[Current])
    start = 1
//Enable the link method
start = 0
} //End of goNext

/*
   This function obtains the appropriate URL and changes the frame
   "Screen" to that URL and assigns the appropriate values to
   arrays and variables in this program.
*/

function go_location() {
    urlString = ""

    //Obtain security access
    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserAccess")
    //Disable link method
    start = 1
    //Check to see if the go_location method was called as a result
    //of a link click or the use of the location box or button.
    //If the result of a link click then get the URL of the
    //"screen" frame and assign it to the variable urlString.

```

```

//If not then the method was called as a result of the use
//of the location box or button in which case the variable
//urlString is assigned the URL given in the location box.
if (link == 1)
{
    urlString = parent.frames[1].location.href

}
else{
//Assign the value of the Locationbox to the variable
urlString = document.forms[0].LocationBox.value

}

if (urlString != "") {
    //Add the Protocol "http://"
    if (add != urlString.substring(0,7))
    {
//        urlString = add + urlString
    }

    //Add a "/" to the end of the URL if one is not present
    len = urlString.length
    part = urlString.substring(len - 3,len)
    if ((urlString.charAt(len - 1) != "/")&&((part == "com")||(part ==
"edu")||(part == "org")||(part == "gov")))
        urlString = urlString + "/"

//If the call to the method was not a result of the user clicking on a
link

//Make the view frame show the URL in the variable
if(link == 0){
    parent.frames[1].location.href = urlString
}
}

```

```

//Make a child array for this URL
Child[Current] = new Array()

//Set the number of children this URL has to zero
Child_Count[Current] = 0

alert("The value of Child_Count is=")
alert(Child_Count[Current])
}
else{

// Assign the integer in the number of children the URL has to the
// child_index variable so the value can be used to add the current
// the child array of the previous URL
Child_index = Child_Count[Current]

//Add the current URL to the parents Child array
Child[Current][Child_index] = urlString
alert("The value of urlString =")
alert(urlString)
//Add one to the value in the Child_Count array
Child_Count[Current] = Child_Count[Current] + 1

//alert("The value of Child[Current][Child_index] is =")
//alert(Child[Current][Child_index])

} //End Else

```

URL to

```

//Assign the "screen" frame's URL to the URL array
URL[index] = urlString
//Assign the index of URL to the correct position

```

```

        //in the back array
        B_URL[B_index] = Current
        ++B_index
        //Make the current indicate the correct position in the array
Current = index
        //Assign the index of URL to the correct position
        //in the forward array
        //The index is added one element behind the back array and
        //two elements behind the URL array
        //If the use of the "Back" Button does not precede the
        //use of the go to then first condition else second
        //second condition

        if (Current > F_index)
        {
            F_URL[F_index] = index
            ++F_index
        }
        else
        {
            F_URL[Current]

            ++index
            //Wait for the URL to load into the "screen" frame
            while (parent.frames[1].location.href != URL[Current])
                start = 1
            //Enable link method
            start = 0
            link = 0
        }
        else {
            alert("Please enter a URL before clicking the Location button.")
        }

    } //End go_location

    function child(){
}
}

```



```

/*      This function deletes a previous cookie if there is one. Then it
        assigns all the cookie values to a string.
*/
function set_Cookie_Info() {

    //Variable to represent Two_Weeks = 12096 * 100000
    var Two_Weeks = 14 * 24 * 60 * 60 * 10000
    var expr = new Date();

    //Set the time for the cookie to expire as two weeks from the current time
    expr.setTime (expr.getTime() + Two_Weeks)
    //Save cookie information in a string object with a ' as a seperator
    //The join method puts array methods into strings seperated by commas
    Cookie_arc += URL.join() + ""
    Cookie_arc += B_URL.join() + ""
    Cookie_arc += F_URL.join() + ""
    Cookie_arc += index + ""
    Cookie_arc += Current + ""
    Cookie_arc += B_index + ""
    Cookie_arc += F_index + ""

    var expString = "; expires=" + expr.toGMTString()

    document.cookie = cookie_Name + "=" + escape (Cookie_arc) +
expString

    alert(document.cookie)
} // end function setCookie_info

/*This function retrieves all the cookie that has been saved.
It places all the saved cookies in a string variable and search through
this variable for the desired cookie*/

function retrieve_Cookie () {
    var components = null

```

```

//Puts a blank at the beginning of cookie and a semicolon at the end of the
cookies saved
var saved_cookie = " " + document.cookie + ";";
// Creates a search string with the name of the cookie and an equal sign
var search_name = " " + cookie_Name + "="
// Searches through all the active cookies that have been saved on machine
for the search string
var start_cookie = saved_cookie.indexOf(search_name)
// This variable is used to store the index of the end of the desired cookie
var end_cookie
// The if statement puts the desired cookie into a string called components
if (start_cookie != -1) {
    start_cookie += search_name.length
    end_cookie = saved_cookie.indexOf(";", start_cookie)
    components = unescape(saved_cookie.substring(start_cookie,
end_cookie))
} // end if

//alert("This is the retrieved cookie string:" + components)

return components
} // end function getCookie

/*
This function receives the cookie string returned by the
function get cookie. The function then parses the string
values into the appropriate program variables.
*/
function retrieve_Cookie_Components() {

    var tmp_1 = 0
    var tmp_2 = 0
    var tmp_string = new String()
    var cookie_comp = new String()
    alert(document.cookie)
    //Call the get_Cookie function to get desired cookie in
string form
    cookie_comp = retrieve_Cookie()

```

```

//Restore all the cookie values to their appropriate positons
//by assigning the values between the single quotes to their
//appropriate variables
tmp_1 = cookie_comp.indexOf("''")
tmp_string = cookie_comp.substring(0, tmp_1)
URL = tmp_string.split(",")

tmp_1 += 1
tmp_2 = cookie_comp.indexOf("''",tmp_1)
tmp_string = cookie_comp.substring(tmp_1, tmp_2)
B_URL = tmp_string.split(",")

tmp_2 += 1
tmp_1 = cookie_comp.indexOf("''",tmp_2)
tmp_string = cookie_comp.substring(tmp_2, tmp_1)
F_URL = tmp_string.split(",")

tmp_1 += 1
tmp_2 = cookie_comp.indexOf("''",tmp_1)
index = cookie_comp.substring(tmp_1, tmp_2)

tmp_2 += 1
tmp_1 = cookie_comp.indexOf("''",tmp_2)
Current = cookie_comp.substring(tmp_2, tmp_1)

tmp_1 += 1
tmp_2 = cookie_comp.indexOf("''",tmp_1)
B_index = cookie_comp.substring(tmp_1, tmp_2)

tmp_2 += 1
tmp_1 = cookie_comp.indexOf("''",tmp_2)
F_index = cookie_comp.substring(tmp_2, tmp_1)

} // end retrieve_Cookie_Components

/*
This function prints the contents of the URL array to
the "Screen" frame with a go besides the element.
*/

```

```

function print_Hist() {

netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserAccess")

//Clear the current time interval
clearInterval(timerID)

top.frames[1].document.clear()
top.frames[1].document.write("<HTML><HEAD><TITLE>Viewing
Category</TITLE>")
top.frames[1].document.write("</HEAD><BODY><FORM>")
top.frames[1].document.write("<CENTER><H1>History
List</H1></CENTER><BR>")

for (var i = 0; i < URL.length; i++) {
    top.frames[1].document.write(URL[i] + " ")
    top.frames[1].document.write(" <INPUT TYPE=\"button\"
VALUE=\"GO\" onClick=\"top.frames[0].go_hist_location(" + i + ")\">")
    top.frames[1].document.write("<BR><BR>")
//
    alert('inside print_Hist for loop')
} // end for

top.frames[1].document.write("</FORM></BODY></HTML>")
top.frames[1].document.close()
} // end function print_History

/*
This function changes the URL viewed in the child frame to the
URL selected by the user
*/

function go_hist_location(index) {
//Change the "Screen" frame to the URL specified by user
parent.frames[1].location.href = URL[index]

//Set a new time interval for one second
timerID = setInterval('link_click()', 1000)
} // end go_hist_location

/*

```

This function checks to see if a click has occurred on a link in the "Screen" frame. If one has occurred the go_location function is called. If no link has been followed no action is taken.

```
*/
function link_click(){
    //Obtain Universal Browser Access so the Url of the "Screen" frame can
be viewed

    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserAccess")

    if (( parent.frames[1].location.href!= URL[Current])&&(start == 0))
    {
        //Set the link variable to one to indicate that the go_location
//method was called as a result of a the user following a link that it
was
        link = 1
        /* alert ("in go location")*/
        go_location()
    }
}

// -->
</SCRIPT>
<form method="post">
<p>
<table border=0 width=100%>
<tr>
<td valign=top width=30%>
    <input
    name="LocationBox"
    type="text"
    size=25
    maxlength=50
    onChange="go_location()">
</td>

<td valign=top width=70%>
    <input
```

```

        type="button"
        value="GO"
        onClick="go_location()">

<input
  type="button"
  value=" BACK "
  onClick="goPrev()">
<input
  type="button"
  value="FORWARD"
  onClick="goNext()">
<input
  type="button"
  value=STOP>
<input
  type="button"
  value=RELOAD>

</td>
</tr>
<tr></tr>
<tr>
<td colspan = 2>
  <input
    type="button"
    value="VIEW HISTORY"
    onClick="print_Hist()">
  <input
    type="button"
    value=PRINT>
  <input
    type="button"
    value=" SAVE "
    onClick="set_Cookie_Info()">
  <input
    type="button"
    value=" RETRIEVE"
    onClick="retrieve_Cookie_Components()">
  <input
    type="button"

```

```
        value=HOME>
    <input
      type="button"
      value=SEARCH>
    <input
      type="button"
      value=HELP>

</td>
</tr>
</table>
</form></P>
</body>
</html>
```

Appendix II

Glossary

Application	Computer programs that perform useful work not relate to the computer itself.
Cookies:	Cookies are a general mechanism which server side connections (such as CGI scripts) can use to both store and retrieve information on the client side of the connection.
HyperText Markup Language (HTML)	A set of codes that can be inserted into text files to indicate special typefaces, inserted images, and links to other hypertext documents.
HyperText Transfer Protocol (HTTP)	A standard method of publishing information as hypertext in HTML.
Internet	A coperative message-forward system linking computer networks all over the world.
Network	A set of computers connected together.
Uniform Resource Locator (URL)	A way of specifying the location of publicly available information on the Internet.

VITA

Ralph A. Grayson

Candidate of the Degree of

Master of Science

Thesis: THE DESIGN AND IMPLEMENTATION OF A WORLD WIDE WEB
NAVIGATION HISTORY TOOL

Major Field: Computer Science

Biographical:

Personal Data: Born in Oklahoma City, Oklahoma, On June 14, 1973,
the son of Jerry and Leonora Grayson.

Education: Graduated from Boley High School, Boley, Oklahoma in
May 1991; received Bachelor of Science degree in Computer Science from
Langston University, Langston, Oklahoma in July 1995. Completed
requirements for the Master of Science degree with a major in Computer
Science at Oklahoma State University in May 2000.

Experience: Employed by Langston University, Department of
Computer Science as an undergraduate research assistant,
1994 - 1995 and as an Instructor 1997 - present. Employed by
Oklahoma State University, Department of Computer Science as a
Graduate teaching assistant, 1995 - 1997 and as an Instructor
1997 - present.

Professional Memberships: Association of Computing Machinery
(ACM)