UNITE - Unified Network Informatics Technology for Education.

by

Cedric Christian Deniau

B.S, University of Kansas 1993

Submitted to the Department of Electrical Engineering and Computer Science and the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Professor in Charge

Committee Members

Date thesis accepted

Abstract

Over the past few years, the world has seen a growing interest in the Internet. E-mail initiated this interest and was the biggest traffic generator for several years. As the Internet grew in popularity, other tools emerged: ftp, gopher, archie, and the World-Wide Web. Connectivity to the Internet blossomed from a few computer specialists at research institutions to include businesses, schools, and home users. At the same time, the ability to create, store, and view multimedia information became widespread. Today, we see a proliferation of sites storing and distributing multimedia information on an ever-increasing range of topics to an exploding number of users.

This thesis describes the UNITE project which provides browsing and search of taxonomically indexed resources in a wide range of media types (text, images, hypercard stacks, etc.). The server provides remote access to any taxonomically structured domain and supports mirroring, which helps distribute the client load, and enables the client to try alternative servers if its first choice is unavailable.

Users are active participants in the project. Through a review mechanism, they can contribute new resources to the database and provide feedback through our comments page. These feedbacks allowed us to research and develop new ways to browse available resources. These efforts in designing the UNITE system indicated that users initially found hierarchical browsing structures to be an easy way to locate information. As users evaluated the browsing mechanism they became familiar with available resources. With this familiarity of the information domain also came a desire to more precisely focus their queries.

Contents

List of Figures

Chapter 1

Introduction

The Information Highway is becoming a reality. The increase in access to the Internet by the public at large, combined with the development of easy to use graphical browsing interfaces such as Mosaic and Netscape, have lead to an explosion in the amount information being added. In particular, the World Wide Web (WWW) is being used to present an exponentially growing amount and range of information through which people can browse. Unfortunately, too much information can be the same thing as not enough information. If the information you seek is buried under an avalanche, is it really there? The WWW is growing at such a rate that it is hard to locate information of interest. To give a feel for the magnitude of the problem, the Lycos system indexes over 860,000 Web documents from 34,000 sites and is able to add 5,000 documents a day [10]. The WWW is growing quickly because it provides an easy to use interface (pointing and clicking at items on the screen) for users, uses simple standards (HTML, MIME) which allow multimedia documents to be exchanged, and provides a simple unified interface to a range of useful tools (ftp, gopher, news. etc.).

The UNITE project chose the collection, management, and exchange of educational resources as its driving application. Questions facing educators trying to use a database of educational resources are: Where is the information? In what form or forms is it presented? Is there any guarantee of its quality? How can I get the information I want? The goal of UNITE is to create a database of educational resources, particularly those in Mathematics and Natural Science, and to create an electronic framework for its distribution to K-12 teachers and students. The target community is a particularly good test of using the Internet for data distribution to the general public for three major reasons: the data spans a wide range of types, the users are widely distributed geographically and are as widely distributed in their knowledge of computer technology.

UNITE provides a central repository for educational resource materials, allowing the information to be easily located. By creating a customized graphical user interface, we have created a system which is accessible to casual computer users. Finally, we involve the users themselves in the evolution of the database by encouraging them to contribute resources that they create. To provide quality control, we have a series of editors which approve and improve the contributed materials.

This thesis first gives an overview of the WWW and how servers and clients work. Then it presents our approach to these problems, focusing on our search capability, the simple interface, and how UNITE supports the sharing of resources.

Chapter 2

Background

2.1 Internet

The Internet is the system that has been interconnected to form one giant network connecting universities, cities, states, countries, and continents together. The literal meaning of the word Internet is "network of networks [5]."

2.1.1 Brief History

RAND Corporation, America's foremost Cold War think-tank, first started the idea of the Internet [9]. Their goal was to find a way to successfully communicate, coast to coast, after any major disaster, especially after a nuclear war. The first proposal made public was in 1964 [9]. The proposal specified that there would not be a central authority. This was a major concern due to the fact that if there was a central authority, then that site would have been the primary target in any attack from an enemy, making communication impossible. The principles involved in the conception of the Internet were extremely simple. It would be designed from the beginning to transcend its own unreliability. All the nodes in the network would be equal in status to all other nodes, each node with its own authority to originate, pass, and receive messages. The messages themselves would be divided into packets, each packet separately addressed. Each packet would begin at some specified source node, and end at some other specified destination node. Each packet would wind its way through the network on an individual basis [9]. As the years went on more groups became interested in the idea. MIT and UCLA joined with RAND to do more research. The National Physical Laboratory in Great Britain set up the first test network in 1968. Then the Advanced Research Project Agency (ARPA) joined in and decided to fund a larger project in the United States. The network that was built in the US consisted of four nodes and was given the name ARPANET. In 1971, the network consisted of 15 nodes. In 1972, the network increased to 37 nodes. Thanks to ARPANET, users could share one another's computer facilities via long-distance and they could share information. By the second year of operation, however, an odd fact became clear. ARPANET's users had warped the computer-sharing network into a dedicated, high speed, federally subsidized electronic post office. One of the first really big mailing list was "SF-LOVERS" for science fiction fans [9]. ARPA used NCP as its original standard for communication, which stood for "Network Control Protocol". As time went by and technology moved on, it was soon noticed that NCP was not very sophisticated and therefore a new protocol suite came about. This protocol suite was known as TCP/IP. TCP, "Transmission Control Protocol", converts messages into streams of packets at the source, then reassembles them back into messages at the destination. IP, "Internet Protocol", handles the addressing, seeing to it that packets are routed across multiple nodes and even across multiple networks with multiple standards like NCP, Ethernet, FDDI, and X.25 [9]. In 1984, the National Science Foundation (NSF), got into the act and set a fast pace for technical advancement through their Office of Advanced Scientific Computing. Figure 2-1 shows a conceptual view of the Internet.

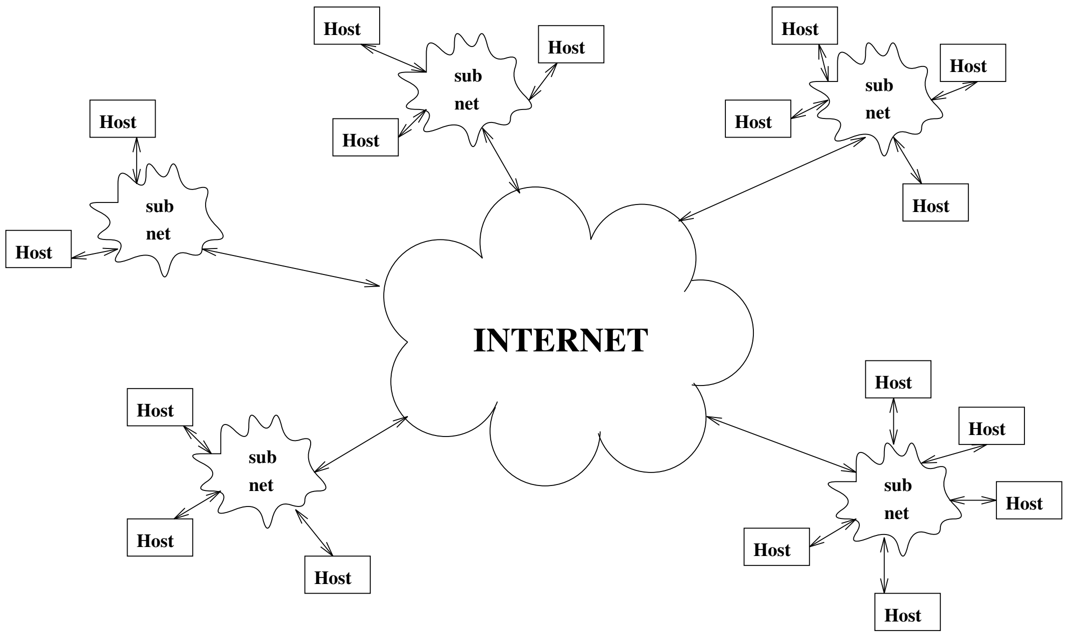

Figure 2-1: Internet

2.1.2 Popularity

On any given day the Internet connects roughly 15 million users in over 50 countries [5]. The use of the Internet is spreading faster than that of cellular phones and fax machines. In 1992, the Internet was growing at a rate of twenty percent a month. The number of "host" machines with direct connection to TCP/IP has been doubling every year since 1988 [5]. Currently, the Internet is growing at a rate of 15% every month.

2.1.3 Uses

The most popular uses of the Internet are: mail, discussion groups, long-distance computing, and file transfer, in that order [9]. Internet mail is called e-mail, electronic mail. Compared to snailmail, i.e. the US Postal service, e-mail is faster by several orders of magnitude and it is also global. E-mail is very similar to the fax machine. The user is sending electronic text to a receiver. The main advantage over a fax machine is that e-mail is free.

The discussion groups are generally known as USENET. Any user with access to USENET can post and read messages. Simply put, USENET is a crowd of news-hungry people. At the moment, there are some 5,000 separate newsgroups on USENET, and their discussions generate about 7 million words of typed commentary every single day [9]. ARPANET's original goal of long-distance computing is still widely used. Programmers have accounts on machines that are more powerful than the machines they have in their facilities. Therefore, they can write, compile, and run their programs on better machines from the comfort of their own office. Also, some libraries will allow users to search their electronic card catalog. This can greatly reduce the amount of time required to search for and find a document.

File transfer allows Internet users to access remote machines and retrieve programs or text. In 1992, there were over a million such public files available to anyone who requested them [9]. Internet file-transfer has become a form of publishing in which the reader simply electronically copies the work on demand, in any quantity he or she wants, for free. Also, writers use file transfer as a means of proofreading their works. The author would say that his book is now available on a specific site. Users will then download it, read it, and give any feedback back to the author.

2.2 Client/Server

The concept of a client/server application is a process (i.e. the server) on a remote machine waiting for a request from another process (i.e. the client). The protocol used is usually TCP/IP which only provides peer-to-peer communication. This means that TCP/IP provides a programmer with the ability to establish a communication channel between two application programs and to pass data back and forth. On the other hand, TCP/IP does not specify when or why peer applications interact, nor does it specify how programmers should organize such application programs in a distributed environment.

2.2.1 The Client/Server Paradigm

Since TCP/IP does not respond to incoming communication requests on its own, a model had to be designed. This paradigm simply states that "a program must be waiting to accept communication before any request arrives [7]." The client/server paradigm, therefore, classifies the communicating applications into two broad categories, depending on whether the application waits for communication or initiates it.

2.2.2 The Client

An application that initiates peer-to-peer communication is called the client. They are usually invoked by the end user when a network service is used. Most client software consists of conventional application programs. Each time a client application executes, it contacts a server, sends a request, and awaits a response. When the response arrives, the client continues processing [7].

2.2.3 The Server

A server is any program that waits for incoming communication requests from a client. The server receives a client's request, performs the necessary computation, and returns the result to the client [7].

2.2.4 Connectionless vs. Connection-Oriented

From the application programmer's point of view, the distinction between connectionless and connection-oriented interactions is critical because it determines the level of reliability that the underlying system provides. TCP provides all the reliability needed to communicate across an internet. It verifies that data arrives, and automatically retransmits segments that do not. It also computes a checksum over the data to guarantee that it is not corrupted during transmission. TCP uses sequence numbers to ensure that the data arrives in order, and automatically eliminates duplicate packets. It provides flow control to ensure that the sender does not transmit data faster than the receiver can consume it. Finally, TCP informs both the client and server if the underlying network becomes inoperable for any reason [7].

By contrast, clients and servers that use UDP do not have any guarantees about reliable delivery. When a client sends a request, the request may be lost, duplicated, delayed, or delivered out of order. Similarly, a response the server sends back to a client may be lost, duplicated, delayed, or delivered out of order. The client and/or server application programs must take appropriate actions to detect and correct such errors [7].

2.2.5 Stateless vs. Stateful

Information that a server maintains about the status of ongoing interactions with clients is called state information. Servers that do not keep any state information are called stateless servers; while others servers are called stateful servers [7].

The desire for efficiency motivates designers to keep state information in servers. Keeping a small amount of information in a server can reduce the size of messages that the client and server exchange, and can allow the server to respond to requests quickly. Essentially, state information allows a server to remember what the client requested previously and to compute an incremental response as each new request arrives. By contrast, the motivation for statelessness lies in protocol reliability: state information in a server can become incorrect if messages are lost, duplicated, or delivered out of order, or if the client computer crashes and reboots. If the server uses incorrect state information when computing a response, it may respond incorrectly [7].

2.3 World-Wide-Web

The World-Wide-Web was started at CERN by Tim Berners-Lee in March 1989 as the HyperText Project, and is officially described as a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents [5].

2.3.1 Overview of the World-Wide-Web

Initially, its main goal was to provide a common (simple) protocol for requesting human readable information stored on remote systems using hypertext as the interface and networks as the access method [6]. Hypertext is similar to regular text since it can be stored, read, searched, or edited, but with an important exception; hypertext contains connections within the text to other documents. The generality and power of the World-Wide-Web becomes apparent when one considers that these links can lead literally anywhere in cyberspace; to a neighboring file, another file system, or another computer in another country.

The World-Wide-Web Project adopted a distributed client/server architecture. The client supports the user as he selects links inside documents by fetching the new document desired, while the server receives the requests generated by selecting a link and responds by providing the client with the required document. At the beginning of the World-Wide-Web Project, the client was a line mode browser which performed the display of hypertext document in the client hardware and software environment. For example, a Macintosh browser uses the Macintosh interface look-and-feel. In September of 1993, NCSA release the Mosaic browser for the most common platforms, X-windows, PC/Windows, and Macintosh. Since Mosaic allowed documents with images to be viewed and also handled new media formats such as video and sound using helper applications, it became the World-Wide-Web browser of choice for those working on computers with graphics capability. However, what may have been Mosaic's most important property was that it effectively subsumed a number of traditional services (i.e. ftp, telnet, gopher, ...), and given its intuitive hypermedia interface, it became the most popular interface to the World-Wide-Web.

Today the World-Wide-Web is growing at an astonishing rate. From January to December 1993, the amount of network traffic across the National Science Foundation's (NSF's) North American network attributed to World-Wide-Web use multiplied by 187 times. In December 1993 the World-Wide-Web was ranked 11th of all network services in terms of sheer traffic - just twelve months earlier, its rank was 127. In June 1993, Matthew Gray's WWWWanderer, which follows links and estimates the number of World-Wide-Web sites and documents, found roughly 100 sites and over two hundred thousand documents. In March 1994 this robot found 1,200 unique sites. A similar program by Brian Pinkerton at the University of Washington, called the WebCrawler, found over 3,800 unique World-Wide-Web sites in mid-May 1994, and found 12,000 World-Wide-Web servers in mid-March of 1995 [5].

The major challenge posed by the World-Wide-Web is clearly one of organizing and making a wealth of information accessible, not of making it merely available. The rest of this section gives an overview of important properties of World-Wide-Web servers and clients, which help determine what services the World-Wide-Web can provide, and the processing and network support required to support them.

2.3.2 World-Wide-Web Server

World-Wide-Web servers are programs running on host computers. They support simultaneous access by multiple users to resources resident on the server's host. In keeping with the client/server paradigm, they respond to a specific set of commands (their protocol) in predictable ways.

Protocol

The World-Wide-Web has used the Hypertext Transfer Protocol (HTTP) since 1990. HTTP is an application-level protocol with the compactness and speed necessary for distributed, collaborative, hypermedia information systems. It is a generic, stateless, object-oriented protocol which can be used for several kinds of tasks [2]. HTTP builds on the discipline of reference provided by the Universal Resource Identifier (URI), such as a location (URL) or name (URN), for identifying the resource upon which a method should be applied. Messages are passed in a format similar to that used by Internet mail and use the Multipurpose Internet Mail Extensions (MIME) [2].

HTTP is based on a request/response between client and server. The client establishes a connection with a server and submits a request consisting of a request method, URI, and protocol version, followed by a MIME-like section containing request modifiers, client information, and optional body. For most implementations, the connection is established by the client prior to each request and closed by the server after each response. The closing of the connection by either or both parties always terminates the current request, regardless of its status [2].

A client request includes the method which should be applied to the resource requested, the resource identifier, and the HTTP version. There are seven different methods allowed in HTTP: GET, HEAD, PUT, POST, DELETE, LINK, UNLINK [2]. The GET method retrieves whatever information is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which is returned as the entity in the response and not the source text of the process [2]. The HEAD method is identical to GET except that the server must not return any entity body in the response. The meta-information contained in the HTTP headers in response to a HEAD request should be identical to the information sent in response to a GET request [2].

The POST method is used to request that the destination server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the request line. POST creates a uniform method to achieve the following functions: annotation of existing resources; posting a message to a bulletin board, newsgroup, mailing list, or similar group articles; providing a block of data (usually a form) to a data handling process; extending a database through an append operation [2].

The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an existing resource, the enclosed entity is considered a modified version of the original. If the Request-URI does not point to an existing resource , and the requesting user agent is permitted to define the URI a new resource, then the server creates the resource with that URI [2].

The DELETE method requests that the server delete the resource identified by the Request-URI [2], while the LINK method establishes one or more link relationships between the resource identified by the Request-URI and other existing resources. The LINK method does not allow any entity body to be sent in the request and does not result in the creation of new resources [2].

The UNLINK method removes one or more link relationships from the existing resource identified by the Request-URI. The removal of a link to a resource does not imply that the resource ceases to exist or becomes inaccessible for future references [2].

Server Features

The features provided by different servers vary, but currently there are two popular servers, those produced by NCSA and CERN. The features discussed in this section are common to both, and are representative of services which any reasonable HTTP server should provide. One feature, directory indexing, allows users to view contents of directories on the server using their World-Wide-Web clients. Depending on how the server was configured, the listing might specify distinct icons for different file formats. A header and trailer file could be included in the listing to give the user more information on the directory contents.

CGI scripts, a particular powerful feature of HTTP servers, are used to run programs on the server side. These scripts are primarily used as gateways between the World-Wide-Web programs and other software like finger, archie, or database software. Image maps, which associate HTTP links with different areas of an image, are another popular use of CGI scripts. The images are virtually segmented so when a user clicks on different parts of the image, he is taken to different URLs. Server features allow the server administrator to include files within all HTML documents provided by the server, creating the ability to include a signature block with every document. When the signature contents change only one file needs to be changed instead of having to change every file containing the signature. The server can also restrict access to certain documents or directories. There are two ways this can be done: (1) in a configuration file, the server administrator can specify certain hosts that are allowed or denied access to documents; or (2) the administrator can specify that the server should ask for a username/password when access to a particular file or directory is requested.

The features mentioned above are a subset of the features implemented by full fledged World-Wide-Web servers. Although these features assist the user in navigating the Internet, the most important feature of a World-Wide-Web server is its understanding and response to a standard protocol, providing access to documents from a variety of browsers.

2.3.3 World-Wide-Web Browser

World-Wide-Web clients, often called browsers, mediate between the user and World-Wide-Web servers by presenting the documents retrieved in a manner best suited to the user's platform, and makes requests to the appropriate server when the user selects a hypertext link. Currently, the most popular browsers are

Netscape and Mosaic, both of which are available for multiple platforms (PC, Mac, UNIX based stations).

HTML

The HyperText Markup Language (HTML) is a simple markup language used to create hypertext documents that are portable across platforms. HTML documents are SGML documents with generic semantics appropriate for representing information from a wide range of applications. HTML can represent hypertext: news, mail, documentation, and hypermedia; menus of option; database query results; simple structured documents with in-lined graphics; and hypertext views of existing bodies of information [3]. HTML has evolved over time, leading clients to render HTML documents differently. Currently there are three versions of HTML, the most common being HTML 2.0. HTML 2.0 introduced forms which support more complex interaction between users and servers by enabling them to supply information beyond simple item selection. For example, forms are commonly used by the user to specify character strings for searching, to provide user-specific data when interacting with a business' World-Wide-Web page, and to provide written text of many kinds in other situations. The Netscape browser has extended HTML by adding extra tags and tag modifiers (i.e. CENTER, BLINK, ...) which provide an enriched set of document formatting controls to the HTML author. Implementations of HTML 3.0 recently became available, which adds the features of tables, mathematical equations, and text wrapping around pictures.

In HTML documents, tags define the start and end of headings, paragraphs, lists, character highlights and links. Most HTML elements are identified in a document as a start tag, which gives the element name and attributes, followed by the content, followed by the end tag. Start tags are delimited by < and >, and end tags are delimited by </ and >. Every HTML document starts with a HTML document identifier which contains two sections, a head and a body. The head contains HTML elements which describe the documents title, usage and relationship with other documents. The body contains other HTML elements with the entire text and graphics of the document. Figure 2-2 gives an example of an HTML document.

```
<HTML>
<HEAD>
<TITLE>Example HTML Document</TITLE>
</HEAD>
<BODY>
<H1>Example HTML Document</H1>
This is a <A HREF=”http://www.tisl.ukans.edu/”>link</A>
to the TISL home page.
<P>
This is an <B>inlined</B> image of a jayhawk:<BR>
<IMG SRC=”jayhawk.gif”>
</BODY>
</HTML>
```

Figure 2-2: HTML example

This example shows the format for a header (<H1> ... </H1>), building a hypertext link (<A HREF=”host.some.where”> link</A>), making a word bold (<B> ... </B>), and adding an inlined image (<IMG SRC=”image.gif”>).

Browser Features

The most popular Web browsers, Netscape and Mosaic, provide similar feature sets. They have a consistent mouse-driven graphical interface and support the idea of using point-and-click actions to navigate through documents. They have the ability to display hypertext and hypermedia documents in a variety of fonts and styles (i.e. bold, italics, ...), layout elements such as paragraphs, lists, numbered and bulleted lists, and quoted paragraphs [5]. All of these are defined in the HTML text of the World-Wide-Web document being rendered.

The browsers have the ability to use external applications to support a wide range of operations. For example, they can be used to view MPEG or QuickTime movies, listen to audio files, or display graphical images. With forms support, they can interact with users via a variety of basic forms elements, such as fields, check boxes and radio buttons. They provide hypermedia links to and support for the following network services: FTP, telnet, gopher, NNTP, and WAIS. In addition, they can: (1) allow remote applications to control the local display; (2) keep a history of hyperlinks traversed; and (3) store and retrieve a list of documents viewed for future use. World-Wide-Web clients often add new abilities along divergent design paths. However, through HTML, they continue to provide a unified and uniform interface to the existing information which is the basis of the WWW's popularity.

2.4 Databases

2.4.1 Database System

A database system is essentially nothing more than a computerized record-keeping system; that is, it is a computerized system whose overall purpose is to maintain information and to make that information available on demand. The information concerned can be anything that is deemed to be significant to the individual or organization the system is intended to serve - anything, in other words, that is needed to assist in the general process of running the business of that individual or organization. Figure 2-3 show a greatly simplified view of a database system. It intends to show that a database system involves four major components, namely data, hardware, software, and users [8].

2.4.2 Database

The database itself can be regarded as a kind of electronic filing cabinet; in other words, it is a repository for a collection of computerized data files [8]. A database

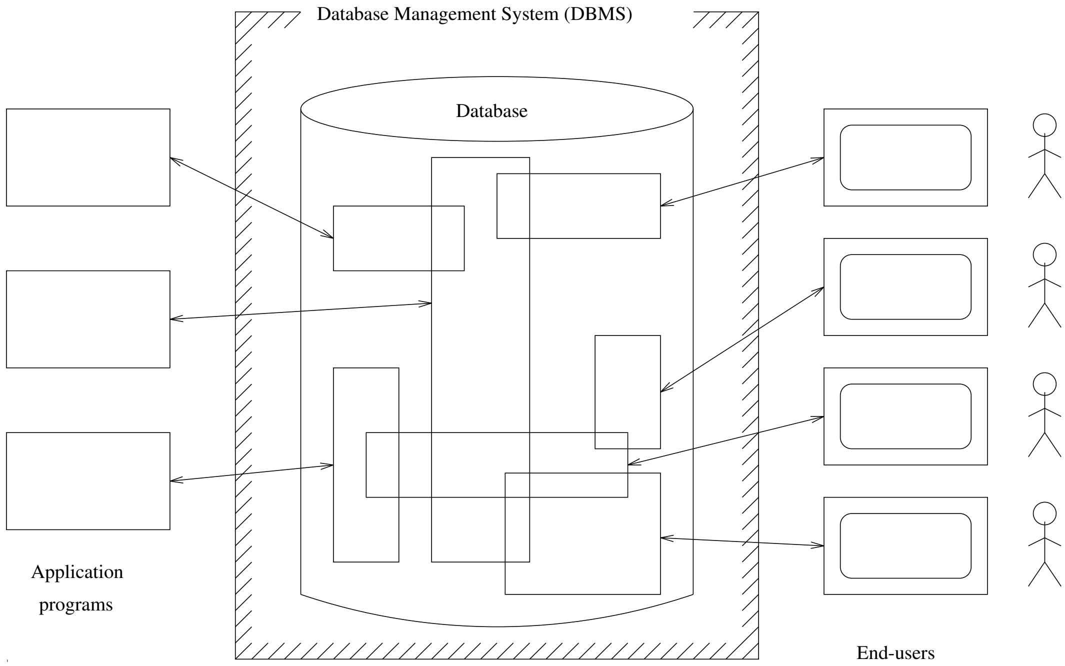

Figure 2-3: Database System Components

consists of some collection of persistent data that is used by the application systems of some enterprise [8]. Here are a few advantages of a database system over traditional, paper-based methods of record-keeping: [8]

1. *Compactness* : No need for possibly voluminous paper files.

2. *Speed* : The machine can retrieve and change data far faster than a human can. In particular, ad hoc, spur-of-the-moment queries can be answered quickly without any need for time-consuming manual or visual searches.

3. *Less drudgery* : Much of the sheer tedium of maintaining files by hand is eliminated. Mechanical tasks are always better done by machines.

4. *Currency* : Accurate, up-to-date information is available on demand at any time.

Currently, the World-Wide-Web is being used to present an exponentially growing amount and range of information through which people can browse. This growth of information makes it impossible to locate documents of interest. A way to solve this problem is to introduce the advantages of databases to Web servers allowing users to search a document structure on a server and find the information.

Chapter 3

Framework

3.1 Overview

The framework of the UNITE project consists of an enhanced WWW server, server tools, and a Macintosh client. These applications provide a mechanism for users to access multimedia resources in a database. The contents of the database are contributed by the user community. In our driving application, a two-stage review process is used to review these contributions (Figure 3-1). Some of the tools provided with the server allow the users to browse the database via a hierarchically structured taxonomy of resources or a graphical search window can be used to intuitively specify Boolean queries.

3.2 The Database

The database is primarily used to organize the resources. Each database has a configuration file associated with it which describes the structure, format, and treatment of the database records. Databases can store several classes of information and must be capable of managing significantly different kinds of data (i.e. software, text, video, audio, etc...). A database configuration language provides

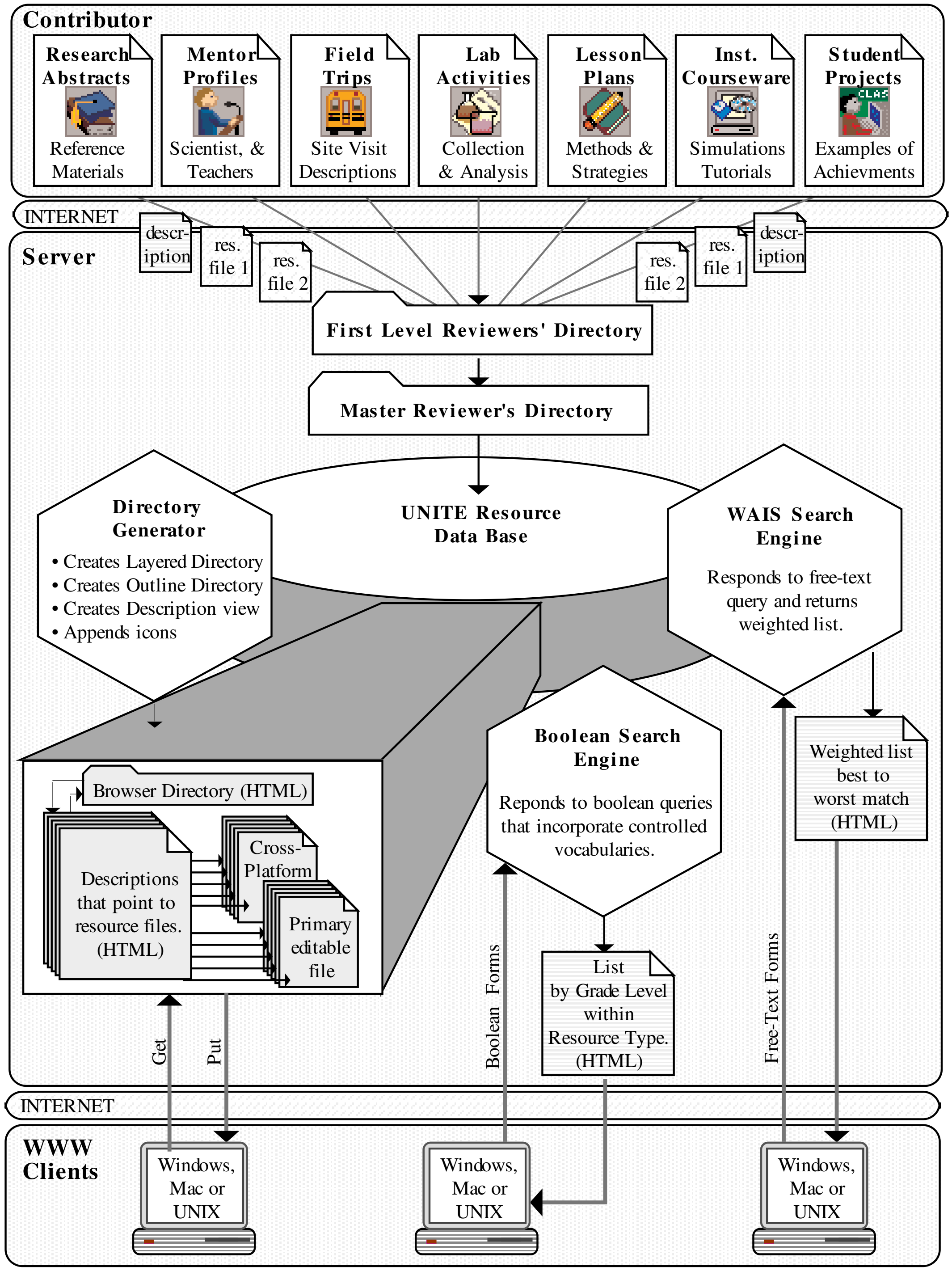

Figure 3-1: UNITE: An Information Service for Contributing, Coordinating and Distributing Educational Resource

a centralized user-readable and modifiable specification of the data stored and its treatment by the system. Following the definition of a database, the records need to be entered and ultimately presented to the user. The records are indexed using the CSO database and are then rendered in HTML. The HTML generation is currently done at contribution time but could be done on-the-fly if it were desirable to trade time for space.

3.3 The UNITE Server

The UNITE server is based on HTTP which has been used by the Web community since 1990 and therefore allows it to be used as a regular Web server. It supports the standard request methods with the addition of the SEARCH method. It runs CGI scripts and supports user directory access. On the other hand, the UNITE server does not support directory indexing, authentication, and a number of other services which were not required for our driving application. The SEARCH method is a unique feature of the UNITE server. It was created to allow the server to directly respond to queries from the client rather than via CGI scripts. It also defines a search syntax, which has yet to be done by the Web community. To support access from other WWW clients, which do not support the SEARCH method, a generic forms interface to the search capability was built. This interface allows the user to select which database and which fields of the database to search on. The current search engine used for UNITE is CSO. CSO was originally written for a simple name service, a computer resident phone book, but required only slight modifications to fit UNITE's needs. It can keep relatively small amounts of information about a relatively large number of objects, and provide fast access to that information over the Internet [4]. CSO also allows for wild card expansion which permits users to be conveniently vague when formulating queries. Another search engine that is currently being integrated into the UNITE server is WAIS. WAIS (Wide Area Information Server) is a free text search engine which would support natural language queries and allow the user to perform inexact searches. Another advantage of WAIS is that it returns a ranked list of matches. This allows the user to select resources that have the best match to the query instead of having to browse through a set of resources to find the best.

3.4 Client: User Interface

The client was based on a prototype developed during earlier pilot projects [1]. This design used a layered approach to represent hierarchal structures similar to the approach used to represent directories in typical graphical user interfaces. Novice users understand how to navigate this structure and they are successful in locating useful resources. The pilot users also provided several suggestions for improving the client interface. Key among these were suggestions for more efficient browsing views of those hierarchical structures and the ability to locate items using multiple selection criteria.

The recent user interface development has been centered on incorporating recent additions to HTML for presenting an easy-to-use interface for constructing Boolean queries using standard WWW clients. Figure 3-2 shows a user constructing part of a Boolean query by specifying, in this case, curriculum values. Additional controlled vocabulary fields may be coupled with remaining text entry fields to form complex queries for specifying resources.

3.5 Distributed Aspects

The success of UNITE as a model for distributed access to collections of information across the Internet depends on a number of factors, but the single most important is ensuring that the system provides good support for adding to the database. Our driving application is a particularly good example of this since the

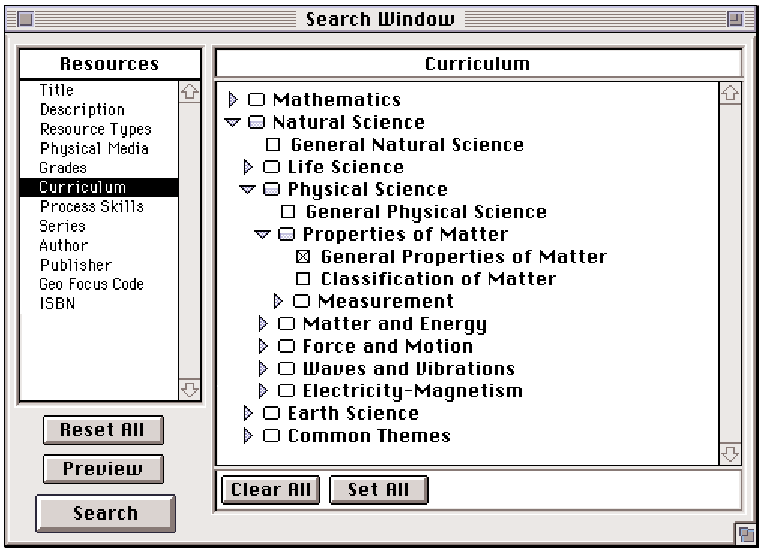

Figure 3-2: The Explorer Client Search Window

educational materials are contributed by the users of the system, rather than by some central authority. However, we believe that this is one of the strengths of the Internet and represents an important aspect of systems which look toward the future of the National Information Infrastructure.

First and foremost, the success of such a database requires the participation of users, who are often the best qualified people to generate source material as practitioners in the field. With this in mind, we implemented a method we called the Contribution Process, supported by software called the Contributor. The Contributor must first know to which database the user wishes to contribute a record. Then the Contributor prompts the user to enter information for each field of the database. The Contributor then sends the newly defined record to a local reviewer. The local reviewer's duties are to make sure the record relates to the application area to which it is being contributed, that it is properly formatted, and is well written. The local reviewer then passes the record along to a master reviewer whose duties are to check the local reviewer's work and approve or reject the record for inclusion in the database. From there, the record is sent to the UNITE server for integration in the database. Currently this is done using FTP but in the future the PUT method will be used. The idea here is that the record is sent to a centralized server, keeping the databases consistent by ensuring that there is only one place where new information is introduced to the system. Once the record is transferred to the server, a series of steps is taken to add the record to the proper database. The first step is to generate an HTML document following the format of the database record definition. Note that this is done on the server and not by the user, keeping a consistent look and feel for all the HTML representations of the database records. Once the database record has been created, it is then added to the database. The final step is to generate a new layered and outline view and to rebuild the database indices. This will allow users to request or search for the newly contributed record.

This Contribution Process is currently run nightly and therefore the time for a newly defined record to appear in the database is usually 24 hours. To distribute server load and improve availability, UNITE supports a method of creating multiple copies of a database on multiple server machines, which is called mirroring. The mirroring process is currently run every night and operates in two modes. The first mode makes a complete copy of the database file structure, including all HTML documents and all indices built by CSO, to the mirrored server. This method is usually used for newly added servers or those that have been inactive for a long period of time. The second method is used for updates to active mirrors. It determines the set of files modified since the last update of the mirrored server and sends. None of the mirrored servers are allowed to receive contributions, thus helping to ensure database consistency.

Chapter 4

UNITE Server

4.1 Overview

The central program of the UNITE project is the server. Everything in the UNITE project interacts with the server in one way or another. Obviously, its main purpose is to serve the client's requests. This brought about the problem of how do the server and client communicate? A protocol had to be defined. When we first started the project in June of 1993, the World-Wide-Web was starting to grow, and its goals were very similar to ours. Therefore, we adopted the HTTP syntax as our protocol. In the intervening years, more features have been added to our server and, therefore, our server has become ideal solution to add, search, and browse resources on the Internet. This chapter will go into detail on the structure of the server and how it can be used to its fullest potential.

4.2 Global Configuration File

The global configuration file provides the UNITE server and tools with the directory structure and other miscellaneous information. Appendix B shows the global configuration file that is actually used by the UNITE project. The first entry file is the 'TopLevelDir' which is the directory in which the software was installed and under which *everything* should be stored.

The 'AuthDir', 'FileSetsDir', and 'UserGroupsDir' are variables used for authorization and authentication. 'AuthDir' is relative to 'TopLevelDir', 'FileSetsDir' and 'UserGroupsDir' are relative to 'AuthDir'.

The 'ContributionDir' is the location in which newly contributed resources are put after being reviewed. 'OldContributionDir' is a directory in which a copy of the original contributed resources is kept. This is done as a safety measure. The 'ReviewDir' is the location in which newly contributed resources are put before they have been reviewed. These three directories are relative to the 'TopLevelDir'.

'MirrorDir', 'MirrorNewFiles', 'MirrorRemovedFiles', 'MirrorUpdatedFiles', 'MirrorServers', and 'MirrorLogs' are directories in which mirroring information is stored.

'ConnectLogs' is a directory in which usage logs are stored. 'DeleteDir' is a directory to which resources deleted using the DELETE method are copied. This is a safety measure provided so that a file deleted by accident can be recovered. 'PutDir' is the directory to which resources are copied when the PUT method is used. This directory is usually the same as the 'ReviewDir' or the 'ContributionDir'.

'ResourceDir' is the directory in which all the HTML files should be stored (i.e. the Document Root). This is where the server will look for any files. 'ScriptDir' is the directory in which the CGI scripts are stored for the server to run. 'DefaultDir' is the name of the script to run when the URL is a slash ('/'). 'GenericDir' is a generic directory where anything can be stored (no special purpose). 'IconDir' is the directory in which the icons are stored. 'BrowserDir' is the directory in which the browser files are stored. 'AuxDir' is a directory used for storing miscellaneous information. 'HomeHTML' is the HTML home page. 'SearchHelp' is the HTML help page for the search interface. 'DeleteMessageFile' is the HTML page displayed to the user after a successful DELETE. 'PutMessageFile' is the

HTML page displayed to the user after a successful PUT. 'DatabaseList' is a file containing a list of all the databases currently used.

'DatabaseCSO' is the directory in which all the databases' indexed files, using the CSO search engine, are stored. 'DatabaseWAIS' is the directory in which all the databases' indexed files, using the WAIS search engine, are stored. 'DatabasePG' is the directory in which all the databases' indexed files, using the Postgres search engine, are stored.

'DbConfigFile' is the database configuration file and is discussed in Section 4.4.1.

'UserDir' is the directory which is appended onto a user's home directory if a user request is received.

'DefaultPage' is the default home page used when a request comes in without a specific file. This is relative to 'ResourceDir'. 'DeletePermission' is the entry in the 'From: ' field that ought to be used for successful use of the DELETE method.

'serverPort' is the port number the server is running on. 'serverHost' is the host name of the server.

'defaultUserGroup' is the user group used when none is specified.

In the 'databaseLocation' table the 'dbName' is the name of the database. The 'engine' is the name of the search engine. The 'dbHost' is the host name on which the search engine is located. The 'dbPort' is the port on which the search engine is listening.

4.3 Structure of the UNITE Server

The structure of the server is shown in Figure 4-1.

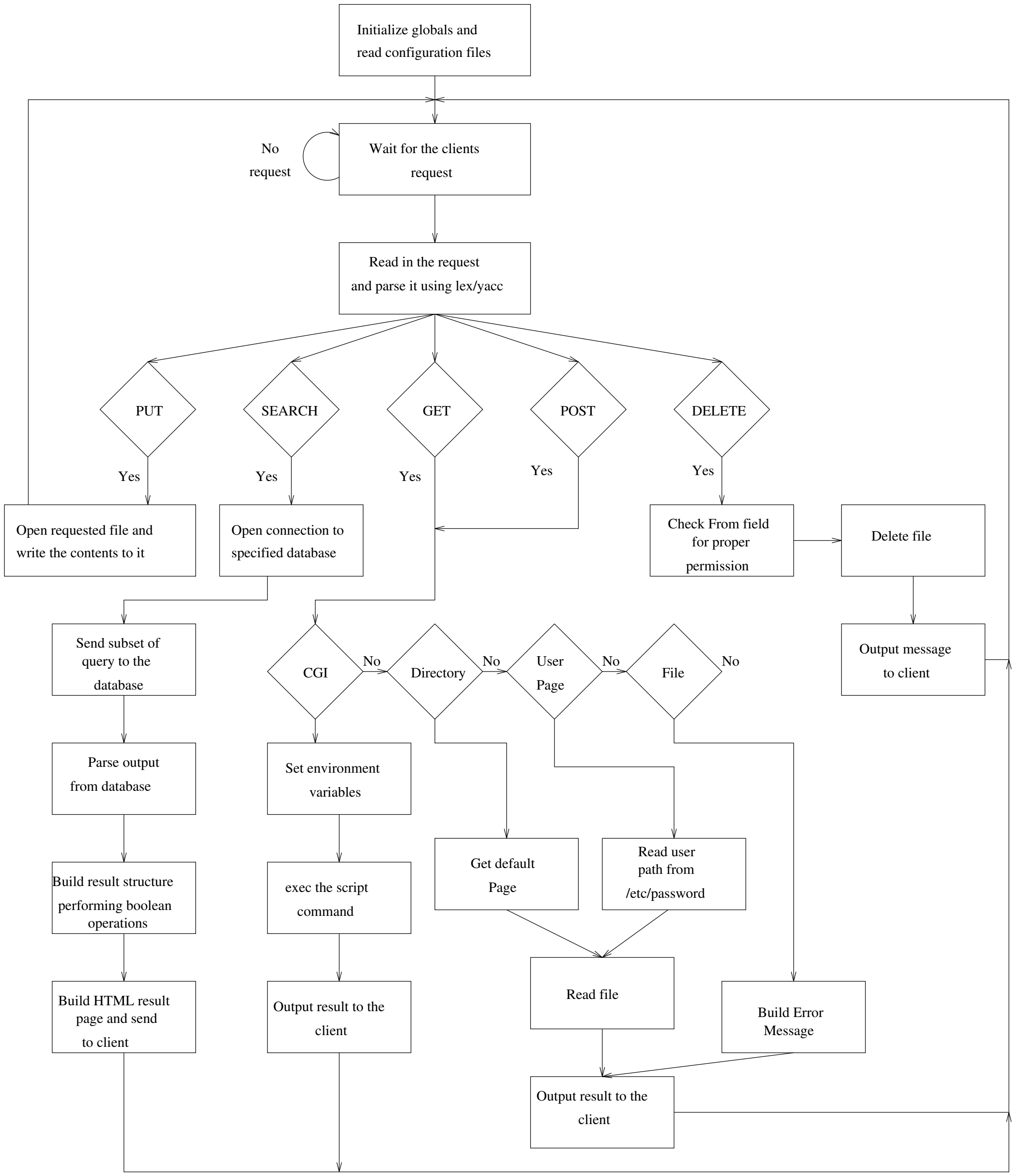

Figure 4-1: UNITE Server Structure

From the client/server paradigm, we have seen that the server has to be waiting for a request to come. Therefore, the first milestone in the server design is to have it listen on a port for a request. This process is known as a daemon and can be achieved in two ways. The first is called a standalone daemon and means that the server is running with no help from any other applications. It performs its own startup tasks: create a socket, bind the server's well-known address to the socket, wait for a connection, then fork. The child process then performs the service while the parent waits for another request. The second method is called inetd and means that the server is running using the BSD UNIX super-server, inetd. The inetd daemon provides two features: (1) it allows a single process (inetd) to be waiting to service multiple connection requests, instead of one process for each potential service, which reduces the total number of processes in the system, and (2) it simplifies the writing of the daemon processes to handle the requests, since many of the start-up details are handled by inetd. However, there is a small price to pay for this in that the inetd daemon has to execute both a fork and exec to invoke the actual server process, while a self-contained daemon that does everything itself only has to execute a fork to handle each request.

On startup, inetd reads the */etc/inetd.conf* file. Therefore to add a service, simply add the proper line in this file and restart the daemon (usually with a *kill -1* signal). Figure 4-2 gives the steps followed by inetd for each services in the configuration file.

Once a connection is accepted, the server reads the content of the socket and parses the request. The protocol used is HTTP with a few modifications added. Figure 4-3 gives the BNF for the protocol used by the UNITE server.

The first symbol is the method used by the client. This can either be a PUT, SEARCH, GET, POST, or DELETE.

The PUT method tells the server that the client wants to add a new file to the server. The name of the new file is the Request-URI. The server creates the file

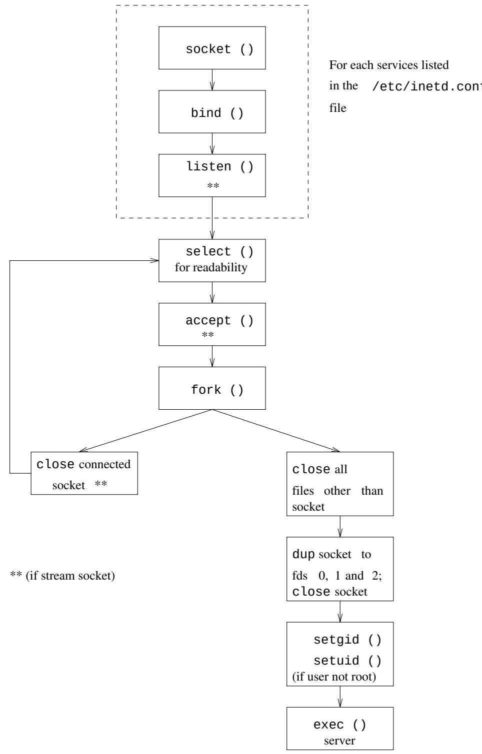

Figure 4-2: Steps performed by inetd

Request = Request-Line
*(Request-Header
| Entity-Header)
CRLF
$[Entity - Body]$

Request-Line = Method SP Request-URI SP HTTP-Version CRLF
Method = "GET" | "PUT" | "POST" | "DELETE" | "SEARCH"
Request-Header = Accept
| From
| Pragma
| Referer
| User-Agent

Entity-Header = Content-Length
| Content-Type

Figure 4-3: BNF for protocol

containing the Entity-Body sent by the client. All the parameters concerning the Entity-Body is given in the Entity-Header: the Content-Length tells the server how many bytes to write to the file and the Content-Type tells the server the type of the Entity-Body. A message is then returned to the client informing the user that the operation was successful. The message displayed to the client is the content of a file, defined in the global configuration file, on the server and therefore can be easily modified. Figure 4-4 gives an example for this method.

```
PUT /new_file.html HTTP/1.0
Accept: text/html
Content-Type: text/html
Content-Length: 84

This is the content of new_file.html. Whatever I type here
will appear in the file.
```

Figure 4-4: Example for PUT method

The DELETE method tells the server that the client wants to remove a file from the server. Figure 4-5 gives an example for this method. This method could be dangerous; we would not want everybody to remove all the files on the server. To make this method more secure, the From field, in the Request-Header, is checked and compared with the value given in the global configuration file. If the value, from the global configuration file, matches the value given by the client then the file is copied to a delete directory. At this point, it is up to the system administrator to delete the file. The response to the client is again the content of a file defined in the global configuration file (similar to the PUT method).

```
DELETE /new_file.html HTTP/1.0
Accept: text/html
From: CedricDeniau
```

Figure 4-5: Example for DELETE method

The GET method is used to retrieve documents from the server and to run CGI scripts. If the Request-URI is a valid file on the server then the server returns the content of the file. If the Request-URI is a directory then the server checks for the default page in that directory. The default page is defined in the global configuration file. If the Request-URI is a user path (using the $\sim$ symbol in the URL) then the server will check in the */etc/passwd* file for the absolute pathname of the user's home directory. Then the server will look in the global configuration file for the "UserDir" which is appended to the user's home directory. Once this is done the server will append the name of the file the client requested and return the content of the file. If the Request-URI specifies a valid CGI script then the server will execute the script passing the program the proper environment variables. For CGI scripts, the GET method appends any client information to the Request-URI. Therefore, the server has to parse the Request-URI for the script name and the client information. This is possible because the two are separated by a question mark (?). The server then passes the client information to the CGI script program through an environment variable called QUERY_STRING. Figure 4-6 shows examples for the different points just discussed.

The POST method is used to perform CGI scripts. When a POST request comes in, the server first checks if the Request-URI is a valid CGI script program. Any information sent by the user is added to the Entity-Body and the Entity-Header fields are properly set by the client. The client information is put in the Entity-Body and passed to the CGI script through standard input (*stdin* in C). The CGI scripts knows how much data to read from standard input from the environment variable CONTENT_LENGTH which is passed by the server to the CGI script. If the Request-URI is a valid CGI script then the server executes the specified program and passes it the proper environment variables. Figure 4-7 shows an example of a POST method.

The SEARCH method is a unique feature of the UNITE server. It was created to allow the server to directly respond to queries from the client rather than via CGI scripts. It also defines a search syntax, which has yet to be done by the Web

1. Client requesting:
   (a) A file
       GET /file.html HTTP/1.0
       Accept: text/html
   (b) A user file
       GET / deniau/file.html HTTP/1.0
       Accept: text/html
   (c) A directory
       GET / deniau HTTP/1.0
       Accept: text/html
   (d) CGI script
       GET /cgi-bin/script_name?name=cedric
       Accept: text/html
2. Server response:
   (a) The content of file.html
   (b) The content of /users/deniau/.public_html/file.html where /users/deniau came form the /etc/password file and .public_html came from the global configuration file specifying the location on the user's HTML files.
   (c) the file /users/deniau/.public_html/index.html the index.html file comes from the global configuration file and, therefore, depends from server to server.
   (d) executes the script script_name with the environment variable QUERY_STRING set to name=cedric.

Figure 4-6: Example for GET method

```
POST /cgi-bin/program_name HTTP/1.0
Accept: text/html
Content-Type: text/html
Content-Length: 83

firstName=Cedric+lastName=Deniau
```

Figure 4-7: Example for POST method

community. Appendix A shows the BNF for this method.

The protocol-type is the protocol the server understands and is used for version control. The DBSpec gives the name of one or more databases to search. If an invalid database name is specified, an error message is returned informing the client of such an error. If the database specified is valid, but unavailable, then a different error message is returned. In either case, an error on a database does not prevent the search process from continuing on other databases specified in the same query. The SessionSpec gives resource control parameters. The time-pair specifies the maximum number of seconds a search may take, the cost-pair specifies the maximum connection cost that a search may take, and the distance-pair specifies the maximum distance at which a database may be and still be searched. If any of these maximums are violated, a message is returned to the client. A user has the option of overriding the maximum, and continuing the search, up to a true maximum, or seeing the incomplete search results. The SearchSpec specifies which records get returned. The operators defined are *and*, *or*, *andnot*, and *contains*. The *and*, *or*, *andnot* perform the standard Boolean operations, while the *contains* allows the user to search for specific values. The ReturnSpec indicates how to present the identified records, including how to present extremely large retrieval sets. The max_num_full specifies the number of full record to present, max_num_sum specifies the number of summary record to present, max_size_full specifies the maximum size of the full records set in bytes, max_size_sum specifies the maximum size of the summary records set in bytes, sort_method is a sort specification with the primary sort key being the one most nested, show_full_method specifies how to present the full records, and show_sum_method specifies how to present the summary records. Figure 4-8 shows an example query sent by the UNITE client.

In the example, the database searched is UNITEResource. The SessionSpec sets the maximum search time to 500 seconds. The SearchSpec tells the server to

```
SEARCH Unite-2.0
 ((UNITEResource)
 (:maxTime 500)
 (and (contains "Title" "animal")
      (contains "Grades" "6")
 )
 (:maxNumSummaryRecords 200
 :sort-by (:alpha "ResourceType" (:alpha "Title") )
 :show-summary ( "ResourceType" "Title" ))
 )
```

Figure 4-8: Example for the SEARCH method

search for records with the word animal in their title and classified as belonging to grade level 6. The ReturnSpec specifies to return no more than 200 records sorted, first, alphabetically by Title and then sorted by their ResourceType.

4.4 The Databases

The objective of the UNITE project is to allow users to browse and search resources on a server. Therefore, a database containing resources had to be configured and a search engine had to be designed. Each database has a configuration file associated with it which describes the structure, format, and treatment of the database records. Databases can store several classes of information and must be capable of managing significantly different kinds of data (i.e. software, text, video, sound, etc.) This section will discuss the database configuration file and the search engine used by the UNITE project.

4.4.1 Database Configuration File

The database configuration language is used to specify record structure, and defines four basic objects: TABLE, ENUMERATION, RECORD, and DATABASE OBJECT. This language provides a centralized user-readable and modifiable specification of the data stored and its treatment by the system. Appendix C illustrates an example of a database configuration file and Appendix D gives the syntax of the database configuration file. Appendix E gives an example of a contributed file built from the database configuration file.

The DATABASE OBJECT section defines a UNITE resource's fields and field attributes, using one line per field. This section is first defined by the keywords DATABASE OBJECT, followed by a STRING, which is the name of the database. A NUMBER then follows, which is the version number. The resource's fields follows, enclosed in braces and delimited by a semi-colon. The first attribute of an entry is the field type, which can either be a predefined or a user defined type. The predefined types are: string, integer, uid, and freetext. A string is defined as a sequence of characters enclosed in double quotes, and an integer as a sequence of numbers from 0 to 9. Freetext is the same as a string except it can contain line feeds. The user defined types are either enumerations or records. The next attribute specifies how many items the field can contain: One, OneOrMore, ZeroOrMore, or Zero. The third attribute specifies how the field is used during a search, while the last attribute is the name of the field used by the database.

In the example of Appendix C, the name of the database is UNITEResource, and the version number is 1994092001. The last entry of the DATABASE OBJECT section specifies that the field "Reviewers" is of type "string", can hold one or more values and is not searchable. As another example, the field "Curriculum" is of type "CurriculumT" which is an ENUMERATION representing a set of values that are hierarchically defined. Therefore, the "Curriculum" field can only contain values that are explicitly defined in the ENUMERATION "CurriculumT". Some possible values could be: "Mathematics", "Mathematics/Problem Solving and Reasoning/Generalize" and "Natural Science/Life Science". "Curriculum" can hold "OneOrMore" values which means that there has to be at least one value defined and it is a "KeywordValue" meaning that it is searchable through a keyword based search engine like CSO.

The other user defined type is a RECORD. This RECORD object uses the same set of parameters as the DATABASE OBJECT. However, the record defined is used as a type for a field in the DATABASE OBJECT rather than defining an object directly. This allows for a more flexible definition of the database. Following our example in Appendix C, the field "FileDescriptions" is of type "FileDescriptionsT" which is a RECORD. This RECORD contains a field "FileDescription" which is of type "FileDescriptionT" which is also a RECORD. This record contains five fields: "FileSizeInKBytes", "FileFormat", "FileName", "FileSet", and "FileEncoding".

The TABLE section gives extra flexibility to the system by defining a mapping from one set of values to another. From the syntax, this section is first defined by the keyword TABLE followed by a STRING, which is the name of the table. The table entries then follow enclosed in braces. Each entry consists of two STRINGs and each entry is delimited by a semicolon. The first STRING in an entry is used as the index and the second STRING is mapped to the value.

The ENUMERATION section defines a set of valid values a database field is allowed to have. The syntax for this section is first defined by the keyword ENUMERATION followed by a STRING which is the enumeration name. The content of the enumeration then follows enclosed in braces. All enumerations are hierarchic. Some hierarchies may just be one level deep making them look like simple lists. For example, the ENUMERATION "ResourceTypeT" is a simple list of valid values for the field "ResourceType". On the other hand, the ENUMERATION "FileFormatT" is a hierarchic list of valid values for the field "FileFormat". Internally, both of these enumerations are represented in the same manner.

Following the definition of a database, the records need to be entered and ultimately presented to the user. The records are indexed using the CSO database and are rendered in HTML. The HTML generation is currently done at contribution time but could be done at runtime if it were desirable to trade time for space.

4.4.2 CSO

The current search engine used for UNITE is CSO. CSO was originally written for a simple name service, a computer resident phone book, but required only slight modifications to fit UNITEs needs. It can keep relatively small amounts of information about a relatively large number of objects, and provide fast access to that information over the Internet [4]. CSO also allows for wild card expansion which permits users to be conveniently vague when formulating queries. The main problem with CSO is that it is inappropriate for large target text items and it does not have boolean search capabilities. This motivated us to implement set operations (i.e. and, or, contains, ...).

4.4.3 Adding Databases Engines

To add a new search engine to the UNITE system, only a handfull of functions would need to be written. First, functions to format and send the query to the new search engine are needed. Then, once the search engine returns the results, functions will have to be written to parse that result in the proper data structures. Finally, the global configuration file would have to be modified by adding an extra line in the "databaseLocation" section (refer to the example in Appendix B) and the database would have to be built.

Chapter 5

UNITE Tools

5.1 Overview

To facilitate the use and enhance the features of the UNITE application, a few tools have been built. Some of these allow users to contribute resources, search the database, and browse the database. This chapter will discuss these tools in depth and suggest possible modifications and enhancements.

5.2 Contributor

The Contributor runs as a CGI script through the UNITE server. When the Contributor is called, it first asks the user to select the database with which he wishes to work (Figure 5-1). The list of these databases are in the "DatabaseList" from the global configuration file (Appendix B). Then the Contributor will prompt the user to enter values (Figure 5-2) for the fields in the database which are specified in the database configuration file, defined in Appendix C. From there, the Contributor will build the DBML document (Appendix E) and put it in the "ReviewDir" from the global configuration file. This directory is used to store contributions that have not yet been through the review process.

Once the file has been reviewed, the file is moved to the "ContributionDir". At this point, another program is run to add the newly contributed resource to the database. This program, the Renderer, parses the content of the DBML file to build the final HTML file and to perform administrative duties. This program only requires one argument: the name of the DBML file. The program will automatically look in the "ContribDir" for the file.

Some resources can be contributed with attached files. These files could be GIFs, MPEGs, or anything the user wants. At this time, this cannot be done through the Contributor on any regular Web browser since file uploads have not yet been incorporated. An Internet draft has been written to address this problem but nothing concrete has been done to solve this problem [11]. The contribution functions are currently done using the UNITE client which was developed concurrently with the UNITE server. To add an attached file to a resource, the "FileDescriptions" field has to be completed. From the database configuration file, this field is of type "FileDescriptionsT", which is a RECORD. This record contains a field called "FileDescription" which is of type "FileDescriptionT" which is also a record. This final record contains five fields: "FileSizeInKBytes", "FileFormat", "FileName", "FileEncoding", "FileSet". These fields **must** be given a value. Note that if multiple databases are built and attached files are needed for these databases, then these exact fields and records have to be defined with the identical values. Any changes will cause the Renderer to work in properly.

The first administrative duty is to generate a unique identifier (a.k.a. uid). This is done so that duplicate resources will not exist. A field in the database configuration file **must** be defined as type "uid". If this is not done, errors will occur. The uid is saved in the DBML file as the "IDNumber" field. When the file is originally contributed, the Contributor sets this field to 0 which means that this is a new resource. If the "IDNumber" field is not 0 then the Renderer will use the given uid as the name of the file and remove any previously existing files

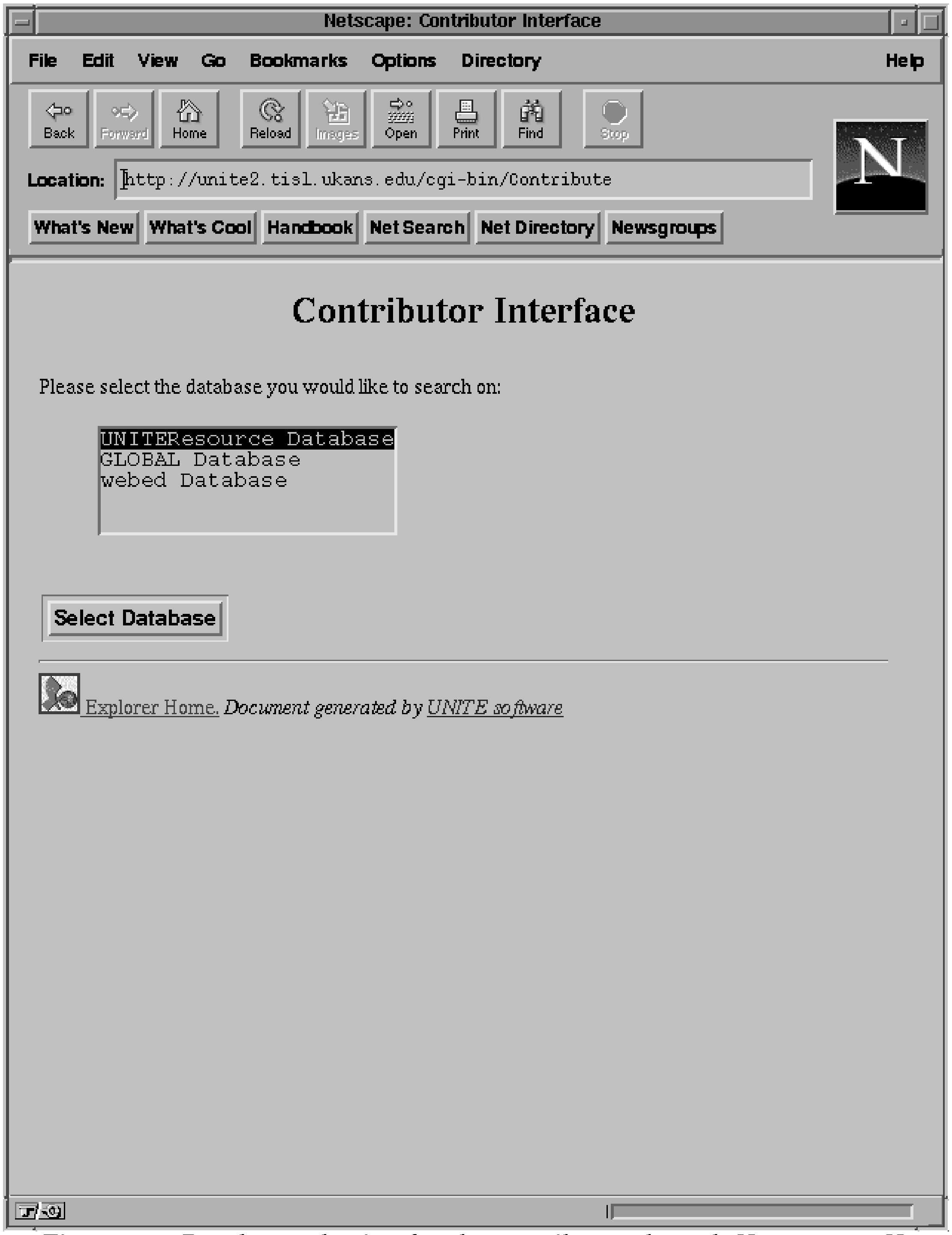

Figure 5-1: Database selection for the contributor through Netscape 1.1N

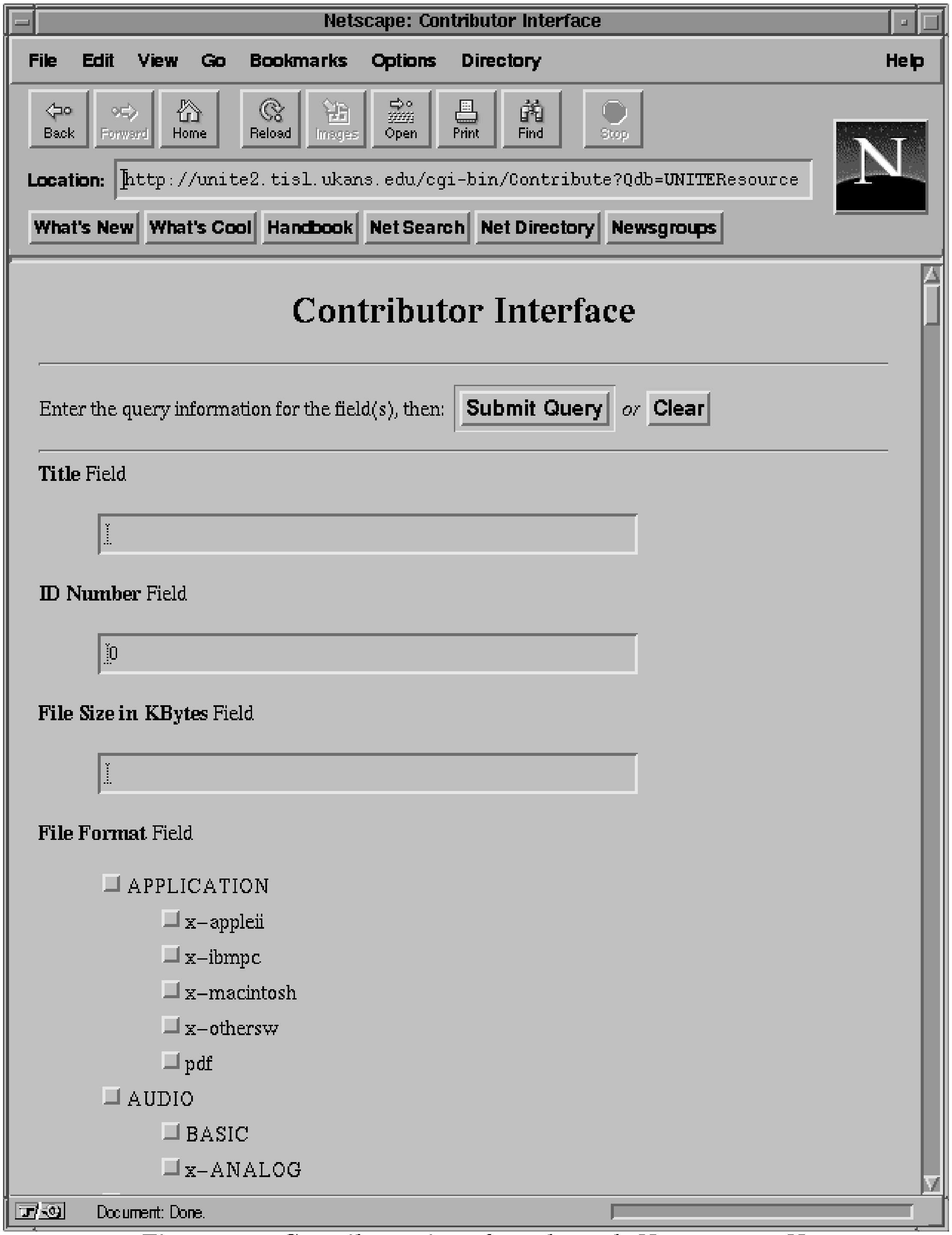

Figure 5-2: Contributor interface through Netscape 1.1N

using the given uid.

Next, the Renderer will add the name of the resource to the "MirrorDir". If the resource is new then a file is created in the "MirrorNewFiles" directory. If the resource is a recontribution then a file is created in the "MirrorUpdatedFiles" directory. The file created is named using the year, month, and day the resource was contributed. This was originally done for mirroring purposes but is now a tool to check what has been contributed and when. The Renderer also adds the name of the resource to the "AuthDir".

Should a contributed resource have an attached file, the Renderer will then read the content of the "FileSet" field and create a file in the "FileSetsDir" containing the name of the resource. The name of the newly created file is the value given in the "FileSet" field.

Following all of this, the Renderer then builds an HTML document from the DBML and moves the file or files (depending on whether or not attached files exist) to its database directory, "ResourceDir". The original file(s) is moved to the "OldContributionDir" as a safeguard.

5.2.1 HTML Builder

To build the HTML file, a library of functions was built. This library can also be used for generating HTML on-the-fly. The HTML is configured using the "htmlPrint.config" file. This file contains methods to build HTML syntax and can, therefore, be changed without having to recompile the program.

All resources are built using the same HTML syntax. Therefore, they all look alike. The attached files are included as a link from the main resource to the attached file. Figure 5-3 shows the rendered HTML version of the DBML file included in Appendix E.

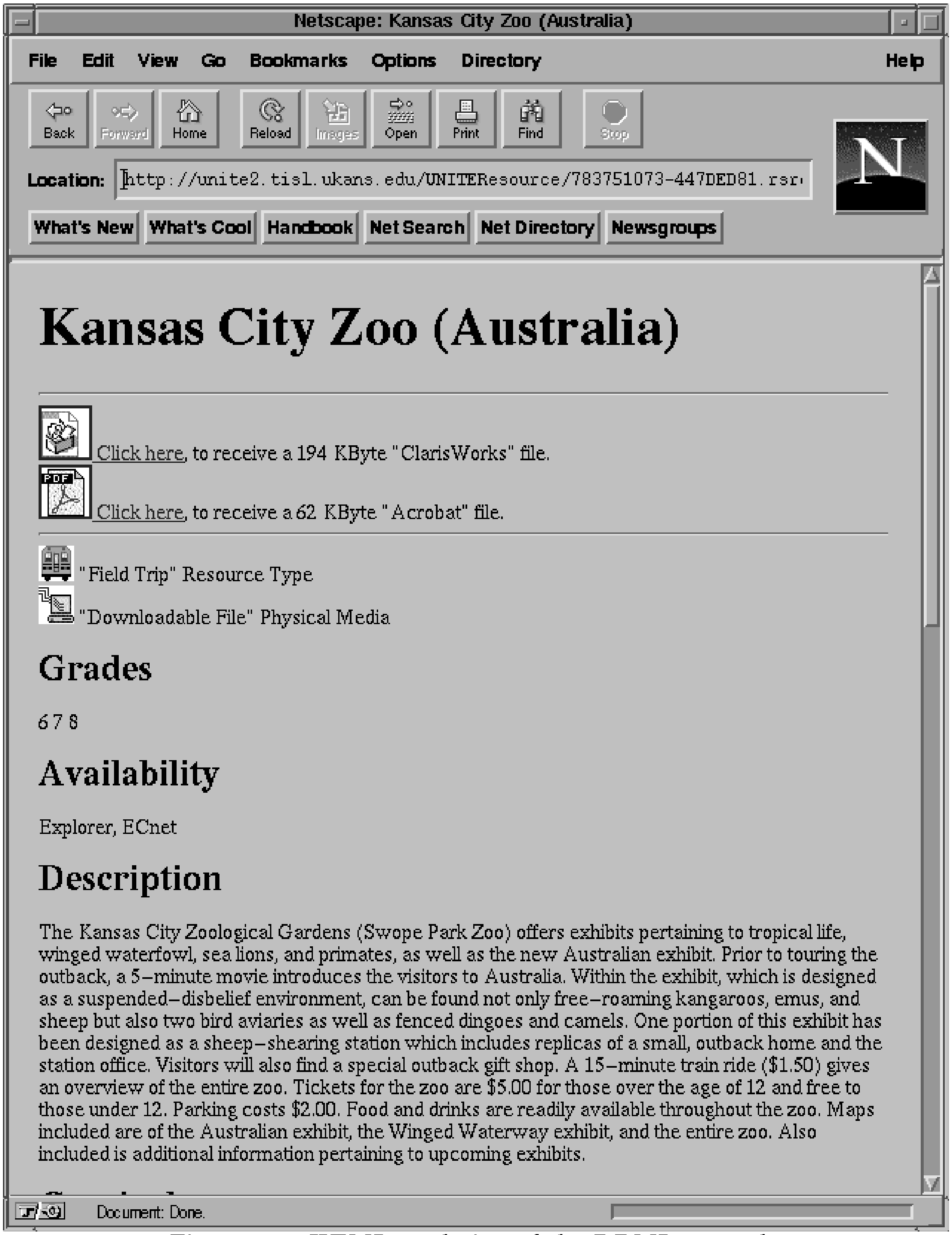

Figure 5-3: HTML rendering of the DBML example

5.3 Browser

The UNITE browser provides views of the database to the user in an HTML format. The two views are the outline and layered views. These views are built using a field in the database. In our application, these views are built using the "Curriculum" field. This field is used because it is a hierarchic enumeration and all resources have to contain a value since it is defined to be a "OneOrMore" field (refer to Appendix C). It is recommended that a hierarchic field be used for the browser since it generates a layered and outline view. If the field is a flat enumeration, there would be no difference between those two views. Figures 5-4 and 5-5 show an example of the two views for our application.

The browser program requires parameters at runtime. The first parameter is the name of the database. The second is the field on which to build the views. The final parameter is the name of the configuration file. This configuration file is special to the browser and will be referred to as the browser configuration file. It is used to configure the output of the views. As the user steps down the layers of the database and finally gets to a leaf of the tree, the user is presented with a list of the resources. This list is configured using the browser configuration file. For our application, the "Title", "Grades", and "ResourceType" fields are used. The resources are first organized by "ResourceType", then by "Grades" and finally alphabetically by "Title". This is shown in Figure 5-6.

The browser configuration file is shown in Figure 5-7. This configuration file shows all the features available. There are a few reserved keywords for this language: ANCHOR, CLOSE, TEXT, ICON, +, and -. The ANCHOR keyword is used to specify that the information enclosed should be used for an HTML anchor (i.e. hyperlink). This keyword requires an argument. This argument is the field that should be used to link to. The keywords ANCHOR CLOSE close the anchor. The TEXT tag specifies that the field's value should be displayed. The + and - keywords are used for tags that contain a list of values. The +

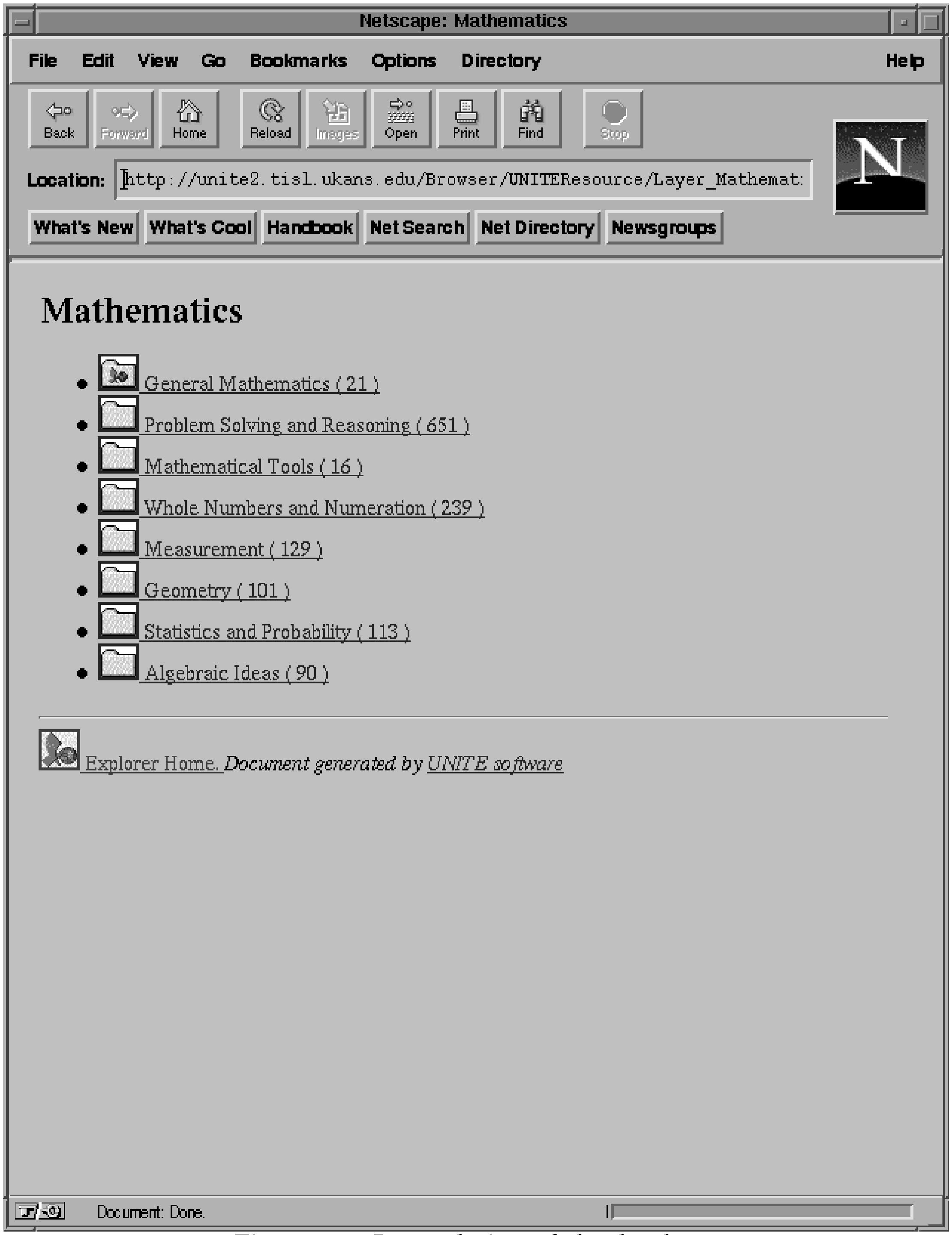

Figure 5-4: Layered view of the database

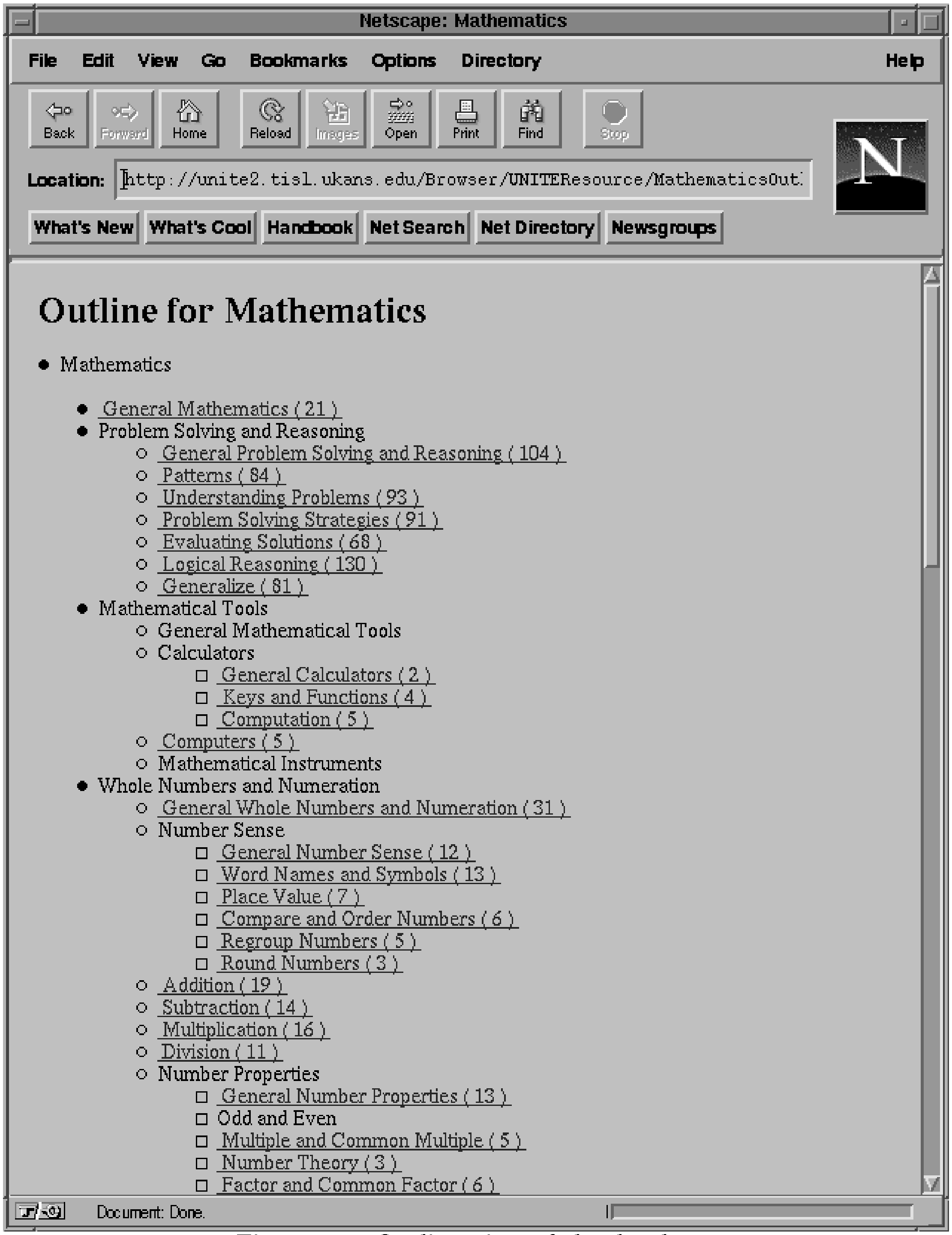

Figure 5-5: Outline view of the database

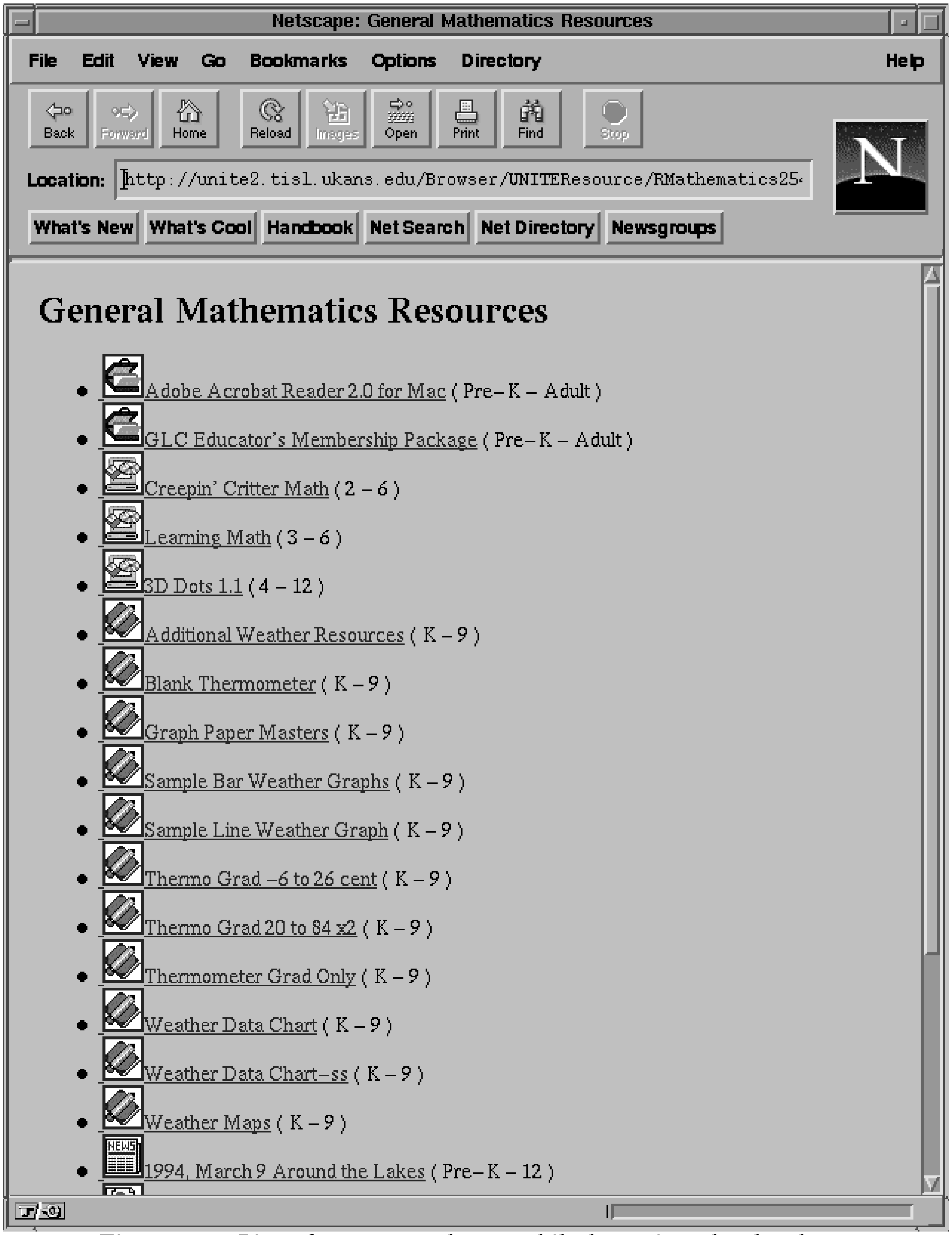

Figure 5-6: List of resources shown while browsing the database

```
(ANCHOR IDNumber)
(ICON 1 ResourceType ResourceType_Table)
(TEXT 3 Title)
(ANCHOR CLOSE)
" ( "
(– 2 Grades)
" - "
(+ 0 Grades)
" )"
```

Figure 5-7: Browser configuration file

specifies the greatest value should be obtained from the list and the - specifies the smallest value from the list. These values can be obtained in two separate ways, depending on how the code was compiled. If the code was compiled with the -DNOT_PRE_SORTED flag, then the values are compared using the C function strcmp. If the flag was not specified, then the values are presumed to be already sorted in ascending order. If this is true then the smallest value is listed first and the greatest value is listed last. The ICON tag specifies that an image should be displayed. This keyword takes an argument which is the name of the field to use to get the value.

The ICON, TEXT, -, and + keywords can all have two additional arguments. The first argument is a number. This number represents the order in which to sort the list. In our example, we are first sorting by the ICON field, then by the - field, and finally by the TEXT field. The number 0 is used to specify not to sort the field. The second argument is the name of a table. This argument is used to look up a match in the table for the value of the field. This is shown in the ICON keyword.

Once the browser is started, HTML formatted files will be created in the current directory. These files contain the information for displaying the views. The beginning file for the outline view is tagged with the name "Outline.html" at the end of the file name and the file for the layered view is tagged with the name

"Layer_" at the beginning of the file name. These are the two files that should be pointed to to initiate the browsing of the database records.

5.4 Database Builder

One of the tasks necessary when contributing resources is to build the search engine's index files. Currently, the search engine being used is CSO. This engine requires two files to be built before it runs its own indexer. The first file is the configuration file. This file contains a description for each field. Figure 5-8 shows an example configuration file using the example database configuration file and shows why a human would not want to build this file himself.

```
55:Title:256:Title:O:Indexed:Lookup:Public:Default:
56:IDNumber:256:ID Number:O:Indexed:Lookup:Public:Default:
57:FileSizeInKBytes:256:File Size in KBytes:O:Indexed:Lookup:Public:Default:
58:FileFormat:256:File Format:O:Indexed:Lookup:Public:Default:
59:FileName:256:File Name:O:Indexed:Lookup:Public:Default:
60:FileEncoding:256:File Encoding:O:Indexed:Lookup:Public:Default:
61:FileSet:256:File Set:O:Indexed:Lookup:Public:Default:
62:ResourceType:256:Resource Type:O:Indexed:Lookup:Public:Default:
63:PhysicalMedia:256:Physical Media:O:Indexed:Lookup:Public:Default:
64:Grades:256:Grades:O:Indexed:Lookup:Public:Default:
65:Series:256:Series:O:Indexed:Lookup:Public:Default:
66:Availability:5000:Availability:O:Indexed:Lookup:Public:Default:
67:Description:5000:Description:O:Indexed:Lookup:Public:Default:
68:Curriculum:256:Curriculum:O:Indexed:Lookup:Public:Default:
69:ProcessSkills:256:Process Skills:O:Indexed:Lookup:Public:Default:
70:Author:256:Author:O:Indexed:Lookup:Public:Default:
71:Publisher:256:Publisher:O:Indexed:Lookup:Public:Default:
72:Reviewers:256:Reviewers:O:Indexed:Lookup:Public:Default:
```

Figure 5-8: Configuration for the CSO search engine

The second file contains the content for each field for each resource. Both of these files are generated automatically from the database configuration file (Appendix C). This program takes one argument to run and that is the name of the database. With that, the program will know where the resource files are and where to build the two files for the CSO search engine. Once this is done, CSO has its own programs to generate its index files. A shell script has been written to run these programs in the proper sequence. Therefore, the final step to building the index files is to run the script. There is one problem with building these index files. To compile CSO, a configuration file had to be created. This configuration file asks for the directory in which the two files generated earlier are located. Therefore, the CSO programs will always look in the same directory for those files. This can cause a problem if there are multiple databases. To solve this a directory called BUILDER was created. This directory was defined in the CSO configuration file as the directory in which to look for those files. Therefore, when building a database, it will be necessary to move the two files generated by the database builder to the BUILDER directory and then run the CSO programs to build the index files. Once this is done, just move all the files generated back to the directory defined for the database. The BUILDER directory can be changed. This is done in the CSO configuration file and it needs to be done before compiling the code for CSO.

5.5 Search Interface

A search interface was built to allow users to search the content of the database using a Web browser capable of supporting forms. This search interface is a simple C program that runs as a CGI script through the UNITE server. The program first asks the user which database he would like to search on. The list of the databases is in the "DatabaseList" file defined in the global configuration file (Appendix B). The user is also asked to select the level of the interface. Once these selections are done, the user is asked to choose which fields he would like to search on. If the user selects the "Dead Simple" level for the interface, then the program will restrict the user to only one entry per field. If the user selects the normal level, then the user is asked to select between 1 and 5 entries per fields. However, not every field can have more than one entry. For example, an enumeration does not need to have more then one entry since all the choices are there to select from. Figures 5-9 and 5-10 give an example of each interface. Figure 5-11 shows the initial page.

Once the user has selected which fields he would like to search on, he then enters the values for each entry. This page also allows the user to select the number of records to be returned from a search. For example, the user may only want to see five records and not all one thousand that could have been returned from a search. This allows the user to tailor his search more to his needs. From this page the user can also look at a help page. This page is a file on the server that can be modified at any time and is defined in the global configuration file as "SearchHelp". Once this is done, the query is sent to the UNITE server and the result is returned. From there, the user can select which resources at which to look.

All of these pages are built from the database configuration file and the "clientPrint.config" file. Therefore, any changes to the database will not require any recompilation. Also, some information had to be passed from page to page. For example, the name of the database had to be passed from the first introduction page to the last page. Since the server is stateless, there was no way of doing this through it. The only way the information could be transferred is through the forms in the HTML pages. Therefore, the information is passed as a hidden form. This hidden form is just like a regular text entry form except it is not visible to the user. The content of the form is passed the same way as any other form therefore causing a state. The hidden forms are an added feature of Netscape and have not yet been implemented by Mosaic.

Another necessity was the need to save a query. This cannot be done on the

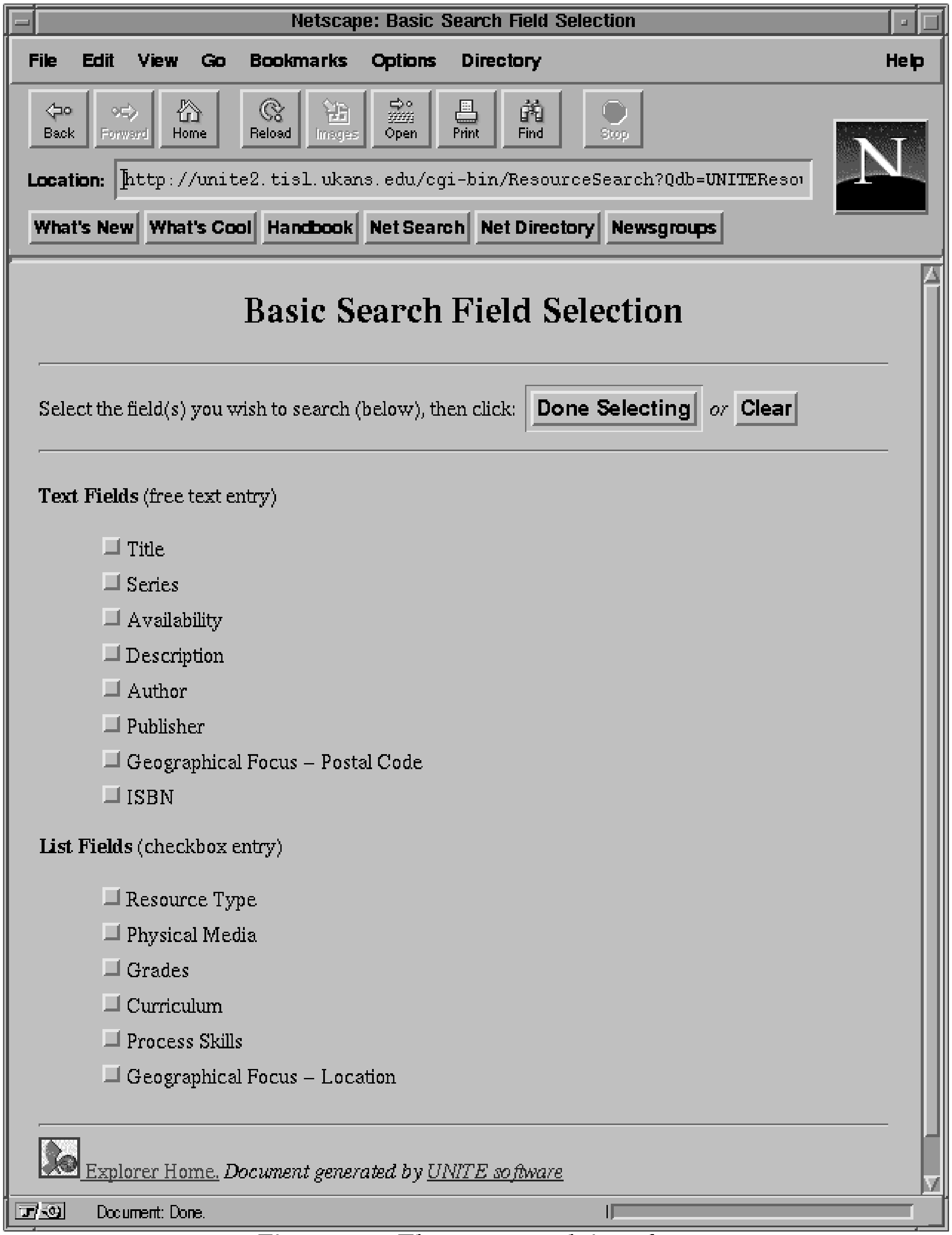

Figure 5-9: The easy search interface

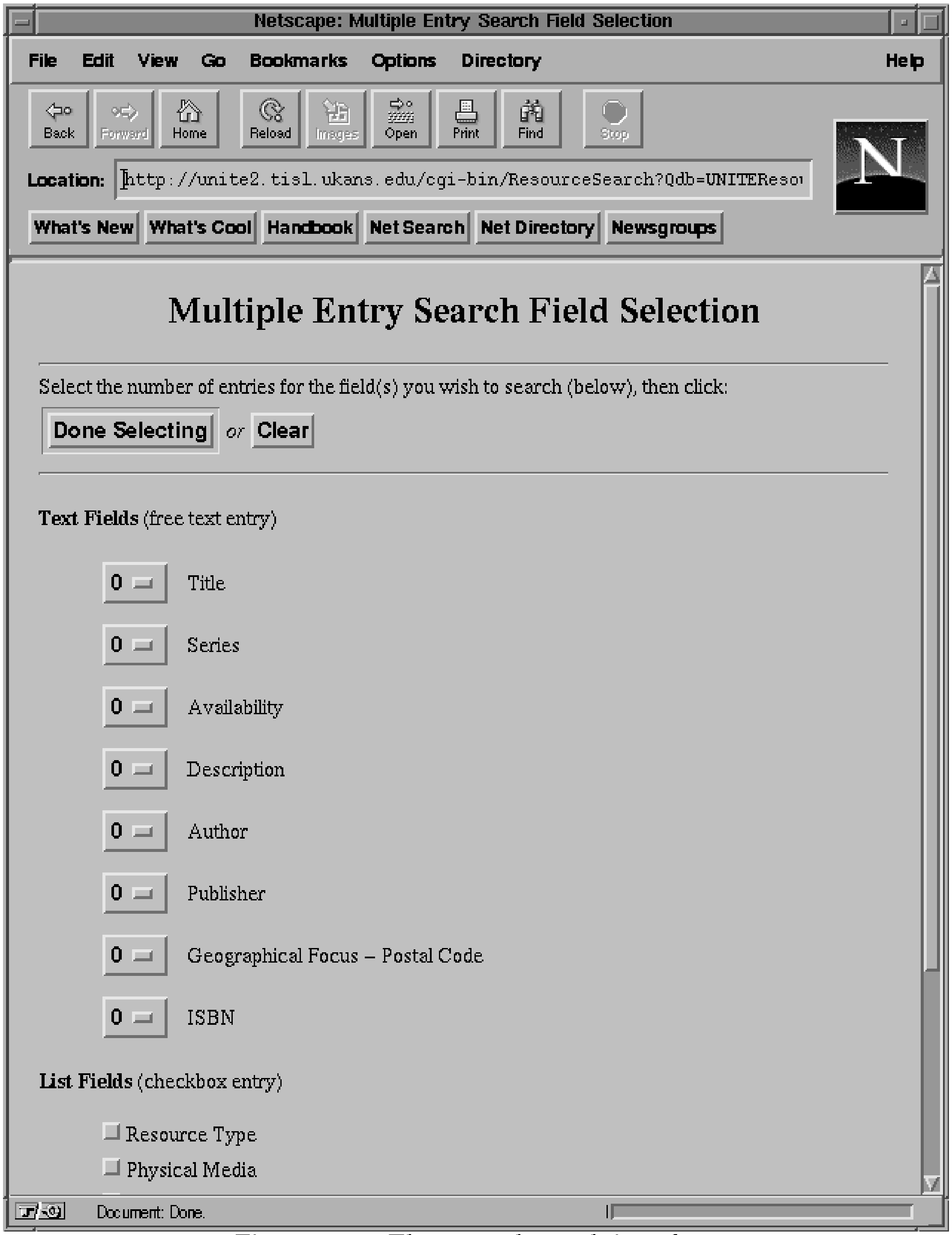

Figure 5-10: The normal search interface

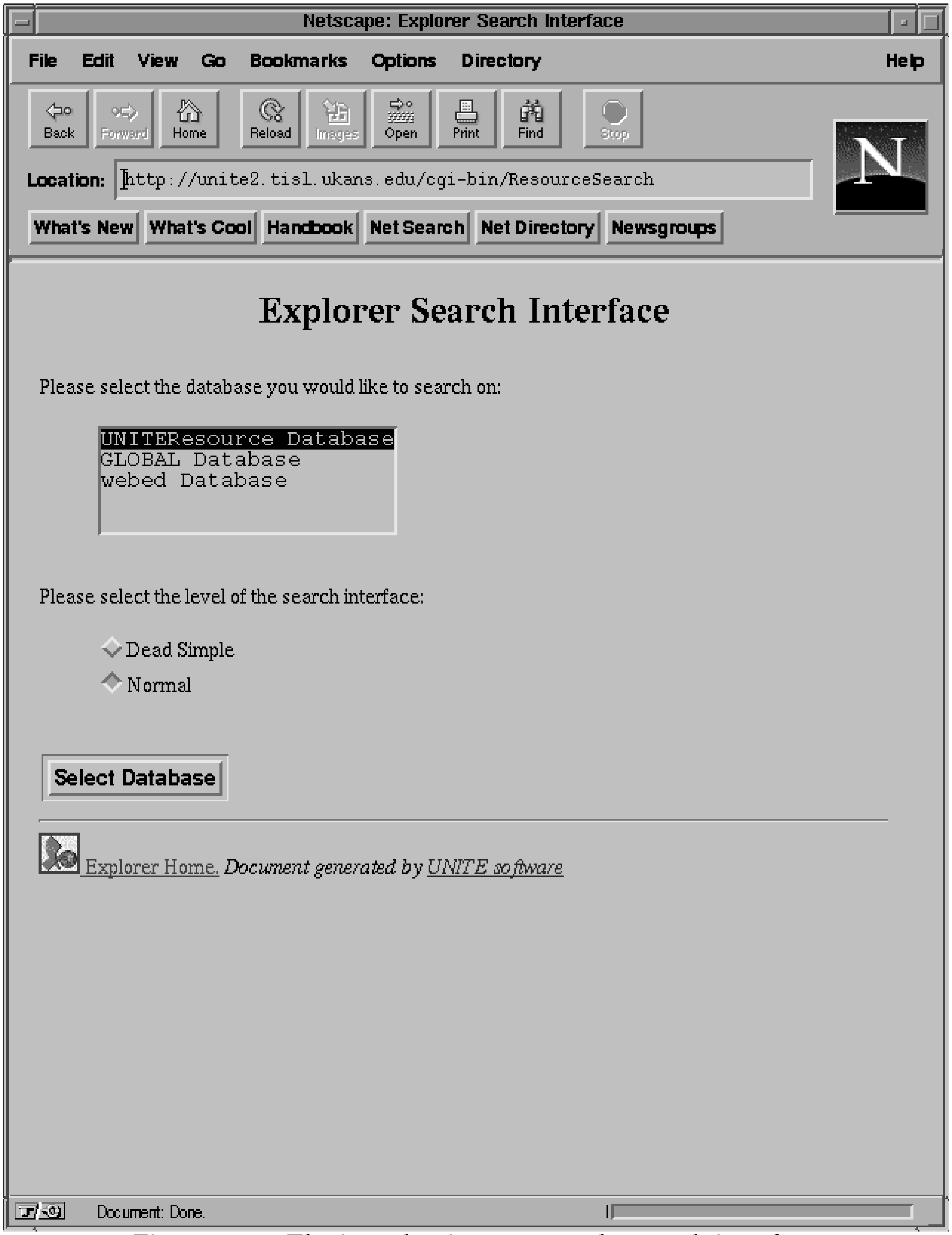

Figure 5-11: The introduction page to the search interface

server since it is stateless and does not know which user is sending a query. Therefore, this has to be done on the client side. This was easily accomplished using the GET method for CGI scripts instead of the POST method. The difference between the two is that the GET method appends all the forms information to the URL and the POST method does not. Therefore, once a query has been sent and the user wants to save his query for future use, all he needs to do is save the page in his bookmark or hotlist. Then when the user later looks through his bookmarks he can resend a query by simply selecting the URL. This will send the query back to the server and the updated result will be returned.

5.6 Mirroring

To distribute server load and improve availability, UNITE supports a method of creating multiple copies of a database on multiple server machines, which is called *mirroring*. The mirroring process operates in two modes. The first mode makes a complete copy of the database file structure, including all HTML documents and all indices built by CSO, to the mirrored server. This method is usually used for newly added servers or those that have been inactive for a long period of time. The second method is used for updates to active mirrors. It determines the set of files modified since the last update of the mirrored server and sends. In order to ensure database consistency, none of mirrored servers should be allowed to receive contributions.

The mirroring script takes three optional arguments and one required argument. The first optional argument is the host of the mirrored server, the second is the directory to mirror, and the final is the archive file. The required argument is the method used.

5.7 CGI Scripts

Included with the UNITE system are a few additional CGI scripts. These scripts are meant as enhancements to the system and are not requirements for the system to work efficiently.

5.7.1 EduLette

EduLette is a C program that allow users to browse through a database randomly. The CGI script takes two arguments. The first is the name of the database and the second is a *yes* or *no*. If a database has a field named "URL" and the second argument is *yes* then the program will automatically take the user to the location specified in the "URL" field. If the second argument is a *no* or the field URL does not exist then the HTML of the resource is returned to the browser.

5.7.2 Home

The Home script is used to segregate Web browsers. This way separate actions can be taken for each browser. The UNITE team uses it to segregate the UNITE browser from other Web browsers. This is done because we did not want links to appear in the UNITE browser. For example, the links to the search interface are not displayed because the UNITE browser has its own search interface built-in. To segregate the browsers the script uses the user_agent field returned by each Web browser.

5.7.3 Imagemap

The Imagemap script was originally written by Kevin Hughes (kevinh@pulua.hcc.hawaii.edu). Its purpose is to virtually segment images so that users can click on the different segments and follow a separate link for each segments. The program was slightly modified to use the environment variable

"QUERY_STRING" instead of passing the information as arguments to the program.

Chapter 6

Conclusion

This document described several aspects of the design and development of the UNITE system at the University of Kansas. The system provides the ability to browse and search hierarchically indexed resources in a wide range of media types (text, images, multimedia, etc.). The server provides remote access to Science and Mathematics resources by geographically distributed K-12 teachers and students, but it can easily be adapted to work with any hierarchically structured domain. For example, we have recently constructed a similar database of information about area businesses for the Chamber of Commerce for the city of Lawrence, Kansas.

The server software supports mirroring, which helps distribute the client load, and enables the client to try alternative servers when its first choice is unavailable. The growth of the database is supported by the Contributor software which helps manage the introduction of material produced by geographically distributed users into the database. The system has been in use by its target audience for over two years and services thousands of requests per week. The experience gained in implementing the system has demonstrated a number of ways in which providing usable services with the WWW presents unique challenges. As such, it has demonstrated the need for modifications of current methods, the need for new abilities, and the fact that the WWW is still a vital and evolving entity.

One area of new research that is underway concerns the relative benefits of different browsing structures on the user's understanding of the information domain. The browsing structure based on a single indexing dimension (e.g. curriculum) is easy to use but provides a somewhat constrained understanding of the scope of the resource. We have recently implemented the "EduLette" browser that randomly selects resources from a given domain. We plan to refine this random browser so that users become actively involved in identifying the dimensions of the domain they wish to investigate. We anticipate that this targeted random browsing coupled with the existing browsing structures will elicit a more robust understanding of the domain and result in the user constructing more meaningful free text queries.

We are continuing to refine the interface and features of the UNITE system based on user recommendations with the goal of developing a useful system for a wide range of users. This includes accessibility from numerous platforms, improvements to the contributing and review functions, and the ability to easily locate meaningful resources in the rapidly expanding collections on the Internet.

We are also investigating the application of the UNITE platform to other possible research areas. We are beginning to apply this technology to the needs of a small working groups. This will give us the opportunity to investigate how to use WWW and HTML methods to provide effective user interfaces for tools supporting group activites. We are also interested in applying this technology to providing user interfaces for sophisticated information retrieval approaches to database access, and for providing access to new types of information including real-time video.

Bibliography

[1] R. Aust. Designing Network Information Services for Educators. *Machine-Mediated Learning*, 4(2&3), 1994, pp. 251-267.

[2] T. Berners-Lee, R. T. Fielding, H. Frystyk Nielsen, K. Hughes. Hypertext Transfer Protocol - HTTP 1.0. *INTERNET-DRAFT*. March 8, 1995
ftp://ietf.cnri.reston.va.us/internet-drafts/draft-ietf-http-v10-spec-00.txt

[3] T. Berners-Lee, D. Connolly. HyperText Markup Language - 2.0. *INTERNET-DRAFT*. March 29, 1995
ftp://ietf.cnri.reston.va.us/internet-drafts/draft-ietf-html-spec-01.txt.

[4] S. Dorner. The CSO Nameserver: A Description. Technical Report, Computing Services Office, University of Illinois at Urbana-Champaign. July 1989.

[5] K. Hughes. Entering the World-Wide-Web: A Guide to Cyberspace. Enterprise Integration Technologies, May 1994.

[6] WorldWideWeb: Proposal for a HyperText Project CERN, 1989. http://www.w3.org/hypertext/WWW/-Proposal.html.

[7] D. Comer, D. Stevens. Internetworking with TCP/IP Vol. III: Client-Server Programming and Applications BSD Socket Version. Prentice Hall. Englewood Cliffs, New Jersey. 1993.

[8] C.J. Date. An Introduction to Database Systems Vol. I. Addison-Wesley Publishing Company. Fifth Edition. Reading, Massachusetts. 1990.

[9] B. Sterling. Short history of the Internet. The Magazine of Fantasy and Science Fiction, February 1993.

[10] J. December. New Spiders Roam the Web. *Computer-Mediated Communications Magazine*, 1 (5), September, 1994.

[11] E. Nebel, L. Masinter. Form-based File Upload in HTML. *INTERNET-DRAFT*. April 19, 1995
ftp://ietf.cnri.reston.va.us/internet-drafts/draft-ietf-html-fileupload-02.txt.

[12] C. Deniau, M. Swink, et al. The UNITE System: Distributed Delivery and Contribution of Multimedia Objects Over the Internet. Inet-95. June 27-30, 1995. http://www.tisl.ukans.edu/Projects/UNITE/.

Appendix A

BNF for the SEARCH method

SEARCH <protocol-type>
(<DBSpec> <SessionSpec> <SearchSpec> <ReturnSpec>)
<protocol-type> := SYMBOL

A.1 DBSpec

<DBSpec> := (<DBNameList>)
<DBNameList> := <DBName>
    | <DBNameList> <DBName>
<DBName> := SYMBOL

A.2 SessionSpec

<SessionSpec> := (<time-pair><cost-pair><distance-pair>)
<time-pair> := /* *empty* */
    | :maxTime <default-int>
<cost-pair> := /* *empty* */
    | :maxCost <default-int>
<distance-pair> := /* *empty* */
    | :maxDistance <default-int>
<default-int> := :default
    | INTEGER

A.3 SearchSpec

```
<SearchSpec> := <search-expr>
<search-expr> := ( <binary-pred-expr> )
     | ( <unary-operator> <search-expr> )
     | ( <mult-operator> <search-expr-list> )
<search-expr-list> := <search-expr>
     | <search-expr-list> <search-expr>
<binary-pred-expr> := <numeric-binary-pred> <field-name>
  <number>
     | <string-binary-pred> <string-pred-flags>
  <field-name> STRING
<numeric-binary-pred> := =
     | <
     | >
     | #
     | >
     | <
<string-binary-pred> := contains
<unary-operator> := andnot
<mult-operator> := and
     | or
<string-pred-flags> := /* empty */
     | ( <string-flag-list> )
<string-flag-list> := <string-flag>
     | <string-flag-list> <string-flag>
<string-flag> := :case-sensitive
     | :case-insensitive
     | :full-field
     | :partial-field
<field-name> := STRING
<number> := REAL
     | INTEGER
```

A.4 ReturnSpec

```
<ReturnSpec> := ( <max_num_full> <max_num_sum>
  <max_size_full> <max_size_sum> <sort_method>
  <show_full_method> <show_sum_method> )
```

```
<max_num_full>  := /* empty */
      | :maxNumFullRecords    <default-int>
<max_num_sum>   := /* empty */
      | :maxNumSummaryRecords <default-int>
<max_size_full> := /* empty */
      | :maxSizeFullSet       <default-int>
<max_size_sum>  := /* empty */
      | :maxSizeSummarySet    <default-int>
<sort_method>   := /* empty */
      | :sort-by  <sort-expression>
<show_full_method> := /* empty */
      | :show-full <show-expression>
<show_sum_method>  := /* empty */
      | :show-summary <show-expression>
<sort-expression> := ( <sort-op> <field-name> )
      | ( <sort-op> <field-name> <sort-expression> )
<sort-op> :=     :chron
      | :revchron
      | :alpha
      | :revalpha
      | :ascending
      | :descending
      | :inorder
      | :revorder
<show-expression> := :all
      | ( <field-list> )
<field-list> := <field-name>
      | <field-list> <field-name>
```

A.5 Lex regular expressions for above types

```
INTEGER :=  -?[0-9]+
REAL    := -?([0-9]+)
        | -?(([0-9]*.[0-9]+)([eE][-+]?[0-9]+)?)

SYMBOL  := [^\r\n\t()"]+
```

NOTE: a symbol cannot start with a number or minus sign, or it will be interpreted as real or integer

STRING := "[^"]*"

NOTE: a backslash preceiding a double quote (") can exist within a string

string flag definitions:

:case–sensitive => case sensitive
:case–insensitive => case insensitive
:full–field => matches the entire field <exact match>
:partial–field => matches part of the field <substring match>

NOTE: For the specific case of UNITE, the following is true:
The <field–name>, <DBName> types must both be valid as per the available db configuration files.
In the <binary–pred–expr>, the field–names and their values (<number> or STRING) must be of compatable types.

Appendix B

Global Configuration File Example

```
#the very top level directory for the server,
#EVERYTHING that the server requires is under this directory
TopLevelDir: /users/unite/Released

#location of authorization information
AuthDir:        /Authorization
FileSetsDir:    /FileSets
UserGroupsDir:  /UserGroups

#location for contributions
ContributionDir:     /Contributions
OldContributionDir:  /Contributions.old
ReviewDir:           /Review

#mirroring directories
MirrorDir:            /FileUpdates
MirrorNewFiles:       /newfiles
MirrorRemovedFiles:   /removedfiles
MirrorUpdatedFiles:   /updatedfiles
MirrorServers:        /.serverlist
MirrorLogs:           /logs

#location for server connect logs
ConnectLogs:  /logs
DeleteDir:    /deleted
PutDir:       /Contributions
```

```
#location for all resources
ResourceDir:     /resources
ScriptDir:       /cgi-bin
DefaultScript:   /home
GenericDir:      /Generic
IconDir:         /Generic/icons
BrowserDir:      /Generic/Browser
AuxDir:          /Generic/auxresources
HomeHTML:        /Generic/Explorer-Home.html
SearchHelp:      /Generic/auxresources/wExpHelp.html
DeleteMessageFile: /Generic/delete.html
PutMessageFile:  /Generic/post.html
DatabaseList:    /.dblist

#location for search engine databases
DatabaseCSO:     /db/cso
DatabaseWAIS:    /db/wais
DatabasePG:      /db/postgress

#filename for each database's configuration file
DbConfigFile: /.dbconfig

UserDir: /.public_html

#Default page (realtive to ResourceDir)
DefaultPage:     wwwhome.html
DeletePermission: CedricDeniau

#the following are defaults for the unite.server
serverPort: 80
serverHost: vader.tisl.ukans.edu

defaultUserGroup: Anonymous

#databaseLocation: dbName         engine   dbHost               dbPort
databaseLocation: UNITEResource  CSO      vader.tisl.ukans.edu 3801
databaseLocation: webed          CSO      vader.tisl.ukans.edu 3802
```

Appendix C

Database Configuration File

```
// UNITEResource.defn – Specifications for UNITE/Explorer Resources

TABLE "ResourceType_Table" {
  "Courseware"              "Courseware_icon.GIF";
  "Lesson Plan"             "LessonPlan_icon.GIF";
  "Lab Activity"            "LabActivity_icon.GIF";
  "Instructional Aid"       "InstructAid_icon.GIF";
  "Instructional Module"    "InstructMod_icon.GIF";
  "Field Trip"              "FieldTrip_icon.GIF";
  "Student Created Material"    "StuCreatMater_icon.GIF";
  "Parent Material"         "ParentMatrial_icon.GIF";
  "Practical Article"       "PractArticle_icon.GIF";
  "Research Article"        "ResearchArtic_icon.GIF";
  "Textbook"                "Textbook_icon.GIF"; }

TABLE "PhysicalMedia_Table" {
  "Audio CD"                "AudioCD_icon.GIF";
  "CD-Interactive"          "CDInteract_icon.GIF";
  "CD-ROM"                  "CDROM_icon.GIF";
  "Downloadable File"       "Downloadable_icon.GIF";
  "Filmstrip"               "FilmStrip_icon.GIF";
  "Floppy Disk 3.5"         "Floppy_icon.GIF";
  "Overhead Transparencies"    "OverheadTransp_icon.GIF";
  "Paper Based Media"       "Paper_Based_icon.GIF";
  "VHS Video Tape"          "VideoTape_icon.GIF";
  "TABLE_DEFAULT"           "Default_icon.GIF"; }

TABLE "FileFormat_Table" {
```

```
 "application/pdf"            "AcrobatDoc˙icon.GIF";
 "document/x-clarisworks"     "ClarisWorksDoc˙icon.GIF";
 "document/x-explorer"        "ExplorerDoc˙icon.GIF";
 "document/x-opendoc"         "OpenDoc˙icon.GIF";
 "document/x-postscript"      "Postscript˙icon.GIF";
 "document/x-replica"         "ReplicaDoc˙icon.GIF";
 "document/x-quicktime"       "QuickTimeDoc˙icon.GIF";
 "image/gif"                  "GifImage˙icon.GIF";
 "image/jpg"                  "JpgImage˙icon.GIF";
 "text/html"                  "HTMLtext˙icon.GIF";
 "text/plain"                 "TeachTextDoc˙icon.GIF"; }

TABLE "FileFormat˙Header˙Table" {
 "application/pdf"            "Acrobat";
 "document/x-clarisworks"     "ClarisWorks";
 "document/x-explorer"        "Explorer";
 "document/x-opendoc"         "OpenDoc";
 "document/x-postscript"      "Postscript";
 "document/x-replica"         "Replica";
 "document/x-quicktime"       "QuickTime";
 "image/gif"                  "Gif Image";
 "image/jpg"                  "JPEG Image";
 "text/html"                  "Text HTML";
 "text/plain"                 "Text Plain"; }

TABLE "Fields˙Table" {
 "IDNumber"             "ID Number";
 "Title"                "Title";
 "ResourceType"         "Resource Type";
 "Grades"               "Grades";
 "Description"          "Description";
 "Curriculum"           "Curriculum";
 "ProcessSkills"        "Process Skills";
 "Reviewers"            "Reviewers";
 "Author"               "Author";
 "Series"               "Series";
 "Publisher"            "Publisher";
 "PhysicalMedia"        "Physical Media";
 "FileSizeInKBytes"     "File Size in KBytes";
 "FileFormat"           "File Format";
 "FileName"             "File Name";
 "FileDescription"      "File Description"; }
```

```
ENUMERATION "GradeT" {
 "Pre-K"  "K"
 "1"  "2"  "3"  "4"   "5"   "6"
 "7"  "8"  "9"  "10"  "11"  "12"
 "Undergraduate"  "Graduate"  "Adult" }

ENUMERATION "FileFormatT" {
 "application" {
           "pdf" }
 "document" {
           "x-clarisworks" "x-explorer" "x-postscript" "x-replica"
           "x-quicktime" "x-opendoc" }
 "image" {
           "gif" "jpeg" }
 "text" {
           "html" "plain" } }

ENUMERATION "FileEncodingT" {
 "NONE" "HQX" "MACBINARY" "PDF" }

ENUMERATION "FileSetT" {
 "KU_FileSet" }


ENUMERATION "PhysicalMediaT" {
 "Audio CD"
 "CD-Interactive"
 "CD-ROM"
 "Downloadable File"
 "Filmstrip"
 "Floppy Disk 3.5"
 "Overhead Transparencies"
 "Paper Based Media"
 "VHS Video Tape" }

ENUMERATION "ResourceTypeT" {
 "Courseware"
 "Field Trip"
 "Instructional Aid"
 "Instructional Module"
```

```
  "Lab Activity"
  "Lesson Plan"
  "Parent Material"
  "Practical Article"
  "Research Article"
  "Student Created Material"
  "Textbook" }

ENUMERATION "StateT" {
  "Hawaii" "Idaho" "Illinois" "Iowa" "Kansas" "Kentucky"
  "Washington" "West Virginia" "Wyoming" "Alberta"
  "British Columbia" "Manitoba" "Newfoundland" }

ENUMERATION "CurriculumT" {
  "Mathematics" {
    "General Mathematics"
    "Problem Solving and Reasoning" {
      "General Problem Solving and Reasoning"
      "Logical Reasoning"
      "Generalize" }
    "Mathematical Tools" {
      "General Mathematical Tools"
      "Calculators" {
        "General Calculators"
        "Computation" }
      "Computers"
      "Mathematical Instruments" }
    "Whole Numbers and Numeration" }

  "Natural Science" {
    "General Natural Science"
    "Life Science" {
      "General Life Science"
      "Cells" {
        "General Cells"
        "Cell Growth"
        "Cell Reproduction" }
      "Living Things" }
    "Physical Science"
    "Earth Science"
    "Common Themes" } }
```

```
ENUMERATION "ProcessSkillsT" {
 "Mathematics Process" {
  "Calculators and Computers"
  "Communication"
  "Computation"
  "Conceptualization"
  "Connections"
  "Estimation"
  "Mental Arithmetic"
  "Problem Solving"
  "Reasoning" }

 "Natural Science Process" {
  "Gather Data"
  "Analysis Synthesis Evaluation"
  "Communicate Ideas"
  "Technology"
  "Values and Attitudes" } }

RECORD "FileDescriptionT" {
 "integer"        "One"        "NotSearchable"  "FileSizeInKBytes";
 "FileFormatT"    "One"        "NotSearchable"  "FileFormat";
 "string"         "One"        "KeywordValue"   "FileName";
 "FileEncodingT"  "One"        "NotSearchable"  "FileEncoding";
 "FileSetT"       "OneOrMore"  "NotSearchable"  "FileSet"; }

RECORD "FileDescriptionsT" {
 "FileDescriptionT" "OneOrMore" "NotSearchable" "FileDescription"; }

DATABASE_OBJECT UNITEResource 1994092001 {
 "string"             "One"        "KeywordValue"   "Title";
 "uid"                "One"        "NotSearchable"  "IDNumber";
 "FileDescriptionsT""ZeroOrOne"    "NotSearchable"  "FileDescriptions";
 "ResourceTypeT"      "One"        "KeywordValue"   "ResourceType";
 "PhysicalMediaT"     "One"        "KeywordValue"   "PhysicalMedia";
 "GradeT"             "OneOrMore"  "KeywordValue"   "Grades";
 "string"             "ZeroOrOne"  "KeywordValue"   "Series";
 "FreeText"           "One"        "KeywordValue"   "Availability";
 "FreeText"           "One"        "KeywordValue"   "Description";
 "CurriculumT"        "OneOrMore"  "KeywordValue"   "Curriculum";
 "ProcessSkillsT"     "OneOrMore"  "KeywordValue"   "ProcessSkills";
 "string"             "OneOrMore"  "KeywordValue"   "Author";
```

```
"string"        "ZeroOrOne"    "KeywordValue"     "Publisher";
"string"          "OneOrMore"    "NotSearchable"    "Reviewers"; }
```

Appendix D

Database Configuration File Syntax

dbconfig := table_list enum_list record_list data_object

table_list := /* *empty* */
| table
| table_list table

enum_list := /* *empty* */
| enumeration
| enum_list enumeration

record_list:= /* *empty* */
| record
| record_list record

data_object := DATABASEOBJECT SYMBOL NUMBER '–' string_semicolon '˝'

enumeration := ENUMERATION STRING '–' hier_lists '˝'

record := RECORD STRING '–' string_semicolon '˝'

table := TABLE STRING '–' string_semicolon '˝'

string_semicolon := string_list SEMICOLON
| string_semicolon string_list SEMICOLON

hier_lists := hier_list

```
           | hier_lists hier_list

string_list := STRING
            | string_list STRING

hier_list := /* empty */
          | string_list
          | string_list '-' hier_list '"'

SYMBOL := [a-zA-Z]+[a-zA-Z0-9_()-&.'/%]*

STRING := "[^"]*

NUMBER := [0-9]+

SEMICOLON := [;]
```

Appendix E

Example DBML File

<UNITEResource>

<Version>1995012601</Version>

<IDNumber>0</IDNumber>

<Title>"Kansas City Zoo (Australia)"</Title>

<ResourceType>"Field Trip"</ResourceType>

<Description>"The Kansas City Zoological Gardens (Swope Park Zoo) offersex-
hibits pertaining to tropical life, winged waterfowl, sea lions, and
primates, as we'll as the new Australian exhibit. Prior to touring theoutback, a 5–minute
movie introduces the visitors to Australia. Within
the exhibit, which is designed as a suspended–disbelief environment, can
be found not only free–roaming kangaroos, emus, and sheep but also two
bird aviaries as well as fenced dingoes and camels. One portion of this
exhibit has been designed as a sheep–shearing station which includes
replicas of a small, outback home and the station office. Visitors will also
find a special outback gift shop. A 15–minute train ride ($1.50) gives an
overview of the entire zoo. Tickets for the zoo are $5.00 for those over the
age of 12 and free to those under 12. Parking costs $2.00. Food and drinks
are readily available throughout the zoo. Maps included are of the Australian
exhibit, the Winged Waterway exhibit, and the entire zoo. Also included is
additional information pertaining to upcoming exhibits."¡/Description¿
<Grades>"6" "7" "8"</Grades>

<Curriculum>

"Natural Science/Life Science/Living Things/Animals/General Animals"
"Natural Science/Life Science/Living Things/Animals/Life Cycles of Animals"
"Natural Science/Life Science/Living Things/Animals/Structure-Function Animals"
"Natural Science/Life Science/Heredity/Parent-Offspring"
"Natural Science/Life Science/Evolution/Origin and Development"
"Natural Science/Life Science/Evolution/Adaptations to the Environment"
"Natural Science/Earth Science/Geosphere/Geology/Tectonics"
</Curriculum>

<ProcessSkills>
"Natural Science Process/Gather Data/Observe"
"Natural Science Process/Gather Data/Measure"
"Natural Science Process/Gather Data/Record"
"Natural Science Process/Gather Data/Research"
"Natural Science Process/Communicate Ideas/Define Ideas"
"Natural Science Process/Communicate Ideas/Describe"
"Natural Science Process/Communicate Ideas/Classify"
"Natural Science Process/Values and Attitides/Teamwork"
</ProcessSkills>

<PhysicalMedia>"Downloadable File"</PhysicalMedia>

<Author>"Nancy Markwell"</Author>

<Availability>"Explorer, ECnet"</Availability>

<Reviewers>"KU UNITE"</Reviewers>

<FileDescriptions>

<FileDescription>

<FileFormat>"document/x-ClarisWorks"</FileFormat>
<FileSizeInKBytes>194</FileSizeInKBytes>
<FileEncoding>"HQX"</FileEncoding>
<FileName>"KC˙Zoo˙Australia˙Exhibit948869.hqx"</FileName>
<FileSet>"KU˙FileSet"</FileSet>

</FileDescription>

<FileDescription>

<FileFormat>"document/x-Acrobat"</FileFormat>
<FileSizeInKBytes>62</FileSizeInKBytes>
<FileEncoding>"PDF"</FileEncoding>
<FileName>"KCZooAus.pdf949004.pdf"</FileName>
<FileSet>"KU˙FileSet"</FileSet>

</FileDescription>

</FileDescriptions>

</UNITEResource>

Appendix F

Installation

The first step to installing this software is downloading it.

1) ftp.tisl.ukans.edu /pub/UNITE/unite_tools.tar.gz

2) uncompress and tar the file.

```
gunzip unite_tools.tar.gz
tar xvf unite_tools.tar
```

This will create a directory called unite_tools_0.9.

3) Edit the global.h file in unite_tools_0.9/src
you should only need to change HOME_DIR unless you decide to change all the directory and file names.

Here is the part of the global.h file you need to change:

```
#define HOME_DIR "/users/unite/unite˙tools˙0.9"
#define CONFIG_DIR "/config"
#define DIRCONFIG "/unite.config"
#define HTMLPRINT "/htmlPrint.config"
#define UNITEPRINT "/unitePrint.config"
#define BUILDCONFIG "/buildPrint.config"
#define CLIENTCONFIG "/clientPrint.config"
#define MIMEFILE "/mime.types"
```

4) Edit the global configuration file in unite_tools_0.9/conf
the name of it should be the same name you used in the global.h

file as the DIRCONFIG variable.

For this file you need to specify the top level directory in TopLevelDir. This needs to be a full path. The rest can be left the same except the databaseLocation. This is where you specify which database is listening on which port and on which host.

5) Edit the Makefile in the src directory. Again all you should have to change is the TOP variable, you may want to change CC and CFLAGS to reflect your C compiler.

6) If this is the first time you are installing this software then you type

```
make extra
```

This will create directories in the unite_tools_0.9 directory.

If you just want to remake everything (i.e. maybe after a make clean) then all you need to do is

```
make all
```

Both of these make commands will compile all the code and install the binaries generated in the unite_tools_0.9/bin directory and some libraries will be created in the unite_tools_0.9/lib directory.

7) If this is the first time you are installing this software the you will need to create the directories you specified in the global configuration file. To do this run:

```
buildDir
```

8) You will need to add a database. To do that run:

```
addDB <dbname>
```

9) You need to edit the CSO configuration file. It is in the src/CSO/src/Configs/server_config . Here you need to change the RootDir variable at the beginning of the file. Also change the Owner, CC, Group, Database, Admin, Passw, MailDomain,

FallBackAddr, FallBackPort, and Host variables. Also look around and change any other variables you believe need changes.

10) Then configure and build CSO. Go to unite_tools_0.9/src/CSO/src and run:

```
Configure server_config
make all
CSOinstall
```

11) You now need to add into your inetd configuration file and in your services file an entry. This is what I entered:

```
uCSO1  stream  tcp   nowait  unite   /users/unite/bin.sun4/qi
       qi -d -Database=/users/unite/Released/db/cso/UNITEResource/csodb
```

you need to make sure that the port and host name are the same you put in the global configuration file.

12) To build a CSO database all you need to do is run:

```
CSObuild <dbname>
```

this will look for the database configuration file and build the cso index file. Now you run:

```
build_db
```

making sure that the directories in that script file are correct. (i.e. BINDIR and DBDIR)

13) To test if CSO is running correctly, you can telnet to the port on the host it is running and type fields. This should give a list of the fields in your database. Type quit to exit.

14) Now all you need is the server. You can run the server in standalone or inetd. To run in standalone run:

```
uServer -s
```

else

```
uServer
```

but then you have to edit your inetd configuration file and add an entry. What I have is

```
http   stream  tcp    nowait  unite
  /users/unite/Released/unite-src/UniteServer/uServer.sun
  uServer.sun -c unite.config
```

Once the installation is done, all the executable files are in the bin directory. These are: unite, CSOBuild, addDB, browser, buildDir, and uServer.

CSOBuild is the program to build the two files CSO need to run its indexer. To run this program simply type:

```
CSOBuild <db_name>
```

unite it the program used to contribute a file. To run this program type:

```
unite <filename>
```

This program will automatically look in the "ContribDir" for the filename.

addDB is a program used to add a database. It creates the directories needed and adds the name to the "DatabaseList" file. To run it:

```
addDB <db_name> <db_type> <db_config>
```

db_type is cso at this time unless you add other search engines and db_config is the name of the database configuration file.

browser is the program used to create the browsing views. To run it:

```
browser <db_name> <enumeration_name> <config_file>
```

the <enumeration_name> is the name of the field to be used to build the views on. <config_file> is the name of the browser configuration file.
A file named trailer should be in the current directory. This file is appended to all the files created by the program.

buildDir is used to build all the directory structure defined in the global configuration file. To run it:

buildDir

this program uses the global.h file to find out which global configuration file to use.

uServer is the server program. This program is usually ran as inetd or standalone. For inetd:

uServer −c <config_file>

for standalone:

uServer −s −c <config_file>

the <config_file> is the name of the global configuration file. If it is not specified then the name is taken from the global.h file. If it is given then it is relative to the CONFIG_DIR in the global.h file.