

**THE DESIGN AND IMPLEMENTATION
OF A HYPERTEXT DOCUMENT MANAGEMENT SYSTEM**

by
Bikram Day

A THESIS

Presented to the Division of Medical Informatics and Outcomes Research
and the Oregon Health Sciences University
School of Medicine

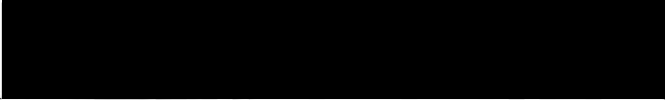
in partial fulfillment
of the requirements for the degree of
Master of Science

May 1998

School of Medicine
Oregon Health Sciences University

CERTIFICATE OF APPROVAL

This is to certify that the M.S. thesis of
Bikram Day
has been approved


Chair: William Hersh, M.D.


Christopher Dubay, Ph.D.


Paul Gorman, M.D.


Andrew Zechnich, M.D. M.P.H.



Associate Dean for Graduate Studies

TABLE OF CONTENTS

I. Acknowledgements.....	v
II. Abstract.....	vi
III. Introduction	1
1. Overview	1
2. Background	3
3. Goals of this project	6
4. Previous work	9
IV. Implementation.....	13
1. Methodology	13
1.1. System Design	13
1.2. System Layout	14
1.3. Document Acquisition Engine (DAE)	17
1.4. Document Control Interface (DCI)	20
1.4.1 DAE user interface.....	21
1.4.2 Cluster creator	21
1.4.3 Browser/selector interface.....	22
1.4.4 Meta-data browser/creator	22
1.5. Document Processing Engine(DPE)	23
1.5.1 RDBMS driver	23
1.5.2 Clustering Module.....	24

1.5.3 Topology Module.....	25
1.5.4 Indexing and Search engine	28
2. Implementation Details	31
2.1. Description of the tables	31
(1) Table 'urls'	31
(2) Table 'dirs'	32
(3) Table 'links'	33
(4) Table 'clusters'	34
(5) Table 'doc_cluster'	34
(6) Table 'meta_data'	34
(7) Table 'alt_title'	35
2.2. Unique Identifiers	35
2.3. Document Titles.....	36
2.4. Generating Alternate Document Titles	37
2.5. Meta-data Extension	37
2.6. Using the Meta-data Table.....	38
2.7. Using Clusters.....	39
V. Discussion	41
1. Scope of this project	41
2. Performance Assessment and Evaluation.....	43
3. Future work	45
VI. Conclusion.....	46
VII. References	48

VIII. Appendices	52
Appendix A. Definitions:	52
Appendix B. Table Structures	53
Appendix C. DAE Codes For Documents	54
Appendix D. Command Line Parameters for the DAE	55
Appendix E. Screen Snapshots of the Program	56

TABLE OF FIGURES

<i>Figure 1. Software layout of the system showing the inter-component interfaces.....</i>	<i>15</i>
<i>Figure 2. Logical layout of the system.</i>	<i>16</i>
<i>Figure 3. Link patching by the DAE to preserve the link integrity.</i>	<i>18</i>
<i>Figure 4. Hypertext documents as a directed graph.....</i>	<i>27</i>
<i>Figure 5. The same directed graph as relational table: 'R_TABLE'.</i>	<i>27</i>
<i>Figure 6. Examples of SQL statements to traverse the document topology.....</i>	<i>27</i>

I. Acknowledgements

I would like to thank my advisor, Bill Hersh for his generous support and encouragement in all my research endeavors in Medical Informatics and Information Retrieval, including this thesis. His experience and guidance have been an enriching experience for me over the past few years.

I would also like to express my appreciation for my thesis committee members Christopher Dubay, Paul Gorman and Andrew Zechnich for their guidance and suggestions.

Thanks are due to all the members of Dr. Hersh's research group, both past and present, whose work has been the foundation for the project.

My wife Sheena has been my inspiration and has always been there for me. I could not have done this without her.

II. Abstract

We have developed a hypertext document management system designed to specifically handle Web pages. Our system should benefit those who need to collect, index, review and organize Web documents from different sites, and use these documents to extract information, create a meta-index, or an archive from these documents. This system will allow the user to utilize information from the document text as well as its hyper-link context and any associated meta-data. We have built this system from small functional and interchangeable modules of software, some of which were pre-existing programs. The backbone of our system is a relational database management system where the extracted information from the documents is stored. All of these components may also be used by a developer as a development platform to build a more powerful and elaborate document management system. This system is not designed for authoring new content for the Web, nor is it a search engine for end-users. To achieve our goal in providing an integrated system for efficient handling of hypertext documents, we have exploited the inherent features and capabilities of the document collections themselves. These attributes have been long ignored due to the lack of proper tools that could utilize these features.

III. Introduction

1. Overview

We have designed and developed an application framework for managing hypertext documents to be utilized by anyone who is using hypertext document collections or Web documents from diverse sources. The user may be using these documents for research, collection and review, or for constructing a meta-index. This system may also be used as a framework to build more powerful and comprehensive hypertext document management tools.

As a development model and test bed, we are using this prototype hypertext management system for managing Cliniweb[6]. Cliniweb is a hierarchical directory of clinically relevant Web sites. Cliniweb's directory hierarchy is based on the MeSH (Medical Subject Headings) hierarchy[14] and it allows a user to browse or search this directory tree using any Web browser. The leaf (terminal) nodes of this directory are the Web pages located at different sites on the World Wide Web (WWW). Since MeSH is based on a controlled vocabulary, the user is provided with unambiguous choices and can usually locate relevant documents quickly and easily by traversing the hierarchy. The problem is that a human indexer or reviewer must classify every new document and determine where it belongs in the hierarchy before it is added to the Cliniweb directory. To maintain and revise Cliniweb, new Web sites have to be added and current sites updated and all the links have to be classified and categorized just like the books in a

library. Since the candidate sites for inclusion in Cliniweb is growing exponentially, it is clearly impossible to tackle this task manually without the help of specialized tools.

We need specific tools and functionality to collect and manage the hypertext documents from the Web, especially those documents in which we are interested. Specifically, we need to:

- Select and save these documents, or save enough information to get back to the document later (e.g., URLs).
- Save information about which other documents a particular document refers to and which other documents refer to it. This information is required to determine the context of a document and to browse the document hierarchy.
- Locate documents that contain specific terms or concepts, using a searching system.
- Annotate the documents with the reviewer's comments or classifications.
- Index and classify the documents into categories or groups.
- Access the meta-data information of the document, or other information not explicitly described in the document text. This and all the other requirements mentioned above might have to be satisfied without the original document itself being present locally.
- Display the documents and all their associated information in a format most useful to the user.

Our system addresses these issues and attempts to build the foundation for a more comprehensive framework that would be able to achieve these goals in a more generalized context.

The specific aims of this project are to:

- Develop algorithms for selection, clustering and retrieval of hypertext documents.
- Implement this framework as an application or toolkit.
- Design evaluation strategies for performance assessment.

This system has not been designed to be an authoring tool for creating novel content for the Web. It is also not a search engine for locating information on the World Wide Web, as used by the final consumers of the information, that is, the end-users. This is a tool for librarians, indexers, reviewers, or anyone who creates an archive of specific Web documents and has the need to organize and effectively utilize this archive.

2. Background

The problems faced by any user trying to locate useful information on the Web are as diverse as the information itself. This multidimensional document space is created by multiple attributes that are intrinsic to the documents, the links between the documents and the document collections as a whole.

The greatest hurdles in locating useful information are:

- The sheer volume of information[16] — both the documents size and their number.
- An adverse signal to noise ratio of the document content in any large collection.
- The dynamic and diverse nature of the content and the format of the web documents.

Apart from the text content of the document, the content quality and the target audience varies widely in a collection as diverse as the WWW.

Document locating strategies today can be one of two types:

- 1) Search engines of varying degrees of complexity, from simple search engines to complex engines, with features such as Boolean and natural language search options and relevance feedback.[26] (*E.g., Altavista [www.altavista.digital.com], excite [www.excite.com], Lycos[lycos.com], Verity[www.verity.com]*[16]). These systems use word-based indexing systems that process web pages downloaded automatically by a program traversing the hyperlinks and build comprehensive indices that are accessible from the web. The search interfaces are the front ends to these indices and these interfaces implement the variety of complex features mentioned above.
- 2) A browsable directory-like structure composed of a hierarchy of hyperlinked documents. (*E.g. Cliniweb[www.ohsu.edu/clinweb], Yahoo[www.yahoo.com]*) These hierarchies are created manually by assigning documents to one or more specific categories, a task accomplished by trained indexers or reviewers.

The popularity of hierarchical or directory-type searching tools on the Internet like Cliniweb or Yahoo indicates that they may have certain advantages over fully automated search engines, which perform Boolean or Natural language searches. The most obvious advantage of these hierarchical-directory based systems is that their contents have been reviewed and classified based on a clearly defined structure or a controlled vocabulary. This structural approach is preferred by many novices as it is less frustrating to reach high quality, reviewed documents without having to navigate through irrelevant pages or without having to phrase effective queries. Most experts prefer this type of searching when they require the summarized overview of topics for preliminary consideration or research.

It may be speculated that the stability and predictability of a hierarchical or directory structure appeals more to the human mind. This impression may be magnified by the perceived instability and randomness of the results of a text-word query on a large and dynamic corpus such as the Web. Since the inclusion criteria for such a hierarchical index may include some measure of quality or legitimacy, it shifts the burden of selection and classification from the consumer of the information to the indexer or reviewer who includes the document in the collection.

Hierarchical or directory type searching tools are not as comprehensive or up-to-date in their content compared to Web search engines that automatically index web pages without human intervention. The major disadvantages of the hierarchical systems can be traced to their singular dependence on the human indexer or reviewer. It is too

expensive to create and maintain a system that has reviewed content and that is always current and as comprehensive as a word-based search engine. The logistics for such an attempt would most probably fail, based simply on the fact that there would not be enough manpower or resources to review the documents for inclusion into a directory, in real-time.

3. Goals of this project.

This thesis proposes to develop tools that would assist and empower the reviewer or indexer to process documents quickly and more comprehensively. This is achieved by eliminating (by automation of) repetitive and redundant tasks and allowing the reviewer to focus more on the exceptions and novel content. Thus, the resource utilization is optimized and the reviewers can work more efficiently.

As mentioned in the introduction, the need to develop this system emerged when Cliniweb's maintenance and expansion was becoming increasingly difficult due to lack of proper tools. Scalability problems of the periodic revisions and updates of the hyperlinks in the Cliniweb website were not addressed by any available application. The problems faced while expanding and enhancing Cliniweb are typical of anyone trying to implement a system that requires the management of a large and dynamic document collection or web site. Most of the tasks to maintain a large site with many hypertext links to documents scattered over the Web are accomplished manually. These tasks include adding new links, deleting redundant or defunct links, or updating changes to

links. These tasks have to be tackled manually because there is great variability in the content and the format of these documents and almost everything occurs asynchronously. Automation of such a process would involve too many variables and would be very complex to implement. Our approach would be to automate the least variable elements of this process and provide effort-multiplier tools to the human reviewer who would then be able to tackle the more variable factors efficiently and exhaustively.

Addition of the links to the collection can be one of the most resource intensive tasks that are currently accomplished entirely manually with no automated help. Although automation of the link reviewing and addition process cannot totally dispense with the human factor, it would simplify the task and can vastly improve the reviewer's efficiency and productivity.

The human reviewer has to browse and sift through almost all the seemingly relevant links in a site before issuing a judgment. This is time and resource consuming, especially when only a few of those links may be actually relevant and get finally selected for the collection. It would help the reviewer if the entire link structure could be visualized as a summarized tree hierarchy. The reviewer can then selectively ignore the obviously irrelevant links and its sub-trees without having to peruse through all the material in those pages. Since the entire hierarchy tree is visible, the reviewer can jump to an interesting or relevant page without having to traverse through the intermediate pages.

The current method of updating Cliniweb is almost entirely manual. These manual procedures are not scalable and are extremely resource intensive, even on a small collection of documents. It is almost impossible to keep up with the proliferation of new sites and pages and incorporate them into the current collection in a timely manner. Consequently, a fraction on the collection soon becomes dated and some of the links may become defunct. A follow-up mechanism that can track changes to an already reviewed site and automatically update the database or, if necessary, alert the reviewer of additions or changes and will shift this burden to the system instead of the reviewer.

Most of the reviewing of the web sites and pages for inclusion into Cliniweb is done online— it is considerably slow due to the network availability and lag-time. This time factor and therefore the reviewer's efficiency can be significantly improved by having a local mirror of the web pages being scrutinized. Thus, the reviewer can browse the pages offline from the local mirror. In addition, this would allow the reviewer to examine all the links exhaustively and if necessary, index the pages or use them as input for any processing program. This will prevent individual pages that are slow to be transmitted over the network or temporarily unavailable, from being lost or ignored. Currently, the causes for failure or non-inclusion of a page are not tracked. An automated system would be able to track protocol or network failures and recover from non-fatal errors. All this would make the reviewing process more comprehensive and exhaustive and make the output more reliable.

There are also specific problems associated with managing large document collections:

- There are no standardized methods for creating or managing document collections.
- The document archives have to be retrievable and relocate-able as well as scalable.
- There are diverse challenges in the reuse, distribution and even the extraction of data from the document archives.
- There are the universal requirements for speed and economy.

One of the major drawbacks of the search systems discussed above is that they mostly ignore the topological information associated with the hypertext.

4. *Previous work.*

Apart from the now ubiquitous search engines on the Internet there are some commercial and research systems that attempt to address some of these issues. No single one of these systems offers complete solutions to all these problems. Each of the concepts and systems discussed below has some novel contribution towards a more powerful document management system. Some of the concepts used in these systems have inspired the development of this project.

Clustering of hypertext documents[3] based on the intrinsic properties of the hypertext content is an effective way to handle large document collections by grouping them into manageable clusters. This technique has been extensively used in our project. Other clustering related techniques include "scatter and gather", a type of feedback technique that is used to browse large document collections. Clustering methodologies in information retrieval are concerned mostly with techniques to automatically categorize

documents into groups[2,12]. This categorization may be based on terms in the documents, or on co-occurring citations used in these documents. When extended to hypertext documents, clustering may be based on the hyper-links between documents[2,28,30]. The factors used to categorize documents into clusters may be any quantitative measure of associativity or similarity or even any measure of distance between two documents. Clustering research has focused on selecting the attributes that are the basis for clustering, on selecting appropriate clustering method and developing efficient algorithms for implementing these methods.

This project uses clusters created manually by selecting or categorizing several documents into groups. These clusters or groups may be arbitrarily created by the user based on any attribute of the document or source. This system will also have the ability to automatically create clusters of documents based on the hyper-link topology. This feature is discussed in detail in a following section.

WebGlimpse [19,20] is a system that combines browsing and searching. It introduces the concept of the neighborhood of a document, allowing localized or directed searching and browsing.

Personal Web notebook [15] is a system that uses the analogy of a scrapbook to collect pertinent fragments from documents on the Web and allows the user to create a composite document incorporating annotations and user-defined markup. This system

defines a very powerful summarization and browsing tool, that extends the bookmark and scrapbook analogies.

WebSQL [23] defines the syntax and semantics of a query language to retrieve web documents. This system is aware of the topology and link structure as well as the document content and meta-information. *WebSQL* may be a reasonable choice for developing structured queries in complex document management systems.

There are also a few commercial systems for managing hypertext or Web documents. Some of these are: *Documentum*-[9] a large document management system, *The Info-Sleuth* (Carnot)project[8] and *Trellix*[29]. Most of these systems have proprietary processing engines and are for use by content creators and end-users. Our target users have slightly different needs and therefore require a more customized solution. These commercial systems have nevertheless provided many useful design paradigms for our prototype.

To summarize the features of an ideal hypertext document management system, we would envision a system with the following capabilities:

- Facilitate saving and archiving individual documents to local storage media.
- Retain all information related to a document's source (e.g., the URL), its size, type and any other information available that is associated with or describes the document.
- Retain the hyper-link information that describes the topology of the links between the hypertext documents. It should be stored in a comprehensive and flexible format.

- Allow for indexing and searching of the entire document collection to locate individual documents that contain the specified term or concept.
- Allow the user to review and add comments to individual documents and create groups of documents. The user may extract, inspect and add to the meta-data content of individual documents. These actions may not modify the original documents in order to maintain the integrity of the repository.
- Provide visualization tools to inspect the documents or any associated attributes.

IV. Implementation

In this section, we will describe the organization, design and implementation of our system. The first half of this section describes the overall organization, architecture and the functions of the various modules of the system. The final half of this section will discuss in more detail the data-structures used and specific design decisions of our approach.

1. Methodology.

1.1. System Design.

The prototype system was developed on a networked cluster of Sun Microsystems Sparc/UltraSparc workstations. A Sun Ultra-1 (143Mhz) with 52 GB of hard disk storage served as the Web-server and the database server for our cluster. The other computers on the cluster were Sun Sparcstation-4s and a Sparcstation-20 which shared the disk storage of the Sun Ultra-1 via NFS mounts over the network.

The development software used for the prototype systems were as follows: the operating system was Solaris 2.5.1.(SPARC). The Perl interpreter used was GNU Perl 5.004. The RDBMS engine used was Oracle 7.3 (for Solaris 2.5), running on the Sun Ultra-1. The Perl-Oracle API used was DBI-DBD (www.hermetica.com/tecnologia/DBI). The C/C++ compiler used was the GNU C/C++ compiler GCC version 2.7.1. The Webserver

for our prototype was the NCSA Webserver running on the Sun Ultra-1 hardware mentioned above.

When this system is used by a developer, as a toolkit for developing information retrieval systems, the programmer may use the Perl API provided to access the repository data structures. Other alternatives available to a developer are: *(i)* using the Perl-DBD interface to the RDBMS, or *(ii)* using SQL or the native procedural language of the RDBMS to access the RDBMS tables directly. In the case of our prototype, we have used Oracle PL/SQL as the procedural language.

Our prototype user interface was constructed using HTML and CGI-forms interfaces which were driven by Perl programs running on our Web-server. Since most modules are independent, they can execute on different CPUs in parallel with other modules. Messages and data are exchanged between these modules through intermediate files. Processes may also use other methods of inter-process communications like pipes and sockets to communicate with each other.

1.2. System Layout.

The system is designed as a set of individual freestanding modules, each of which provides a different functionality to the system. This modular black box design allows the system to be versatile, powerful and easily maintained. Adding new features and functionality would simply involve plugging in the new modules to the system. The

inter-relationship between the top-level modules is described in Figure 2. Logically the framework may be broken down into three functional parts:

- The Document processing engine- DPE.
- The Document acquisition engine- DAE.
- The Document control interface- DCI.

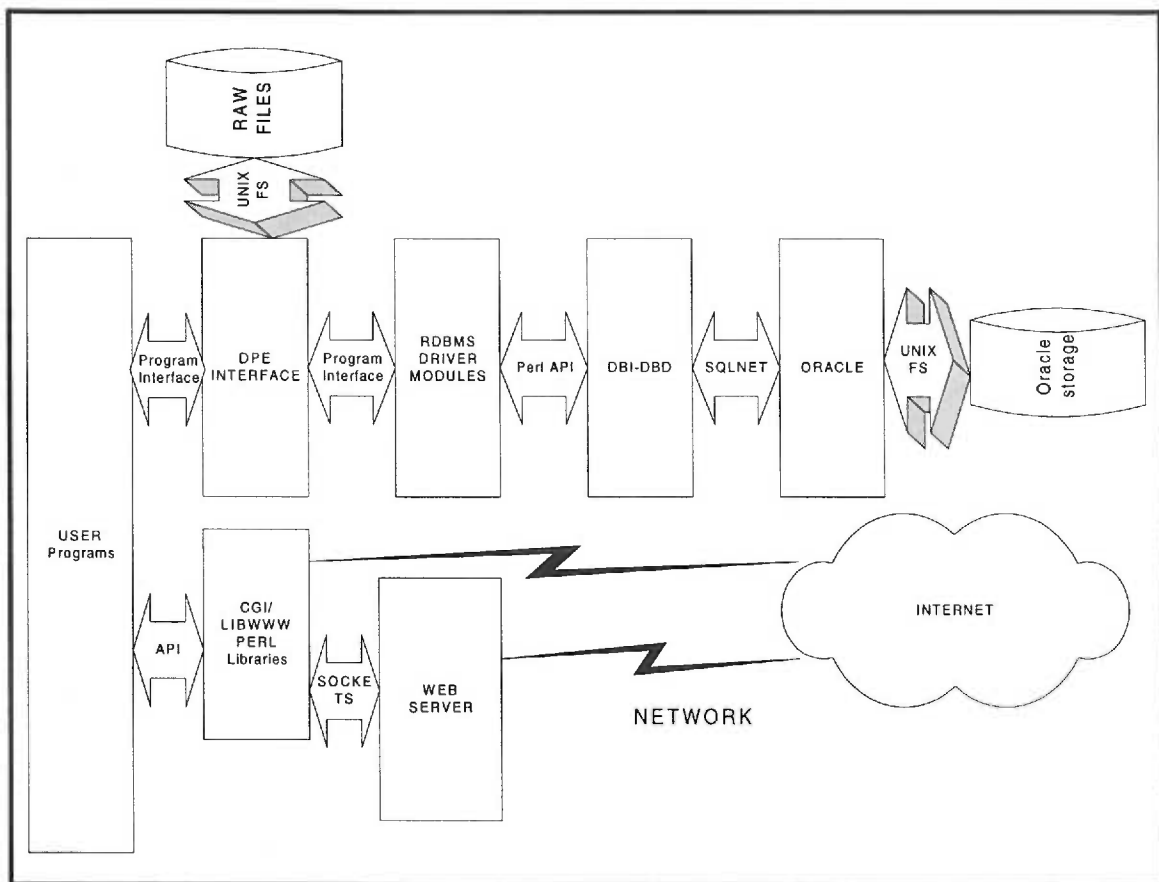
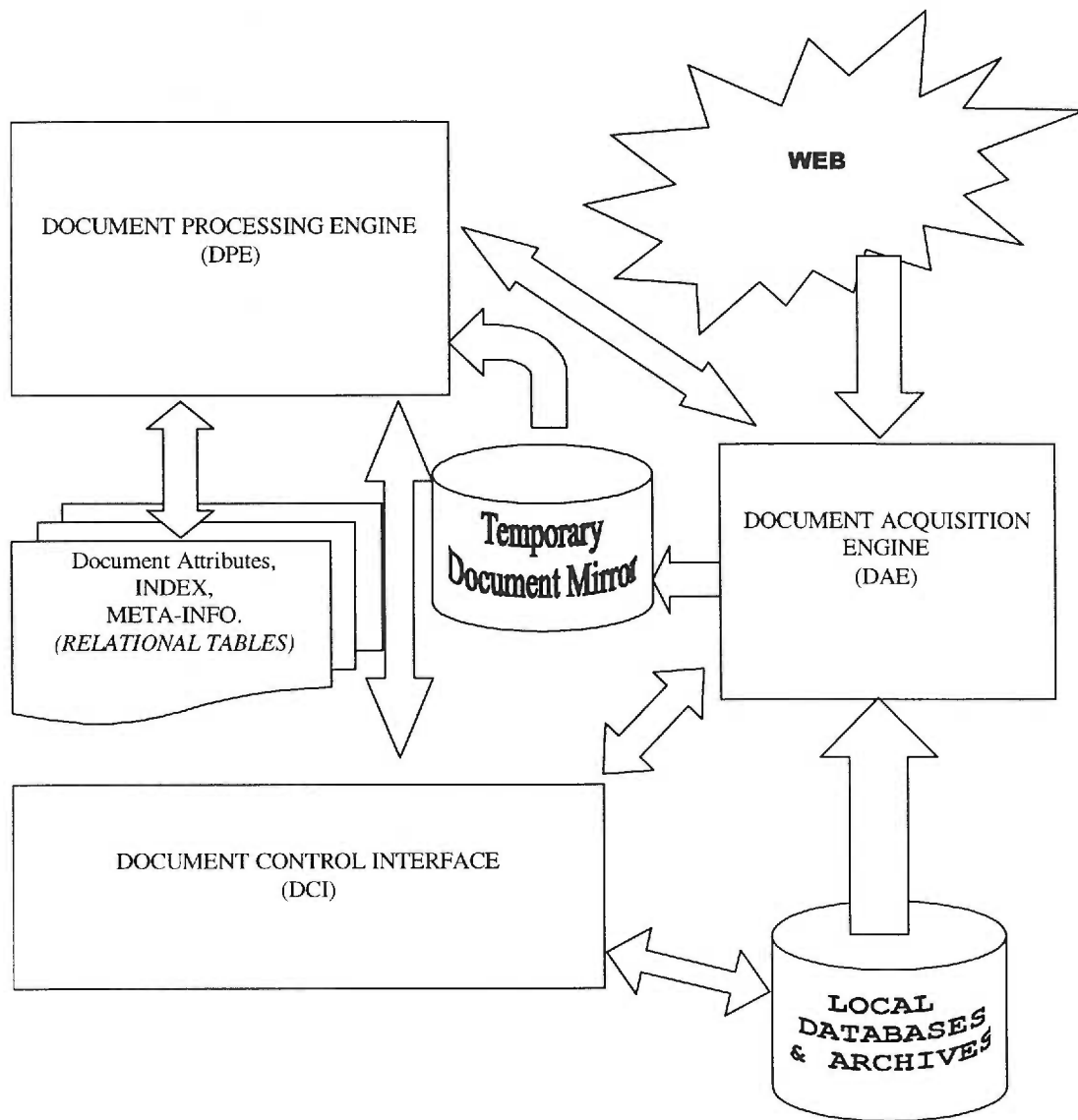


Figure 1. Software layout of the system showing the inter-component interfaces.



Top Level Schematic: Document Management System

Figure 2. Logical layout of the system.

1.3. Document Acquisition Engine (DAE)

The Document Acquisition Engine (DAE) is essentially a program that collects the documents from the Web or archives and does some pre-processing and data scrubbing before feeding this document input to the DPE. The DAE has the responsibility of capturing the topology information and any other information that is not present in the document itself. In this implementation, the DAE consists of three modules:

- A Web crawler (a robot).
- The Document Pre-processor.
- The Database interface.

This engine, in its current implementation, uses a Perl library (*lib-www-perl API: <http://www.linpro.no/lwp>*) as its underlying engine. This program is based on the "lwp-rget" program by Gisle Aas and has been modified to enhance and extend its capabilities to suit our requirements.

This robot visits the top-most (root node) document and downloads it to a local file. It then proceeds to traverse each link in that page and attempts to download the contents of that link, subject to command parameters. In case it does download a linked document, the original link is patched to point to the local copy of the document. Documents or links that are not fetched and stored locally, still point to the original copy on the Web. This patching process is illustrated in figure 3. This traversing and downloading

continues recursively until there are no more links to visit or the program achieves anyone of the preset limiting parameters.

```
Web document: http://medir.ohsu.edu/~dayb/index.html

      bgl.GIF
      bd-header.JPG

      http://medir.ohsu.edu/~dayb/medinf/medirusage.html
      http://medir.ohsu.edu/~dayb/comp.html
      http://www.vanderbilt.edu/HomePage/Internet.html
      http://www.intermarket.net/laughweb/master_by_score_ns.html
      http://iinwww.ira.uka.de/bibliography/index.html
```

```
As Downloaded by the DAE:  Local file: index000.html

      http://medir.ohsu.edu/~dayb/bgl.GIF
      http://medir.ohsu.edu/~dayb/bd-header.JPG

      medirusage.html
      comp.html
      http://www.vanderbilt.edu/HomePage/Internet.html
      http://www.intermarket.net/laughweb/master_by_score_ns.html
      index001.html

Shading Key:
Patched links to local files, Back-patched links to remote files, Untouched links.
```

Figure 3. Link patching by the DAE to preserve the link integrity.

The DAE robot can be constrained to fetch only a certain maximum number of documents per link as well as by limiting the depth that it would explore from the top-most supplied link. The system may also be constrained within a specific host, domain or directory structure (the URL prefix). All these limits prevent the robot from aimlessly wandering into the Web. These parameters are listed in Appendix D.

The documents that are downloaded are all stored in one subdirectory per root (top level) node, with unique file names (that are derived from their actual file name). This results in flattening a complex hierarchy into a simple one-dimensional directory structure. This eliminates the variability of complex directory hierarchies and allows for a simple file-naming scheme. As this program traverses each link and downloads certain documents from the Web, it simultaneously logs the document or link information in a relational table via the database interface. Thus, all the topological and hierarchical information of every link is retained, including the original hyperlink of each document, the time-stamps and any associated mime-types. Consequently, no information is lost in harvesting the documents and mirroring them locally. The details of this relational information are described in a following section.

The robot does not loop indefinitely or get confused when the URLs are self-referential or circular, since the system downloads any page only once. If it has seen the page before, it simply sets the link pointer to the unique local copy and moves onto the next link.

The advantages of using this robot are:

- There is no redundancy in the storage of documents or content in the local mirror.
- The system can be selective in fetching only specific (mime) types of documents; the robot can filter out graphics and multimedia content, saving network bandwidth and local storage space.
- Loops and self-referential links are handled gracefully; the content and links are transferred non-destructively.

This local mirroring of documents is exact; it is seamless and transparent to the browser, irrespective of the constraints used to download the documents. The DAE has the responsibility of capturing the topology information from the source, which may be the Web or a local digital-document archive. For example, the domain and path information for each document, its parent and children links and other related meta-data will be saved by the DAE into respective RDBMS tables.

1.4. Document Control Interface (DCI)

The document control interface (DCI) is the user interface to the system. The DCI allows multiple views of the document repository. It allows the user to visualize the topology of the document space. It also allows for the browsing of the meta-data tags and links. This interface allows the user to cluster, sort or rank them based on any available attribute.

The prototype system implements the DCI as a CGI/forms front-end client, which may run on a Web browser. This client connects to the DCI server running on a Web server. The DCI server forwards the user requests to the DPE, which returns the user replies to the DCI server in an open format. The DCI has the responsibility for creating the user-defined views and their rendition. This design allows for greater flexibility and efficiency due to the modularization and decentralization of the processing.

1.4.1 DAE user interface

This section discusses the user interface that drives the DAE. The prototype implementation uses CGI-forms as the user interface. The CGI script is a Perl program that parses the form input and uses the parameters specified to invoke the DAE robot. Appendix E shows a screen snapshot of this forms interface as displayed on a Web browser.

1.4.2 Cluster creator

The cluster creator interface is used to create clusters of documents manually, by selecting from a list of document titles. The titles themselves are hyperlinks so that the user may examine the document before making a selection. A document may be included in more than one cluster. Selections are made by selecting appropriate check boxes beside each document title. Appendix E shows an example of such a selection screen.

1.4.3 Browser/selector interface

This interface provides the basic capability of browsing the document repository. It shows the document hierarchy as a "table of contents" view, with a cascading list of links, a few (by default 3) levels deep. If the user wants to examine a document, the title hyperlinks will display the document contents in a separate frame below. The interface provides a column of check boxes beside the document links to create a cluster of "selected" documents.

1.4.4 Meta-data browser/creator

This interface allows the user to browse the meta-data information that is stored in the relational tables and associated with a specific document. It should be noted that this meta-data may be inclusive of the meta-data information embedded in the document itself and also contain additional meta-data present only in the relational table. The user may modify or add new meta-data information to be stored in the DPE tables using this interface.

Although the DCI provides a Web-based forms interface to certain features and functionality of this system, all this and more can also be accomplished using a command-line or program/API interface. In fact, the program/API interface is more suitable and efficient for batch processing by using a script that automates the individual processes and manages the parameters to the program. Another option available is that

the user may directly access the relational tables in the database, thereby bypassing both the API and the DCI.

1.5. Document Processing Engine(DPE)

The DPE is itself composed of many smaller modular engines. Each of these modules provides a different functionality to the system. Some of the sub-modules of this system are the:

- RDBMS Driver,
- Clustering Module,
- Topology Module,
- Indexing and Search Engine.

1.5.1 RDBMS driver

To handle the large volume of data that this system generates, its data storage has to be managed efficiently and flexibly. There should be no artificial or system dependent barriers (e.g., scalability) in the flow or manipulation of this information. Rather than develop the data structures and associated algorithms from scratch, we decided to use a commercial RDBMS to store and manipulate the data that may be readily converted into tables without loss of flexibility or function. The optimized and efficient database engine of a commercial RDBMS frees us from developing and troubleshooting the techniques to store all the information and makes the system modular and versatile. As described above, all the document information may be easily translated into normalized

relational tables. SQL statements and operations may be used to query these tables and derive composite information scattered over different sources. This type of data storage scheme creates an extremely robust and efficient data repository that can be extended almost indefinitely.

1.5.2 Clustering Module

This module is used to group document clusters so that an action may be performed collectively on that cluster of documents. The clusters may be created in the following ways:

- Manually, by selecting a group of documents (using the DCI), and assigning them to a cluster. A common example of a manually created cluster would be that of a user grouping documents into three clusters: relevant, non-relevant and undecided. The relevant cluster can be forwarded to an indexing engine, while the undecided cluster may be saved for future review.
- Automatically, by explicitly defining the criteria (based on a document attribute) that results in the selection of a group of documents from a larger pool. The cluster information is stored in relational tables and therefore, any relational algebra or set operation may be performed on these clusters. This empowers the user with extremely powerful and flexible tools to manipulate clusters of documents and perform complex operations without having to handle individual documents.
- Implicitly, the results or output of any operation, e.g., a search engine result, or a topology query output, may be interpreted as a cluster.

As discussed, these clusters may be used by the clustering module for other composite actions.

The document details in a cluster may be easily obtained by creating a relational "join" over the cluster table and the document detail tables, which are described in §2.1.

1.5.3 Topology Module

This module is used to analyze and process the physical and logical layout of a hypertext document collection. The hyperlink information present in every document determines the corresponding documents with which it is linked. The link implies that there may be some relationship between the contents of these documents. We may define a document attribute of "relevance by proximity" from such a hypertext link. This referential information, which is actually a directed graph (Figures 4-5), is captured by the topology engine and translated into a relational table. (Table 'links' in §2.1(3)). This allows us to traverse links in a document without actually browsing through the documents. Succinct relational operations can determine complex link hierarchies and create document clusters with similar topological characteristics (Figure 6).

The referential information that constitutes the links of a hypertext document collection and ultimately the Web, exists only in the documents themselves. This referential information of inter-linked documents may be represented mathematically as a directed graph. See Figure 4 for an illustration of this concept.

We have translated this directed graph representation of hypertext documents into a relational table format that retains all the information of the directed graph form. In essence, the source and target for each link is stored as an entry in the relational table (Figure 5.).

The benefits of using this relational table format are:

- The referential information can exist independently, without the document itself being stored. This economizes both storage and processing resources. The hypertext documents are not required once the tables are loaded with the link information (usually accomplished by the DAE when downloading a site).
- The data is stored as normalized relational tables. Therefore there is no redundancy in the structure, allowing for straightforward and efficient additions and updates.
- Topology data may be extracted from these relational tables using SQL statements (Figure 6). These operations are usually highly optimized by the RDBMS system to efficiently handle large queries. This is of particular concern since the hierarchical nature of these links often leads to exponential increases in the number of children nodes.
- Using table information to traverse the hypertext space is more robust and comprehensive than using the Web documents and their hyper-links directly.

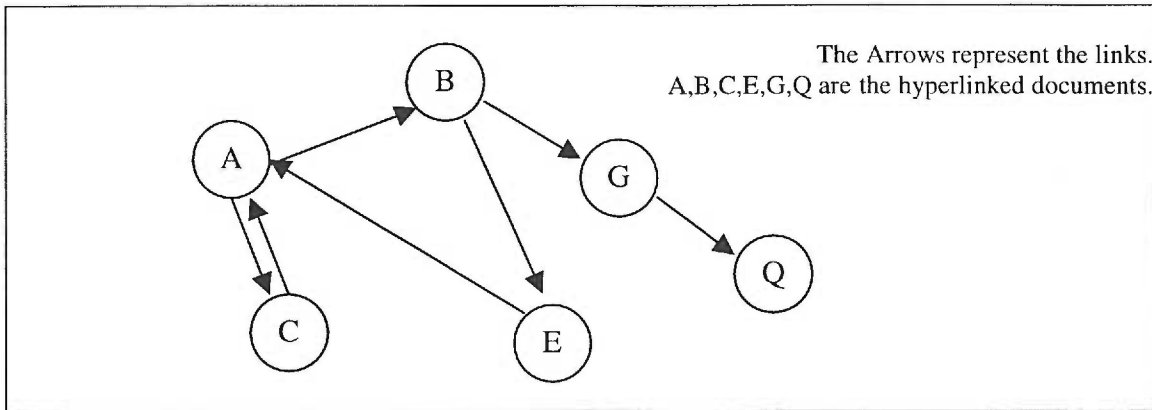


Figure 4. Hypertext documents as a directed graph

R_TABLE

Source	Target
A	B
A	C
B	E
B	G
C	A
E	A
G	Q

Figure 5. The same directed graph as relational table: 'R_TABLE'.

1. One level indirection:

```
select TARGET from R_TABLE where SOURCE in "B".
```

2. Reverse one level indirection:

```
select SOURCE from R_TABLE where TARGET in "A".
```

3. Two levels indirection:

```
select TARGET from R_TABLE where SOURCE
    in (select TARGET from R_TABLE where SOURCE in "B" ).
```

4. Discover all 1st generation siblings:

```
select TARGET from R_TABLE where SOURCE
    in (select SOURCE from R_TABLE where TARGET in "E").
```

Figure 6. Examples of SQL statements to traverse the document topology.

1.5.4 Indexing and Search engine

A search engine is required to provide a Boolean and/or a Natural language search environment for the terms present in the documents. This module has to be very fast and scalable to handle the ever-expanding document domain of the Web. We did not intend to provide a complex engine with advanced features in the prototype. Instead, the modular design of this system will allow for the substitution of an advanced search engine module to replace a more rudimentary module providing the basic functionality.

In this case, the primary requirement of a search engine module would have to be that the engine should be able to index the document collection incrementally. This would enable the system to download chunks of documents from the Web and index them, accumulating the index information. The complete document corpus will not be required to exist in entirety on this system at any point of time. A moving window to the document corpus should be sufficient to allow the indexing engine to accomplish its task.

We have achieved incremental indexing indirectly by using clusters, using a completely different approach, as described in §2.7.

Since it is not feasible to retain all the local copies of the indexed and processed documents, most of these documents will be deleted from the local mirror. The index pointers would be adjusted to point to the original Web version of the document.

Instead of developing our own search engine, we have used available search engines as the indexing and searching module of the DPE.

Search Engine Interface

We have developed a generalized program interface module for search engines. Any search engine that can conform to this interface may be used by the DPE. This is achieved by an interface layer that translates command and data between the DPE and a search engine. Such interface layers have been developed (by other members of Dr. Hersh's research group) for two search engines- **MG** (<ftp://munnari.oz.au/pub/mg>) and **SWISH-E** (<http://sunsite.berkeley.edu/SWISH-E>).

Since the interface is standardized, the specific search engines are completely interchangeable. Either or both may be used and the specifics of the search engines are transparent to the user. When used with the clustering functionality of the DPE, the system can function like a meta-search engine, pooling results for many engines or queries. This and the Boolean functionality of this interface will be discussed in more detail in §2.7

Apart from being a translation and compatibility layer, these interface modules provide additional functionality to the search engines. This is achieved by using the DPE clustering system to create logical partitions of the document repository. This will allow us to overcome a serious disadvantage of some search engines- their inability to index

collections incrementally (i.e. the collection has to be completely re-indexed if new documents are to be included in the index).

The DPE may create a second cluster of the new documents and builds an index only for the second cluster. Any query to the repository is duplicated by the interface layer and both the clusters queried individually. The results from the two queries are pooled together and returned as a composite result. These actions are transparent to the user, the DPE-interface handling the details of the separate indices and the reconstitution of the results. This not only saves computation time and storage by not having to re-build and re-index large document collections, but also the smaller indices are more efficient and their individual searching processes may be distributed over multiple CPUs to execute simultaneously. This may allow for load distribution and parallel processing by multiple computers or CPUs.

The searching capability on the RDBMS fields (using SQL), including the meta-data tables, provides an additional feature whereby the search may be limited to specific fields. This is in addition to the full text searching capability that is provided by the search engine modules.

2. Implementation Details

2.1. Description of the tables

The database tables are the most important component of this entire system. We have attempted to use standard SQL code to build and query the database tables so that the code is fully compatible with any commercial database that supports the standard SQL syntax. For our prototype, we have used Oracle7.3 as the database engine. The database interface that provides the Perl API is the Perl - DBD-DBI package. This package is also database independent. (i.e. any commercial or freeware database maybe used, as long as the DBD driver for the database engine exists).

Comments on field data types: The fields in the database that contain strings are defined as "varchar2" datatypes. This storage type is defined to be the largest possible string length expected for that field. (The Oracle database does not store any unused trailing spaces after the string). It should be noted that all the time (date) fields are converted to and stored in the universal time (GMT) standard.

(1) Table 'urls'

```
create table urls(  
    docid          number not null,  
    url            varchar2(500) not null,  
    title          varchar2(1000),  
    local_flag     char(1),  
    local_stat     varchar2(500),  
    mime           varchar2(80),  
    stime          date,  
    fsize          number,
```

```
links          number,  
  UNIQUE(docid)  
);
```

`docid` — the unique document identifier.

`url` — normalized URL for the document.

`title` — original title from the title tag of the document.

`local_flag` — if this flag is 'Y', it indicates that the file exists in the local repository and the 'local_stat' field contains the file name for the local copy.

`local_stat`— if the `local_flag` value is 'Y' this variable contains the local filename. Otherwise, this field contains the error code for the transport protocol or the reason why the file was not stored in the repository. A list of these codes may be found in Appendix C.

`mime` — the mime-type of the document. From HTTP:Header.

`stime` — the time the document was downloaded or accessed.

`fsize` — the size of the document in bytes. From HTTP:Header.

`Links` — the number of hyperlinks that refer to this document, i.e. every time this document is referred to and a download is attempted, the DAE returns the duplicate (DUP/DUPBF) flag and increments this link count by one.

(2) Table 'dirs'

Each time the DAE is invoked with a new starting link, it creates a new directory and proceeds to download the hyperlinked files into this directory. This is not an absolute

requirement, but it is helpful since it separates the downloaded files from each invocation into different directories.

This directory information and the details from each run are stored in this dirs table.

```
table dirs(  
    dirid          number not null,  
    path           varchar2(500),  
    stime          date,  
    range_s       number,  
    range_e       number,  
    UNIQUE(dirid)  
);
```

`dirid` — unique directory/run identifier.

`path` — absolute path to the repository directory.

`stime` — timestamp of the run.

`range_s` — starting docid of the run: (i.e. docid of the first document).

`range_e` — ending docid of the run.

(3) Table 'links'

```
table links(  
    source         number not null,  
    target         number not null  
);
```

`source` — the docid of the document with the link.

`target` — the docid of the document the link refers to.

Note: To preserve the referential integrity, all the starting documents will not have an entry in this table as a 'target' since that would imply an undefined 'source'. Later, all

excluding the first references will be treated normally.

(4) Table 'clusters'

```
table clusters(  
    clusterid      number not null,  
    clustername    varchar2(100),  
    ctime          date,  
    detail         varchar2(1000),  
    UNIQUE(clusterid)  
);
```

Clusterid – unique cluster identifier.

Clustername – arbitrarily chosen name (preferably unique).

Ctime – time stamp.

Detail – description or comments.

(5) Table 'doc_cluster'

This table identifies the documents and their relationship to the clusters.

```
create table doc_cluster(  
    clusterid      number not null,  
    docid          number not null  
);
```

Clusterid — cluster identifier.

Docid — the unique document identifier.

(6) Table 'meta_data'

```
table meta_data(  

```

```
    docid          number not null,  
    meta_tag      varchar2(1000),  
    meta_value    varchar2(1000)  
);
```

Please refer to §2.5 for the description and usage of the table.

(7) Table 'alt_title'

```
table alt_title(  
    docid          number not null,  
    title          varchar2(1000)  
);
```

Please refer to §2.4 for the description and usage of the table.

2.2. Unique Identifiers

This section describes the assignment of unique serial numbers for documents and other objects (e.g., docid, clusterid etc.)

Each object in the repository has a unique number that identifies it exclusively and completely within its type. The uniqueness of this field is ensured by enforcing the "unique" attribute for this field and the primary database tables. A new serial number is generated by the DPE program by incrementing by one, the largest assigned serial number in the table.

There can be another approach for generating unique serial numbers. Some databases may generate a sequence number internally (e.g., Microsoft Access), or indirectly via some procedural code fragment that is executed when a new row is created in the table

(e.g., Triggers in Oracle). This approach was not used since it makes the DPE code dependent on a specific RDBMS and less portable, as different RDBMSs implement the generation of serial numbers in different ways.

As the DPE generates its own unique serial numbers, it has the responsibility to ensure synchronization and concurrency in multi-process mode by exclusive locking (using available RDBMS locking features). An added benefit in having the DPE generate serial numbers, is that the DPE, if required, may recycle old serial numbers that are no longer used by the system (using a reliable algorithm to reclaim all dangling references to that identifier).

2.3. Document Titles

The 'title' field of a document is very useful for displaying as link anchors in table of contents pages or lists. A descriptive and self-explanatory title is very helpful in identifying documents. Sometimes this title field is 'null' or contains the string 'untitled'. In these cases, an alternative title may be created and used in place of the document title.

The title field in the table 'urls' is extracted from the HTML document header by the HTTP:head command by the DAE. In case this value is 'null', the DAE stores that 'null' value in the 'title' field of the 'urls' table entry for this document. In order to generate alternative titles, the DPE will identify all 'null' and 'untitled' entries in the 'urls' table and attempt to generate synthetic titles only for these. In order to

retain the integrity of the 'urls' table, generated titles are not inserted back into the title field of the 'urls' table (thereby overwriting the 'null' and 'untitled' entries). Instead, the generated titles are stored in a separate table 'alt_titles'. This table has two fields, 'docid' and 'title'. Applications that require the document title to be displayed, first query the 'alt_title' table to check if an entry exists for a particular docid. If one is found, it is used, otherwise the actual 'title' from the 'urls' table is used.

2.4. Generating Alternate Document Titles

The logical alternative to the <TITLE> tag in an HTML document is the <H1> tag contents (or the largest header tag <H*> available). If there are no header tags available, possible alternatives include the first line (30-80 characters) of the text of the file, or the file-name itself.

The program selects the most suitable alternate title using the logic described above and creates an entry for the document in the 'alt_title' table (the user may select or provide an alternative title if necessary).

2.5. Meta-data Extension

The Meta-data content of an HTML document is usually embedded in the document as tag-value pairs. To extract this information, the document is processed and the tag-value records saved as rows in the table 'meta_data'.

Additional comments or meta-information (not embedded in the document), that is provided or generated by the reviewer/indexer (either automatically or manually), may also be saved in this table as tag-value pairs. This additional information may be indexing terms from a controlled vocabulary such as MeSH or SNOMED terms, or reviewer comments and judgments. It should be noted here that the additional meta-information not present in the original document is retained only in the meta_data table; the original document remains unchanged.

If at some point the user wants to generate a new version of the document that contains all the meta-information embedded in the HTML file, a program may be used to inject the meta-data tag-value pairs into the HTML document (similar to the methods described by Munoz et. al.[25]).

2.6. Using the Meta-data Table

The meta-data table may be used to search for terms in the meta_value field. The search may be limited to specific meta_tag fields like "Author", "MeSH", etc. A reviewer may use the system to add one or more meta-data entries to all the documents in a cluster by performing the action on a selected cluster, instead of a single document. Examples of such meta-data entries are: a cluster of documents authored by an individual tagged as: `<META NAME= "Author" CONTENT= "John Doe">`; documents of type "reviews" tagged as: `<META NAME= "Document Type" CONTENT= "Review">`.

The meta-data extension, along with the clustering functionality of the DPE, will allow the reviewer or indexer to operate efficiently on entire groups of documents in selecting and classifying them, instead of one document at a time. While searching for, or selecting documents from the repository, the meta-data fields may be used to narrow selections to specific documents or groups of documents having similar meta-data tag-value pairs. If the values of meta-data contain controlled vocabulary terms, searching for specific terms in this table instead of the document text may greatly improve the precision of searching.

2.7. Using Clusters

Often the reviewer has to perform the same set of actions on numerous individual documents. If these same documents were grouped together into a cluster, the reviewer may perform the actions only once on the entire set. The changes would affect all the documents in the cluster, thereby reducing the reviewer's task of handling individual documents.

Clusters will allow the user to perform Boolean operations on groups of documents. This is especially useful if the sub program (e.g., a search engine) does not support Boolean operations. The results of more than one operation (as clusters) may be 'AND'ed or 'OR'ed together in any combination to produce another cluster. This will allow the user to construct complex operations from simple functional modules. Search

engines without Boolean capabilities may be used to perform Boolean searches indirectly by deconstructing the query into simple components. An interface may be developed to parse a Boolean query into its components and subsequently reconstruct the results from the component clusters, all handled by the interface layer and the DPE, transparent to the user.

When used as an experimental system, the cluster operations will allow a researcher to compare the results of two or more search engines (or any two processes that creates clusters). For example, a Boolean "AND" operation will indicate the overlap between the two results sets. This facility may also be used to create a meta-search engine that can pool the result from two or more search engines and return a composite list of results.

V. Discussion

1. *Scope of this project*

We have developed a system that will allow any reviewer, librarian and Web site maintainer to manage large hypertext collections. It provides them with tools to index, annotate and group the documents for improved accessibility by the final consumer of the information. Without this system, most of these tasks are accomplished manually, if at all. A manual process is not only inefficient and labor-intensive, but also error prone and much less comprehensive compared to our system.

A few automated tools in use today perform some of these functions, but most have been adapted from other uses and are not perfectly suitable or integrated for these particular applications. These systems often do not exploit the full potential of the information-rich hypertext documents, mostly because their predecessors were developed prior to the introduction of hypertext and SGML.

Our document management system has been designed specifically to handle Web and hypertext documents. We have designed it to integrate seamlessly with all the components within the system and provide flexible interfaces to external components and other programs. To achieve a high degree of flexibility without sacrificing the powerful features, the system was assembled from small functional modules. These

modules were pre-existing component software, components developed for this project, or components that were modified to integrate with the rest of the framework.

Apart from designing and developing this system as a tool, we have made the individual components of this framework available, so that an information retrieval system developer may use it as a toolkit for developing other systems.

The features offered by this system to the reviewer or indexer:

- Visualize a document structure overview as a table of contents/index.
- Operate on clusters instead of single documents.
- Markup documents with meta-data.
- Ability to work offline from the Web.
- Maintain a local archive or mirror of the remote site, which may also serve as a snapshot of a dynamic or volatile website.
- Have exhaustive error tracking and audit trails.

The features offered by this system to a developer using the system as a toolkit:

- Develop systems using pre-fabricated components.
- Exploit the document information infrastructure in the RDBMS.
- Access to the pre-processed document repository.

2. Performance Assessment and Evaluation

In this part of the thesis, we will attempt to design and describe evaluation strategies for assessing the performance of this document processing system. The performance evaluation of a system as large as this can be a project all on its own merit. We did not intend to perform any of the assessment or evaluation strategies discussed below due to time and resource constraints.

The primary goals for such an evaluation would be to determine if this toolkit can provide an improved signal to noise ratio over more conventional tools and techniques. We may also attempt to discover if the topological clustering does actually help in predicting other document attributes, like quality and content. The following are possible scenarios for evaluation.

To evaluate the ability of this system to improve the recall and precision by limiting the search domain to predetermined clusters of documents, arbitrary selections from one or two test collections based on actual Web content may be used. It must be emphasized that such results may not be extendible to large collections or the entire WWW.

The experiment may use test collections with a known number of relevant judgments of documents and their corresponding queries. The queries may be run against document clusters created based on different attributes, and their recall and precision computed. The results may indicate if any attribute used to generate a cluster has an effect on

improving the recall or precision. A positive result may indicate the existence of a clustering technique that may be used to fine-tune a search based on that particular attribute.

As a specific case, a test collection that has been clustered manually on the content quality, and automatically based on topological information, may be studied. It would be instructive to examine the correlation of documents between these clusters. A strong correlation would allow us to predict the quality of documents in any cluster, if we have some topological information and the quality judgments on only a few documents.

This may even be the basis for the development of a statistically valid comparative scale for document quality that would depend on topological and domain information as input. There are many other possible assessment studies that are beyond the scope of this thesis. Some of these possibilities may include: (1) Usability studies to examine if this system improves the efficiency of the user in locating information in a diffuse document space. (2) If the system can improve the quality of documents retrieved by applying some machine learning techniques to its clustering logic.

3. Future work

Plans for future development include:

- Developing an interface to the Saphire[14] system, which would allow users to automatically generate clinical terms from a controlled vocabulary like MeSH or SNOMED, associated with the document being processed. These terms may be saved as meta-data associated with that document.
- Adding facilities to handle additional mime-types of non-text documents like images, documents in specialized formats like PDF(Portable Document Format) and documents in multiple languages.
- Developing a comprehensive and consistent programming API to the entire system in a high-level structured language like C++ or Java. This would enable developers to easily integrate components from this system into their own applications.
- Developing an integrated front-end, written completely in Java (for portability), that would provide the user with a powerful and flexible user interface. This front-end program should be able to communicate with the rest of the system -- specifically the RDBMS repository through a JDBC channel.

VI. Conclusion

Over the last decade, the World Wide Web has been growing exponentially, along with its contents, diversity and complexity. The tools to locate and manage specific information in this growing environment have just not been able to keep up. Most of the systems that are currently used were actually designed for other non-hypertext uses and subsequently adapted for the Web. Consequently, many of the features and facilities that are unique to the Web are underutilized or ignored due to the lack of appropriate and specific tools and programs.

In this project, we have tried to address these issues by developing an application framework designed specifically to handle hypertext documents of the World Wide Web. Our development strategy has been to use small component modules to provide specific functionality to the system. This makes the system flexible and allows the user to modify the system to suit their specific needs. Apart from using this system as an application framework, a developer or researcher may use this system as a toolkit to build a more complex document management system, or a research system.

Our primary goal in developing this system was to provide a powerful and flexible tool to maintain and update Cliniweb. This system may also be used as a general application framework for collecting, indexing and reviewing hypertext and Web documents from multiple sources.

Our system strives to improve both the quality and the quantity of the content of the material generated by reviewers and indexers compared to their manual efforts. It provides for a more comprehensive and complete solution compared to isolated tools and programs. We attain these goals by exploiting the unique features of the hypertext and the World Wide Web that have often been ignored by other systems.

VII. References

1. Abiteboul, S. *et.al.* Querying documents in object databases. , **Intl. J. Digital Libraries** (1997) pp. 5-19.
2. Allan, James. Building Hypertext using Information Retrieval. **Info. Processing & Mgmt**, 1997. 33 (2) pp145-159
3. Botafogo R.A. Cluster Analysis in Hypertext Systems. , **Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval** (1993), pp.116-125.
4. Bowman C. M., P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz, "The Harvest Information Discovery and Access System," **Computer Networks and ISDN Systems**. 28 (1995), pp. 119-125.
5. Boyle James, A blueprint for managing documents. **Byte**, May 1997.
6. Hersh WR, Brown KE, Donohoe LC, Campbell EM, Horacek AE, CliniWeb: Managing clinical information on the World Wide Web, **Journal of the American Medical Informatics Association**, 1996, 3: 273-280. Cliniweb: (www.ohsu.edu/clinweb).
7. Cutting D. R., D. R. Karger and J. O. Pedersen, "Constant Interaction-Time Scatter/Gather Browsing of Very Large Document Collections," **Proceedings of the**

Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (1993), pp. 126-134.

8. Darrell Woelk and Christine Tomlinson, The Infosleuth project: intelligent search management via Semantic agents. **Proceedings of the International WWW Conference, October 1994.** (www.mcc.com/projects/infosleuth)
9. Documentum Inc. (<http://www.documentum.com>)
10. Domel, P., "WebMap - A Graphical Hypertext Navigation Tool," **Proceedings of the Second International World Wide Web Conference, Chicago, Illinois (October 1994).**
11. Epstein J. A., J. A. Kans, and G. D. Schuler, "WWW Entrez: A Hypertext Retrieval Tool for Molecular Biology," **Proceedings of the Second International World Wide Web Conference, Chicago, Illinois (October 1994).**
12. William B. Frakes and Ricardo Baeza-Yates, Eds. **Information Retrieval: Data Structures & Algorithms**, 1992, Prentice-Hall, Inc.
13. Hackathorn Richard, Farming the Web. **Byte**, October 1997.
14. Hersh William R., **"Information Retrieval- A Health Care Perspective"** Springer 1996.

15. Klark P., and U. Manber, "Developing a Personal Internet Assistant," **Proceedings of ED-Media 95, World Conf. on Multimedia and Hypermedia**, Graz, Austria (June 1995), pp. 372-377. (www.cs.arizona.edu/people/udi/nabbit/nabbit.html)
16. Lawrence S, Lee Giles C. Searching the World Wide Web. **Science** 280 April 3, 1998, pp 98-100.
17. Lynch Clifford, "Searching the Internet" **Scientific Am.** March 1997.
18. Malet G, *et.al.* Medical core Metadata elements. Personal communications.
19. Manber U. and S. Wu, "GLIMPSE: A Tool to Search Through Entire File Systems", **Usenix Winter 1994 Technical Conference**, San Francisco (January 1994), pp. 23-32. See also Glimpse Home Pages at:
(glimpse.cs.arizona.edu),(donkey.cs.Arizona.edu/webglimpse)
20. Manber Udi, Smith M, Gopal B, Webglimpse- Combining Browsing and Searching. , **Proc. of Usenix Technical Conference** Jan.1997.
21. Marcel Holsheimer, *et.al.* Data mining, the search for knowledge in databases. **C.W.I. Amsterdam, Technical report.**
22. Mendelzon, A.O., *et.al.* Querying the World Wide Web. **Intl. J. Digital Libraries** (1997) pp54-56.
23. Mihaila G.A. WebSQL - an SQL like query language for the WWW. **Masters thesis, Univ. of Toronto**, 1996.

24. Morton D., and S. Silcot, "Systems for providing searchable access to collections of HTML documents," **First Australian WWW conference**, July 1995.
(<http://www.scu.edu.au/ausweb95/papers/indexing/morton>)
25. Munoz F R, Hersh W., "MCM Generator: a Java-based Tool for Generating Medical Metadata. **Submitted**. 1998.
26. Salton G, McGill M. J. **Introduction to Modern Information Retrieval**, McGraw-Hill, New York 1983.
27. "The Internet: Bringing Order from Chaos." Special Issue on the Internet. **Scientific American**, March 1997.
28. Small, H., E. Sweeney. 1985. "Clustering the Science Citation Index Using Co-citations. I. A Comparison of Methods." **Scientometrics**, 7, 391-409.
29. Trellix Inc. (<http://www.trellix.com>).
30. Willett, P. 1988. "Recent Trends in Hierarchic Document Clustering: A Critical Review." **Information Processing & Management**, 24(5), 577-97.

VIII. Appendices

Appendix A. Definitions:

Action: any operation or modification on any document sub-structure or super-structure

Archive: (repository) a collection of documents stored together (usually physically).

Attribute: an intrinsic or extrinsic property of the document based on its content or context.

Cluster: (aggregate) a logical set of documents, grouped together manually or automatically, based on some document property.

Document: a single file with human and machine-readable content.

Domain: a physically or administratively distinct zone or collection of documents encompassing one or more hierarchies or archives.

Hierarchy: a linked and logically arranged collection of documents with one root (or top most node) and many children (leaf) nodes.

Structure: layout or arrangement of the elements of a document or cluster.

URL: Uniform Resource Locator.

View: one of many representations of the contents of the document or document space(or cluster)

Appendix B. Table Structures

```
create table urls(  
    docid          number not null,  
    url            varchar2(500) not null,  
    title          varchar2(1000),  
    local_flag     char(1),  
    local_stat     varchar2(500),  
    mime           varchar2(80),  
    stime          date,  
    fsize          number,  
    links          number,  
    UNIQUE(docid)  
);  
  
create table links(  
    source         number not null,  
    target         number not null  
);  
  
create table clusters(  
    clusterid      number not null,  
    clustername    varchar2(100),  
    ctime          date,  
    detail         varchar2(1000),  
    UNIQUE(clusterid)  
);  
  
create table doc_cluster(  
    clusterid      number not null,  
    docid          number not null  
);  
  
create table dirs(  
    dirid          number not null,  
    path           varchar2(100),  
    stime          date,  
    range_s        number,  
    range_e        number,  
    UNIQUE(dirid)  
);  
  
create table meta_data(  
    docid          number not null,  
    meta_tag       varchar2(1000),  
    meta_value     varchar2(1000)  
);  
  
create table alt_title(  
    docid          number not null,  
    title          varchar2(1000)  
);
```

Appendix C. DAE Codes For Documents

CODE	Description
DEEP	Too many levels deep.
DUP	*Document seen before in current run.
DUPBF	*Document seen in a previous run.
ERROR	HTTP protocol error – error code follows:
IMAGE	Document is a graphic or binary file.
MAIL	Mail link.
OUTSIDE	Link is outside the current prefix specification.
OVERLIMIT	Too many files have been downloaded, limit exceeded.

*These codes are not stored in the tables- they are for diagnostic facilities only.

Appendix D. Command Line Parameters for the DAE

<code>-depth=N</code>	Maximum depth to traverse (default: \$MAX_DEPTH)
<code>-limit=N</code>	A limit on the number documents to get (default: \$MAX_DOCS)
<code>-version</code>	Print version number and quit
<code>-verbose</code>	More output
<code>-quiet</code>	No output
<code>-sleep=SECS</code>	Sleep between gets, ie. go slowly
<code>-prefix=PREFIX</code>	Limit URLs to follow to those which begin with PREFIX
<code>-noimg</code>	Do not fetch Graphics/Binaries.

These command line parameters are extensions of the 'lwp-rget' program parameters.

Appendix E. Screen Snapshots of the Program

