

DESIGNING A WEB *EES* MODIFICATION
REQUEST (*EMR*) SYSTEM

By

WEIMING GAN

Bachelor of Engineering

Xi'an Highway Institute

Xi'an, Shanxi

P. R China

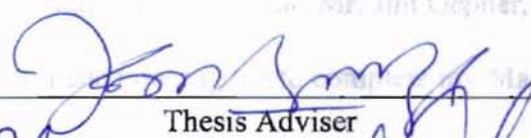
1982

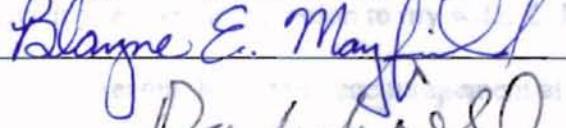
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2000

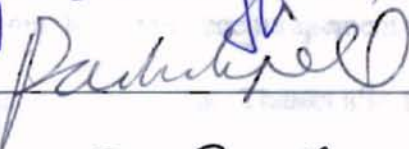
DESIGNING A WEB EES MODIFICATION


REQUEST (EMR) SYSTEM

Thesis Approved:



Thesis Adviser






Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. K. M George for his intelligent supervision, constructive guidance, inspiration and friendship. My sincere appreciation extends to my other committee members Dr. Jacques LaFrance and Dr. Nohpill Park, whose guidance, assistance, encouragement, and friendship are also invaluable.

More over I would like to thank my supervisor, Mr. Jim Gepner, for providing me with this research opportunity and encouraging me to complete my Master Degree.

I would also like to give my special appreciation to my wife, Z. Julia Jin, for her precious suggestions to my research, her strong encouragement at times of difficulty, love and understanding throughout this whole process. Thanks also to my parents for their support and encouragement.

Finally, I would like to thank the EES group of the Lucent Technologies for supporting me with a lot of helpful resources.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.....	1
II. RALATED WORK.....	5
The EMR System.....	5
Work Related to EMR.....	7
The First Design of the <i>emr</i> Tool.....	7
Release 2 of the <i>emr</i> Tool.....	11
Related Tools.....	17
<i>Hybrowse</i>	17
III. NEW DESIGN: WEB <i>EMR</i> SYSTEM.....	18
The Architecture.....	18
Retrieve and Restore a File.....	19
The Web Server of the New EMR System.....	19
The CGI Programing of the EMR System.....	20
The Interface for Searching an EMR.....	20
The CGI program for Searching an EMR.....	21
The Interface for Creating an EMR.....	22
Restore an EMR Document.....	25
IV. COMPARISON OF WEB <i>EMR</i> and <i>EMR</i>	26
V. CONCLUSION AND FUTURE WORK.....	27
REFERENCE.....	28
APPENDIXES.....	30
APPENDIX A--THE SOURCE CODE OF THE PROGRAM <i>findemr.cgi</i>	30
APPENDIX B --THE SOURCE CODE OF THE PROGRAM <i>uploademr.cgi</i>	40

LIST OF FIGURES

Figure	Page
1. EES Development Phases with Major Outputs.....	2
2. The Architecture of the Original Electronic EMR System.....	5
3. The EMR File Structure.....	6
4. EMR States.....	12
5. The Architecture of the Web EMR System.....	18
6. The Web EMR System Interface and the source.....	20
7. The Interface to Show All EMRs.....	21
8. The Interface to Show All Items of an EMR.....	22
9. The Interface to Create an EMR.....	23
10. The Source of the EMR Creating Interface	24
11. The Interface for Uploading an EMR File	25

CHAPTER I

INTRODUCTION

The 5ESS™ Switching System is a major product of the Lucent Technologies Inc. It is a digital time division, distributed control system [1]. There are two test environments for the 5ESS™ Switch development community: the Laboratory Test System (LTS), which is a small office rigged with debugging equipment, and the Execution Environment System (EES), which is a software simulation of a small office that runs under the UNIX® operation system [2]. LTS is more complete and covers more tests accurately, but EES is more flexible and economical.

The developments of EES software products include the new products or enhancements to existing products. The EES Software Development Methodology is based on the ISO 9000-3 guidelines. Generally, most terms have been retained from the ISO standards/guidelines [3]. Figure 1 shows that there are 6 major phases from the original request for the software to the actual customer acceptance of the final product. Major outputs (primarily documentation) are shown for each stage with the output(s) requiring customer participation in *italics*.

There are two primary classifications of EES documentation produced during the development of software: (1) those produced as a result of driving or following the EES Software Development Methodology itself, e.g., Design, Test Plan, and (2) those

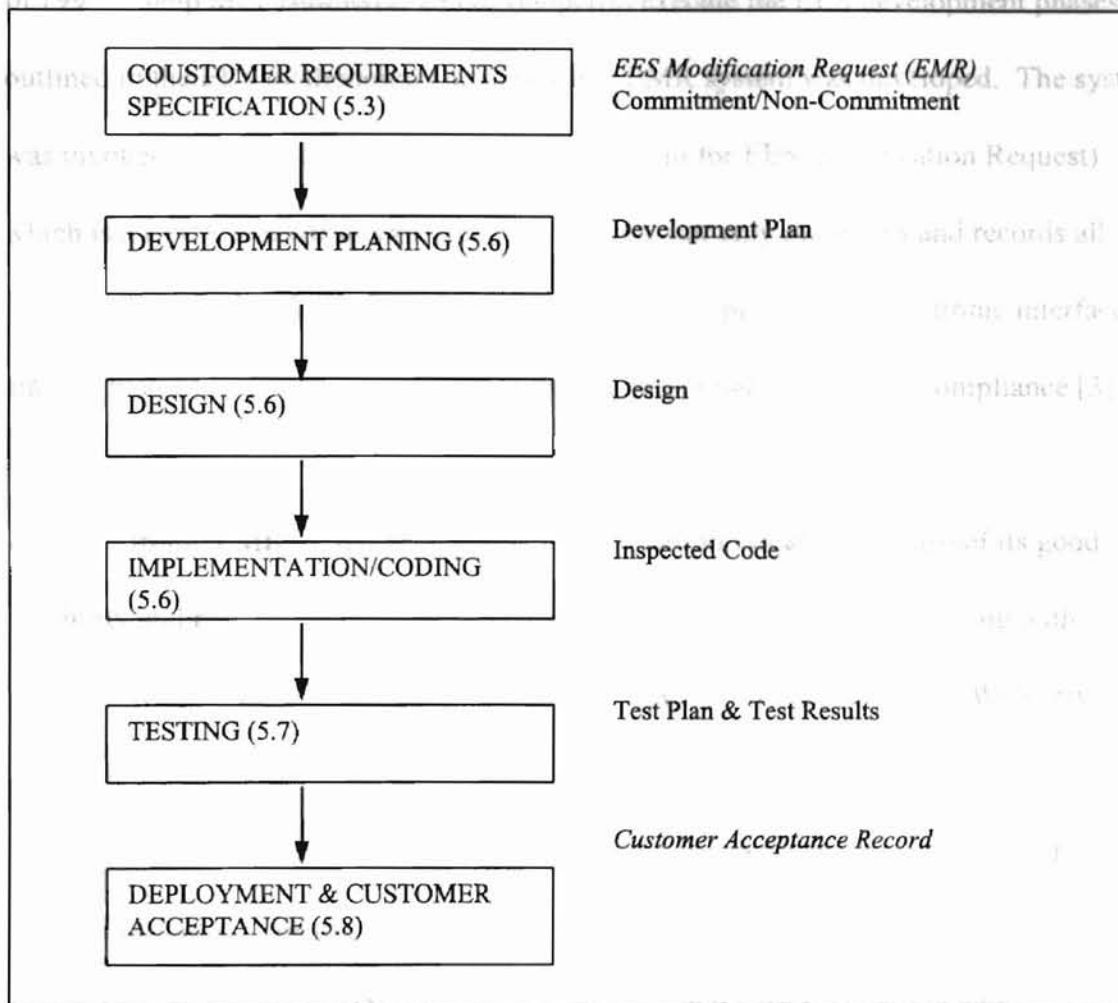


Figure 1. EES Development Phases with Major Outputs (ISO 9000-3 clauses are indicated in parenthesis)

primarily for users to be able to utilize the final product, e.g., man pages, user manuals.

In the beginning of the EES development, the EES Modification Requests (EMRs) was a documentation system for the EES Software Development Methodology. It was a paper-based system that included the US EMR system and the international Problem Statement-based system. The paper-based EMR system had been in use for a long time.

In 1993 to help the customer(s) and developer(s) execute the EES development phases outlined in the EES methodology, an electronic EMR system was developed. The system was invoked by a tool called *emr* (*emr* is an acronym for EES Modification Request) which is a UNIX shell program. The EMR system not only maintains and records all necessary documentation during development, but also provides an electronic interface and facilitates the kind of development tracking that is needed for ISO compliance [3].

The electronic EMR system works like an information system. Because of its good reliability and convenience, it has been used until today. However comparing with current information system, for example, an information system running a Web server with CGI (Common Gateway Interface), the EMR system has several disadvantages:

- Complexity: The command-line interface with too many keywords is not friendly.
- Limitation: The execution is restricted in UNIX system only.
- Difficulty: The UNIX text file with *nroff* or *troff* format is hard to edit.

To solve the above problems, a new design of the EMR system was given in this thesis. The new design used the similar EMR file structure, but replaced the *emr* tool with the Web server and the CGI programs. This approach allows the customer(s) and the developer(s) to use a Web browser to run the EMR system via the Internet. It also gave the possibility that an EMR document could be an HTML format.

Since EMR system includes a lot of functionality and features, it is impossible to cover all designs in this thesis. Based on the major requirements, the thesis have given the following solutions:

- Creating or opening an EMR: A user can create or open an EMR.
- Searching an EMR: A user can search all documents in the EMR file system via the Internet. The documents could be HTML format.
- Updating or modifying an EMR: A user can download or create an EMR document into his/her local machine, edit or modify it by using a Web browser, then send or upload it back to the EMR file system.

RELATED WORK

The EMR System

Figure 2 shows the architecture of the original electronic EMR system. It includes a browser called *Hybrowser*, the *emr* tool, and the EMR files system.

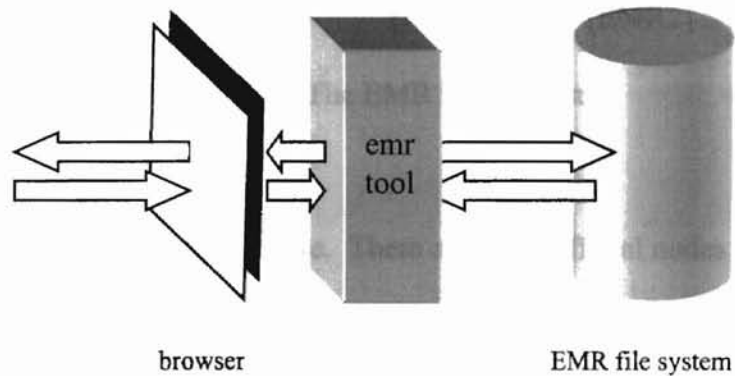


Figure 2. The Architecture of the Original Electronic EMR System

Hybrowser is a menu interface where the users can browse information or perform various operations by entering certain key combinations.

The *emr* tool was used to invoke the browser and connect to the EMR file system. Its primary functions are:

- To provide a centralized, electronic means of tracking EES development.
- To aid the writing process by supplying templates and/or guidelines.

- To aid the execution of the methodology by supporting operations on those documents.

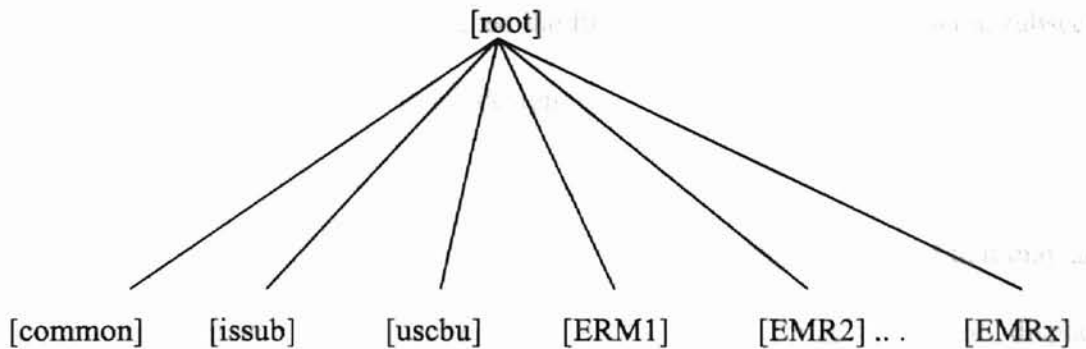


Figure 3. The EMR File Structure

Figure 3 shows the EMR file structure. There are three official nodes *common*, *issub*, and *uscbu*. Each node has a different purpose.

- *common*: The node serves as the root for all base system files. In all decisions, it serves as the default.
- *issub*: The node serves as the root for the international EES domain. It contains any variance from the base system.
- *uscbu*: The node serves as the root for the U.S. EES domain. It contains any variance from the base system.

When an EMR is entered into the system, its root node appears at the same level as these three.

The First Design of the *emr* Tool

In October 1993, Gary Barrett designed the first *emr* tool [4]. The following subsections describe the major components in his design.

Viewpathing. The *viewpathing* tool is used to resolve any conflict that may arise surrounding template contents, naming conventions, and anything that the tool displays to the user. The choice to viewpath influences the node structure within which information is stored.

A Generalized Algorithm. The *emr* tool's specific behavior is to be driven by information contained in files found down the viewpath. These files come in the following varieties:

- Rules files: These specify the action to take.
- Informative files: These supply information to be displayed.
- Menus: These provide the user with a choice of alternative actions.
- Naming conventions: These determine the name of phase and source files.
- Templates: These provide the user with starting points to writing documents

The algorithm is simple. It consists of a loop, which executes until the user chooses to quit. Each loop locates the *rules* file, gives the current node and the viewpath, and executes, in sequence, each rule that it finds in that file.

Node Structure. There is one node under the root node for each development phase. A node is a directory and contains some subdirectories (subnode) and files. The nodes are named as follows.

Node	Phase
<i>rqs</i>	requirement
<i>plan</i>	development plan
<i>des</i>	design
<i>code</i>	implementation
<i>to</i>	customer acceptance

To accommodate the differing levels of testing and to assign the proper perspective to those levels, nodes that pertain to testing are contained within the various phase nodes.

Phase nodes typically contain the following subnodes.

Node	Purpose
<i>test</i>	Information for this phase-level of testing
<i>review</i>	Information about reviews
<i>change</i>	Change records
<i>doc</i>	phase document source

The *rqs* node also contains the *commit* subnode, for the commitment record. The *plan* contains no *test* node. The *to* node contains only the *doc* node.

A *test* node contains the following subnodes.

Node	Purpose
<i>review</i>	Information about reviews
<i>change</i>	Change records
<i>result</i>	Test results
<i>doc</i>	Test plan document source

The following table enumerates the files that can exist within system nodes.

File	Number allowed	Purpose
<i>rules</i>	one node has one	drives the <i>emr</i> toll
<i>informative</i>	one node may has any	
<i>menus</i>	one node may has any	
<i>docname</i>	one node has one	specifies source name
<i>name</i>	one node may has one	specifies title of node

The names of menu and informative files are determined by the content of the *rules* files.

All document sources is kept in EMR terminal node. Terminal nodes can also contain the following files.

File	Purpose
<i>partitions</i>	Design unit information
<i>versions</i>	Version history
<i>original</i>	Original document source
<i>save</i>	Saved document source
<i>alist</i>	Approver list
<i>approved</i>	Approval record
<i>sent</i>	Timestamp for commitment record sending
<i>out</i>	Document source lock file

Additionally, EMR root nodes can contain the following information source files.

File	Purpose
<i>who</i>	Originator identity, copy-to list
<i>owners</i>	EES developer identifies
<i>state</i>	EMR state
<i>abstract</i>	EMR abstract
<i>ack</i>	EMR receipt acknowledgment

Convention. The EMR number is constructed of a six-digit date, the originator's login id, and a two-digit numeric suffix. Login id's that are longer than six characters are truncated to six so that the EMR number is no longer than fourteen. The date portion of the EMR is constructed of the year, month and day of month so that EMRs can be easily sorted in chronological order. The EMR number is used as the name for the EMR's root node.

Locks are implemented as files that contain the login id of the person who has the file out for edit. The lock files, called *out*, are co-located with the document source. A file is locked when it is taken out for edit. It is unlocked when it is put back if any of the following conditions is true:

- The approval process for that document has not yet begun.
- The document is approved and the change made to it requires no additional approval.

Under all other conditions, the file remains locked. While locked, it is only the owner of the lock who can continue to take the file out for edit.

The file can also be unlocked by the lock's owner via manual procedures.

The Main Program Loop. The program begins by initializing variables. It is this point where the CBU must be known so that a viewpath can later be constructed. Before entering the main program loop, the command line is scanned for input arguments and parsed. If there are no arguments, then the high level interface is invoked. Low-level invocations of the tool are made interactively, by passing arguments, or from the high level via *hybrowse* application macros. At the low level, after input commands are parsed and additional information is received, if needed, from the user, the program enters a large switch block where preliminary handling differs from one operation to another. Some operations can be completed right there. Most others, however, complete by invoking the main program loop that locates the *rules* file and processes it one line at a time.

Release 2 of the *emr* Tool

In 1994, David Newkirk described a simplified user interface for the *emr* tool. The following subsections are the major components in Newkirk's description.

EMR states. An EMR can have the following states:

- *open* A new EMR that has not been submitted. An open EMR is created and owned by the originator. The originator can submit or cancel an open EMR.

- *submitted* An EMR that has been submitted but is unassigned. The administrator can reject, close, acknowledge or assign submitted EMR's.
- *in-progress* An EMR that has been assigned. The developer or administrator can close or hold an in-progress EMR. The administrator can re-assign an EMR to a different developer.
- *hold* An uncommitted EMR is being held until it can be re-evaluated. The administrator can assign or close an EMR in the hold state.
- *closed* An EMR is closed after development is complete or it's uncommitted. The administrator can reopen a closed EMR, putting it in its previous state. The administrator can also delete a closed EMR from the database when it is no longer needed.

Figure 4 shows the state transition allowed for EMR's:

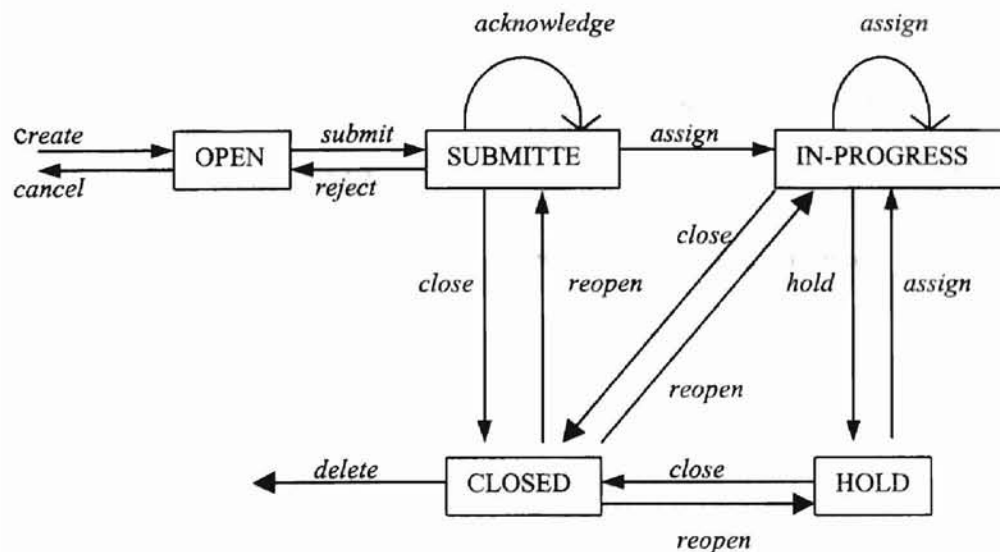


Figure 4. EMR States

Menu System. The *emr* tool includes eight menus as follows:

- MAIN MENU Select an existing EMR to work on or create a new one.

- **EMR OPERATIONS** Perform operations on the EMR (such as printing its status).
- **DOCUMENTS** A list of EMR documents the user can choose to access.
- **DOCUMENT OPERATIONS** Operations specific to individual documents (create, browse, edit, etc).
- **REVIEW & APPROVAL MENU** Access the document review and approval data.
- **CODE INSPECTION MENU** Access data about a code inspection session.
- **VERSION HISTORY MENU** Access previous versions of approved documents and their review data.
- **ADMINISTRATOR MENU** Operations for the EMR Administrator, such as assigning or closing EMR's.

Main Menu is the first menu presented by the *emr* tool unless EMR environment variable is set to a valid EMR identifier. The current EMR number is displayed in the menu header field. The following list is the operations in the menu:

- a* access an existing EMR (ACCESS EMR dialog)
- c* create a new EMR (CREATE EMR dialog)
- s* show summary of all EMR's (SUMMARY output)
- f* send feedback about EMR tool (FEEDBACK dialog)
- A* administrator menu (ADMINISTRATOR menu)
- h* help (run the pager on the Main Menu Help file)

q quit (QUIT dialog)

The "A" operation is only visible if the user is the EMR administrator or the alternate.

EMR Operations menu lets the user choose a document to examine or perform an operation on the EMR. If the EMR environment variable is set to a valid EMR identifier, this is the first menu displayed. The EMR number appears in the menu header field. The following list is the operations in menu:

s short EMR status (STATUS output)

f full EMR history (HISTORY output)

d choose a document (DOCUMENT menu)

p print all EMR documents available (PRINT ALL dialog)

*r** submit the EMR requirements to the EES

*c** cancel EMR (CANCEL EMR dialog)

*a** change the abstract (CHANGE ABSTRACT dialog)

*i** change the related IMR number (CHANGE IMR dialog)

*H** hold EMR (HOLD EMR dialog)

*C** close EMR (CLOSE EMR dialog)

m return to MAIN MENU

h help

q quit (QUIT dialog)

Documents menu lets the user choose a document to examine. The EMR number appears in the menu header field. The following list is the operations in the menu:

r requirements
c commitment
n non-commitment
p development plan
d design
i code inspection menu (CODE INSPECTION menu)
t test plan
T test plan results
u user documentation
a acceptance
o return to EMR OPERATIONS menu
h help
q quit (QUIT dialog)

The '*c*', '*n*', '*p*', '*d*', '*i*', '*t*', '*T*', '*u*' and '*a*' operations are visible if the user owns the corresponding document or the EMR, or official documents are available.

Document Operations menu contains all the operations relevant to individual documents. Besides viewing and updating documents, this includes their review and version information, and ownership. The EMR identifier and the selected document name appear in the menu header field. The following list is the operations in the menu:

b browser (BROWSER DOCUMENT dialog)
p print (PRINT DOCUMENT dialog)
*c** create (create a private document using the online

and lock the document)

*e** edit (EDIT DOCUMENT dialog)

r review & approval menu

*s** submit change (SUBMIT DOCUMENT dialog)

v version history menu

*u** undo private change (UNDO CHANGE dialog)

*B** backout official change (BACKOUT CHANGE dialog)

*P** change owners (OWNERSHIP dialog)

*C** create design unit (CREATE DU dialog)

O open design unit (OPEN DU dialog)

E end design unit

Close the current design unit, returning to high-level document and updating the menu header.

*D** delete design unit (DELETE DU dialog)

d return to DOCUMENTS menu

o go to EMR OPERATIONS menu

h help (run the pager on the Document operation Help file)

q quit (QUIT dialog)

The operations marked by an asterisk (*) do not appear in the menu unless the user owns the current document. In addition, the 'c' operation is visible if no private or official document exists. The 'b', 'p', 'e' and 'r' operations are visible if an official or private document exist. The 's' and 'u' operations are visible if a private document exist. The 'v' operation is visible if there is more than one version of the official document. The 'B'

operation is visible if there is an unlocked and unapproved official document. The 'C' operation is visible only when the current document is a design. The 'O' and 'D' operation are visible if a design unit exists and the current document is a design, test plan or test result. The 'E' operation is visible if the current document is inside a design unit.

The other menus are ignored since this thesis doesn't cover the designing for those features. The information could be found in Newkirk's paper [5].

Related Tools

Hybrowse

The *emr* tool uses *Hybrowse* as the browser interface. The *Hybrowse* is a program that simplifies access to online documentation by taking advantage of the structure provided by memorandum (mm) macros [6]. It provides readers with an annotated catalog of available documents, a Table of Contents for each document, commands to jump to selected sections, automatic lookup of references, and a context display (section heading and page number) on each screen of information.

Hybrowse was designed to minimize the cost of conversion, training, and administration, so it is practical for small applications. *Hybrowse* is useful for implementing online reference manuals, help systems, interactive tutorials, and browsing systems for collections of existing documents.

NEW DESIGN: THE WEB EMR SYSTEM

The Architecture of the New EMR System

The new EMR system is a Web EMR system. The design adopted the traditional architecture of a Web documentation information system shown in Figure 5. It is a client/server structure. The client site is a Web browser running on a local machine. The server site is composed of Web server, Common Gateway Interface (CGI), and the EMR file system [7,10,11].

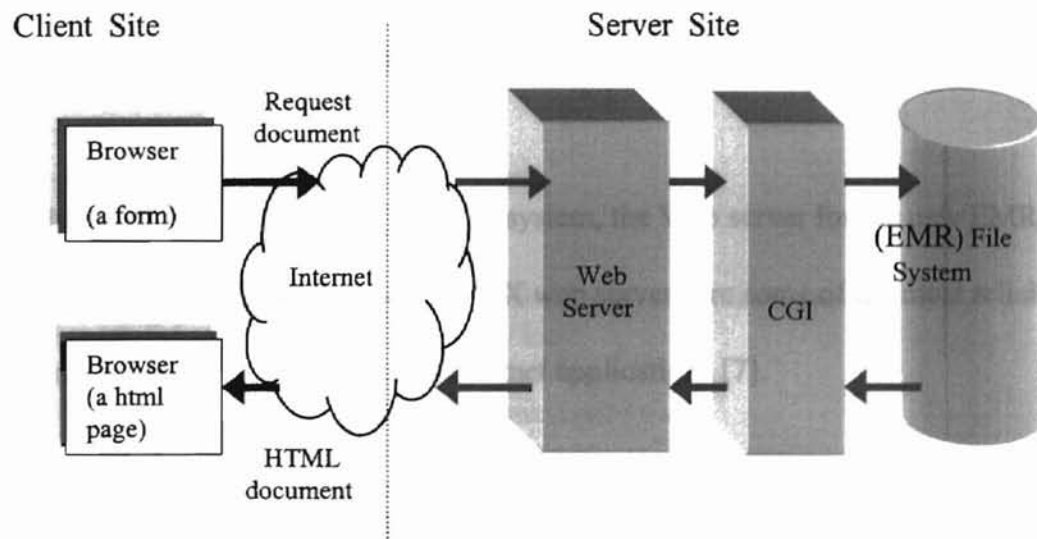


Figure 5. The Architecture of the Web EMR System

Retrieve and Restore a File

To retrieve the documentation, a user can input the request in the form that the Web server supplies. After getting the request of the user, the Web server will invoke the CGI program, retrieve the documentation from the EMR file system, and send it back to the user.

To restore a file back to the EMR file system, the user uses the Web browser to edit the document first, then saves this document somewhere in the local machine. Finally, the user can use the upload feature that Web browser supports to send the document back to the EMR file system to the remote machine.

The Web Server of the New EMR System

Since the EMR file system is a UNIX file system, the Web server for the new EMR server should be a UNIX Web server. UNIX web servers are some of the most reliable and secure servers that support a lot of Internet applications [7].

The new design selected the NCSA Web server 1.5.2a for the Web EMR system. NCSA HTTPd is an HTTP/1.0 compatible Web server for making hypertext and other documents available to Web Browsers [8,9]. Because of the very good reliability and compatibility, it is widely used by the UNIX community. NCSA HTTPd server is written in C language, and the all source code is free.

The Interface for Searching an EMR

The Web EMR System works like an Internet search engine. It uses a form to get the request. Figure 6 is the interface and the source code. When a user links to the URL of

Web EMR System -- A global EES Modification Request System.

Please click the following keywords to do the EMR operations on an EMR.

Search -- for searching an EMR

Create -- for opening the space to store a new EMR

Update -- for uploading a document of an EMR

```
<html>
<head><title>Web EMR System</title>
</head>
<body>
<b><font size=+2>WEB EMR SYSTEM </font></b>- - A<font size=+0>
global EES Modification Request System.</font>
<p>Please click a keyword to do the operation on an EMR.
<a href="http://127.0.0.1:8000/cgi-bin/findemr.cgi"></a>
<p><a href="http://127.0.0.1:8000/cgi-bin/findemr.cgi">Search</a> -- for
searching an EMR
<br><a href="http://127.0.0.1:8000/create.html">Create</a> -- for opening
the space to store a new EMR
<br><a href="http://127.0.0.1:8000/uploademr.html">Update</a> -- for uploading
a document of an EMR
</body>
</html>
```

Figure 6. Web EMR System Interface and the Source

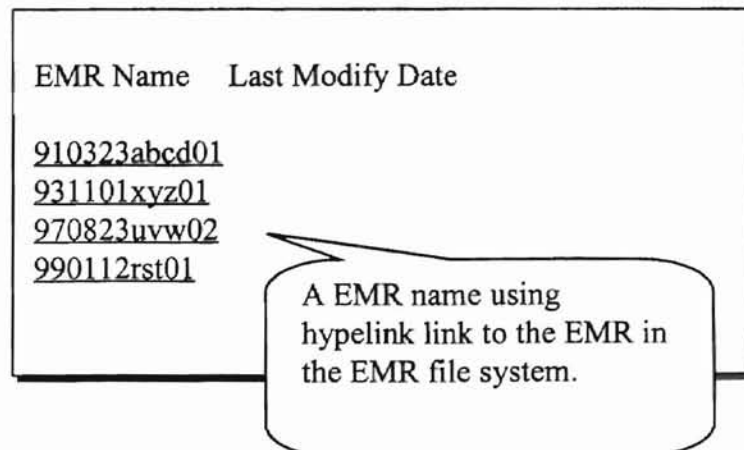
the Web EMR system, this interface will be displayed to the user. The user can enter an EMR name into the open bar, and click the search button, then the request will send back to the server. After decoding the request, the server will invoke CGI script, the program *findemr.cgi*, search the EMR in the EMR file system, and send the result back to the user.

The CGI Program for Searching an EMR

The CGI program *findemr.cgi* can support the following functionality,

1. Search all EMRs and display them.
2. Search one EMR and display it.
3. Search any content of one EMR, such as a design document, a test plan, or a review report, and display it.

Figure 7 is an example that shows all EMRs. If the user uses the mouse click an EMR's name, all items of the EMR will be displayed, as shown in Figure 8. If the user clicks one



EMR Name	Last Modify Date
<u>910323abcd01</u>	
<u>931101xyz01</u>	
<u>970823uvw02</u>	
<u>990112rst01</u>	

Figure 7. The Interface to Shows All EMRs

item, such as the *design* item, he or she will see the design documentation of that EMR via the Internet.

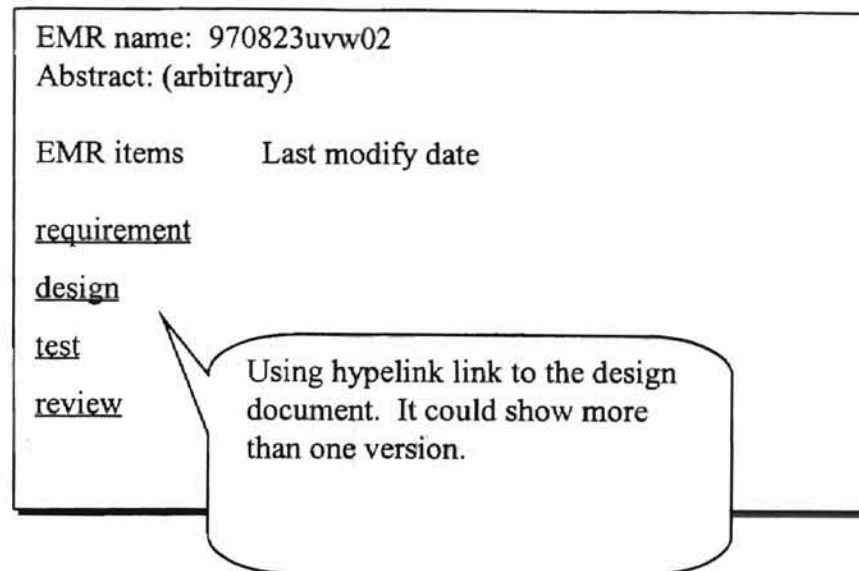


Figure 8. The Interface to Shows the All Items of an EMR

The source code of the program *findemr.cgi* is shown in Appendix A. This program is a good example for a small Internet searching engine [12].

The Interface for Creating an EMR

To create a new EMR, the user can click *Create* button in Figure 6. Then the EMR Web server will display the EMR creating interface, as shown in Figure 9 to the user. Figure 10 shows the source of the EMR creating interface.

From EMR creating interface, the user can enter the name, login ID, telephone number, and EMR name, etc. The circular buttons are RADIO buttons for the RADIO input field. A user can check the RADIO button to get the EMR ownership or the access permission. For example, a user can select *requirement* button and submit the request for opening an EMR. After checking the request, the EMR administrator will set up the permission for the user. Then the user can put the requirement documentation back to the EMR file system. Similarly a developer should select the *design* button if he or she has a design assignment and wants to send the design document back to the EMR file system.

The form is enclosed in a rectangular border. It contains the following elements:

- Your Name**: A text input field.
- Phone**: A text input field.
- Login ID**: A text input field.
- Enter an EMR name**: A text input field with a placeholder text below it: *(Format:yymmdd<keyword>)*.
- RADIO buttons**: A vertical list of four radio buttons with labels: *requirement*, *design*, *testplan*, and *review*.
- Enter the abstract**: A label above a large, empty text area.
- Submit**: A button with a dark background and white text.
- Clear**: A button with a dark background and white text.

Figure 9. The Interface to Create an EMR

```

<head><title> Create an EMR </title>
</head>
<br>
<h2>Create an EMR</h2>
<p><form METHOD="POST" ACTION="/cgi-bin/createemr.cgi">

First Name:<input TYPE="TEXT" NAME="first_name" MAXLEN="50" SIZE="15">
Last Name:<input TYPE="TEXT" NAME="last_name" MAXLEN="50" SIZE="15">
<br>Login ID:<input TYPE="TEXT" NAME="login_id" MAXLEN="20" SIZE="15">
<br>Phone:<input TYPE="TEXT" NAME="phone" MAXLEN="20" SIZE="15">
<ul>
<li>
    <input TYPE="radio" NAME="Document" VALUE="EMR requirement document">
    requirement document</li>
<li>
    <input TYPE="radio" NAME="Document" VALUE="EMR design document">
    design document</li>
<li>
    <input TYPE="radio" NAME="Document" VALUE="EMR testplan">testplan
    document</li>
<li>
    <input TYPE="radio" NAME="Document" VALUE="EMR review">
    review document</li>
</ul>
Enter the EMR name
<input TYPE="TEXT" NAME="emr_dir" MAXLEN="50" SIZE="15">
<p>
Enter the abstract:
<p><textarea NAME="Abstract" ROWS=15 COLS=60></textarea>
<p>
<INPUT TYPE="hidden" NAME="log_filename"
    VALUE="/home/col/httpd/htdocs/log.txt">
<INPUT TYPE="hidden" NAME="log_fields"
    VALUE="first_name,last_name,login_id,phone,Document,Abstract">
<INPUT TYPE="hidden" NAME="log_delimiter" VALUE="|">
<INPUT TYPE="hidden" NAME="log_uid" VALUE="/home/col/httpd/htdocs/uid.txt">
<INPUT TYPE="hidden" NAME="recipient" VALUE="col@blueston.com">
<p><input TYPE="submit" VALUE="Submit"> <input TYPE="reset"
VALUE="Clear">
<p></form>
</body>
</html>

```

Figure 10. The Source of the EMR Creating Interface

Restore an HTML Document Back to EMR System

The EMR creating interface doesn't support the remote edit feature. It only allocates the space for a new EMR and gives the directory permission to a user or a developer. To restore an *html* file back to the EMR system, the design uses upload feature that the Netscape has. After clicking the keyword *Update* button from the Web EMR System interface, the upload interface will be brought up, as shown in Figure 10. The EMR name and the document type are for locating the path. After clicking the button *Browse*, the user can highlight a file that he or she wants to restore, click the upload button. Then that file will be send to the serve and written into the EMR file system. Before uploading the documentation, the user should use Netscape to edit the document, and save it in the local machine somewhere.

Upload an EMR file
To upload an EMR file, fill out the form below:
EMR Name :

☐ requirement document
☐ design document
☐ testplan document
☐ review document

Figure 11. The Interface for Uploading an EMR File

CHAPTER VI

COMPARISON OF NEW EMR AND EMR

The new EMR system used HTTP server and CGI technologies. It makes the EMR system more powerful and convenience. Comparing with the old EMR system, The Web EMR system has the following advantages:

- Widely access: A user can get the information of an EMR, or do an EMR operation via the Internet.
- Ease of operation: The Web EMR system replaced the menu system in the old EMR system by using a friendly GUI interface. For EMR operations, a user doesn't need to type a lot of keywords.
- HTML format: An EMR document can has HTML format. A user can change the font of a document easily.

The disadvantages of the Web EMR system are:

- The old EMR documents with *nroff* or *troff* format can not be displayed.
- The benefits that *nroff* or *troff* supported, such as some useful macros, are lost.

CHAPTER V

CONCLUSION AND FUTURE WORK

The Web EMR system is valuable because it inherits many benefits from the Internet technologies. This thesis gave the solutions for the Web EMR system as the follows:

- Search an EMR or a document of an EMR.
- Create an EMR node.
- Send a document back to an EMR node.

To completely build a Web EMR system, there is still more work to be done. The following topics are the major components for the future works.

- Add the others EMR operations, such *approved*, *close*, etc.
- Add the function to lock a document.
- Add the administration feature.
- Add the feature to display the old documents that have *nroff* or *troff* format.

REFERENCE

1. Aditham, R., "5ESS™ Operating System and Communications - An introductory description," AT&T Bell Laboratories, June 1984.
2. Lied, R., "Simulating 5ESS™ Switch Network Path Signals in the Execution Environment System," AT&T Bell Laboratories, March 1990.
3. Barrett, G., Jones, J., Phillips, R., Taylor, D., Turek, M., "EES software Development Methodology," 5ESS™ Switching Process Documentation, Lucent Technologies Inc., May 1995.
4. Barrett, G., "File Case Design of the *emr* Tool," EMR 931020biker01, Lucent Technologies Inc., October 1993.
5. Newkirk, D., "Release 2 of *emr* tool," EMR 940222dcn01, Lucent Technologies Inc., February 1994.
6. Stanfield, L., "Hybrowse - A Tool for Accessing Online Documentation," 55513-880629-01TM, AT&T Bell Laboratories, June 1988.
7. Stanek, R. W., "HTML CGI SGML VRML JAVA Web Publishing Unleashed," *Sams.net* Publishing, Indianapolis, 1996.
8. NCSA HTTPd Development Team, "NCSA HTTPd," January 1998.
<http://hoohoo.ncsa.uiuc.edu/doc>
9. Berners-Lee, T., Fielding, R., Frystyk, H., "Hypertext Transfer Protocol -- HTTP/1.0," RFC: 1945 <http://www.ics.uci.edu/pub/ietf/http/rfc1945.html>
10. NCSA HTTPd Development Team, "The Common Gateway Interface," January 1998. <http://hoohoo.ncsa.uiuc.edu/cgi/>

11. Robinson, T., "The WWW Common Gateway Interface Version 1.1," Internet-DRAFT, University of Cambridge, February 1996.
12. Patchett, C., Wright, A., "CGI/Perl Cook Book," Wiley Computer Publishing, New York, 1998.

APPENDIX A

THE SOURCE CODE OF THE PROGRAM *findemr.cgi*

```
#!/usr/bin/perl
# File name: findemr.cgi
#####
# Define configuration constants                                     #
#####

# $ROOT_DIR is the full path to the root directory that FileSeek will look
# for files in. (Note that it should end with a directory delimiter.)

$ROOT_DIR = '/home/col/httpd/htdocs/emr/';

# $ROOT_URL is the URL of the directory specified by $ROOT_DIR

$ROOT_URL = 'http://127.0.0.1:8000/emr';

# $ROOT_NAME is the name of the root directory that will be used within the
# program.

$ROOT_NAME = 'EMR';

# $ICON_DIR is relative path from your server's root directory where the
# icons for the different file types will be kept. (Note that it should end
# with a forward slash.)

$ICON_DIR = '/fileseek/icons/';

# %TYPES matches file types to the corresponding icon files. With the
# exception of 'directory' and 'binary' default file types, each file
# type corresponds to a filename extension.

%TYPES = ('parent', $ICON_DIR . 'parent.gif',
          'directory', $ICON_DIR . 'directory.gif',
          'binary', $ICON_DIR . 'binary.gif',
          'txt', $ICON_DIR . 'text.gif',
          'gif', $ICON_DIR . 'graphic.gif',
          'jpg', $ICON_DIR . 'graphic.gif',
          'jpeg', $ICON_DIR . 'graphic.gif',
          'htm', $ICON_DIR . 'html.gif',
          'html', $ICON_DIR . 'html.gif',
```

```

        'pdf',    $ICON_DIR . 'pdf.gif');

# $ALLOWED_DIR is the full path to the root directory that any files
# specified by the user must be in (e.g., template files). (Note that
# it should include a directory delimiter at the end.)

$ALLOWED_DIR = '/home/col/httpd/htdocs/emr/';

# $DATE_FORMAT holds the &format_date() format for the file mod date & time

$DATE_FORMAT = '<wday>, <mon> <d>, <year>, <h>:<0n> <AP>';

# $PROGRAM_URL is the URL of this program

$PROGRAM_URL = 'http://127.0.0.1:8000/cgi-bin/findemr.cgi';

# $ERROR_PAGE is the error page template that will be used by the error
# subroutine.

$ERROR_PAGE = "/home/col/httpd/error_page.html";

# $REQUIRE_DIR is the directory in which all of your required files are
# placed. On most systems, if you leave the required files in the same
# directory as the CGI program, you can leave this variable blank.
# Otherwise, if you move the required files to another directory, specify
# the full or relative path here.

$REQUIRE_DIR = 'require';

#####
# Get required subroutines which need to be included.          #
#####

# Push $REQUIRE_DIR onto the @INC array for include file directories
# and list required files.

push(@INC, $REQUIRE_DIR) if $REQUIRE_DIR;

require 'formdate.pl';
require 'template.pl';
require 'error.pl';

```

```
#####
# Initialize other constants                                     #
#####

@DAYS = ('Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
        'Saturday');
@MONTHS = ('January', 'February', 'March', 'April', 'May', 'June', 'July',
        'August', 'September', 'October', 'November', 'December');
$DAY_SECS = 24 * 60 * 60;

if ($ENV{'QUERY_STRING'} !~ /sort=\d/ && $ENV{'QUERY_STRING'}) {
    $adjust = '&' . $ENV{'QUERY_STRING'};
}
$NAME_TITLE = "<A HREF=\"$PROGRAM_URL?$`sort=0'$adjust\">Name</A>";
$SIZE_TITLE = "<A HREF=\"$PROGRAM_URL?$`sort=1'$adjust\">Size</A>";
$DATE_TITLE = "<A HREF=\"$PROGRAM_URL?$`sort=2'$adjust\">Last
Modified</A>";

#####
# Parse emrname string                                         #
#####

(@args) = split(/&/, $ENV{'QUERY_STRING'});
foreach $arg (@args) {
    ($sarg, $value) = split(/=/, $arg);
    $value =~ tr/+ / /;
    $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;
    if (($sarg eq 'emrname') && ($value eq "")) { $ARGS{$sarg} = '.' }
    else { $ARGS{$sarg} = $value }
}
$sort_order = ($ARGS{'sort'} || 0);
$directory = $ARGS{'dir'};
$VAR{'emrname'} = $ARGS{'emrname'} if $ARGS{'emrname'};
$VAR{'root_name'} = $ROOT_NAME;

#####
# Perform security and validity checks on directories and files #
#####

# Set the directory delimiter based on $ROOT_DIR and make sure the
# different directory variables end or don't end with a delimiter as
# called for.
$DD = substr($ROOT_DIR, -1);
```

```

if ($DD !~ /[V:]/) {
    &error("\$ICON_DIR must end with a directory delimiter.");
}
if ($ALLOWED_DIR !~ /$DD$/) { $ALLOWED_DIR .= $DD }
if ($directory && (substr($directory, 0, 1) ne $DD)) {
    $directory = "$DD$directory";
}
$directory =~ s/$DD$/g;
$VAR{'directory'} = $directory if $directory;
$VAR{'full_dir'} = "$ROOT_NAME$directory";

# Change to root in case an invalid directory was specified

if (!$ROOT_DIR) { &error('$ROOT_DIR has not been defined.') }
if (!( -d $ROOT_DIR )) { &error("$ROOT_DIR is not a valid directory ($!).") }
if (!( -r $ROOT_DIR )) { &error("$ROOT_DIR is not readable ($!).") }
chdir($ROOT_DIR);

# Make sure they're not trying to access an invalid directory

if ($directory =~ /$DD\\.\/) { $directory = " }
$ARGS{'head'} =~ s/^(^$ALLOWED_DIR)(^$DD)(\\.\\.$DD|$)/g;
$ARGS{'foot'} =~ s/^(^$ALLOWED_DIR)(^$DD)(\\.\\.$DD|$)/g;

#####
# Perform search or get directory listing #
#####

if ($ARGS{'emrname'}) {
    $VAR{'page_title'} = "Search of $ROOT_NAME$directory for
\"$ARGS{'emrname'}\"";

    # Recursively search directory

    &search("$ROOT_DIR$directory", "", $ARGS{'emrname'});
    chdir("$ROOT_DIR$directory");
}
else {
    $VAR{'page_title'} = "Directory of $ROOT_NAME$directory";

    # Read list of files

    chdir("$ROOT_DIR$directory");
    @link_files = @files = <*>;

```

```

}
$num_items = @files;

#####
# Get file information                                     #
#####

$count = 0;
foreach $file (@link_files) {
    $filename = $files[$count];
    $directory{$filename} = -d $file;                # Directory flag
    if (!$directory{$filename}) { $size{$filename} = -s $file } # Size (in bytes)
    $modify{$filename} = $^T - int((-M $file) * $DAY_SECS); # Modification date
    $readable{$filename} = -r $file;                  # Read permission
    $full_path{$filename} = $file;                    # Path from dir

    # Determine file type

    $_ = $file;
    tr/A-Z/a-z/;
    if ((/\.[^.]$/) && $TYPES{$1}) { $type{$filename} = $1 } # Use extension
    elsif (-B $file) { $type{$filename} = 'binary' }        # Check for binary
    else { $type{$filename} = 'txt' }                       # Otherwise text
    ++$count;
}

#####
# Sort files                                              #
#####

if ($sort_order == 1) {
    @sorted_files = sort by_size @files;
    $SIZE_TITLE = "<B>Size</B>";
}
elseif ($sort_order == 2) {
    @sorted_files = sort by_date @files;
    $DATE_TITLE = "<B>Last Modified</B>";
}
else {
    @sorted_files = sort by_name @files;
    $NAME_TITLE = "<B>Name</B>";
}

```

```
#####
# Generate the HTML page #
#####

# Generate the arguments for links back to the program

if ($ARGS{'sort'}) { $sort_arg = "sort=$sort_order&" } else { $sort_arg = " }
if ($ARGS{'head'}) {
    $temp_arg .= "head=$ARGS{'head'}&";
}
if ($ARGS{'foot'}) {
    $temp_arg .= "foot=$ARGS{'foot'}&";
}

# Generate the HTML header

print "Content-type: text/html\n\n";

# Insert the page header if specified

if ($ARGS{'head'}) {
    if (!&parse_template("$ALLOWED_DIR$ARGS{'head'}", *STDOUT)) {
        &error($Error_Message, "", 1);
    }
}
else {
    print <<END_HTML;
<HTML>
<HEAD>
<TITLE>$VAR{'page_title'}</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
END_HTML
}

# Generate the directory header

print <<END_HTML;
<B>$VAR{'page_title'}:</B>
<P>
<TABLE BORDER=0 CELLPADDING=0 WIDTH=100%>
<TR>
<TD HEIGHT=17 ALIGN=CENTER WIDTH=50 NOWRAP>
    $num_items items
```



```

        </TD><TD WIDTH=5 ALIGN=LEFT>
            &nbsp;
        </TD><TD ALIGN=LEFT>
            $NAME_TITLE
        </TD><TD ALIGN=RIGHT WIDTH=100>
            $SIZE_TITLE
        </TD><TD WIDTH=5 ALIGN=LEFT>
            &nbsp;
        </TD><TD ALIGN=LEFT WIDTH=900>
            $DATE_TITLE
        </TD></TR>
    <TR>
        <TD COLSPAN=6 HEIGHT=10>
            <HR>
        </TD></TR>
END_HTML

```

Generate the link to the parent directory if appropriate

```

if ($directory && !$ARGS{'emname'}) {
    $parent = substr($directory, 0, rindex($directory, $DD));
    $file_link = "<A HREF=\"\$PROGRAM_URL?$sort_arg$temp_arg\"";
    $file_link .= "dir=$parent\">\n";
    $file = '<I>Parent Directory</I>';
    $link_close = '</A>';
}

```

Generate the directory listing

```

foreach $file (@sorted_files) {

```

If the file is a directory, link back to the program

```

if ($directory{$file}) {
    $file_link = "<A HREF=\"\$PROGRAM_URL?$sort_arg$temp_arg\"";
    $file_link .= "dir=$directory$DD$full_path{$file}\">";
    $image_link = $TYPES{'directory'};
    $file_size = '-';
}

```

```

else {

```

Otherwise link to the file and format the file size

```

    $file_link = "<A HREF=\"\$ROOT_URL$directory$DD$full_path{$file}\">";
    $image_link = $TYPES{$type{$file}};

```

```

    $file_size = &size_format($size{$file});
}

# Remove the link if the file isn't readable by the user

if (!$readable{$file}) { $file_link = $link_close = " }
else { $link_close = '</A>' }

# Calculate the modification date

$mod_date = &format_date($modify{$file}, $DATE_FORMAT);

if ($ARGS{'emname'}) { $file = $full_path{$file} }

# Print the file information

print <<END_HTML;
    <TR>
        <TD HEIGHT=17 ALIGN=CENTER>
            $file_link
            <IMG SRC="$image_link" WIDTH=16 HEIGHT=16 BORDER=0>$link_close
        </TD><TD ALIGN=RIGHT>
            &nbsp;
        </TD><TD ALIGN=LEFT>
            $file_link$file$link_close
        </TD><TD ALIGN=RIGHT>
            $file_size
        </TD><TD WIDTH=5 ALIGN=LEFT>
            &nbsp;
        </TD><TD ALIGN=LEFT>
            $mod_date
        </TD></TR>
END_HTML
}
print "    </TABLE>\n    <HR>\n";

# Insert the page footer if specified

if ($ARGS{'foot'}) {
    if (!&parse_template("$ALLOWED_DIR$ARGS{'foot'}", *STDOUT)) {
        &error($Error_Message, "", 1);
    }
}
else { print "    </BODY>\n</HTML>" }

```

```
#####
# Search subroutine                                     #
#####

sub search {

    # Initialize

    local($SEARCH_ROOT, $search_dir, $search_query) = @_;
    chdir("$SEARCH_ROOT$DD$search_dir");
    local(@filenames) = <*>;
    local($file);

    # Scan directory & subdirectories

    foreach $file (@filenames) {
        if (-d $file && -r $file) {
            if ($search_dir) {
                &search($SEARCH_ROOT, "$search_dir$DD$file", $search_query);
            }
            else { &search($SEARCH_ROOT, $file, $search_query) }
            chdir("$SEARCH_ROOT$DD$search_dir");
        }
        if ($file =~ /$search_query/i) {
            if ($search_dir) { push(@link_files, "$search_dir$DD$file") }
            else { push(@link_files, $file) }
            push(@files, $file);
        }
    }
}

#####
# File size format subroutine                             #
#####

sub size_format {

    # Initialize

    local($size) = $_[0];

    # Format

```

```

if ($size < 1024) { return("1K") }
elseif ($size < 1048576) { return(int($size/1024 + .5) . "K") }
else { return((int(10 * $size/1048576 + .5) / 10) . " MB") }
}

```

```

#####
# Custom sort subroutines                                     #
#####

```

```

sub by_size {
    $a2 = $a;
    $a2 =~ tr/A-Z/a-z/;
    $b2 = $b;
    $b2 =~ tr/A-Z/a-z/;
    ($size{$b} <=> $size{$a}) || ($a2 cmp $b2) || ($modify{$b} <=> $modify{$a});
}

```

```

sub by_date {
    $a2 = $a;
    $a2 =~ tr/A-Z/a-z/;
    $b2 = $b;
    $b2 =~ tr/A-Z/a-z/;
    ($modify{$b} <=> $modify{$a}) || ($a2 cmp $b2) || ($size{$b} <=> $size{$a});
}

```

```

sub by_name {
    $a2 = $a;
    $a2 =~ tr/A-Z/a-z/;
    $b2 = $b;
    $b2 =~ tr/A-Z/a-z/;
    ($a2 cmp $b2) || ($a cmp $b);
}

```

APPENDIX B

THE SOURCE CODE OF THE PROGRAM *uploademr.cgi*

```
#!/usr/bin/perl
#
# File: uploademr.cgi
#
# set up configurable options.

BEGIN {
    $SAVE_DIRECTORY = "/home/col/httpd/update";
    $ALLOW_INDEX = 0;
    $SUCCESS_LOCATION = ""
}

#####
$_ = 1;
chop $SAVE_DIRECTORY if ($SAVE_DIRECTORY =~ /\$/);
use CGI qw(:standard);
$query = new CGI;

if ( !(-e $SAVE_DIRECTORY) ||
    !(-w $SAVE_DIRECTORY) ||
    !(-d $SAVE_DIRECTORY) ) {
    print header;
    print <<__END_OF_HTML_CODE__;

    <HTML>
    <HEAD>
    <TITLE>Error: Bad Directory</TITLE>
    </HEAD>
    <BODY BGCOLOR="#FFFFFF">
    <H1>Bad Directory</H1>
    <P>
    The directory you specified:
    <BR>
    <BLOCKQUOTE>
    <TT>\$SAVE_DIRECTORY =
        "<B>$SAVE_DIRECTORY</B>";</TT>
    </BLOCKQUOTE>
    <BR>
```

is invalid. This problem is caused by one of the three following reasons:

The directory doesn't exist. Make sure that this directory is a complete path name, not a URL or something similar. It should look similar to

<TT>/home/username/public_html/uploads</TT>

<P>

The directory isn't writable. Make sure that this directory is writable by all users. At your UNIX command prompt, type <TT>chmod 777 \$SAVE_DIRECTORY</TT>

<P>

The directory you specified isn't really a directory. Make sure that this is indeed a directory and not a file.

<HR SIZE=1>

</BODY>

</HTML>

exit;

}

foreach \$key (sort { \$a <=> \$b } \$query->param()) {

next if (\$key =~ /^s*\$/);

next if (\$query->param(\$key) =~ /^s*\$/);

next if (\$key !~ /^file-to-upload-(\d+)/);

\$Number = \$1;

if (\$query->param(\$key) =~ /([^\v\\]+)/)

{

\$Filename = \$1;

\$Filename =~ s/^\./+//;

\$File_Handle = \$query->param(\$key);

if (!\$ALLOW_INDEX && \$Filename =~ /^index/i) {

print header;

print

<<__END_OF_HTML_CODE__;

<HTML>

<HEAD>

<TITLE>Error: Filename

Problem</TITLE>

```

        </HEAD>
        <BODY BGCOLOR="#FFFFFF">
        <H1>Filename Problem</H1>
        <P>
        You attempted to upload a
        file that isn't properly
        formatted. The system
        administrator has decided
        that you can't upload
        files that begin with the
        word '<B>index</B>'.
        Please rename the file on
        your computer, and try
        uploading it again.
        <P>
        <HR SIZE=1>
        </BODY>
        </HTML>
__END_OF_HTML_CODE__
        exit;
    }
} else {
    $FILENAME_IN_QUESTION = $query-
        >param($key);
    print header;
    print <<__END_OF_HTML_CODE__;
    <HTML>
    <HEAD>
        <TITLE>Error: Filename
        Problem</TITLE>
    </HEAD>
    <BODY BGCOLOR="#FFFFFF">
    <H1>Filename Problem</H1>
    <P>
    You attempted to upload a file
    that isn't properly formatted.
    The file in question is
    <TT><B>$FILENAME_IN_QUESTION</B></TT>
    Please rename the file on your
    computer, and attempt to upload it again. Files may not have forward or
backward slashes in their
    names. Also, they may not be prefixed with one (or more) periods.
    <P>
    <HR SIZE=1>
    </BODY>

```

</HTML>

07:52:28.61

```
__END_OF_HTML_CODE__
    exit;
}
```

```
if (!open(OUTFILE, ">$SAVE_DIRECTORYV$Filename")) {
    print "Content-type: text/plain\n\n";
    print "-----\n";
    print "Error:\n";
    print "-----\n";
    print "File: $SAVE_DIRECTORYV$Filename\n";
    print "-----\n";
    print "There was an error opening the Output File\n";
    print "for Writing.\n\n";
    print "Make sure that the directory:\n";
    print "$SAVE_DIRECTORY\n";
    print "has been chmodded with the permissions '777'.\n\n";
    print "Also, make sure that if your attempting\n";
    print "to overwrite an existing file, that the\n";
    print "existing file is chmodded '666' or better.\n\n";
    print "The Error message below should help you diagnose\n";
    print "the problem.\n\n";
    print "Error: $!\n";
    exit;
}

undef $BytesRead;
undef $Buffer;

while ($Bytes = read($File_Handle,$Buffer,1024)) {
    $BytesRead += $Bytes;
    print OUTFILE $Buffer;
}

push(@Files_Written, "$SAVE_DIRECTORYV$Filename");
$TOTAL_BYTES += $BytesRead;
$Confirmation{$File_Handle} = $BytesRead;

close($File_Handle);
close(OUTFILE);

chmod (0666, "$SAVE_DIRECTORYV$Filename");
}
```



```

$FILES_UPLOADED = scalar(keys(%Confirmation));

if ($TOTAL_BYTES > $MAXIMUM_UPLOAD && $MAXIMUM_UPLOAD >
0) {
    foreach $File (@Files_Written) {
        unlink $File;
    }

    print header;
    print <<__END_OF_HTML_CODE__;

    <HTML>
    <HEAD>
        <TITLE>Error: Limit Reached</TITLE>
    </HEAD>
    <BODY BGCOLOR="#FFFFFF">
    <H1>Limit Reached</H1>
    <P>
        You have reached your upload limit. You attempted to upload
    <B>$FILES_UPLOADED</B> files, totalling
        <B>$TOTAL_BYTES</B>. This exceeds the maximum limit of
    <B>$MAXIMUM_UPLOAD</B> bytes, set by the system
        administrator. <B>None</B> of your files were successfully saved. Please try
again.
    <P>
    <HR SIZE=1>
        </BODY>
    </HTML>

__END_OF_HTML_CODE__
    exit;
}

if ($$SUCCESS_LOCATION !~ /^\/s*$/) {
    print $query->redirect($$SUCCESS_LOCATION);
} else {
    print header;
    print <<__END_OF_HTML_CODE__;
    <HTML>
    <HEAD>
        <TITLE>Upload Finished</TITLE>
    </HEAD>
    <BODY BGCOLOR="#FFFFFF">
    <H1>Upload Finished</H1>

```

```

    <P>
    You uploaded <B>${FILES_UPLOADED}</B> files totalling
    <B>${TOTAL_BYTES}</B> total bytes. Individual
    file information is listed below:
    <PRE>

```

```

__END_OF_HTML_CODE__

```

```

    foreach $key (keys (%Confirmation)) {
        print "$key - $Confirmation{$key} bytes\n";
    }

```

```

    print <<__END_OF_HTML_CODE__;

```

```

    </PRE>

```

```

    <P>

```

```

    Thank you for using the File Upload! system.

```

```

    <P>

```

```

    <HR SIZE=1>

```

```

    </BODY>

```

```

    </HTML>

```

```

__END_OF_HTML_CODE__

```

```

    exit;

```

```

}

```

VITA

Weiming Gan

Candidate for the Degree of
Master of Science

Thesis: DESIGNING A WEB *EES* MODIFICATION REQUEST (*EMR*) SYSTEM

Major Field: Computer Science

Biographical:

Education: Received Bachelor of Engineering degree in Mechanical Engineering from Xi'an Institute of Highway Technology, Xi'an City, Shanxi Province, People's Republic of China in Jun 1982. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December, 1999.

Experience: Employed by China No 2 Automotive Manufacture as a mechanical engineer, Beijing, P. R. China; employed by Oklahoma State University, Department of Computer Science as a graduate research assistant; employed by the Lucent Technologies Inc. as a software engineer, 1997 to present.

Professional Memberships: Asian/Pacific American Association for Advancement-
Lucent Technologies