A System for Structured Management of Hypermedia Resources for the World Wide Web

Kevin L. Marlowe
Old Dominion University

Follow this and additional works at: https://digitalcommons.odu.edu/computerscience_etds

Part of the Databases and Information Systems Commons, and the Software Engineering Commons

Recommended Citation
https://digitalcommons.odu.edu/computerscience_etds/158
A SYSTEM FOR STRUCTURED MANAGEMENT OF HYPERMEDIA RESOURCES FOR THE WORLD WIDE WEB

by

Kevin L. Marlowe
B.A. May 1987, University of Virginia

A Thesis submitted to the Faculty of Old Dominion University in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
March 1996

Approved by:

Stephan Olariu (Director)

James L. Schwing (Member)

Stewart N.T. Shen (Member)
ABSTRACT

A SYSTEM FOR STRUCTURED MANAGEMENT OF HYPERMEDIA RESOURCES FOR THE WORLD WIDE WEB.

Kevin L. Marlowe
Old Dominion University, 1996
Director: Dr. Stephan Olariu

The World Wide Web (WWW) is arguably the preferred method for disseminating information across the Internet. Most of the work to support the advancement of WWW technology has focused on servers used in storing and retrieving information, browsers for viewing this information, and editors or filters for creating the information.

One area which has received little attention is that of actually managing this information at the local host. This thesis describes the development of a means for organizing local data prior to its publishing on the WWW, a method for gathering local pages together and preparing them for distribution, and a system for managing such data once published. As a proof-of-concept, this thesis describes a prototype software utility, the PageManager, which provides an intuitive, hierarchical method of organizing Hypertext Markup Language (HTML) pages, images, and other file-based objects. The PageManager software supports the management of such objects through the creation, editing and exporting of these page hierarchies.
ACKNOWLEDGMENTS

- Don Randall, Cathy Cronin, and Ray Gates, for having the intellectual spark that eventually became PageManager, and for hearing me out when I wanted to steer the project in a new, previously uncharted direction;

- John Davis, for allowing me the time and professional latitude to work on PageManager when other, more pressing, less interesting projects loomed;

- The industry producing applications for Internet-connected devices, for validating my ideas about a year after I first thought of and started coding them;

- Drs. Jim Schwing and Stewart Shen for their comments and thoughts on the ideas presented herein;

- Dr. Stephan Olariu, for directing my efforts in the writing of this thesis and the completion of requirements for this degree;

- My children Daniel and Shannon for granting me the time to pore over books and papers when they would rather I read them stories, and

- My wonderful wife Jill, M.S., B.S., B.S., for giving me the chance to catch up.

-KLM
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>i</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td><strong>1. INTRODUCTION</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Organization</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Background</td>
<td>4</td>
</tr>
<tr>
<td>1.3.1 The Birth and Growth of the WWW</td>
<td>6</td>
</tr>
<tr>
<td>1.3.2 The Problem</td>
<td>9</td>
</tr>
<tr>
<td>1.3.3 The Need for an Organizer for the WWW</td>
<td>10</td>
</tr>
<tr>
<td><strong>2. FEATURES AND FUNCTIONS</strong></td>
<td>13</td>
</tr>
<tr>
<td>2.1 Features Overview</td>
<td>13</td>
</tr>
<tr>
<td>2.2 Subnotebooks</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Main Window</td>
<td>16</td>
</tr>
<tr>
<td>2.4 Functions</td>
<td>16</td>
</tr>
<tr>
<td>2.4.1 The “File” Menu</td>
<td>16</td>
</tr>
<tr>
<td>2.4.1.1 New</td>
<td>16</td>
</tr>
<tr>
<td>2.4.1.2 Open</td>
<td>17</td>
</tr>
<tr>
<td>2.4.1.3 Save</td>
<td>17</td>
</tr>
<tr>
<td>2.4.1.3.1 Save</td>
<td>17</td>
</tr>
<tr>
<td>2.4.1.3.2 Save As</td>
<td>17</td>
</tr>
<tr>
<td>2.4.1.3.3 Save Subtree</td>
<td>17</td>
</tr>
<tr>
<td>2.4.1.4 Export</td>
<td>17</td>
</tr>
<tr>
<td>2.4.1.5 List</td>
<td>19</td>
</tr>
<tr>
<td>2.4.1.6 Preferences</td>
<td>19</td>
</tr>
<tr>
<td>2.4.2 The “Page” Menu</td>
<td>20</td>
</tr>
</tbody>
</table>
3. IMPLEMENTATION ........................................................................................................... 24

3.1 Background .................................................................................................................. 24
3.2 Data Structures: Selection and Design Issues .......................................................... 25
  3.2.1 The PM Tree ......................................................................................................... 25
    3.2.1.1 Overview ...................................................................................................... 25
    3.2.1.2 Display ........................................................................................................ 25
    3.2.1.3 Orientation and Rotations ......................................................................... 25
    3.2.1.4 Height and Width ....................................................................................... 26
    3.2.1.5 Moving Nodes ........................................................................................... 27
    3.2.1.6 Internals ...................................................................................................... 29
    3.2.1.7 Complexity ................................................................................................. 30
  3.2.2 Globals .................................................................................................................... 32
  3.2.3 User Interface Independence ................................................................................ 33
  3.2.4 Expanding and Collapsing Subnotebooks ........................................................... 34
  3.2.5 Outlining ............................................................................................................... 35
  3.2.6 Export .................................................................................................................. 36
  3.2.7 Import Filenames ("AutoTOC") ......................................................................... 37
3.3 The Application Program Interface ........................................................................... 38
  3.3.1 API Functions ..................................................................................................... 39
3.4 Operating Environment and Component Libraries .................................................. 41
  3.4.1 Operating Environment ..................................................................................... 41
  3.4.2 The HTML Widget ............................................................................................ 41
  3.4.3 Other Libraries ................................................................................................. 41
3.5 Similar Research .......................................................................................................... 42

4. CONCLUSIONS AND FUTURE WORK ........................................................................ 46

REFERENCES .................................................................................................................. 47

APPENDIX I: PAGEMANAGER SCREENSHOTS ................................................................ 50
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PageManager Main Window, bottom</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>Leaf Node with No Siblings</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>Same Node Promoted</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>PageManager Tree Data Structure</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>PageManager Main Window</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>TOC with all data stored in the same notebook</td>
<td>51</td>
</tr>
<tr>
<td>7</td>
<td>TOC with data stored in subnotebooks, selectively displayed</td>
<td>51</td>
</tr>
<tr>
<td>8</td>
<td>TOC with only certain subtrees displayed</td>
<td>52</td>
</tr>
<tr>
<td>9</td>
<td>PageManager Export Dialog</td>
<td>52</td>
</tr>
<tr>
<td>10</td>
<td>PageManager List Dialog</td>
<td>53</td>
</tr>
<tr>
<td>11</td>
<td>PageManager Preferences Dialog</td>
<td>53</td>
</tr>
<tr>
<td>12</td>
<td>Edit Table of Contents Dialog</td>
<td>54</td>
</tr>
<tr>
<td>13</td>
<td>Main Window, node collapsed</td>
<td>54</td>
</tr>
<tr>
<td>14</td>
<td>Main Window, node expanded</td>
<td>54</td>
</tr>
<tr>
<td>15</td>
<td>PageManager Preferences: &quot;None&quot;</td>
<td>55</td>
</tr>
<tr>
<td>16</td>
<td>PageManager Preferences: &quot;Paragraph&quot;</td>
<td>55</td>
</tr>
<tr>
<td>17</td>
<td>PageManager Preferences: &quot;Legal&quot;</td>
<td>56</td>
</tr>
<tr>
<td>18</td>
<td>PageManager Preferences: &quot;Outline&quot;</td>
<td>56</td>
</tr>
<tr>
<td>19</td>
<td>Sample Directory Structure</td>
<td>57</td>
</tr>
</tbody>
</table>
Figure 20: Imported Directory Structure .................................................................58
Figure 21: PageManager Operational Functions .........................................................59
Figure 22: FrontPage Default Home Page .................................................................60
Figure 23: FrontPage Main Window .........................................................................61
Figure 24: FrontPage Interview ..............................................................................61
Figure 25: LiveWire Default Home Page .................................................................62
Figure 26: LiveWire "Guru" ..................................................................................63
Figure 27: LiveWire Site Manager ..........................................................................63
Figure 28: LiveWire Application Manager ..............................................................64
1. INTRODUCTION

1.1 Overview

The World Wide Web (WWW) is a distributed hypermedia infrastructure based on the client-server model. Information disseminated over the WWW typically includes formatted text, images, and, less frequently, executable programs, animations, and other multimedia objects. Additionally, diverse hypermedia resources are accessed via the WWW for ease of use, even when other means are available. These media include USENET newsgroups, and electronic mail. The technical specifications for Uniform Resource Locators (URL), which provide the addressing scheme for the WWW, currently provide for transportation to any of more than fifteen different types of hypermedia objects [9] [10].

A textual object prepared for WWW use is usually written in or translated to the Hypertext Markup Language (HTML), and a "page" written in HTML is practically the smallest possible unit of information that is provided from a WWW server. Typically, an HTML page may include dozens of references to files on the local file system or external systems. Less frequently, the source of a document specified by a URL may be something other than an HTML text page in its native format; however, these objects are provided to client applications as HTML pages, so the term "page" may still be used to represent a hypermedia object on the WWW. One should note, however, that there is not necessarily
any relationship between a physical “page” (i.e., a piece of paper with writing on it) and a WWW “page” (a hypermedia object).

Managing these pages and the references therein can quickly become a nightmare as the number of referenced objects grows into the hundreds. Many authors typically design a directory structure on the local file system and assign a directory to contain the files required for a given page. Depending on the application, they may nest these directories in a manner that is clear to the originator but is not necessarily clear to another reader or associate. At worst, the hierarchy chosen may be entirely useless to others and the pages become out-of-date, their content unmaintainable.

1.2 Organization

In this thesis, the factors leading up to the development of an application (and associated application programming interface, or API) called PageManager are described and discussed. The PageManager is an application that manages data intended for dissemination via the WWW with minimal learning curve for the user. Its development was predicated by a requirement for management of local data prior to its distribution to the general public on the WWW and the PageManager application validates the assumptions made during discussion of this problem. Section 2 presents some background on the project for which this application was developed, although the solution is general; Section 3 gives an overview of the Motif version of the product,
Section 4 describes some details and techniques used in this implementation, and Section 5 mentions some possible enhancements.

The PageManager application software was developed in support of the Scientific Applications Branch, Information Systems and Services Division, NASA Langley Research Center. The contribution of the author of this thesis to the PageManager project was substantial throughout and continues as this product is developed further. Ideas discussed herein were largely generated in group discussions and it is probably impossible to assign credit or blame for any of the individual features of PageManager to any of the participants; however, the participation of the author in these discussions was integral and a significant portion of the code was developed and debugged by the author.

The development of a solution to the problem of organizing data for the WWW brought about the genesis of several tools previously unseen in Web development products. While several of these have appeared in commercial products since they were first devised as part of this research, their application in PageManager is still, in most cases, the first example of their use in the Unix environment. These novel features include (but are not limited to):

- The organization of local files into a hierarchical structure (a "notebook");
- The ability to reorganize a representation of the local “webspace” without actually moving data files on disk, including pruning, moving, and collapsing parts of the data structure ("Move", "Expand/Collapse", "Delete");
• The management of local and remote files of any data types in one hierarchy
  ("Import");
• The customization of the indexing and display of entries in a given hierarchy
  ("Outline Style");
• The automated generation of a representation of the local filesystem for further use
  ("AutoTOC").

While the development of the PageManager application by NASA may continue after the
publication of this thesis, the state of the prototype executable code as described herein
may be considered frozen in its current “beta” state for non-NASA applications. If
enhancements are made and NASA deems the source code publicly releasable, it will be
made available through the Technology Applications Group at the NASA Langley
Research Center. Use of the executable as a prototype for evaluation and testing,
recognizing that it exists as a proof-of-concept and is not distribution-ready, is not
restricted and comments to the author are appreciated and encouraged.

1.3 Background

The NASA Langley Research Center (LaRC) is responsible for supporting aerospace
research in support of national priorities and assisting industry involved in similar
research [15]. Within LaRC, the Information Systems and Services Division (ISSD)
consists of several branches; this work was performed for the Scientific Applications
Branch (SAB) beginning in the summer of 1995.
SAB was tasked with providing data management support for a wind tunnel test which was being conducted by a project team consisting of members from LaRC and the aerospace industry. The researchers needed to archive and access both raw data from the tunnel and information from the daily summary logs in a secure environment. The daily summaries were historically kept in paper notebooks called “logbooks”. A logbook generally contains text, drawings, and plots on paper, which summarize one day’s worth of activities in the tunnel. Since the members of the project team were in physically distant locations, the traditional method of recording information on paper in a logbook was inadequate for effective team communication.

The researchers’ original plan was to photocopy logbook pages, along with any other supporting documentation, fax them to the other team members, and hold a telephone conference to discuss their findings. This method would leave the organization and management of the pages to various individuals, create multiple versions of the same information in various locations and use an insecure method of transmission. Recognizing the shortcomings of this approach, the researchers wanted a technology-based solution that was as easy to use as their original plan.

In a similar vein, it is becoming de rigeur for professors to make class materials of varying types available on a local file system for use by their classes. These materials typically include syllabi, lecture notes, examples, code samples, and presentations. The
standard means of serving these items is to store them in a central directory for copy and ftp access, with the organization dependent on the capabilities of the local system. Typically, some segregation of material by type is performed through the use of the file system’s directory structure, but filenames are as apt to be undescriptive as useful, and there is no good way to ensure presentation of the files in any useful manner (Figure 19). Furthermore, access to these materials cannot be conveniently controlled nor monitored when they are made available in this manner.

The World Wide Web provides a convenient medium for transmission of an electronic version of the logbook, or for serving class materials in a well-organized and maintainable format. In fact, the timely and wide distribution of newly-available information was an intention of the founders of the WWW [2].

1.3.1 The Birth and Growth of the WWW

Before the advent of the WWW in 1990, the distribution of information via the global Internet was accomplished in several disparate ways, depending on the characteristics of the data to be passed. General text and binary data could be passed using a standard “file transfer protocol”, or “ftp”; once received, the recipient was responsible for determining how the information should be viewed and used. Specialized protocols for specialized information were developed, e.g. “archie”, “wais”, and “gopher” for searching indexed remote databases, “simple mail transfer protocol” or “smtp” for transmitting electronic
mail, “news” for group discussions and collaborations. All of these protocols exist and are in general use today.

This content-based method of dealing with data transfer is cumbersome, however. Different client platforms require different programs to deal with each of these types of data, and a sender has no guarantee that a piece of information will appear the same to the recipient without validating the platform that the recipient will use to view it first. Additionally, use of these applications and protocols required some degree of experience or education that kept their use concentrated in the education and government sectors.

In 1990, Tim Berners-Lee and Robert Cailliau, physicists at the European Particle Physics Laboratory (“CERN”) in Geneva perceived a need for a universal client interface to the Internet - a “look and feel” that would be more or less the same on any computer and would obviate the need for content-dependent applications for the viewing of data received. Their original intent was to develop a system for the handling of textual information over CERN’s global network of computers; by October, the general architecture and name “World Wide Web” had been agreed upon and the first general-purpose browsing application was being written for the NeXT operating system.

The concept of “hypertext”, first proposed some twenty years earlier, lent itself to the developing transmission standard. Simple text could be passed from a server to any client “browser” as text; each text item might represent another data item that may or may not
be textual. A browser would know how to handle any given object type by an identifying
tag, or "uniform resource locator" ("URL"). Browsers would handle disparate data types
either by displaying them natively or by passing the data to a "helper application" for
further processing; in either case, the user was relieved of the responsibility of knowing
what application to use to view data of different types. By mid-1991, working browsers
were available for four different computing environments and the specifications for the
WWW were released by CERN.

By early 1993, only about 50 WWW servers were in place worldwide and traffic in the
hypertext transfer protocol ("http") accounted for less than 0.1% of all Internet traffic. In
February, the National Center for Supercomputing Applications at the University of
Illinois at Urbana/Champaign ("NCSA") released the first version of a graphical web
browser ("Xmosaic") written using the Motif widget set for the X Window System. In
September, NCSA released versions of Mosaic for Microsoft Windows and MacOS.
From this point, the popular use of the WWW for distribution of online text and graphics
began to explode. Commercial browsers and mentions in the mass media began to appear
in early 1994, and by January 1995 nearly 20,000 WWW servers and untold clients were
operating on the Internet.

It is of note that the merging of text and graphics into one integrated package for general
Internet distribution and the burgeoning field of desktop publishing entered a phase of
heavy growth at about the same time. With the ability to create high-quality
communications from nearly any computing platform by nearly any user and to distribute those communications globally on demand, the initial growth of the WWW seems assured. In the case at hand, the WWW was an obvious choice for the conversion of paper documents to electronic text that formed the basis for the PageManager project.

1.3.2 The Problem

The logbook problem described above could be separated into three parts:

- The researcher would enter the information onto a page to be included in the electronic notebook;
- The researcher would insert the page into the appropriate section of the notebook;
- The information would be stored and transmitted in an encrypted form.

Entering information on a page would require the researcher to learn the standard "hypertext markup language" ("HTML"), which was unacceptable to research staff at the time. A WYSIWYG editor for HTML was thus required; its development is documented in [6]. Inserting pages into the appropriate section of the electronic notebook would require the researcher to understand the file structure of the local computer system, which was also unacceptable to them; therefore, a system of managing the pages was needed. This system is described herein.
The commercial client required that the information needed to be encrypted when stored and automatically decrypted when accessed. The automated encryption/decryption method has been developed but is not yet documented.

This method of managing information developed and stored independently is sufficiently general that it can easily be extended to other disciplines. In an academic environment, for example, this method could be used to manage the individual components of a course catalog or a set of class notes. As a collaborative tool, a central file repository could contain data files without regard to their naming convention or format; the PageManager would track and organize them; incorporating additions and deletions as required. For the examples in this thesis, the file hierarchy on the author's local system containing the PageManager executable, its source and library directories, and ancillary files is used. For some sections, extracts from filesystems at ODU are used.

1.3.3 The Need for an Organizer for the WWW

In early 1995, when the impetus for the PageManager project was first being discussed, the Web was a relatively young technology with a dearth of user tools. The preeminent browser was NCSA Mosaic 1.0, server software was generally run on Unix machines, and the mass-marketing of the WWW was yet to begin. At the time, there was no commercially-available HTML editor and no means for organizing content intended for publishing on the Web. When the development of PageManager begun, it was generally
understood that, in time, commercial applications would become available that filled its niche.

Today, the commercialization of the WWW and the increasing presence of corporate entities on the Web has fostered the birth of a new genre of software - the "Web publishing kit." In general, these suites of applications are intended as plug-and-play solutions for small businesses looking for a WWW presence with minimal support and effort. In general, an administrator installs WWW publishing software on a PC at the company site, answers a few general questions about the company, and the software configures and installs a Web server and sample HTML pages containing company information. With some effort, an author can customize and add to these pages to produce a customized Web presence.

PageManager's role as an organizer for WWW content has not been replaced by these applications, however. There are two main arenas of difference between PageManager and any of the several commercial Web publishing packages. First, the commercial applications are intended primarily for use with new content sites; they do not directly address the problem of organization of existing sites. A server with significant amounts of existing content would have to be manually ported to the commercial product's organization hierarchy. Although all of these products allow that, none do it automatically as PageManager does. Second, none of these products are currently available for the Unix environment. All are available for PC's, and a few for Macintosh,
but there is no web authoring environment currently available for Unix. This will surely change with time, but PageManager still fills its niche alone.
2. FEATURES AND FUNCTIONS

2.1 Features Overview

The current implementation of the PageManager is on a UNIX platform and provides a graphical user interface (GUI) based on the X Window System and Open Software Foundation’s (OSF) Motif toolkit. While the central concept of this thesis is the set of features that were developed in support of the publishing problem, the prototype PageManager application provides ample illustration of these concepts at work. The features and functions of the PageManager are best described and illustrated by examining functions from this GUI.

The PageManager generates a hierarchical list of files and Uniform Resource Locators (URL’s) that represent the organization of the data. This list of files is referred to as a “table-of-contents” (TOC); the individual files are referred to as “pages”; and a collection of pages is a “notebook”. An entry in a TOC (also referred to as a “node”) may be a file containing HTML text, an image file, a URL without an associated file, or even another notebook (usually called a “subnotebook” when used this way).

The TOC represents a “view” of the notebook, implying that multiple TOC’s can be used to represent alternate views of the same collection of files. A TOC is displayed as an outline in the PageManager main window (Figure 5). Note that the visual representation of the notebook’s TOC resembles a standard HTML “index page”, although the
underlying data structures provide additional information for managing the pages. The user manages the pages in the notebook by manipulating the TOC.

The individual files may reside anywhere on the file system (or may be absent altogether), but the hierarchical display of hypertext describing each file clearly shows the relationship between the files. Adjacent files within a “level” of the hierarchy are usually closely-related by content.

2.2 Subnotebooks

The Subnotebooks facility provides a very useful means for selecting specific data while viewing it in the context of the larger set of related data. For example, consider a hypothetical set of class notes, handouts, displays, etc. stored hierarchically in a PageManager notebook. If a professor teaches several classes, each set of files for each class might be stored in separate directories, perhaps segregating types of files (sample exams, class handouts, etc.) into even more subdirectories. If the user then uses the “Import TOC” facility, the resulting tree is potentially very large and cumbersome (Figure 6).

This representation is almost useless. One could certainly scroll to the page desired, but to do so would be cumbersome and time-consuming. Furthermore, the instructor generating this hierarchy may wish to make some of it available to one class and not to another, or to present the items in the list in a different order.
Two additional operations are provided for tree nodes which are not leaves: Expand and Collapse (see “Expanding and Collapsing Subnotebooks” on page 34). Expanding a subnotebook makes all of its pages part of the current (main) notebook; collapsing removes all pages in an expanded subnotebook from the (main) notebook. Neither operation affects the files on disk. This has the effect of splitting the large tree into several TOC files and allowing a concise view of a given class’ data in the context of the whole, or of providing a top-level directory of all of this instructor’s classes, arranged in an order he considers useful (Figure 7). For students in a certain class, he might wish to provide only a directory of files for that class (Figure 8). This tree efficiently shows where an individual page fits into the whole at a glance.

Also supported is the selective saving and retrieval of subnotebooks. Any node may become the root of a new subtree by selecting it and either collapsing it (which implicitly builds a new subtree) or by choosing the menu selection “Save Subtree” (see “Save Subtree” on page 17).

Functionally, the act of collapsing a subtree actually saves a subnotebook, and expanding a subtree reads the associated subnotebook file. This improves execution time in the long run at the cost of a small amount of disk space. Since the notebook saves files as formatted ASCII, a given page’s record in a notebook file rarely exceeds 128 characters.
2.3 Main Window

The main window is the control center for all actions on the notebook. It provides a palette on which the user may "build" a notebook. A "ROOT" node is maintained as the entry point for adding items to the TOC. It is displayed to give a position for adding new nodes in a new TOC.

At the bottom of the Main Window are text fields marked Reference and Type (Figure 1). These are updated with the motion of the pointer to display the filename and data type of the object under the pointer.

![Reference: HtxAliases.h Type: unknown](image)

Figure 1: PageManager Main Window, bottom

2.4 Functions

2.4.1 The "File" Menu

Items in this menu manipulate the current TOC.

2.4.1.1 New

Clears the current TOC and display, leaving only a new ROOT node.
2.4.1.2 Open

Activates a file selection box for selection of a TOC file to load.

2.4.1.3 Save...

2.4.1.3.1 Save

Saves the current TOC in the current file.

2.4.1.3.2 Save As

Activates a file selection box for designation of a new filename and saves the current TOC in the selected filename.

2.4.1.3.3 Save Subtree

As "Save As", above, but the TOC is saved with the currently selected node as the root. This effectively creates a new subnotebook. This action does not affect the current TOC; only the saved file references the new root node.

2.4.1.4 Export...

Activates the Export dialog box.
Since references in a given page may be local to the host where the selected page was written, they are by their nature non-portable. The Window

Export function parses each HTML page in a given notebook, converts local references to URL’s (based on user-specified environment variables), and saves the updated HTML pages in a specified directory. URL’s to external pages are unaffected. The entire contents of the export directory can then be archived or transferred to the server specified by the user in the environment variables.

To perform an export, the user must specify server information and/or header/footer information via environment variables. They also must select options in the Export dialog (Figure 9). At the bottom of each exported page, a reference to the next/previous page in the notebook and/or a reference to the home page may be created. The home page may be specified to be the first page defined in the table of contents or a standard index page (which resembles the PageManager’s table of contents display). A destination directory where the updated pages will be placed must also be specified.

The URL provided at the bottom of the Export dialog may be used to access the exported notebook, assuming that the user physically moves the exported directory to a location recognizable by the server.

The URL given will not work if the user does not move the whole exported directory to the location they defined in the environment variables $SERVERNAME,
$PORTNUMBER and $URLPATH. These are combined as
http://SERVERNAME:PORTNUMBER/URLPATH and are provided in the Export
dialog for reference. The index page will be named index.html by default.

For more details, see “Export” on page 36.

2.4.1.5 List

The List function creates a dialog box that lists files of a selected type in the current
notebook (Figure 10). This is intended for use in building notebooks with several pages
repeated, such as header pages. Pages in the lister can be viewed by selecting the “Apply”
bUTTON, which starts the WWW browser of choice and displays the page selected.

A possible enhancement might include the ability of the Lister to reference lists of
hyperlinks, as in a browser’s “Hotlist” or some other hierarchical list of URL’s, such as
E-Mosaic’s “Flexible Interest List” [17]. These could then be included in an exported
TOC as if they were originated locally. Additionally, some kind of link between a
browser (i.e. a Netscape “plug-in” or Microsoft Internet Explorer “ActiveX” application)
and PageManager to include the currently active browser page in the PM display might
be convenient.

2.4.1.6 Preferences...

Pops up the Preferences dialog box (Figure 11).
The only Preferences option currently supported changes the display of outlining counters in the main display. Changing these values affects only the display. The nodes themselves and TOC are not modified.

For more details, see “Outlining” on page 35.

2.4.2 The “Page” Menu

Functions in the Page menu manipulate individual notebook pages.

2.4.2.1 Edit...

2.4.2.1.1 TOC Entry...

Creates a dialog that allows editing of the elements in the TOC node structure (Figure 12).

The “Hypertext” field refers to the string that will appear in the PageManager Main Window. This same window is used for the Page/Insert New Page/Import... functions (below) and the hypertext defaults to the name of the file object minus any period-delimited extension.
The file type radio buttons are informational and are used to populate the data structure. Eventually, the code should make an educated guess as to the file type based on its contents.

2.4.2.1.2 Page

Calls the editor of choice, as specified in the environment variable $PAGEEDITOR. This feature is intended for use with HTML or text files; in most cases, choosing this option while the currently-selected page is a binary will not prove productive.

2.4.2.1.3 View

Calls the appropriate viewer, depending on the MIME type of the object selected. If the object is of type URL, the WWW browser of choice is called; it is set by an environment variable $WWW_BROWSER. Other viewers are currently hard-coded.

2.4.2.2 Insert New Page...

The distinction between “inserting” and “importing” is maintained for historical reasons. To “insert” a page means to add a reference to a page that is not in the local file directory, including URL’s. To “import” a page means to add a reference to a page that is local to the PageManager or is already referenced in the notebook.

Insertions at the same level in the node tree are displayed at the same level of indentation in the Main Window. Insertions “under” are indented one level from the currently
selected node. Note that any node can be moved to any indentation level after it is added to the tree, regardless of how it is added.

These functions all use the same dialog as the Edit/TOC Entry... box above.

- Insert After (same level)
- Insert Under (indented)
- Import After (same level)
- Import Under (indented)

2.4.2.3 Expand

A subnotebook can be "expanded" or "collapsed" at will. When a node marking a subnotebook is "expanded", the sub-TOC file is read and all the nodes in it are appended to the current tree at the point where the expanded node is attached.

For more details, see "Expanding and Collapsing Subnotebooks" on page 34.

2.4.2.4 Collapse

Collapse is the opposite of "Expand". If the current node is the root of a subnotebook, the nodes below it are pruned. In the GUI, this appears as if the subnotebook node was "collapsed". Any node with children can be considered a subnotebook and collapsed and later expanded. In this way, large notebook files can be modularized for easier management (Figure and 13).
2.4.2.5 Move...

These functions all move a selected node in the direction indicated relative to the current position in the tree. If the desired move operation is impossible (for example, promoting a node to the same level as the root), no action is taken.

- In ->
- Out <-
- Up
- Down
3. IMPLEMENTATION

3.1 Background

In general, the development team decided upon several paradigms to characterize the entire development effort in support of this project. These goals included:

- Tools developed should be general-purpose
- Where possible, existing infrastructure and tools should be used
- Modularity of system components should be maintained
- Individual system elements should be expandable and/or replaceable with commercial or externally-developed equivalents if desired
- Tools developed must be locally maintainable
- Support for diverse data representations should be provided
- An intuitive, hierarchical management system should form the infrastructure
- System should be easily accessible to entry-level users
- Modules that could not be obtained commercially would be coded locally in C
- Existing libraries and tools would be used as much as possible.

Tools for the structuring of information to assist in its dissemination via the WWW seem few and far between. Some work has been done in the area of expanding and collapsing outlined lists of information [10], but the approach taken here more closely resembles the Microsoft Windows File Manager paradigm [13] for displaying directory structures.
3.2 Data Structures: Selection and Design Issues

3.2.1 The PM Tree

3.2.1.1 Overview

The PageManager uses an "outline" paradigm for internal management of the WWW resources inserted by the user. Internally, HTML pages, images, and other hypermedia resource files are stored in an n-ary tree, with an arbitrary root providing a starting point for any new tree. New pages can be attached as leaf nodes at any point in the tree without disturbing the underlying structure. This "outline" scheme makes the PageManager "notebook" seem like an outline in the traditional sense when viewed sideways, indented appropriately.

3.2.1.2 Display

The hierarchy is stored internally as a tree; the display shows the tree rotated 90 degrees counterclockwise. The "root" of the tree is at the top of the display, with the files ("nodes") on the next indented level representing its immediate children, and so forth. A node at the innermost level of indentation is referred to as a "leaf" node.

3.2.1.3 Orientation and Rotations

The tree, in its default orientation, uses a no-content node as the arbitrary ROOT. The implementation of content storage to this node has been discussed, but not implemented.
This orientation is generally (but not necessarily) a representation of the intuitive structure of the file hierarchy on disk. Experience has shown that users tend to think of the ROOT node as a handle to the topmost file directory and that adding content to it is counterintuitive to most.

Since the ROOT node is null, it is nonsensical to consider the possibility that the null "handle" that it represents could be moved anywhere else in the tree. The act of moving the ROOT node, effectively a rotation of the tree, would serve only to reorder the nodes on the display (and would not affect the disk representation of the content). Since this facility is already available by arbitrary moving of nodes anywhere in the tree the user desires (i.e. there are no constraints on the content of any node in the tree at any position), the implementation of automatic rotations would be redundant and confusing.

3.2.1.4 Height and Width

No practical restrictions are placed on the maximum height or width of the internal representation. In practice, however, a tree height of 34 nodes and a width of 10 nodes fills a default-sized window. Excessive height makes for unwieldy HTML representation upon export, however; careful use of the Collapse facility has been shown in practice to make trees of thousands of nodes workable in one screen-space. Once a tree is exported, users tend to be unhappy about scrolling through large HTML screens, so judicious use of Collapse is recommended.
3.2.1.5 Moving Nodes

The moving of nodes within the tree is supported only as a unary operation; that is, a selected node (pmSelectedTreeNode, a global) is permitted to move no more than one position in any allowable degree of freedom. For example, consider a leaf node with no siblings (Figure 2, node “tell11”). Since the only possible one-position move is to promote the leaf to the same level as its parent, the only PageManager control that has any effect is “Move Out <-”. Selecting this node and moving it results in the display shown in Figure 3.

Figure 2. Leaf Node with No Siblings

Figure 3: Same Node Promoted
In the representation shown in Figure 2, “tell11” is the sole child of the node marked “software” (i.e., “tell11” is the “name” field of the structure pointed to by the leftChild pointer of the node named “software”). The promotion is effected by examining the node that is actually two levels higher (“demos” in this example), determining the correct placement of the node to be promoted (after “software”), and inserting the selected node as the rightSibling of its former parent. A temporary pointer is stored to the new rightSibling of the promoted node until it can be connected.

A node that is in the middle of a group of siblings has potentially four degrees of freedom. Promoting such a node (“tell11” in Figure 3) ("Move Out <-") works as in the example above. Moving “Up” or “Down” exchanges such a node with its leftSibling or rightSibling, respectively, and requires about three lines of code. Demoting a node ("Move In->") is somewhat more complicated. The selected node is changed from the rightSibling of its inorder predecessor to its leftChild; any existing children are moved to become rightSiblings (inorder successors) of the newly demoted node. The former rightSibling of the demoted node becomes the rightSibling of the demoted node’s new parent.
3.2.1.6 Internals

The internal structure of the tree permits the storage of the data associated with a node and the information required for the maintenance of the tree itself in each node (Figure 4):

```c
typedef struct pmtree_info {
    int gridX;
    int gridY;
    int numChildren;
    int start;
    int subtreeWidth;
    int x, y;
} sPMTreeNodeInfo;

typedef struct pmtree_data {
    char *name;
    char *htref;
    char *type;
    int select_flag;
} sPMTreeData;

typedef struct pmtree_node {
    sPMTreeNodeInfo info;
    sPMTreeData data;
    struct pmtree_node *parent;
    struct pmtree_node *leftChild;
    struct pmtree_node *rightSibling;
    struct pmtree_node *leftSibling;
} sPMTreeNode;
```

Figure 4: PageManager Tree Data Structure

Elements in struct sPMTreeNodeInfo are primarily used for determination of where to draw nodes and are not currently used. They are maintained for compatibility with a sister
project, the HTML Page Editor [6]. The elements of sPMTreeData are, in this implementation, passed to editing dialogs for maintenance of individual tree elements; the sPMTreeNode struct incorporates the above along with pointers to neighboring nodes to make each node a complete entity. Note that no pointer to a “rightChild” is provided, since any child of a given node can be located by starting with its leftmost child (node->leftChild) and moving to its right sibling (node->leftChild->rightSibling).

3.2.1.7 Complexity

The choice of data structures is a major factor in the design process. Early in the development of PageManager it became apparent that the data structures selected for use needed to satisfy a number of apparently conflicting goals:

• data access needed to be fast, when viewed from the users’ perspective;
• the coding complexity of the structures had to be relatively simple to accommodate future maintainance of the system;
• structures chose needed to be expandable to accomodate future enhancements without necessitating a complete rewrite of the code.

Since implementing an internal search engine was not an initial design goal (although it was anticipated as a future enhancement), some flexibility was possible in the decision behind a general data structure. It was obvious from the start that whatever data structure was chosen would be implemented as a linked list of nodes [18]; some deliberation narrowed the choices to binary search trees, either balanced or unbalanced [1].
The most elegant solution to this problem would have been the implementation of a balanced binary search tree [11]. While the ideal binary search tree has a performance of $\Theta(lg n)$, where $n$ is the number of nodes in the tree, there is a rather large overhead involved in maintaining the tree "in balance". On the other hand, Knuth has shown [11] that the performance of random binary search tree is, on the average, only 28% worse that that of ideal binary search trees. In other words, the average depth of a randomly created binary search tree is only about 28% higher than that of the ideal such tree. The author’s suspicions, based on the actual structure of some sample trees constructed during development, were that actual user-generated trees would have an even lower average depth than an optimal tree.

Experiments that we have conducted with the application at hand confirmed that this is, indeed, the case. As a result, we have decided to forgo all attempts at rebalancing the resulting search trees. Development of the trees in a more balanced manner without maintaining strict balance, i.e. as AVL trees [12], would provide some algorithmic quality, but the savings would probably not outweigh the effort required to implement. Furthermore, there is an intuitive advantage to maintaining the internal tree with an identical structure to the apparent file hierarchy.

Yet another important practical aspect that diverges somewhat from theoretical results on data structures is that in case the data structures involved contain a relatively small
number of elements, even more direct implementations are of interest. The author
suspects that searches, when implemented, will generally find their target "close to" the
currently selected node. Users may search for nodes for many reasons, but one of the
most common searches will probably be to effect similar edits on files in the local
directory. It would be academically interesting to store a binary, sorted representation of
the tree internally and exercise searches on it, then translating the results to the unordered
display tree. However, the size of typical trees is sufficiently small that \( \Theta(n) \) searches will
not be appreciably slow; furthermore, the overhead to store a sorted tree to effect a faster
search would probably exceed the savings in search time for the small trees we typically
see. An excellent discussion of the tradeoffs inherent in internal sorting and disk access is
presented in [7]. The largest tree developed in trials to date (about 14,000 nodes) can
typically be parsed inorder in about 80 seconds during average machine load conditions;
the time it would take to create an internal, sorted tree of this size during its initial
building is no better than \( a[n(lg n)] \), where \( a \) is the machine overhead to add a single node
to a internal tree. For a tree of the size described above, \( a \) would have to be less than
about .04ms to make an arbitrary session with ten searches more efficient; it seems
reasonable to assume that \( a \) is typically greater than that.

3.2.2 Globals

Three global variables are maintained by the API: the first, "sPMTreeNode
*pmtreeRootNode", always points to the root node of the current tree. If the tree is empty,
it is still initialized (i.e., there are no "empty" trees; a tree has at least one node). The
second, "int TOC_mod_save_flag" is set positive if the current tree has been modified since it was last saved. This permits some communication among modules as to the current state of the notebook.

Finally, a global error structure is maintained for tracking of the current error state of the application. Errors are categorized as either PMWARNING or PMFATAL and the handling of these error conditions is left to the interface to manage.

3.2.3 User Interface Independence

A major advantage to this type of representation is that it is independent of the underlying GUI. While current efforts focus on the Motif implementation, a text-only version of the application is being planned. Since the location of a node in the tree is independent of its representation on a screen, a PageManager user on a character-based terminal could presumably perform most of the functions that a user in a graphical environment could. The author has endeavored to allow maximum functionality for the hypothetical user in the minimal environment.

The treatment of nodes as being of heterogeneous data types (struct sPMTreeData->type) allows avoidance of hard-coded support logic for the file types that are anticipated at compile time. While the current version does, in fact, support several file, the PageManager references the client's MIME configuration files to determine how to
manage any individual page. Thus a user's image viewer of choice can be used for
display of a node of type "image".

Another potential advantage of the outline structure described above is the similarity it
shares with a generic directory structure. To a user accustomed to "managing" files by
storing them in cascading directories, the outline provides a familiar way of looking at
their data. An enhancement is being discussed to store files in a directory structure
mimicking the tree as viewed; see "Conclusions and Future Work" on page 42.

3.2.4 Expanding and Collapsing Subnotebooks

Individual notebooks are stored on the file system as "tables-of-contents" (TOC) in files
typically identified with the suffix.toc. A TOC entry contains all the information
necessary to reconstruct its place in the tree, including its name, hypertext reference, and
filename or URL. An indentation scheme in the TOC files allows the parser to reconstruct
the tree as it appeared when it was stored.

Since a given TOC file does not contain any information relative to the specific ordering
of nodes in the global set of all trees, any TOC may be used as a node in another TOC.
This recursive storage of successively deeper levels of trees provides a mechanism for
specifying a desired view of the data incorporated in a notebook. Any node with children
may be represented as the root of a "subnotebook" with all the properties of the parent
notebook.
No distinction is made between subnotebooks and leaf nodes in the main display. This display would look the same even if all of the nodes shown were subnotebooks.

Likewise, there is no distinction between types of file objects in the notebook at this level of abstraction. The currently selected node is designated by a dashed line appearing under its hypertext marker in the main window. Depth in the tree is denoted by indentation and graphic symbols to the left of each entry, which may be toggled off or to one of a number of standard outline styles.

Expanding a subnotebook imports all of its nodes into the current tree; collapsing prunes the tree at the selected point.

For more details, see “The Page Menu” on page 20.

3.2.5 Outlining

Several outlining styles (methods of numbering lines in the TOC) are available (Figures 15 through 18). These “standard” styles are borrowed from WordPerfect and are consistent with WordPerfect and the U.S. Government Style Manual [19] down to the eighth level of any given tree, where the order repeats. For more details, see “The “File” Menu” on page 16.
3.2.6 Export

The Export concept provided in the PageManager was designed to allow a user flexibility in managing pages of information and disseminating that information on the WWW. A user may create a notebook of information in a user-accessible area (i.e., their user directory). A user may then select a location where the information will be placed to allow WWW access.

The Export process gathers the information together and places it in the user-selected location. The Export function operates on the current table of contents. A user may provide different versions of the information to different locations using multiple TOC’s. The power of multiple TOC’s is evident when a user is maintaining related information that needs to be shared with different users. For example, the notebook maintainer may need to disperse certain information to one set of users, other information to another set of users, and possibly another set of information to both sets of users. The maintainer of the information would have access to all of the data, but could set up and export different versions of the information through different table of contents.

The Export function traverses the Page Manager tree, locating each HTML page, processing each reference on that page, and then saving the updated page to a user-selected area. The processing of a page includes:

- Correcting each reference to point to the user-selected area
- Adding a header/footer to the page
• Adding user-selected references at the bottom of the page
• Return to home page, move forward a page, or move backward a page

Source libraries for handling of HTML tags and features were developed in conjunction with the BullDozer HTML editor project [6]. The page processing function uses routines to load an HTML document into a tree structure, locate a URL within the tree, manipulate the URL information in the tree, and write the internal tree structure into an HTML file.

It seems reasonable that some ease-of-use enhancing features could be added to the Export function. Currently, PageManager includes functions that add external references and custome headers and footers to an exported page. Stewart Shen [17] discusses the facility of making resource discovery engines and WWW directories easily available to users, and these could easily be integrated into the Export function to allow their inclusion in an exported page. This would be especially useful if they could be tailored to specific search engines or categories; i.e. an exported series of documents containing academic data could include an automatic link to a search engine that could be passed a search string to reference related data.

3.2.7 Import Filenames ("AutoTOC")

Recognizing that PageManager may be called upon to manage the structure of an existing Web site, the AutoTOC function reads a directory structure (Figure 19) and replicates it in the PageManager main window (Figure 20). All filenames in the given directory and
all of its subdirectories are extracted and TOC entries built. Subdirectories are shown in
the PageManager display as levels of indentation from the ROOT. The application attempts
to discern the type of the file being referenced from its name and assigns a type of
"Unknown" if necessary. A user can then modify the TOC as required, pruning and
moving entries, until the final structure matches the presence desired.

3.3 The Application Program Interface

In the initial implementation of the PageManager, the PageManager operations were
integrated with the user interface. This approach was acceptable for users running in a
UNIX computing environment using the X Window System and the OSF/Motif toolkit,
but too restrictive for users running on non-UNIX platforms or those desiring an
alternative user interface. For this reason, a C language Application Program Interface
(API) for the PageManager has been developed which is both platform- and user
interface-independent.

The current X/Motif (under UNIX) implementation of the PageManager has been re-
written using this API. It is anticipated that alternative user interfaces to the PageManager
functions on UNIX or cross-platform ports may be developed using this API in the future.
A command line version and an interface based on the HTML forms mechanisms are
definite possibilities.
3.3.1 API Functions

The API provides functions for all the major PageManager operations as well as a collection of support utilities to be used by the application programmer. The API also defines major data structures, specifies a method for error identification and includes other indirectly-referenced support libraries. Although it is beyond the scope of this thesis to provide a detailed discussion of the API, a list of PageManager operator functions which follows may be compared with menu structure described in Section 3. PageManager operational function API's are provided in Figure 21.

These are generally self-explanatory. The functions above make liberal use of the PageManager utility library, which is also part of the API. PMUtil includes the following functions:

- **PMCatRef**
  
  Determines the number of absolute URL's, relative URL's, and local file references in a given notebook.

- **PMFindNode**
  
  Returns a pointer to a page in the notebook determined by a search criteria passed to the procedure. Currently, defined searches include internal notebook label, hypertext, or URL/filename.

- **PMGetErrorInfo**
  
  Error information is defined in a header file, PMerr.h. Errors are maintained in a global variable whose value can be access through this procedure.
- **PMGetFileType**
  Returns the data type of the file whose name is passed. Valid types include HTML, image (binary), ASCII, PostScript, TOC, URL, and unknown.

- **PMGetNextPage**
  Returns a pointer to the next node in a preorder traversal of the PM tree.

- **PMGetTreeStatus**
  Boolean; returns false if the notebook has changed since it was last saved.

- **PMIsFileInNB**
  Returns positive if the file passed is in the current notebook and is in the same directory as the notebook index ("local"); zero if the file passed is in the current notebook and is in a different directory from the index ("external"); negative if the file is not in the current notebook.

- **PMIsFileURL**
  Returns positive if the filename passed is a valid URL (syntactically correct, not necessarily accessible), zero if invalid, and negative if indeterminable.

- **PMIsSubTree**
  Boolean; returns true if the passed page has children.

- **PMIsValidFileType**
  Boolean; returns true if the passed file type is valid.

- **PMNameNode**
  Populates a given tree node with the information required to make it valid.

- **PMValidatePage**
3.4 Operating Environment and Component Libraries

3.4.1 Operating Environment

This application was written to run under version 5 of the X Window System (X11R5), using version 1.1 of the OSF/Motif widget library. The source code is written in C and compiles under SunOS 4.1.3 using gcc version 2.6.3.

3.4.2 The HTML Widget

The implementation of PageManager for Motif takes advantage of the capabilities of NCSA's HTML Widget [14] for displaying node information. The tree is parsed and written to a file in HTML; the contents are then passed to the HTML widget for display. The appearance of the outline on the PageManager display is thus limited to the capabilities of the HTML widget, ibid.

3.4.3 Other Libraries

Several external libraries for string manipulation, tree node manipulation, and memory management were written by Rick Boykin of Computer Sciences Corporation in conjunction with the project and are referenced by this application.
Scott Deerwester of the University of Chicago has developed a program called number [8] that counts in different languages, given a grammar and an integer. The application ships with support for nearly twenty languages, including Cantonese, Japanese, and Esperanto; a grammar was locally developed to describe the counting sequence in roman numbers (both upper- and lower-case) and it is used in the outlining code. A user who required a different numbering scheme could, with a little effort, write a grammar that described any regular numbering scheme and PageManager could then number in that format.

A portable library of short routines to standardize an application’s interaction with the file system on UNIX machines was developed in conjunction with another concurrent effort and used in PageManager. Use of these routines effectively isolates application code from the operating system.

3.5 Similar Research

The topic of authoring (including publishing) on the WWW is burgeoning and has immediate relevance to Internet commerce. The popular Yahoo! index of WWW sites (<http://www.yahoo.com>) lists some 30 sites in its “Computers and Internet:World Wide Web:Authoring” topic category. The Fourth International Conference on the WWW, held last December in Boston, devoted two tutorial sessions to and presented more than five papers on WWW authoring. Clearly, some interest in the problem of organizing Webspace exists.
Several approaches to the organization problem are discussed in the literature. The WebMake environment described in [2] addresses the issue of organizing software in development into a graph that can be modified by a collaborating team of researchers and built in place. This approach supersedes efforts in [9] to organize source code and adds compilation and linking support. It is similar to PageManager in that it permits reorganization of the source hierarchy into user-specific notebooks ("views"), but does not integrate editing and viewing capabilities in the browser.

James Pitkow of Georgia Tech describes [16] a centralized authoring environment similar to PageManager; in Publish, an author submits an input file name and URL via an HTML form to a central repository. The central site verifies the integrity of embedded hyperlinks, validates HTML source in the submitted page, extracts embedded hyperlinks into a database, and finally copies the submitted source to a server directory. It also maintains some version control and permits updates when requested by the author. This methodology is considerably more workgroup-oriented than is PageManager, which was designed for small groups but works well for individual use. The process of copying source files makes some scheme for version control necessary in Publish; in PageManager, source files which are subject to modification can be "linked" into the notebook and version control exercised by the author.
Several authoring environments for the PC platform have appeared on the market. These generally take advantage of the comfortable MS- Windows operating environment look and feel, and seem to be targeted to the rapid prototyping of corporate web presences. These are also the most well-developed of all the authoring tools, usually integrating a suite including an editor, a manager, and often a WWW server into one package.

The two giants in the PC Internet software market, Microsoft Corp. and Netscape Communications, both have offerings for the Intel platform in this market. Microsoft’s purchase of Vermeer Technologies brought Vermeer’s FrontPage authoring environment into the Microsoft Office suite. Netscape’s LiveWire provides an authoring environment tailored to Netscape’s Communications Server and medium-sized businesses.

FrontPage was one of the first authoring suites on the market and shared many features with PageManager in its early incarnations; its development has far exceeded the capabilities of PageManager of late, however, with its integrated PC-based HTTP server, management tools, clean user interface, and similarity in “look and feel” to most MS- Windows applications. FrontPage manages each site’s WWW presence as a “web”, giving the user a graphical display of the web as currently installed as well as authoring tools for individual pages. The initial setup is small-company oriented, with fill-in-the-blank pages and an initial interview to enter the most common information directly into the pages as they are generated. The main desktop of FrontPage is not unlike PageManager’s hierarchical list of component files, and it adds a “web-like”
representation (really a graph) of the entire "site". Including a built-in HTML editor that is full-featured and easy to use, FrontPage is an excellent authoring environment. Its main limitations include its availability only in the MS-Windows operating environment, its lack of support for preexisting "webs", and its insistence on using its own web server during page development. It is reasonable to expect a port to MacOS (but probably not Unix) in the near future. Sample screens from Microsoft FrontPage are included in Appendix II.

Netscape's LiveWire places more emphasis on the management of the site than on its development. Like PageManager, special attention is paid to eventual detachment of a hierarchy of files from the place where they were developed and their reinstallation elsewhere; Netscape calls this "site management" and LiveWire maintains the integrity of internal hyperlinks despite moves; a feature PageManager does not yet incorporate. Since Netscape's WWW server and HTML editing tools are already available separately (as Netscape Communications Server, Netscape Commerce Server, and Netscape Navigator Gold), these tools are not included with LiveWire, but work well with it if installed. LiveWire also adds value by including several sample applications that make use of complex cgi scripts and server capabilities not often exercised; these applications can be copied into pages and reused or modified. Several screen examples from LiveWire (beta release) are included in Appendix III.
4. CONCLUSIONS AND FUTURE WORK

Sophisticated WWW servers, browsers and editors do not provide the information management functionality required for certain projects and by some users. The management of HTML pages, as well as images and other WWW supported information types, is typically a manual process left to the individual using the features of the native file system. The PageManager addresses some of these information management concerns by allowing for the construction, editing and exporting of hierarchical organizations of HTML pages, images, etc.

Future work on the PageManager will focus on expanding the functionality of the API and on the development of alternative user interfaces. In the case of expanding capability, additional editing functionality, such as a more general “Move”, and improved validation of external references in both imported and exported pages would be beneficial. With respect to alternative user interfaces, a command-line interface could provide for background, as well as interactive, execution, and may be a first step towards a cross-platform solution.
REFERENCES


Any trade names used are the property of their respective owners.
Figure 5: PageManager Main Window
Figure 6: TOC with all data stored in the same notebook

Figure 7: TOC with data stored in subnotebooks, selectively displayed
Figure 8: TOC with only certain subtrees displayed

Figure 9: PageManager Export Dialog
Figure 10: PageManager List Dialog

Figure 11: PageManager Preferences Dialog
Figure 12: Edit Table of Contents Dialog

- PM API(…)
- PM XMotif
  - src
    - BX convenience
    - PMmain
    - Makefile

Figure 13: Main Window, node collapsed

- PM API
  - doc
    - include(…)
  - src(…)
  - lib(…)
- PM XMotif
  - src
    - BX convenience
    - PMmain
    - Makefile

Figure 14: Main Window, node expanded
Figure 15: PageManager Preferences: "None"

Figure 16: PageManager Preferences: "Paragraph"
Figure 17: PageManager Preferences: "Legal"

Figure 18: PageManager Preferences: "Outline"
<table>
<thead>
<tr>
<th>demos/</th>
<th>older/</th>
<th>review-q.txt</th>
<th>textfiles/</th>
</tr>
</thead>
<tbody>
<tr>
<td>old/</td>
<td>private/</td>
<td>syllabus/</td>
<td></td>
</tr>
</tbody>
</table>

**demos:**

<table>
<thead>
<tr>
<th>0intro/</th>
<th>lovervu/</th>
<th>2mgmt/</th>
<th>3testing/</th>
<th>software/</th>
</tr>
</thead>
</table>

**demos/0intro:**

<table>
<thead>
<tr>
<th>intro.cfg</th>
<th>intro2.tll</th>
<th>nosilv1.otl</th>
<th>spiral2.tll</th>
<th>swprcs1.otl</th>
</tr>
</thead>
<tbody>
<tr>
<td>intro.go2</td>
<td>intro3.otl</td>
<td>nosilv1.tll</td>
<td>spiral3.tll</td>
<td>swprcs1.tll</td>
</tr>
<tr>
<td>intro1.otl</td>
<td>intro3.tll</td>
<td>nosilv2.otl</td>
<td>spiral4.tll</td>
<td>waterfall.tll</td>
</tr>
<tr>
<td>intro1.tll</td>
<td>maturity.otl</td>
<td>nosilv2.tll</td>
<td>swprcs.cfg</td>
<td>waterfall2.tll</td>
</tr>
<tr>
<td>intro2.otl</td>
<td>maturity.tll</td>
<td>spiral1.tll</td>
<td>swprcs.go2</td>
<td>waterfall3.tll</td>
</tr>
</tbody>
</table>

**demos/lovervu:**

<table>
<thead>
<tr>
<th>actnet1.tll</th>
<th>design.tll</th>
<th>plnschd2.otl</th>
<th>spec&amp;dsn.otl</th>
<th>v&amp;v1.tll</th>
</tr>
</thead>
<tbody>
<tr>
<td>barchart.tll</td>
<td>overview.cfg</td>
<td>plnschd2.tll</td>
<td>spec&amp;dsn.tll</td>
<td>v&amp;v2.tll</td>
</tr>
<tr>
<td>cbsyse.otl</td>
<td>overview.go2</td>
<td>polar.tll</td>
<td>swmgmt1.otl</td>
<td>v&amp;v2.tll</td>
</tr>
<tr>
<td>cbsyse.tll</td>
<td>plnschdl.otl</td>
<td>reqeng.otl</td>
<td>swmgmt1.tll</td>
<td></td>
</tr>
<tr>
<td>design.otl</td>
<td>plnschdl.tll</td>
<td>reqeng.tll</td>
<td>v&amp;v1.otl</td>
<td></td>
</tr>
</tbody>
</table>

**demos/2mgmt:**

<table>
<thead>
<tr>
<th>cocograf.tll</th>
<th>costest1.tll</th>
<th>mangmt.cfg</th>
<th>progpod.tll</th>
<th>swrev.tll</th>
</tr>
</thead>
<tbody>
<tr>
<td>cocomo1.otl</td>
<td>costestm.tll</td>
<td>mangmt.go2</td>
<td>qualass1.otl</td>
<td>swstd.tll</td>
</tr>
<tr>
<td>cocomo1.tll</td>
<td>funcpts.otl</td>
<td>pimpcht.tll</td>
<td>qualass1.tll</td>
<td>swstd.tll</td>
</tr>
<tr>
<td>cocomo2.otl</td>
<td>funcpts.tll</td>
<td>procimp.otl</td>
<td>swmet.otl</td>
<td>teamorg.otl</td>
</tr>
<tr>
<td>cocomo2.tll</td>
<td>grpwrk.otl</td>
<td>procimp.tll</td>
<td>swmet.tll</td>
<td>teamorg.tll</td>
</tr>
<tr>
<td>costest1.otl</td>
<td>grpwrk.tll</td>
<td>progpod.otl</td>
<td>swrev.otl</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 19: Sample Directory Structure**
Figure 20: Imported Directory Structure
Figure 21: PageManager Operational Functions
APPENDIX II: VERMEER FRONTPAGE SCREENSHOTS

This is the default home page FrontPage creates. I've added the ODU logo at the top.

Our Mission: To Complete This Thesis

Write one or two short sentences that describe your company's philosophy and ambitions. Something like, 'To become the leading provider of ...'.

Company Profile

Describe who you are, and how you provide value to your customers.

If you belong to any industry associations, list them here.

Figure 22: FrontPage Default Home Page
Figure 23: FrontPage Main Window

Figure 24: FrontPage Interview
APPENDIX III: NETSCAPE LIVEWIRE SCREENSHOTS

Marlowe's Thesis Home Page

This is the central page for an organization or a firm that provides a service. The focus here becomes the valuable information you can offer your visitor/clients. (For some general information and suggestions on setting up your web site read "suggest html".)

Brief Information about the Organization

Give a brief description of your organization for the newcomers.

Figure 25: LiveWire Default Home Page
Figure 26: LiveWire "Guru"

Figure 27: LiveWire Site Manager
Figure 28: LiveWire Application Manager
VITA

Kevin L. Marlowe is a systems analyst with Computer Sciences Corporation at the NASA Langley Research Center in Hampton, Virginia and a commissioned officer in the United States Navy Reserve. He was awarded the degree of Bachelor of Arts from the University of Virginia in May, 1987. His research currently includes the use of the World Wide Web to provide platform-independent access to local databases and the development of intercorporate fiber optic networks. Current mailing address: NASA Langley Research Center (MS 931), Hampton, VA 23681.