INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

CINDI: THE VIRTUAL LIBRARY GRAPHICAL USER INTERFACE

Youquan Zhou

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

April 1997

Abstract

CINDI: The Virtual Library Graphical User Interface

Youquan Zhou

The search for information on the Internet is often not an easy job for many Internet users. Because of the lack of a standard indexing scheme and an informative query interface, a Net search could have thousands of hits returned and the number of search misses is high. This thesis is part of the work to develop an Indexing and Searching System for the Internet called the CINDI (Concordia INdexing and DIscovery) system. It is aimed at providing a standard index scheme called Semantic-Header and informative query interface for users and providers of resources published on the Internet.

This thesis presents the architectural design of the CINDI system, the design and implementation of the client part of the expert system, and the design and implementation of the graphical user interface (GUI) for the CINDI system. As the interface is an important part for the quality of a software, it has been carefully designed and implemented in an effort to be easy to use, user friendly, and consistent. The interface has been implemented under the UNIX system using the Motif toolkit and C programming language.

The heart of the indexing system is the record called Semantic-Header that is kept for each item being indexed. The grammar of the Semantic-Header and that of the search query are also discussed in this thesis. An interface between the CINDI system and Netscape Navigator is also implemented.

Finally, some directions for future work related to CINDI are described.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor, Dr. Desai for active discussions and valuable suggestions. This work would not have been possible without his conceptional design of the semantic header, enthusiastic support and consistent guidance.

I would like to sincere thanks to Dr. Shinghal, R. and Dr. P.G. Chander. for their guidance in the design and implementation of the expert system.

I would also like to thank Rajabien Shayan Nader who, as part of the CINDI project, suggested many ideas to me during the valuable discussions between us. His amicable personality also made the project more enjoyable.

I would like to express my thanks to Yves Saintillan. In the early stages of the project, we had many discussions which helped to shape the specification and architecture of the system.

Finally, I would like to dedicate this work to my wife, Yijuan Liu, my daughters, Baiyin and Emily, for their support, great patience, and encouragement.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview of the Virtual Library

Virtual Library

> A virtual library [LS93] can be defined as the concept of remote access to the contents and services of libraries and other information resources, combining an on–site collection of current and heavily used materials in both print and electronic form, with an electronic network which provides access to, and delivery from, external worldwide library and commercial information and knowledge sources. In essence, the user is provided the effect of a library which is a synergy created by bringing together technologically the resources of many many libraries and information services.

An increasing number of research institutes, universities and business organizations are currently providing their reports, articles, catalogs and other information resources on the Internet in general and the World Wide Web, which can be considered as a virtual library.

Hypertext

> Hypertext [Nie95] provides a non-sequential method for reading a document displayed on a computer screen. Instead of reading a document in sequence from beginning to end, the reader can jump to topics by selecting a highlighted word or phase (called an **anchor**). This activates a link to

another place in the same document, or to another document altogether. The resulting matrix of links within and among documents is termed a **Web**.

Multimedia

In a computer system or a computer network, the term " multimedia" signifies presentation of information by means of more than one medium at a time (including, for example, graphics, video clips, and animations sound in addition to text).

Hyper-media

When a hypertext system incorporates multimedia resources, it is called a hyper-media. Hyper-media extends hypertext in two ways. First, it incorporates multimedia documents into hypertext documents. Second, it allows graphic, audio, and video elements — rather than just text elements — to become links to other documents or multimedia elements.

Hypertext Markup Language (HTML)

HTML [Web96] is a variant of the Standard Generalised Markup Language (SGML). It is a language used to describe hypertext documents. It is a standard for formatting documents so that, when accessed by Mosaic or any other HTML-capable editor, they look good on-screen. It effectively defines what Web documents may look like and how Web resources can present themselves.

Universal Resource Locators (URL)

A URL is a statement, written according to strict rules of syntax, that specifies where a Web resource can be found.A URL is the address of any multimedia resource on the WWW. A URL can point to an HTML file, a GIF image, an MPEG movie, an AU sound file, and so on.

1.1.1 The Internet

The Internet can be described as the federation of interconnected digital networks (LANs and WANs) that communicate through the TCP/IP protocol [Car93]. The Internet is a vast telecommunication network of networks involving millions of computers worldwide.

The Internet began as a federally funded research network [Lan93]. In 1969, the Department of Defence's Advanced Research Projects Agency (DARPA) funded the establishment of a packet switching network, the ARPANET. In its beginning as the ARPANET, the Internet was conceived primarily as a means of remote login and experimentation with data communication protocols. However, the predominant usage quickly became electronic mail (E–Mail) in order to support collaboration. This trend continues into the present incarnation of the Internet, but with increasingly diverse support for collaborative data-sharing. E–mail has been supplemented by a variety of wide-area filing, information retrieval, publishing, and library access systems. At present, the Internet provides access to hundreds of gigabytes of software, documents, sounds, images, and other file system data; library catalog and user directory data; weather, geography, telemetry, and other physical science data; and many other types of information. This information can be either private or public. Private information is available for authorised users, who have a particular status or have paid a fee. Public information is available to any user of the Internet without any charge. In 1993 the Internet was chosen The Landmark Technology of the Year by InfoWorld. And the Internet Society has declared 1993 as "The Year the Internet Happened" [Mas94]. Its growth continues at an exponential rate or higher. Whereas Internet users historically had been mostly researchers and university students, there are now a larger number of commercial than educational sites. Protocols and services for this network enable new forms of information sharing and communication.

Many people, however, found Internet services difficult to use, primarily because of the awkward user interface. Until about 1990, the only way to share information on the Internet was through FTP, E–mail, and Usenet. Later, the University of Minnesota came up with a novel system named **Gopher**. This is a hierarchical menu system in which people use Gopher client programs to connect to Gopher servers

running at many different sites throughout the Internet. It allowed organizations to have more of a "presence" on the network than was previously possible. They could make information available about themselves and include "pointers" to other related Gopher servers. Another feature of Gopher was that it provided access to other Internet services such as FTP, Telnet, and index searching. Users no longer had to deal with the awkward command line FTP client program. Transferring data became easier than ever before.

1.1.2 The World-Wide Web (WWW)

Without question the most popular part of cyberspace at this time is the World-Wide Web [BCL+94, Ber96], whose development began at CERN, the European Particle Physics Laboratory in Geneva, as an information service on the Internet. Its initial goal was to allow physists to share information. WWW is a concept, not a program or a system, not even a specific protocol. It provides the technology needed to offer a manageable, attractive interface to the Internet's vast sea of resources. WWW brings together two distinct ideas—that of hyper-media documents for non-sequential access to multimedia information and that of GUI-based client programs capable of scouring a world–wide network in search of information. Capable of linking dissimilar computers worldwide into a hyper-media network, WWW permits any Internet user to provide hypertext–based information and make it available to the WWW community.

The Web is a mix of the Internet's client-server handshaking and communication protocol, the HyperText Transfer Protocol (HTTP) [GF95], together with a protocol for document formatting, the HyperText Markup Language (HTML) [Web96]. The World-Wide Web is characterized by the following properties:

- The address system: Use of a Unique Resource Locator (URL) as a common addressing syntax.
- A Network protocol: Use of the HTTP protocol to transfer information and requests between computers. This is the native WWW server's protocol. The World-Wide Web browser can also get information from FTP Archives or WAIS servers [Koc95].

- A markup language (HTML): Every World-Wide Web client browser is required to understand HTML.

The flashy NCSA Mosaic graphical WWW browser captured a lot of attention in 1993 and 1994. Since then, Netscape Navigator, a commercial version WWW browser, has become the most popular one.

1.2 The CINDI System

1.2.1 Introduction

The vast amount of information available today on the Internet has a great potential to improve the quality of life. Currently available information sources include a growing number of digital libraries.

To make effective use of the wealth of information on the Internet, users need ways to locate pertinent information. In the past few years, many such search systems have been developed and some of them have gained wide acceptance on the Internet. These include AltaVista, Yahoo, WebCrawler, Lycos, Harvest, EINet Galaxy, Veronica etc. Some of these are manually generated indices while others are generated by robots [Kosb]. Some of these robot–based systems also allow manual index entry. Moreover, some of these are specialized for the WWW, while others are designed to locate files on Anonymous FTP sites. The search interface of these systems provides users little flexibility and the search results are not always pertinent. As an example, a test was conducted in June 1995 on some of the existing search systems [Des95d]. Table 1 summarizes the test results of some of the existing search systems using the keywords **Bipin (AND) Desai.**

There were 24 URLs containing Bipin (and) Desai on the WWW at the time the test was taken. The terminology used in the table is as follows:

- Search System: The search system used to conduct the test.
- Number of Hits: The number of document containing *Bipin (and) Desai* found.
- Number of Duplicates: The number of times the same document was retrieved more than once.

- Number of Mis-hits: The number of irrelevant documents found.
- Number of Missed: The number of documents not found even through they exist on the WWW.

Search System	Number of Hits	Number of Duplicates	Number of Mis-hits	Number of Items missed
Aliweb	none	-	-	24
DA-CLOD	none	-	-	24
EINet	6	0	4	18
GNA Meta Lib.	none	-	-	24
Harvest	none	-	-	24
InfoSeek	7	0	0	17
Lycos	231	2	222	17
Nikos	none	-	-	24
RBSE	8	-	8	24
W3 Catalog	none	-	-	24
WebCrawler	7	3	0	20
WWWW	2	0	0	22
Yahoo	none	-	-	24

Table 1: Sample Search Statistics for searching on Bipin (AND) Desai

The table shows that none of these systems was always successful in retrieving the documents sought. Even systems such as Lycos [Mau95], which was known to have indexed over four million documents at the time this test was conducted had only partial success in locating all relevant documents. The reason for these unexpected results is that many of these systems attempt to match the specified search terms without regard for the context in which the words appear in the target information resource.

It should be noted that some of these systems are no longer accessible and newer systems have since been introduced. If these tests were conducted today, the target items would most likely have been found by using *web-robots*. However, they would be hard to locate among the total number of entries found, since the number of duplicates and the number of mis–hits would be in the hundreds or even thousands.

Some of the drawbacks in the existing search systems are listed below:

- Heterogeneous: Different virtual libraries may describe the same document in different ways.
- Not easy to use for new users: A new user may navigate a long time through the menus if he does not know how to express his query perfectly.
- Documents are usually referenced by words and random phrases instead of by title, author, subject etc.
- Many systems do not have exact information about the content of a document or file. A user often has to retrieve the pertinent source document/file and browse it before deciding if a specific file is pertinent.
- It is often difficult to find relevant information. To address this problem, many people have built indexing systems (such as WWW robots), but most of these systems place significant unnecessary load on information servers and network links, because they use expensive object retrieval protocols to gather indexing information, and they do not coordinate information gathering among each other. Each indexer gathers all of the information it needs, without trying to share overlapping information with other indexers.
- Most systems offer little support for customising how they handle different information formats and index/search schemes.

Creating indices based on search robots, worms, spiders, or crawlers have the following disadvantages [DS96]:

- Repeated attempts by robots to find new resources increases the traffic on the network. The number of these robots is increasing, and system administrators may disallow visits by robots.

- A robot–based approach will become difficult to justify if the network switches to a fee–for–use mode of operation [Bro95].
- The type of data gathered by a robot is not useful because it is too simple to support discovery. This is the case in spite of the fact that such indices, for the lack of a better tool, have been useful to date.
- Natural language understanding systems are not advanced enough to extract meaning from a resource.
- It is more difficult to generate automatic identification of features and concepts from resources such as program code, digital images and complex systems.
- Automatic indexing tends towards the simplistic approach and supporting discovery or even finding required information will become increasingly non–manageable. This trend for existing systems to exhibit poor selectivity is illustrated above.

1.2.2 CINDI: The Project

The search for informations on the Internet is often not an easy job for many Internet users. Because of the lack of a standard index scheme and an informative query interface, a Net search could have thousands of hits returned and the number of search misses is high. According to the sixth World–Wide Web survey [Too96], the number one problem that Web users face is "speed" (76.55%), The next big problems are "finding known info" (34.09%), and organizing collected information (31.03%). If documents are not catalogued properly, then during a search, one may not be able to retrieve the resource sought even though it exists in the virtual library. Proper cataloging thus becomes essential. Employing professional cataloguers will be too time consuming and costly. A better approach is to provide a system that helps the contributors catalog their own documents.

In the library domain, there are some systems that address the issue of cataloging. CORE [Cro94], MARC [Byr91], MLC [Hor86], and TEL [Gay94] are examples of this approach. These existing and proposed indexing systems range from minimum to full level of bibliographic information. However, such systems are designed for professional cataloguers, and many of the elements included in them are beyond the level

of familiarity of most providers or use of information.

Metadata

Metadata is the information which records the characteristics and relationships of the source data. It helps provide succinct information about the source data which may not be recorded in the source itself due to its nature or an oversight [Des90].

The Metadata Invitational Workshop was held from in March, 1995 in Dublin, OH [Des95b]. The participants were actively involved in various aspects of the Digital or Virtual Library project, primarily in North America. The intent of the meeting was to try to work towards the definition of a minimum common set of elements for a network object. The main objective was to address the problem of cataloging network resources with the adoption, extension, or modification of the current standards and protocols in order to facilitate their discovery and access.

The goals of the workshop were: to achieve a consensus on a set of core data elements for **document–like objects**(DLO); to map these and related elements to accepted standards; and to devise an extension scheme for registering other types of network objects. The elements proposed are subject, title, author, other–agent, publisher, date, identifier, object–type, form, relation, language, source, and coverage. Detailed discussion of these elements can be found in [DS96].

The problem with current automatically generated index databases is their inadequate semantic information. Yet, it is evident that professional cataloging of the ever–growing information resources would be prohibitively expensive. Thus the design of adequate metadata to describe and establish the semantic content of resources and to establish their semantic dependencies on other resources is of utmost importance. This, along with a registering system, would establish a basis for later search and discovery.

Metadata has become an intensely debated issue in the Internet community, and many have come to the conclusion that the concept of a catalogued–based approach would be the most appropriate paradigm. Metadata is seen to provide a template for

describing information about a resource in order to facilitate later discovery.

The Project

The CINDI (Concordia INdexing and DIscovery System) project is a system proposed by Desai et al [DS94]. The objective of the project is to build a system that enables any resource provider to catalog his/her own resource and any user to search for hypermedia documents using a typical search items such as Author, Title, Subject etc. The system will offer a bibliographic database that provides information about documents available on the Internet. A standardized index scheme will be used to ensure homogeneity of the syntax and semantics of such an index. These index entries are stored in a database system (Semantic Header Database System). In addition the Catalog database will provide help when the user catalogs his/her index. The system is based on a set of nodes connected to the Internet, each node contains a graphical user interface (based on an expert system). The proposed system is characterized by the following:

- A standardized metadata to describe each information resource – the Semantic–Header [Des95c]
- A Semantic–Header registering system to register the Semantic–Header into the Semantic Header Database System.
- A search system that allow query entry for information discovery.
- An annotation system.
- A Semantic Header Database that stores the Semantic–Headers.
- A Catalog Database that stores information about subjects classified using a standard cataloging scheme.

Using a set of graphical user interfaces, a user can register/update an index entry, make annotations to any index entry, and execute search. Furthermore, a user may select one of the entry from a search result to view the complete metadata. The system will increase the metadata access counter by one. The user can also browse the source resource using Netscape. In this case, the system will increase the associate resource access counter by one to keep record of the total number of times the

associate resource has been accessed.

There are many advantages of the proposed system over existing systems:

- The Semantic-Header allows the indexation of documents accessible by any protocol.
- The Semantic-Header includes annotations of reviewers and other users thus offering the possibility of a more informed decision as to the pertinence of the source resource.
- The Semantic-Header syntax offers a way to register standardised keywords chosen by the provider of the resource. These make searching uniform. The existing systems often hack terms from title and/or content resulting in unpredictable results.
- The Semantic-Header is designed to be part of each resource. The HTML/SGML [Nie95, Gol90] based syntax allows its display by appropriate browsers.
- The Semantic-Header has an "abstract" element which provides a better idea of the resource than an excerpt prepared by systems such as Lycos.
- The distributed [Des90] nature of the Semantic-Header database provides for scalability.
- In existing indexing systems, one of the limitations is the low number of indexed documents. This is illustrated by the disappointing results for manual index systems such as ALIWEB[Des95c] as illustrated in [Kosa]. The difficulty lies in convincing people to register information regarding their resources. This problem would be solved in the CINDI system by providing an easy–to–use interface to register metadata of resources, thereby assuring the presence of metadata in resources by browsers.
- Since the registration of the Semantic-Header in the database is performed by the provider of the resource, it has the following advantages: cost, accuracy, and efficiency.

1.3 Organization of the Thesis

This thesis describes the architecture of the CINDI system, and the design and implementation of the Graphical User Interfaces, implementation of the client part of the expert system, and implementation the interface between the CINDI System and Netscape browser.

Chapter 2 presents the general idea of the CINDI System. The first section gives an overview of graphical user interface design principles and an overview of X–Motif, which is the toolkit used to implement the interfaces. The client–server concept used in our system is presented in section 2.2. The expert system is discussed in section 2.3. The semantic header database and the catalog database are briefly discussed in section 2.4. Section 2.5 presents the system design the CINDI system and its main components: the registering sub–system, the search sub–system, and the annotation sub–system.

The details of the indexing/registering sub–system are presented in Chapter 3. The index structure for our indexing system, the Semantic–Header, is given in section 3.1. The semantic rules are also discussed in this section. The rules for the expert system and implementation detail of the expert system for the indexing/registering subsystem are presented in section 3.2. In section 3.3, details of the GUI are discussed.

Our search sub–system is presented in Chapter 4. The structure of the search query is described in Section 4.1. In section 4.2, we discuss expert assistance for searching, which incorporate the elementary expertise used by a reference librarian. We also discuss the implementation details of the search sub–system in this section. Section 4.3 is the implementation to view the indexed metadata and connecting to Netscape browser.

The annotation sub–system is discussed in Chapter 5. An overview of this sub–system is given in section 5.1. The implementation details are given in section 5.2.

A brief summary of the CINDI system, the contribution of this thesis to the CINDI system, and some directions for future work are given in Chapter 6.

Chapter 2

System Architecture of the CINDI System

2.1 Graphical User Interface

2.1.1 GUI Design Principle

A user interface is simply the interface between an application and the user of an application. The primary goal of a user interface is to help the user interact with an application simply and naturally. Successful user interface designers keep the user in mind when designing an application.

User interface software is difficult and expensive to implement. Highly interactive interfaces are among the hardest to create, since they must include at least two asynchronous input devices (such as a mouse and keyboard), real–time feedback, multiple windows, and elaborate, dynamic graphics. Most graphical interface are created using toolkits such as X-Motif [JR93], Open Look [Dub96], and UIM/X [Blu96] etc..

Graphical user interfaces (GUIs) [BGBG95, Ped92, Thi90] have been used since the late 1960s. However, it did not start to become popular until 1981, when Xerox introduced the 8100Star workstation. The first truly commercial success of a GUI–based system was the Macintosh. GUIs were soon developed for UNIX workstations by Sun Micro systems and for PCs by Microsoft. By now, most computer users interact through GUI. GUIs are popular, because they make applications easier to learn

and use. By standardizing the way users interact with a program and input information, errors are reduced. As a result, GUI applications not only are favoured by end users, but also save companies both time and money by increasing productivity and ensuring accuracy.

GUI design is a kind of art. A GUI developer must take the responsibility of using interface elements effectively and helping the user be as productive as possible. The designer must take care to keep things as simple as possible for beginner and, at the same time, not restrict the more experienced user. This task is perhaps the most difficult one facing the developer in application design. In order to design an effective graphical user interface for an application, the designer must evaluate both the goals of your particular application and your intended audience. Only with a complete understanding of these issues will the designer be able to determine the best interface to use.

Since errors are a fact of life, the designer have to design the interface to deal with the possibility of error input, and guide the user accordingly in order to reduce error input. The user interface must be easy to use, user friendly, and consistent. The principles of good user interface design are as follows:

- **Adopt the user's perspective:** Good design is rooted in an understanding of the user's work. A well-designed application solves a user's problems, makes his work easier, and offer him new capabilities. The two most effective ways of understanding the user's work are to involve users in the design and to be a user oneself.

- **Give the user control:** Users want and need to be in control of the tools they use to perform their work. The user can be in control when an application is flexible and uses progressive disclosure.

- **Use real–world Metaphors:** A good user interface allows the user to transfer skills from real–world experiences. For example, push buttons, and scales slides. This makes it easier for the user to infer how to use an application.

- **Keep the interface natural:** Each screen object needs to have a distinct

appearance such that the user can easily recognize and quickly understand it. At the same time, the style of the interface needs to graphically unify these elements and ensure a consistent and attractive appearance at any screen resolution.

- **Keep interfaces consistent:** Similar components operate similarly and have similar use. The same action should always have the same result.

- **Communicate application action to the user:** Effective applications let the user know what is happening with the application, but without revealing implementation details. Proper communication between the user and the application increases user satisfaction. There are three types of communication: provide feedback, anticipate errors, and provide warnings.

While designing our application, we follow the above principles. For example, we adopt context-sensitive help and pull-down menu selection to make the user's work easier. We also offer users with flexibility. For example, the user can select an item from a pull-down menu or enter a value through the keyboard. We also try to keep our interface consistent. For example, all the repeatable entry blocks have **Prev** and **Next** buttons and they have the same meaning in all blocks. We keep our user interface to be use friendly by appropriate pop-up messages to interact and inform the user.

2.1.2 Overview of X-Motif

Since X–Motif is the tool used to implement the Graphical User Interfaces for the CINDI system, an overview of Motif is necessary.

2.1.2.1 What is Motif ?

Motif [JR93, HF94]was developed by the Open Software Foundation (OSF), an industry consortium that includes Digital Equipment Corp., Hewlett-Packard, and IBM. When someone refers to Motif, they can be referring to any one of the following:

- A look-and-feel Style Guide for applications, based on IBM's Common User Access (CUA) guidelines.

- A window manager, mwm, to help enforce the Style Guide.

- A User Interface Language (UIL) interpreter, which places much of the user interface code into interpreted files.
- A toolkit (C library) for building Style Guide-compliant applications.

In this thesis, Motif refers to a toolkit. This toolkit is based on the Xt Intrinsics [NO92], which provides an Object–Oriented Framework for creating reusable, configurable user–interface components called widgets, supporting a convenient interface for creating and manipulating X windows, colormaps, events, and other cosmetic attributes of the display. Motif allows the developer to create applications with a graphical user interface that can run on a wide variety of computer platforms, including those from Sun Micro systems, Hewlett-Packard, IBM, Silicon Graphics, DEC, and a slew of clone vendors who support UNIX on the 386/486/586/Pentium architectures.

A widget is a generic abstraction for user interface components in the Motif toolkit. Widgets are objects that operate independently of applications. They know how to draw themselves and how to respond to certain events. Widgets are used for scroll bars, push buttons, dialogs boxes and just about everything else in the toolkit. Most widgets have an associated window on the X display that holds the user interface elements of the widget. This allows Motif programs to create a great many windows on the X server.

Each widget has a C data structure that is dynamically allocated via malloc at creation time. This structure, which is supposed to be treated as an opaque data type, contains the widget's data attributes, called resources, and pointers to the widget's functions, such as callbacks. Each widget is of a certain class, and the class also defines a set of functions and resources that apply to every widget of that class.

Some widgets display graphics on the screen. Others serve as container widgets that group other widgets, which are called child widgets.

Xt and Motif follow the Object-Oriented approach, wherein the application program is completely insulated from the code inside the widgets. There is a whole hierarchy of widget classes. For example, the XmLabel class inherits features of all

ancestor classes. In this case, the label widget inherits from the Core and XmPrimitive class. Other classes, like XmPushButton, inherit in turn from the XmLabel widget class. A programmer has access to functions that create, manage, and destroy widgets, in addition to certain public widget variables known as resources. Normally, though, the developer is not concerned about which widgets inherit from which. What is really needed is a list of a widget's resources, including the resources inherited from ancestor widgets. These resources lists in the official OSF/Motif Programmer's Reference Manual [FB93]. If the application programmer wishes to keep the user from modifying resources, the resource values of a widget may be set at creation time.

The Motif toolkit sits on top of the Xt Intrinsics library, which means that Motif uses many of the Xt features and functions. Motif does, however, have a set of functions of its own, and Motif has its own widget set with its own look and feel. The Xt Intrinsics provide a basic mechanism for many widget sets that do not provide look and feel. For more information about Xt Intrinsics and Motif, the reader may refer to [JR93, HF94, FB93, NO92]

2.1.2.2 Advantages of using Motif

There are a number of advantages in using Motif:

- Motif is one of the major interface standards in the UNIX world.
- Motif was adopted as part of the Common Open Software Environment, or COSE, led by IBM, Hewlett-Packard, Sun, SCO, and UNIX System Laboratories.
- Speeds up programming: Motif allows programmers to build Graphical User Interfaces for their applications with little effort.
- Motif provides 3D effects.
- Motif fits in reasonably well with X standards through the use of window managers and resource files.
- Program interfaces are more consistent across applications.
- People find Motif applications easier to learn and use.

2.1.2.3 Basic Steps of Motif Programming

Motif programs usually follow five basic development steps:

1. Initialize the Xt Intrinsics (which also sets up the connection to the X server) with XtAppInitialize
2. Create the widgets
3. Set up any required callback functions
4. Realize the widget hierarchy with XtRealizeWidget
5. Enter the event-handling loop

Initialize the Xt Intrinsics

The first step in any Motif program is to initialize the Xt Intrinsics.

```
Widget XtAppInitialize(
    XtAppContext* appcontext,
    String app_class_name,
    XrmOptionDesclist xrm_options,
    Cardinal number_xrm_options,
    int *argc,
    String* argv,
    String* fallback_resources,
    ArgList args,
    Cardinal number_args)
```

The application context is set up by XtAppInitialize. The developer uses the application context, appcontext, in the call to XtAppMainLoop. The application-class name specifies a name used for looking up X resource values. For most of the parameters, the developer can safely pass NULL or zero. That's because XtAppInitialize offers many more options than most Motif programs need.

Create the Widgets

There are two primary ways to create Motif widgets. One is to use the Xt-provided functions XtCreateWidget. Another is to use Motif-provided functions, one for each widget type. For menus, dialog boxes, scrolled-text and other combination widgets, it is better to use Motif-provided functions to create them, because there is often a complex hidden code for each of them.

Here is an example that illustrates the creation of a widget using Motif-provided function:

XmCreatePushButton creates a push button widget. All Motif XmCreate type functions take the same set of parameters:

```
Widget XmCreatePushButton(Widget parent,
        char* widget_name,
        ArgList args,
        cardinal num_args)
```

The parameter parent widget (all widgets, except for top-level widgets, must have a parent), controls the size and location of the child widget created by XmCreate-PushButton. *widget_name*, is the name of the push button to be created. This name is important for resource–setting commands. The *args* and *num_args* parameters are the list of hard–coded resource values the developer want to set for the widget.

The XmCreate... functions create, but do not manage, widgets. Therefore, the developer must later call XtManageChild. An unmanaged widget does not appear in a window on the screen. When a widget is managed, the widget is then placed under control of its parent. Usually managed widgets are visible, but sometimes the parent sets the size and location of a managed widget to something that can not be seen. In addition, if the parent widget isn't managed, then the child widget also will not be visible-even if the child widget is managed. This is very useful for working with dialog windows.

Callback Functions

Xt and Motif intercept and handle most X events, freeing program code from this responsibility. If the developer wish to be notified of an event, he/she can set up an event-handling callback function to handle an event. Motif makes extensive use of these callback functions to notify the application code when a high-level event occurs. The Xt function XtAddCallback registers a function as a callback for a widget:

```
void XtAddCallback(Widget widget,
    String which_callback,
    XtCallbackProc callback_function,
    XtPointer client_data)
```

The client_data is a pointer to an extra data that to be passed to the callback function.

All basic callback functions set up with XtAddCallback take the same parameters. The first one is the widget ID of the widget the callback was set up on. The second one is a pointer (normally XtPointer) to the data called client data. This data was originally passed to XtAddCallback. The third one is a pointer to a Motif structure that includes specific information about the callback.

The *client_data* is application data, whereas *call_data* comes from Motif and Xt toolkit. Many callbacks provide a specialized structure with information useful to user application functions. Those structures are always in the call_data parameter. Any data that passed from XtAddCallback is passed as the client_data parameter. The easiest and most common mistake with callback functions is to reverse client_data and call_data. If the callback doesn't seem to work properly, always check that the parameters are correct. This is a really easy error to make, and even easier error to correct.

Realize the Widget

After setting up all the widgets and managing them, it is time to make them visible on the screen. XtRealizeWidget takes care of all the initializations necessary for a

widget and all its managed children. XtRealizeWidget takes one parameter, a high-level widget to be realized. Normally, this is the widget returned from XtAppInitialize:

void XtRealizeWidget(Widget parent)

XtAppMainLoop

XtAppMainLoop executes the main event-handling loop of the Motif application. This function executes forever, so there must be at least one callback function that will exit the program.

Looping forever allows the Xt Intrinsics to handle most of the work of an X application. With XtAppMainLoop, the Xt Intrinsics essentially take over the application. When an input event arrives, XtAppMainLoop determines which widget in the application should get the event and then passes the event on to the widget. The widget, then, determines whether to execute a callback function or to handle the event on its own.

2.2 The Client–Server Concept

Client/server [Cho94] computing refers to the method of distributing computer applications across many platforms. Typically, applications are divided between a server providing access to a central data repository and numerous clients containing graphical user interfaces (GUI's) that permit access to and manipulation of the data. The client application sends a request across the network for data or the use of a resource. The server application sits idle, waiting to service incoming requests from clients. When the server receives a request, it performs the necessary action. One server application can support numerous clients, giving each client access to all of the application data.

Client/server systems operate over LANs, WANs, or a combination of the two [Car93]. Client applications are typically placed on workstations with graphics capability. The primary purpose of the client application is to provide an interface between the application/data and the user. The server application usually resides

on a more powerful computer containing a database. While fulfilling requests, the server application prevents inconsistent data conditions caused by more than one client attempting to update the same data. Numerous client applications can run simultaneously on the same workstation. In addition, numerous server applications, or multiple instances of one server application, can run on a server. Due to the distributed nature of client/server applications, client/server architecture implies a great deal of flexibility and portability.

Client/server computing facilitates the development of applications with enhanced access to data. The implementation of client/server systems has enabled organizations to achieve breakthroughs in performance and significantly higher levels of customer satisfaction. The flexibility inherent in client/server architecture allows for more competitive procurements, and easy upgrading of hardware. Client/server implementations force organizations to focus on how information moves within their organization, and how to best use that information to meet the needs of users.

It is difficult to succinctly define exactly what client/server systems are. Different people and organizations have different definitions. However, Bochenski [Boc94] points out that there are ten commonly accepted characteristics that help to define the meaning of client/server computing:

1. A client/server architecture consists of a client process and a server process that can be distinguished from each other, yet that can interact seamlessly.

2. The client and server portions can operate on separate computer platforms - but do not have to.

3. Either the client or the server platform can be upgraded without having to upgrade the other platform.

4. The server is able to service multiple clients concurrently; clients can access multiple servers.

5. The client/server system includes a networking capability.

6. A significant portion of the application logic resides at the client end.

7. Action is usually initiated at the client end. However, database servers can perform tasks based on triggers, business rules, and stored procedures.

8. A user-friendly graphical user interface (GUI) generally resides at the client end.

9. A structured query language (SQL) capability is characteristic of the majority of client/server systems.

10. The database server should provide data protection and security.

Regardless of the definition used, the computer industry is going through a period of rapid and dynamic change – and client/server computing is a key component of this evolution. 100% of the corporate respondents to a recent survey on information systems (IS) spending for 1996 are budgeting significant funds for client/server development and deployment [Wil96]. There are several reasons for the intense interest in client/server computing:

- Clients are increasingly powerful and cheaper than mainframes and minicomputers.

- Servers are becoming powerful enough to handle the workload of many clients at low cost.

- Data storage can be moved closer to the end user where it becomes a more valuable business resource.

- Client/server applications are generally easier to construct and maintain.

CINDI is a *client-server* application where programs runs on both the client and server computers. The program that runs on the server in no way limits the number of clients that the server can support simultaneously (thee server machine could be difference from the client machine). In fact, a single server can in principle manage an arbitrary number of active clients. We provide graphical user interface on our client application.

Once a client has established a connection to a server, it can issue a series of transactions each of which is invoked by a simple function call. One such transaction

actually performs the connection between client and server. With the exception of the connect transaction, which must also establish a connection, each transaction provokes a sequence of actions as follows:

1. A message identifying the transaction to be executed and the associate data are sent by the client to the server.
2. The server processes identified transaction using the associate data.
3. A message containing the response to the transaction is sent by the server to the client along with the ID of the original transaction..

This scenario is repeated until the client initiates a terminating transaction.

All the data pertaining to the indexing metadata is maintained in one or more sites of the distributed databases.

2.3 Use of an Expert System

Expert Systems have been used extensively to model human intelligence [Shi92]. For example, a medical diagnosis system can mimic a doctor by capturing his/her knowledge that enables the doctor to diagnose a disease from a given set of symptoms. Capturing the mental view of the domain expert, the acquired knowledge is encoded as a set of if...then... rules.

In our application, resources providers need to classify their documents by subject. In cataloging a new resource, the librarians use the knowledge of accepted norms for classification. They are familiar with the classification schemes, terms, index structures, and resources available in the domain of the user's need. From this knowledge, and their perception of the resource to be cataloged, they choose the term to describe the document. The expertise of a cataloging librarian had to be replicated to assist the users of the registering sub-system in cataloging their resources. The subject lookup manager guides the user in selecting the most correct subject terms from the catalog database [DS96, CSDR]. The rules for cataloging were designed to be *re-usable* for searching as well.

Users searching for a information resource often have only incomplete information about what they are looking for; for example, a user wants an article written by Desai on the topic of *information system*, but the full name of the author and exact title of the document are not known. Typically, a reference librarian would be able to guide the user in the search; this is what an expert system sets out to do. In general, the user input to the expert system can be at different levels of detail, and depending upon the level of detail entered, one task of the expert system is to provide the required amount of help to complete the input. In general, the expert system exhibits the following properties:

Indexing sub–system

A semantic header is created by the author of the resource. In addition to standard bibliographic information, the semantic header includes author–generated subject terms and an indication of document type. The expert system guides the user in selecting the most correct subject terms from an on-line thesaurus. This ensures an automatic conversion of author terms to the controlled vocabulary drawn from the established lists of subject headings and descriptions.

Search sub–system

A graphical user interface allows the user to enter search queries. The expert system provides automatic conversion of user search terms to the controlled vocabulary. In searching for a given set of documents, often the users offer vague, partial, and incorrect information in identifying the index terms for the documents they are searching. In other words, the user search specification is often "ill–structured"; hence, expertise is needed to help users better articulate their needs. In addition, the total number of input field combinations occurs at any given time. For example, in a given search, the user may be interested only in the documents of a particular subject hierarchy (Computer Science–Artificial Intelligence–Expert System) without caring for the titles of the documents. In this case, it does not make sense to consider a search situation that also explicitly requires the titles of the documents. On the other hand, if the user

is interested in articles written by a specific author without caring for the titles of the documents or how they are cataloged, the system should not require subject hierarchy entry and title entry. Thus, by isolating and encoding the input combinations and handling them through rules, only a subset of rules needs to be active to process the user input and provide appropriate help. This also improves the system's modularity and understandability [AW94, JF90]. Detail implementation of the expert system and explanations of the rules are given in Section 3.2.2.

2.4 The Database Systems

The index entries registered by a provider of a resource are stored in a Semantic Header distributed Database system (SHDDB). From the point of view of the user of the system, the underlying Semantic Header Database (SHDB) may be considered to be a monolithic system. In reality, it could be distributed and replicated, allowing for reliable and failure-tolerant operations. The interface hides the distributed and replicated nature of the database. The distribution is based on subject areas and, as such, the database is considered to be horizontally partitioned [Des90].

It is envisaged that the Semantic Header Database on different subjects will be maintained at different nodes of the Internet. The locations of such nodes need only be known by the intrinsic interface. A database Catalog would be used to distribute this information. However, this Catalog itself could be distributed and replicated as is done for distributed database systems.

Database Catalogs will also be used to store information about subject areas maintained in the SHDBs so that the users can select subject items for indexing and retrieving Semantic Headers. Thus, each node will contain a Catalog consisting of all subjects as well as information relating to the locations of Semantic Headers, pertaining to a subject, in the distributed system. The overall structure of the CINDI System is illustrated in Figure 1.

The Semantic Header information entered by the provider of the resource using

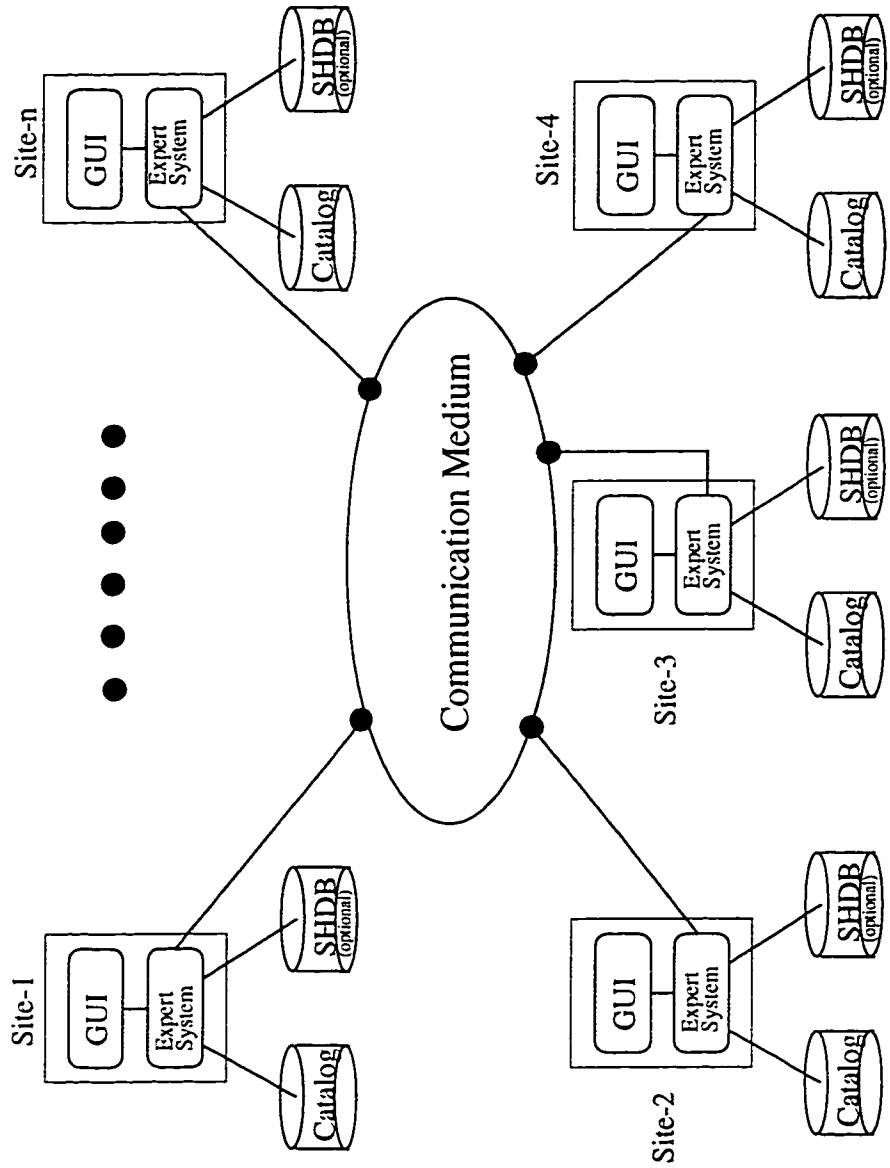

Figure 1: General structure of the CINDI system

a graphical interface is relayed from the user's workstation by a client process to the database server process at one of the nodes of the SHDDB. The node is chosen based on its proximity to the workstation or on the subject of the index record. On receipt of the information, the server verifies the correctness and authenticity of the information and after finding everything in order, sends an acknowledgement to the client.

The server node is responsible for locating the partitions of the SHDDB where the entry should be stored and for forwarding the replicated information to appropriate nodes. In addition, the database server process is responsible for providing the catalog information for the search system. In this way the various sites of the database work in a cooperating mode to maintain consistency of the replicated portion. The replicated nature of the database also ensures distribution of load and ensures continued access to the bibliography when one or more sites are temporarily nonfunctional.

The Semantic Header Database and the Catalog Database are implemented using the ODE (Object Database Environment)[AGGL95] database system. In our system. we used a modified Library of Congress subject classification scheme for the general subject level. We produced a thesaurus for the subject *Computer Science* based on ACM Computing Review classification [Ass95].

When making a search request, the client process communicates with the nearest Catalog to determine the appropriate site of the SHDB. Subsequently, the client process communicates with this database and retrieves one or more Semantic Headers. The results of the query could then be collected and sent to the user's workstation. The contents of these headers are displayed, on demand, to the user who may decide to access one or more of the actual resources. It may happen that the item in question may be available from a number of sources. In such a case, the best source is chosen based on optimum costs. The client process would attempt to use the appropriate hardware/software to retrieve the desired resources.

2.5 System Design

System design [RBPEL91, Pre95] is the high–level strategy for solving the target problem and building a solution. System design includes decisions about the organization of the system into subsystems, the allocation of subsystems to hardware and software components, and major conceptual and policy decisions that form the framework for detailed design. The overall organization of a system is called the *system architecture.*

System design is the first stage during which the basic approach to solving the problem is selected. The first step in system design is to divide the system into sub–systems. A sub–system is not an object or a function but a package of functions, associations, operations, events, and constraints that are interrelated and that have a reasonably well-defined and small interface with other subsystems. A sub–system is usually identified by the services it provides. A service is a group of related functions that share some common purpose, such as I/O processing, drawing pictures, or performing arithmetic.

The CINDI system is divided into five sub–systems: The Index Registering Sub–system, the Search Sub–system, the Annotations Sub–system, the Semantic–Header Database Sub–system and the Catalog Database Sub–system. Figure 2 shows the architecture of the CINDI system.

Authors/providers of information resources create and register the metadata of their resources through the Registering Sub–system's user interface. The expert system of the Registering Sub–system guides the user in making the metadata entry and interacts with the Semantic header distributed database system. Internet users search for pertinent information through the Search Sub–system. A search query is entered through the user interface of the Search Sub–system. The expert system of the Search Sub–system helps the user to enter more precise query by providing context–sensitive help to focus the search. The user can access the actual on–line source resources through the search system. The search system, on the user's demand, passes the on–line access identifier of the resource to an existing browser, which display the actual resource to the user. Reviewer's comments on a specific document can be made through the Annotation Sub–system.

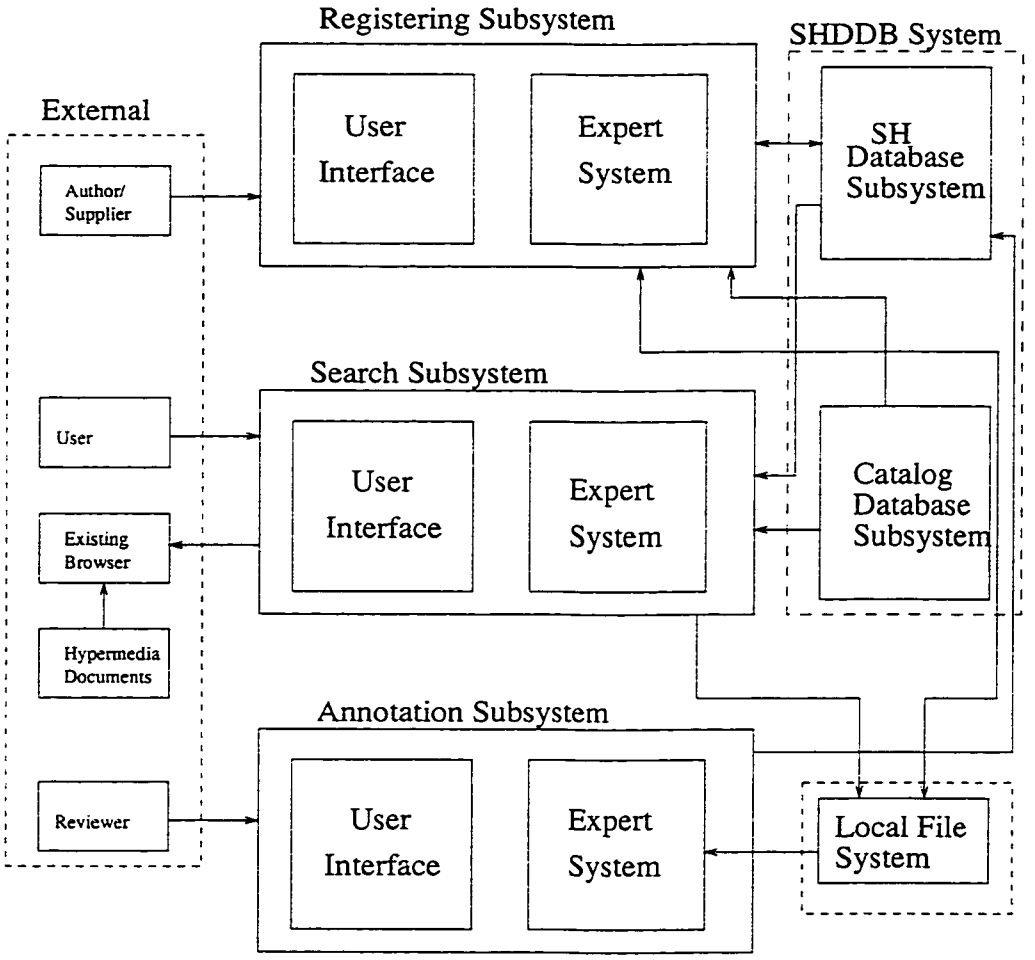

Figure 2: Architecture of the CINDI system

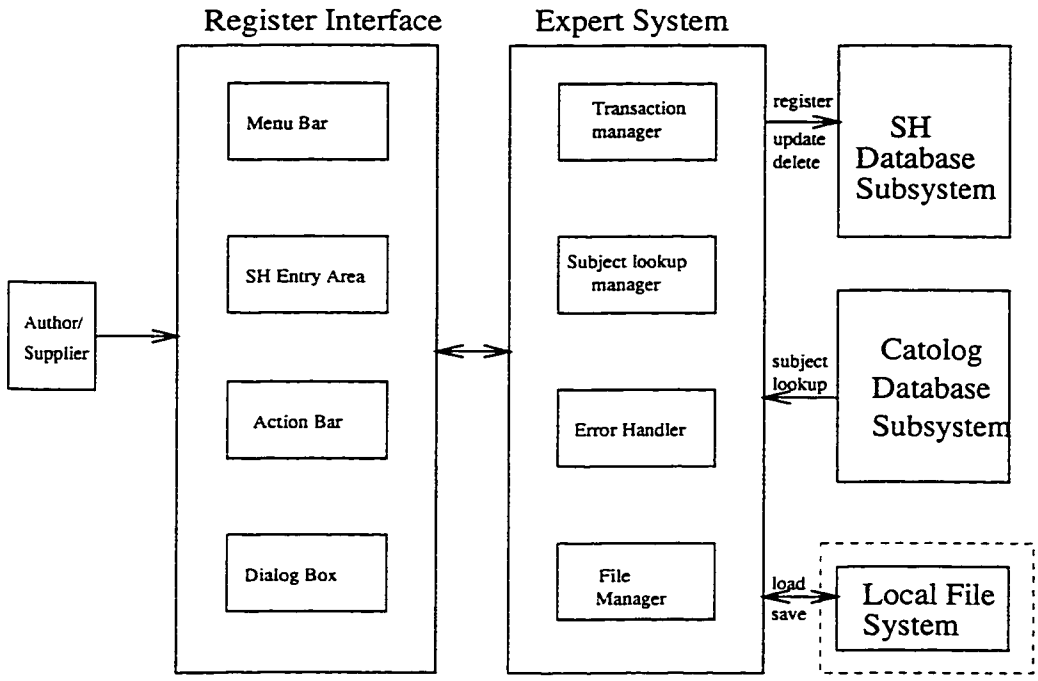

Figure 3: Architecture of the Registering sub–system

The Registering Sub–System

The Registering Sub–system (Figure 3) is the system that accepts metadata from the resources provider. This sub–system is further divided into two main components, the graphical user interface (GUI) and the expert system (Section 3.2.2). Resource providers enter metadata through the GUI. The expert system mimics the behaviour of a reference librarian to help the user during the process of data entry. It provides context–sensitive help for entering the catalog subject fields. Given a synonym or substring, the expert system will provide all matching terms for selection through a pop up window. It interacts with the Catalog Database Sub–system by sending subject search query and by receiving search results from the Catalog Database Sub–system. After completely entering the Semantic–Header fields, the author/supplier can register it in the Semantic–Header database (SHDB). He can also update an existing Semantic–Header in the SHDB or delete a Semantic–Header from the SHDB provided he enter the same user ID and password pair that was used at the time of the registration. However some fields cannot be updated (Section 3.3.4). The GUI has a main menu bar, a data entry area (SH Entry Area), an action bar, and dialog boxes.

The menu bar provides menu option for users to open or save a Semantic–Header file, edit the entry area, and access online help. The Semantic–Header entry area is the fields where the corresponding entries of the information resource's metadata are entered. The fields are title, alt–title, subject, language, character set, author information, keywords, identifiers, classification, coverage, genre, system requirement, source/reference, abstract, and annotation. Detailed description of the metadata is given in Chapter 3. An action bar provides push button for various actions to be taken, i.e., register, update, or delete the metadata. The Transaction manager is responsible for interacting with the remote Semantic–Header database system for registering, updating, and deleting an index. It also checks the metadata entered before sending the data to the Semantic–Header Database Sub–system for registering. The subject lookup manager is the expert system engine. The error handler validates the metadata entered by the user to make sure that the data are acceptable according to the rules of the Semantic–Header. It interacts with users through popup messages if necessary. The file manager provides services for loading or saving a Semantic–Header in the local file system.

The Search Sub–system

As shown in Figure 4, a user utilize the Search Sub–system to enter the search query for pertinent resources. This sub–system is also divided into two main components, the graphical user interface and an expert system. The GUI allows the user to enter a search query. The expert system consists of the query manager, the error handler, subject look–up manager, and the file manager. The query manager interacts with the Semantic–Header database system by sending search query and receiving search result. The Subject Look–Up Manager interacts with the Catalog Database Sub–system by conducting standard subject terms look–up. The error handler ensures that the query entered is valid and provide the user with appropriate pop up message. The local file manager can save a Semantic–Header entry in the display window in the local file system.

The search interface has a query entry area, an action bar, a Semantic–Header display window, and dialog boxes. The query entry area is the area where the search

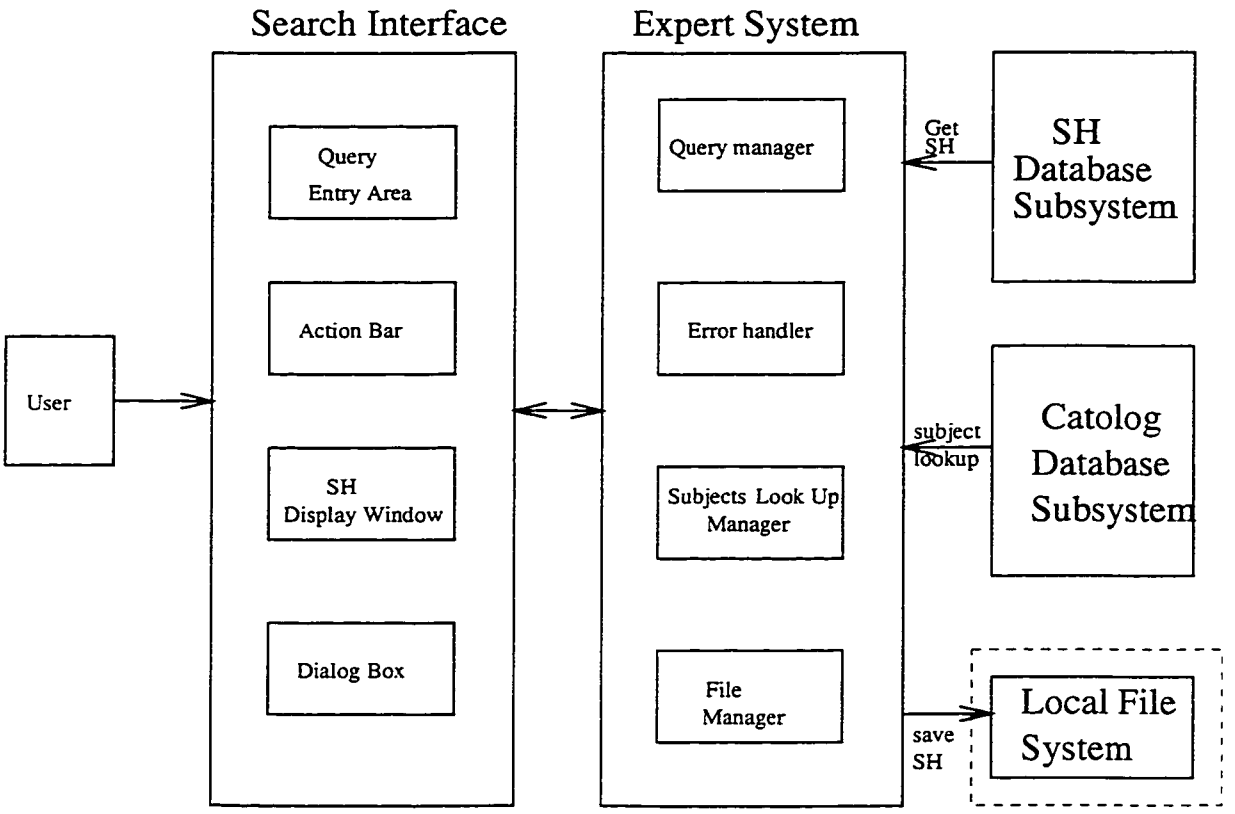

Figure 4: Architecture of the Search sub–system

query is entered. The search entry can be a document's title, author, subject, keywords, identifier, words in abstract or the combination of them. Detailed description on the search query format will be given in Chapter 4. Action bar will provide push button for actions to be taken, i.e Search, Clear, Help, Exit. SH Display Window is the popup window to display a specific Semantic–Header selected by the user during the search process. From the display window, the user can access the actual resource by connecting to Netscape browser, save the Semantic–Header to local disk. Dialog box will provide error messages, warning messages etc.

The Annotation Sub–system

Annotations is a reviewer's comments on a information resource. Any user can make annotation on a resource through the Annotation Sub–system (Figure 5). Such annotations will be registered with the Semantic–Header and would be displayed with it. The graphical user interface of this sub–system displays the existing annotations of the resource and the Semantic–Header name, which is composed of the title, first

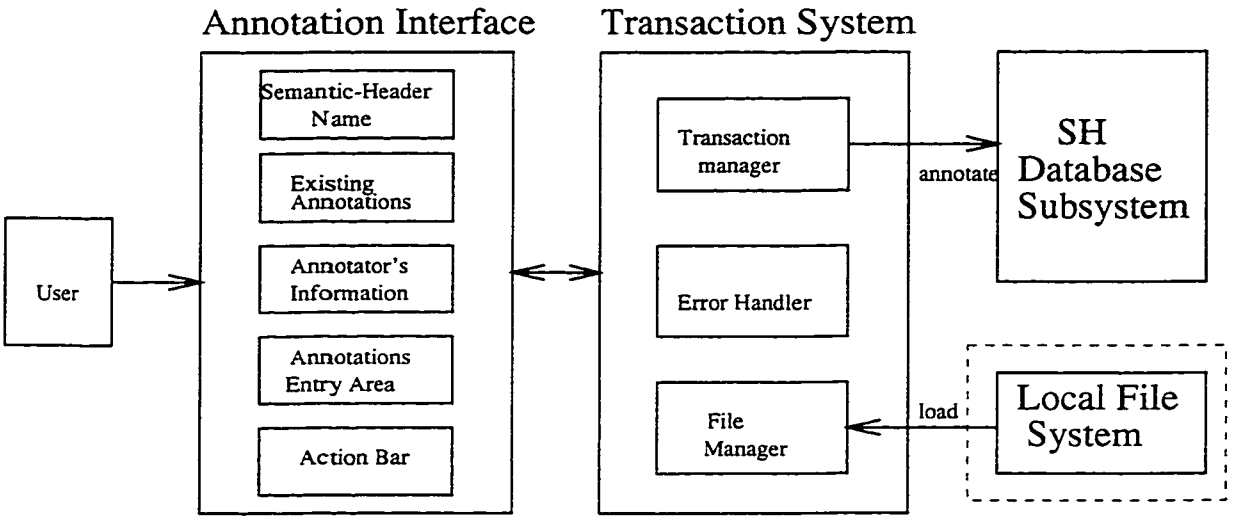

Figure 5: Architecture of the Annotation sub-system

subject, first author, posted date, and version number. It also allows the user to enter new annotations. The transaction manager checks the user's name entry through the interface against the user's actual login name. A warning message, "Name entered is different from user login name. Both name will be recorded", will be given if it is necessary.

Chapter 3

Indexing/Registering Sub–system

The index entry and registering sub–system provides a graphical interface (Figure 6) to facilitate the provider (author/creator) of a resource to register the bibliographic information about the resource. The interface allows the provider to enter this information and it provides help by means of pop-up selection windows and an expert engine to suggest controlled terms (Section 3.2.2). Once the information is correctly entered, the author can decide to register the Semantic–Header entry in the Semantic Header database. Once the header information is accepted by the database, the author/creator is notified. An user ID and password are to be provided when the Semantic–Header is first registered and for all changes made to it. Since the user ID and the associate encoded password is not accessible by anyone other than the original registrar of the index entry, the entry can only be updated by person(s) who are cognizant of them.

3.1 Semantic–Header

The heart of any bibliography or indexing system is the record that is kept for each item being indexed. Standardization of a bibliographic entry better allows people to exchange information through a virtual library. The index structure for our indexing system was proposed by Desai [Des95c, Des95b, Des95a] and is called the **Semantic–Header**. It was originally presented at the First International World Wide Web Conference in Geneva for WWW resources. Since then, it has been extended to other resources accessible directly on the Internet.

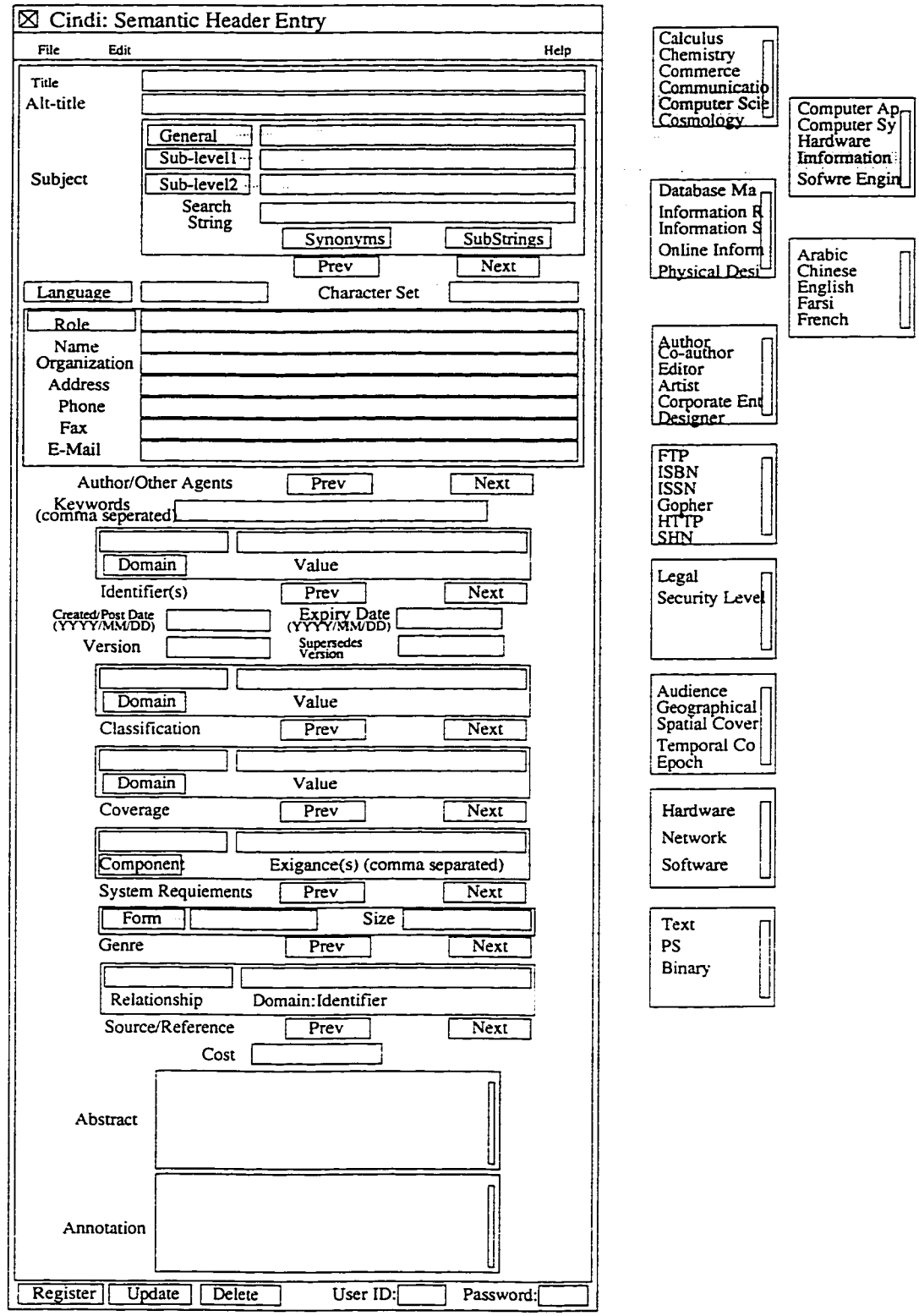

Figure 6: Semantic–Header Entry Interface: Overall Design

The Semantic–Header may be considered as an application of the Standard Generalized Markup Language (SGML)[Gol90].The objective of the Semantic–Header is to include those elements that are most often used in the search for an information resource. Since the majority of searches begin with a title, the name of the author(s) (70%), subject and sub–subject (50%) [Kat87], we have made the entry of these elements mandatory in the Semantic–Header. The abstract and annotations are, also, relevant in deciding whether or not the resource would be useful. These items are also included. However, a user working with the index entry system is guided through the process by an expert system. This system guides the user in the choice of standardized terms through an easy–to–use graphical user interface (Figure 6).

3.1.1 Semantic Header Structure

The structure of the index is similar to the ones used for most library indices and includes other information deemed useful for on–line systems. The contents of a Semantic–Header are defined in the following grammar. The grammar follows extended BNF rules [GM86]: tags are given in quotes, optional items are bracketed by square brackets "[" and "]", alternative items are separated by a bar "|", and the superscript plus sign $^+$ means that items may be repeated one or more times. The grammar for the Semantic–Header follows:

sem_header := "<semhdr>" contents "</semhdr>"
contents := userID password title alt–title subjects language character_set
authors keywords identifiers dates version sup_version
classifications coverages system_reqs genre source_ref cost
abstract annotation
userID := "<userid>" ... "</userid>"
password := "<password>" ... "</password>"
title := "<title>" ... "</title>"
alt–title := "<alttitle>" [...] "</alttitle>"
subjects := "<subject>" subject_item$^+$ "</subject>"
subject_item := general sublevel1 sublevel2
general := "<general>" ... "</general>"

sublevel1 := “<sublevel1>” [...] “</sublevel1>”

sublevel2 := “<sublevel2>” [...] “</sublevel2>”

language := “<languag>” [Arabic|Chinese|English|...] “</language>”

character_set := “<char-set>” [...] “</char-set>”

authors := “<author>” author_item^{+} “</author>”

author_item := role name orgnization address phone fax email

role := “<arole>” Author|Co-author|Editor|Artist| Corporate Entity| Designer|Programmer “</arole>”

name := “<aname>” [...] “</aname>”

orgnization := “<aorg>” [...] “</aorg>”

address := “<aaddress>” [...] “</aaddress>”

phone := “<aphone>” [...] “</aphone>”

fax := “<afax>” [...] “</afax>”

email := “<aemail>” [...] “</aemail>”

keywords := “<keyword>” ... “</keyword>”

identifiers := “<identifier>” iden_item^{+} “</identifier>”

iden_item := domain value

domain := “<iden-domain>” FTP|ISBN|ISSN |Gopher|HTTP|ULC|USA|ULN “</iden-domain>”

value := “<iden-value>” ... “</iden-value>”

dates := “<dates>” date_post date_expiry “</dates>”

date_post := “<created>” ... “</created>”

date_expiry := “<expiry>” [...] “</expiry>”

version := “<version>” [...] “</versionE>”

sup_version := “<spversion>” [...] “</spversion>”

classifications := “<classification>” cls_item^{+} “</classification>”

cls_item := domain value

domain := “<cls-domain>” Legal|Security Level “</cls-domain>”

value := “<cls-value>” ... “</cls-value>”

coverages := “<coverag>” coverage_item^{+}“</coverage>”

coverage_item := domain value

domain := “<cover-domain>” Geographical|Spartial |Temporal|Epoch

"</cover-domain>"

value := "<cover-value>" ... "</cover-value>"

system_reqs := "<system-requirements>" system_item$^+$
"</system-requirements>"

system_item := component exigance

component := "<component>" [Hardware|Software|Network]
"</component>"

exigance := "<exigance>" [...] "</exigance>"

genre := "<genr>" genre_item$^+$ "</genre>"

genre_item := form size

form := "<form>" [Text|PS|Binary] "</form>"

size := "<size>" [...] "</size>"

source-refs := "<source-reference>" ref_item$^+$"</source-reference>"

ref_item := relation domain_iden

relation := "<relation>" [...] "</relation>"

domain_iden := "<domain-identifier>" [...] "</domain-identifier>"

cost := "<cost>" [...] "</cost>"

abstract := "<abstract>" [...] "</abstract>"

annotation := "<annotation>" [...] "</annotation>"

In general, every field or block of the Semantic–Header will be started with a 'begin' tag and terminated by an 'end' tag. For example *abstract* begins with *<abstract>* and ends with *</abstract>*.

A Semantic–Header begins with the tag *<semhdr>* and ends with the tag *</semhdr>*. In between these are fields: userID password, title, alt-title, a list of subjects, language, character set, a list of authors/agents, a list of key words, a list of identifiers, date created, expiry date, version, supsedes version, a list of classifications, a list of coverages, a list of system requirements, a list of genres, a list of sources/references, cost, abstract, and annotations. The significance of these fields are given below:

User ID: This is a required field. It is a self selected value chosen at the outset by the provider along with an associate password. The Semantic Header database

will record this information and would require it for all sunsquent interaction to identify the provider of the resource. The user should enter the same userID and password pair every time he/she register or modifies a Semantic–Header.

Password: This is a required field. It appears between the tags *<password>* and *</password>*. When first registering a Semantic–Header, the user must select a password . This password must be used later along with the user ID to update or delete the corresponding Semantic–Header.

Title: Title signifies name or short description of the indexing resource. It is a required field. It begins with the tag *<title>* and ends with the tag *</title>*.

Alt-Title: Alt-title is an alternate name or short description of the resource to be indexed. Alt-title is an optional field.

Subject: Subject denotes the words or phrases indicative of the information content. Subject begins with the tag *<subject>* and ends with *</subject>*. Within subject we have three levels of hierarchy. Subject is a repeating group (a multi-part field with one or more occurrences of items in the group). All resources must have at least one occurrence of this field. An expert system guides the user when entering all subject levels.

Language and Character Set: The character set used and the language in which the resource was composed are given in the next two optional fields.

A *character set* is the set of all characters, alphabetic and otherwise, necessary to construct words and sentences in a given language. For example, ASCII is a 7–bit character set. ISO 8859–1, also called Latin–1, is an 8–bit character set that represents most Western European languages (except Greek), including French, German, and English.

Author: The details about the author(s) and/or other agent(s) responsible for the resource are given in the next repeating group. The sub–fields are: role of the agent,

name, organization, address, phone and fax numbers, and e–mail address. All sub–fields except the name are optional, except in instances of corporate entries in which case the orgnization must be given. By using the role sub–field and giving it the appropriate value, semantics for agents such as editor or publisher are incorporated in this repeating group.

Keyword: Keywords that can be used for searching the resource are kept in between the tags *<keyword>* and *</keyword>*. There could be one or more keywords separated by a comma.

Identifier: The next element is a repeating group for recording the identifiers of the resource. Each occurrence of this group consists of two sub–fields: one for the domain and the other for the corresponding value.

Date: The dates of creation(required), expiry and update (if any) are given next. Any updates made are indicated by a system–generated date.

Version: The version number, and the version number being superseded, if any, are given in these optional elements.

Classification: The intended classification is indicated in the next optional repeating group. It consists of a domain (nature of resource, security or distribution restriction, copyright status etc.) and the corresponding value.

Coverage: The coverage is indicated in the next repeating group. It consists of a domain (target audience, coverage in a spatial and/or temporal sense, etc.) and the corresponding value.

System Requirements: A list of system requirements such as the software required to access, use, display or operate the resource is included in the Semantic–Header as an optional repeating group. It consists of a domain of the system requirements (possible value are: hardware, software, network, protocol, etc.) and the corresponding exigance.

Genre: This optional element is used to describe the physical or electronic format of the resource. It consists of a domain (type of representation or form which in the case of a file could be a format such as ASCII, Postscript, TeX, GIF, etc.,) and the corresponding value or size of the resource.

Source/Reference: The relationship of the resource to other resources may be indicated by this optional repeating group. It contains the relationships, domains, and identifiers of related resources. A related object may be used in deriving the resource being described, or it may be its sub/super components. Such information is usually found in the body of a document–like resource. However, this optional group permits an option for this type of resource and an opportunity to register it for resources of other formats.

Cost: In the case of a resource accessible for a fee, the cost of accessing it is given next. It consists of a currency and the cost for accessing the resource.

Abstraction and Annotation: The abstraction and annotations are given in the next fields. The abstract is provided by the author of the resource, the annotations are made by the author and/or independent users of the resource and include their identities along with their login user names. Once registered, the annotation cannot be modified.

3.1.2 An Semantic–Header Example

Here is an example of a Semantic–Header.

<semhdr>
<userid>
bcdesai
</userid>
<password>
ciuth4g
</password>
<title>

A System for Seamless Search of Distributed Information Sources
</title>
<alttitle>
Sailing the Internet with a navigational System
</alttitle>
<subject>
<general>
computer science
</general>
<sublevel1>
information systems
</sublevel1>
<sublevel2>
search process
</sublevel2>
<general>
computer science
</general>
<sublevel1>
information systems
</sublevel1>
<sublevel2>
distributed systems (database management)
</sublevel2>
</subject>
<languag>
English
</language>
<char-set>
iso-8859-1
</char-set>
<author>
<arole>
Author

</arole>
<aname>
Bipin C. Desai
</aname>
<aorg>
Department of Computer Science, Concordia University
</aorg>
<aaddress>
1455 De Maisonneuve Blvd. West, Montreal, Quebec, Canada H3G 1M8
</aaddress>
<aphone>
848-3000
</aphone>
<afax>
848-3026
</afax>
<aemail>
bcdesai@cs.concordia.ca
</aemail>
<arole>
Co-author
</arole>
<aname>
Rajjan Shighal
</aname>
<aorg>
Department of Computer Science, Concordia University
</aorg>
<aaddress>
1455 De Maisonneuve Blvd. West, Montreal, Quebec, Canada H3G 1M8
</aaddress>
<aphone>

</aphone>
<afax>

</afax>
<aemail>
shinghal@cs.concordia.ca
</aemail>
</author>
<keyword>
distributed information technology, hypermedia, artificial intelligence, expert systems
</keyword>
<identifier>
<iden-domain>
HTTP
</iden-domain>
<iden-value>
http://www.cs.concordia.ca/old/w3-paper.html
</iden-value>
</identifier>
<dates>
<created>
1994/06/15
</created>
<expiry>

</expiry>
</dates>
<version>
1.0
</version>
<spversion>

</spversion>
<classification>
<cls-domain>
Security Level
</cls-domain>
<cls-value>
Normal
</cls-value>
</classification>
<coverag>
<cover-domain>
Audience
</cover-domain>
<cover-value>
engineers, scientists
</cover-value>
<cover-domain>
Geographical Coverage
</cover-domain>
<cover-value>
planet earth
</cover-value>
</coverage>
<system-requirements>
<component>
Software
</component>
<exigance>
Netscape Browser
</exigance>
</system-requirements>
<genr>
<form>
Text

</form>
<size>
19919 bytes
</size>
<form>
PS
</form>
<size>
79919 bytes
</size>
</genre>
<source-reference>
<relation>

</relation>
<domain-identifier>
</domain-identifier>
</source-reference>
<cost>

</cost>
<abstract>
This article discusses the issues in developing a system that will provide users with desktop access to the world's digital information resources. It provides a more focused approach to searching, retrieving and perusing hyper-media documents. The documents are stored in heterogeneous distributed information systems representing virtual libraries. The system should allow users to search and obtain information from systems exhibiting a range of categorizing and organizing conflicts. Thus users, at a given workstation, perceive their local information system as having been augmented by further data rather than having to deal with a number of unfamiliar systems. For example, a user at a workstation could locate and peruse any document stored electronically anywhere across a wide network

of virtual libraries.
</abstract>
<annotation>

</annotation>
</semhdr>

3.1.3 Semantic Rules

To register a Semantic–Header, certain rules must be respected.

1. The following fields are required:
   - Title.
   - At least the General level for one subject.
   - At least one author/other agents.
   - At least one keyword.
   - At least one identifier.
   - Date created/Posted.
   - User ID (Must be at least 6 characters.)
   - Password (Must be 4 to 8 characters.)
2. Semantic Header Name: To unique identify a Semantic–Header, we define the concatenation of the title, the first author's name, first author's role, the first subject, the date created, and the version number as the **Semantic Header Name (SHN).** In change mode the SHN cannot be changed.
3. The value for the *role* between the tags *<arole>* and *</arole>* can be author, co-author, editor, artist, corporate entity, designer, programmer, or publisher. If *role* is a corporate entity, the name of the organization must be given between the tags *<aorg>* and *</aorg>*. Otherwise, the name of a person must be given between the tags *<aname>* and *</aname>*.

4. Subject entries are selected from standard terms in the thesaurus and provided by the expert system. Presently we provide only lower subject levels for computer science in the Catalog Database. The Details of this classification can be found in [Ass95].

5. The possible value for identifier domain between the tags *<iden-domain>* and *</iden-domain>* can be FTP, ISBN, ISSN, Gopher, HTTP, SHN, UAS, or URN.

6. Possible values for the classification domain between the tags *<cls-domain>* and *</cls-domain>* must be Legal or Security level.

7. Possible values for the coverage domain between the tags *<cover-domain>* and *</cover-domain>* must be Audience, Geographical, Spatial, Temporal, or Epoch.

8. Possible values for the system requirement component must be Hardware, Software, Protocol, or Networks.

9. Possible values for the genre form could be Text, PS, or Binary etc.

3.2 Subject Entry: Modelling the Expertise of a Librarian

The advice offered by a librarian is made available to the user of our application while cataloging a resource. During cataloging, the provider often may not know to which subject, sub–subject, sub-sub-subject a particular document belongs. In cataloging a new resource, the cataloging librarian uses knowledge of authority and accepted norms for classification. From such knowledge s/he chooses terms to describe the resource. Reference librarians are aware of the conventions used by cataloguers as well. They are typically familiar with the classification schemes, terms, index structures, and resources available in the domain of the user's need.

To model the expertise of a reference librarian, we use a small embedded expert system in our current implementation for cataloging. We focus our discussion solely on context–sensitive help for subject entry from the graphical user interface.

3.2.1 Context Sensitive Help

The user interfaces for registering require the user to input information about the subject classification to which the document to be indexed belongs. The user entry can be at different levels of detail, and, depending upon the level of detail entered, one of the tasks of our expert system is to provide the required amount of help to complete the input. This is equivalent to a reference librarian's guiding a novice user in entering additional data, based on what has already been entered by the user. For example, if the user had entered the subject *Computer Science*, then the help for other fields of the subject hierarchy would be tailored to *Computer Science*. Later, if the user changes the subject to, say, *Physics*, then the help information would change accordingly. The context sensitivity of the help is complicated, however, because the user can input a synonym for an input field. In general, synonyms can be entered for any of the fields corresponding to the subject hierarchy: a three level hierarchy for document classification. Moreover, a synonym entered at one level of the subject hierarchy may resolve at one or more levels. A synonym must be resolved using a control thesaurus so that it can match with an appropriate document–identification keyword.

Suppose, for example, that the user entered KBS: it can denote Knowledge Base Systems, Expert Systems, or Deductive Data Base Systems that are part of the control thesaurus. In general, searching the subject hierarchy for a control vocabulary can be modelled as a tree traversal. Synonyms, however, complicate this search for a control thesaurus term because they make a tree traversal into a directed–acyclic–graph traversal, which is computationally more demanding. Synonym resolution, however, allows the system to automatically fill in higher levels of the subject hierarchy, as these can be determined once a synonym at a lower level is resolved in the control thesaurus. This aids focusing the search to a specific set of documents [CSDR]. Such synonyms. For example. if entry KBS for sub–subject was resolved to *Expert Systems*, then the subject of the entry would be automatically updated to contain *Artificial Intelligence* under which *Expert Systems* is a sub–subject.The synonyms and the associated control thesaurus are stored in a local database.

A complication arises in synonym resolution when the user enters a synonym for a subject that is actually a sub–subject synonym [DS96]. Such synonyms are said to be *non–contextual.* Upon detecting a non–contextual synonym entered for a field, the system warns the user of this mismatch and resolves the synonym by traversing up/down the subject hierarchy. The expert system then displays a list of control thesaurus items found for the user to choose.

When a synonym resolves at multiple levels of the subject hierarchy, the synonym is said to be *overloaded.* For example, nothing prevents a generic synonym such as DB to be used as a shorthand for the sub–subject and the sub–sub–subject levels of the subject hierarchy. Suppose that as a sub-subject it resolves to {Data Base Systems, Knowledge Base Systems}, and as a sub–sub–subject it resolves to {Relational Data Bases, Horn Clause Systems}. The context of the input entry is then used to resolve the synonym for the level it corresponds to instead of resolving it into all matching terms. Handling non–contextual and overloaded synonyms mimics the discerning judgement of a reference librarian in interpreting user input appropriately.

Another complication arises when the user enters *partial values:* a sub–string for a subject, such as, *Data Bas.* Although one can display a list of subjects that have this substring, context sensitivity implies that the values already entered in other

fields must also be considered in providing help to the user. Thus, the help would be based on not only the partial values of the current field, but also on existing values of the other related fields. For example, if the user has entered *Hybrid relational* in sub–subject and *Frame* in sub–sub–subject field, then the context–sensitive help for subject should take into account the current values in sub–subject and sub–sub–subject fields before providing appropriate help to the user. Thus, the help would be based on not only the partial values of the current field, but also on existing values of other related fields. For example, if the user has entered **Hybrid relational** in sub–subject and **Frame** in sub–sub–subject field, then the context–sensitive help for subject would take into account the current values in sub–subject and sub–sub–subject fields before providing appropriate help to the user. If the current values of sub–subject and sub–sub–subject entries are ignored, then the user would be provided with a long list of subjects, many of which would have sub–sub–subjects that may not match with the current values entered in these respective fields.

When a partial value matches entries at multiple levels, then a *collision* is said to have occurred in the expansion of this partial value. In this situation, the system gives the matches for each subject level. If, however, no match can be found on the hierarchy based on the input value, a warning is issued.

3.2.2 Implementation of the Expert System

The knowledge that an expert system uses is represented as a set of if .. then .. production rules [Shi92]. In the current implementation of the expert system, some restrictions are applied to reduce the complexity of the rules. First of all, entry fields for subject, sub–subject, and sub–sub–subject are not editable. The user has to select an entry from the context–sensitive help list. Secondly, the synonym resolution and partial value entry (sub–string) are carried out through a separate entry field. The third restriction is that when the user looks for subject entry terms through synonym resolution or partial value entry, the values for current subject entry fields are ignored. The rules for our current implementation are as follows:

Rules for the inference engine

- If the search is at a general level, then invoke the top level subject of classification based on the Library of Congress subject headings from the Catalog

Database.

- If the search is at sub–level1, then use the selected general level entry as the key, search from the catalog database, and bring up the children subject level for the selected general level.
- If the search is at sub–level2, then use the selected general level and sub–level1 as a basis, and provide the lower level of the selected general level and sub–level1.
- If the string entry is a substring, then for each subject level of the subject hierarchy in the catalog database, display the control terms for each subject level containing the substring entered in the GUI.
- If the string entry is a synonym, then for each subject level of the subject hierarchy in the catalog database, display the control terms for each subject level having synonym with the substring entered in the GUI.

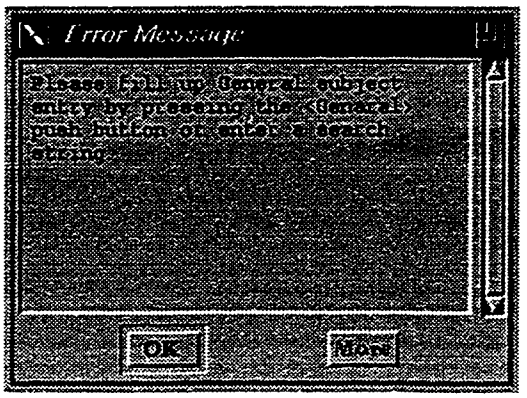

Figure 7: Error message for out of sequence lookup

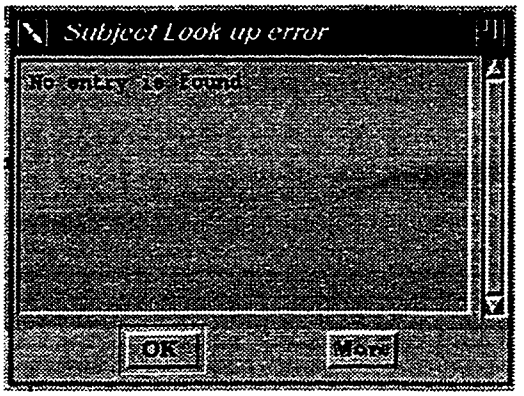

Figure 8: Subject look–up: No entry found

Rules enforced by the GUI

- Subject entry fields are not editable. The user cannot manually enter a subject, sub–subject, or sub–sub–subject.
- The user must select a higher level in the subject hierarchy before selecting a lower level. He must fill out the general level before looking up sub–level1 and, similarly, choose sub–level1 before looking up sub–level2. Otherwise, an error message window is displayed (Figure 7) to inform the user.
- When a subject button (**General, Sub–level1** or **Sub–level2**) is pressed, a popup window containing a list of standard subject terms of the appropriate subject hierarchy level appears. The user can then select a standard subject

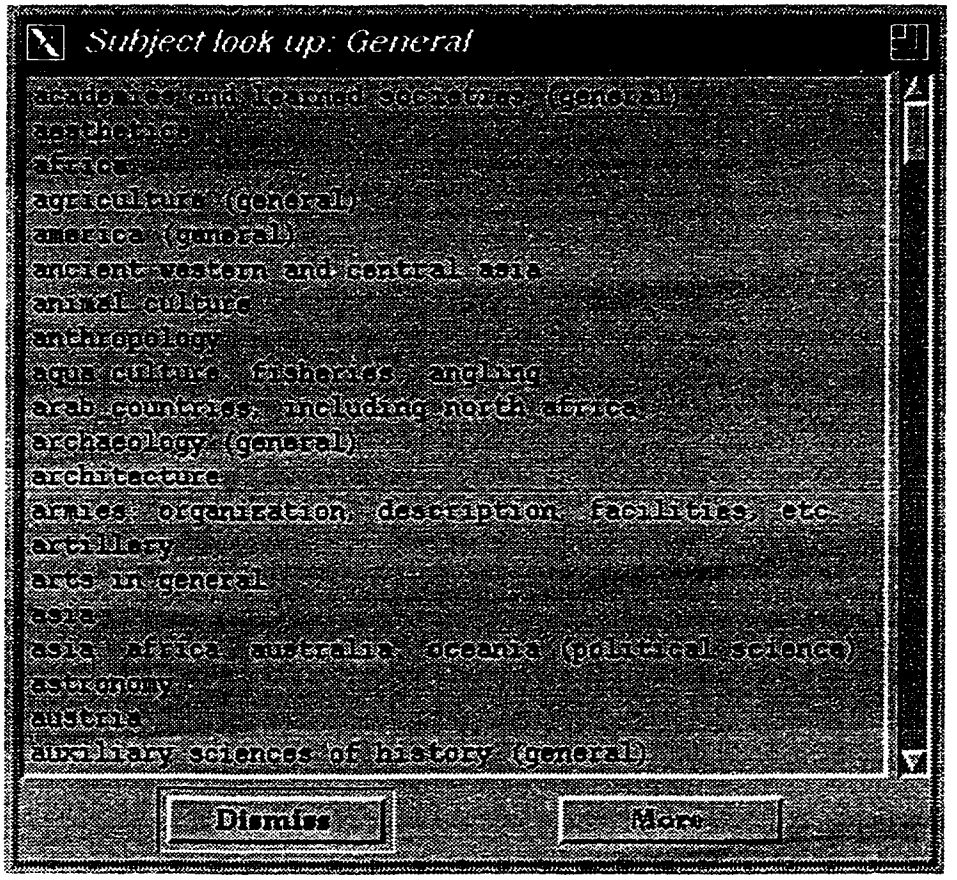

Figure 9: Subject look–up: General

term from the list by clicking on the appropriate line in the pop up window. In event that no standard term exists, an error message appears in a popup window (Figure 8).

- Selecting the **General** button invokes the top level of subject classification based on the Library of Congress subject headings. Figure 9 is the popup window for General level look–up. The items in the list are sorted in alphabetic order.

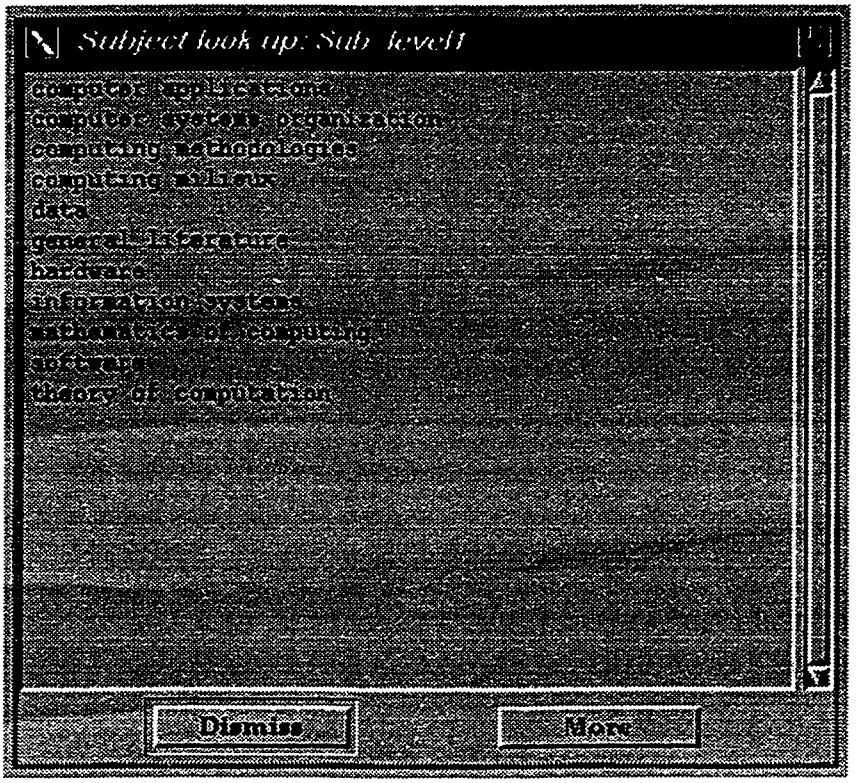

Figure 10: Subject look–up: Sub–level1

- Selecting the **Sub–level1** button provides the child subject level terms for the selected general level. Figure 10 is the popup window for the Sub–level1 look–up when the General level is *Computer Science.* In our current implementation of the catalog database, we provide only lower level of subject hierarchies for *Computer Science.*

- Selecting the **Sub–level2** button will display the lower subject level terms for the current general level and sub-level1 entries. Figure 11 is the popup window for the Sub–level2 look–up when the General level is *Computer Science* and the Sub–level1 is *Information Systems.*

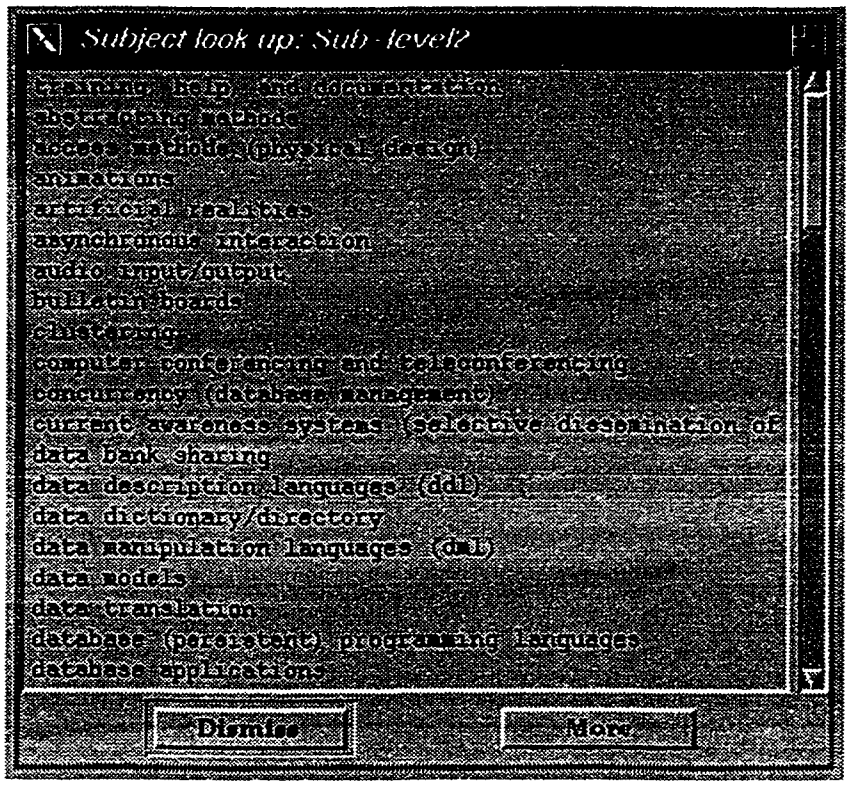

Figure 11: Subject look–up: Sub–level2

- If the user re–selects a subject level, the lower level(s) are cleared. He must re–select the lower subject level(s) if necessary.

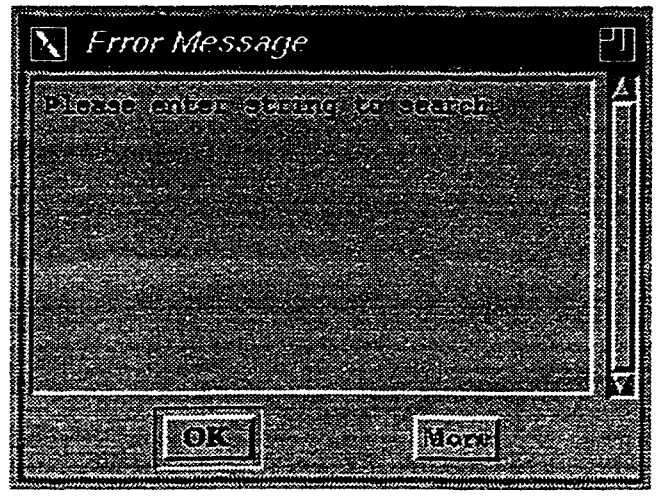

Figure 12: Search by Substring: Error message-1

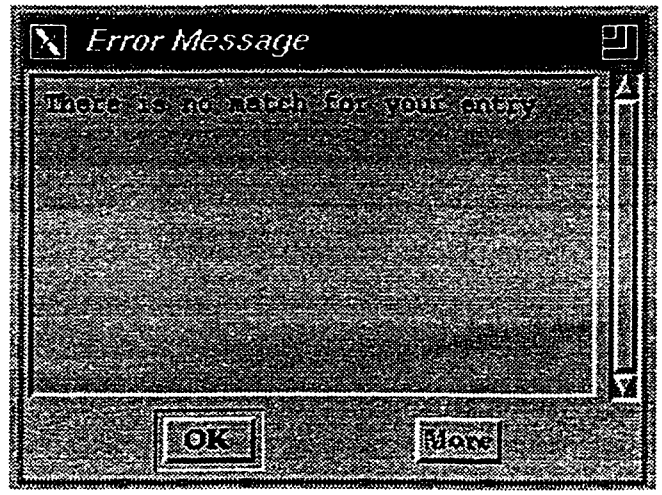

Figure 13: Search by Substring: Error message-2

Synonym resolution and substring entry rules

- To look for synonyms or sub–strings, a string must be entered in the search string entry field. If no string is entered, the system will pop up an error message (Figure 12).

- Looking for synonyms and substrings are separate operations. If **Synonyms** button is pressed, the system will look for synonyms of the string entered only. On the other hand, if **SubStrings** button is pressed, the system will look for terms in the thesaurus with substring matching the string entered by the user without looking for synonyms.

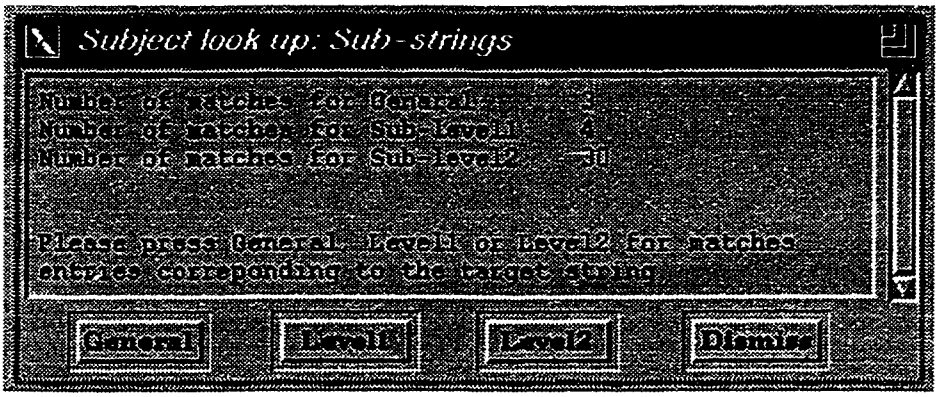

Figure 14: Example of sub–string look–up: search string 'com'

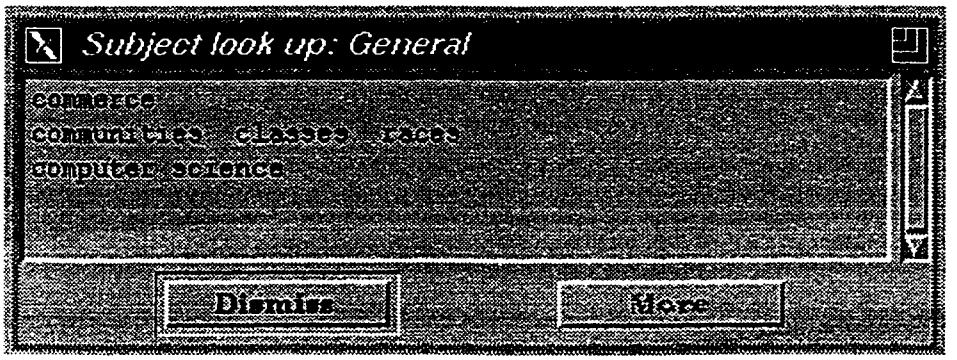

Figure 15: Example of sub–string look–up: display General level

- When **Synonyms** or **SubStrings** is pressed and the search string is not empty, search will be conducted by the expert system at the catalog database server. If no matching term is found, an error message will pop up (Figure 13). If matching terms are found, a popup window will display the number of matching items in each subject level (Figure 14). The user can then view the matching subject terms for each matching level (Figure 15,16,17) by selecting the corresponding push button.

- For synonym and sub–string look–ups, selecting a lower subject level will automatically select the higher parent level(s) for the selected entry. For example, if a user were to select Expert System as sub–level2, Artificial Intelligence and Computer Science will be the sub–level1 and the general level respectively.

The following example illustrates the mechanism: Suppose the user enters "com" for the search string. When **SubStrings** button is pressed, a popup window will display the number of matching items for each subject level (Figure 14).

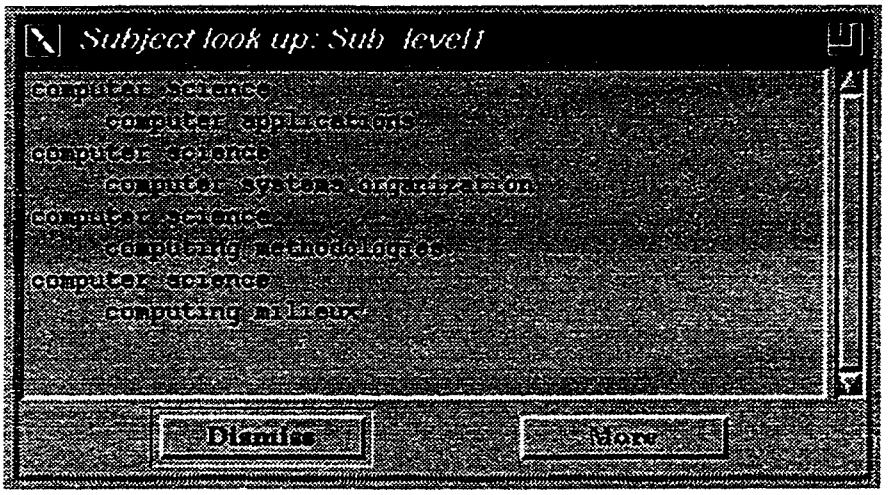

Figure 16: Example of sub-string look-up: display Sub-level1

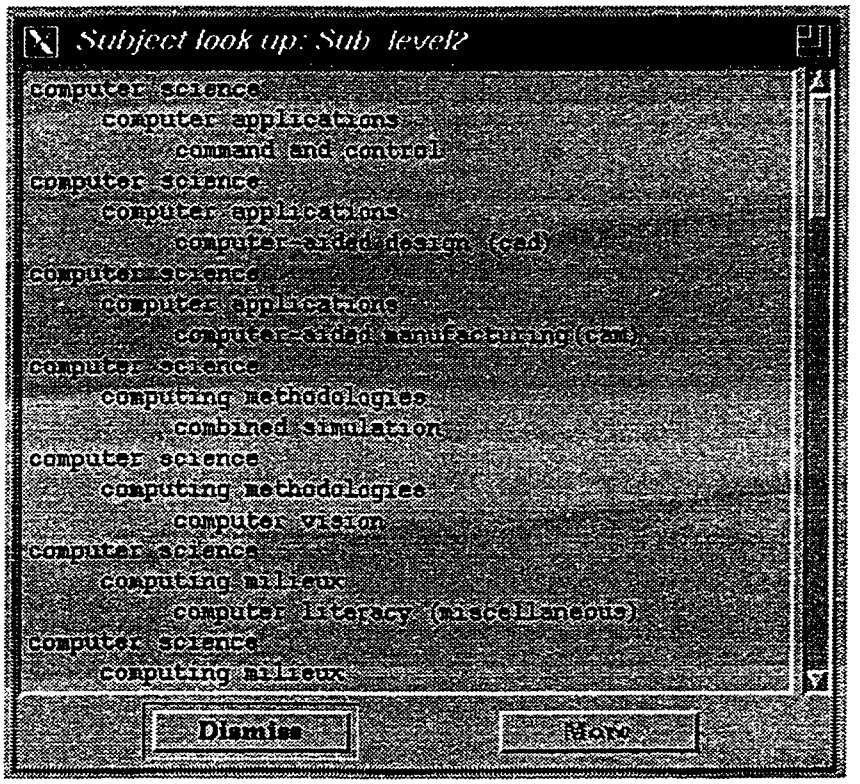

Figure 17: Example of sub-string look-up: display Sub-level2

- If the user selects button **General** a popup window will display the matching terms for general level (Figure 15). If the user selects an item from the item list in Figure 15, only the general field is filled up and any entry for lower levels will be cleared.

- Selecting **Level1** button will display the matching items for subject level1 (Figure 16). If the user selects an item from the item list in Figure 16, both the general level and sub–level1 are filled up and sub–level2 is cleared.

- Pressing on **Level2** button will popup a window displaying matching terms for subject level2 (Figure 17). The user can select any single item from the popup window. If you select an item from the item list in Figure 17, all the three subject levels will be filled up.

3.3 Other Parts of the GUI

As mentioned in Chapter 2, X Motif was used to implement the graphical user interface. The overall design of the Indexing/Registering Sub–system interface is shown in Figure 6. The screen dumps for the actual interface are shown in Figures 18, 19, 20. Since one single screen is not large enough to accomodate the whole interface, a scrollable window is used. The viewport may be adjusted by the user through the use of scrollbars that are attached to the scrolled window.

3.3.1 Pulldown Menus

Menu selection has many advantages. This includes shorter training time for new user, reduced keystrokes, rapid, accurate entry of items from standard choices. The user interface provides pulldown menus for the following entry fields:

- **Role:** A field for author information entry. Current available terms for selection are Author, Co-author, Editor, Artist, Corporate Entity, Designer, and Programmer.

- **Identifier Domain:** Currently available terms for selection are FTP. ISBN. ISSN, Gopher, HTTP, ULR, UAS, and ULN. This field is not editable, and the user has to choose from the pull down menu to select value for this field. Once a domain is chosen, the user must enter the associate value. For example, *http://www.cs.concordia.ca/w3-paper.html* is a value for HTTP domain, and *0-201-15790-X* is a value for ISBN domain etc.

- **Language:** The language in which the resource is written. Currently available languages for selection are: Arabic, Chinese, English, Farsi,French, German, Haryani, Hindi, Italian, Japanese, Korean, and Spanish. The user can also enter other languages from the keyboard.

- **Classification Domain:** Currently available terms for selection are Legal and Security level. The value for *Legal* could be copy right, or free to use and distribution etc. The value for *Security level* could be, personal, and public etc.

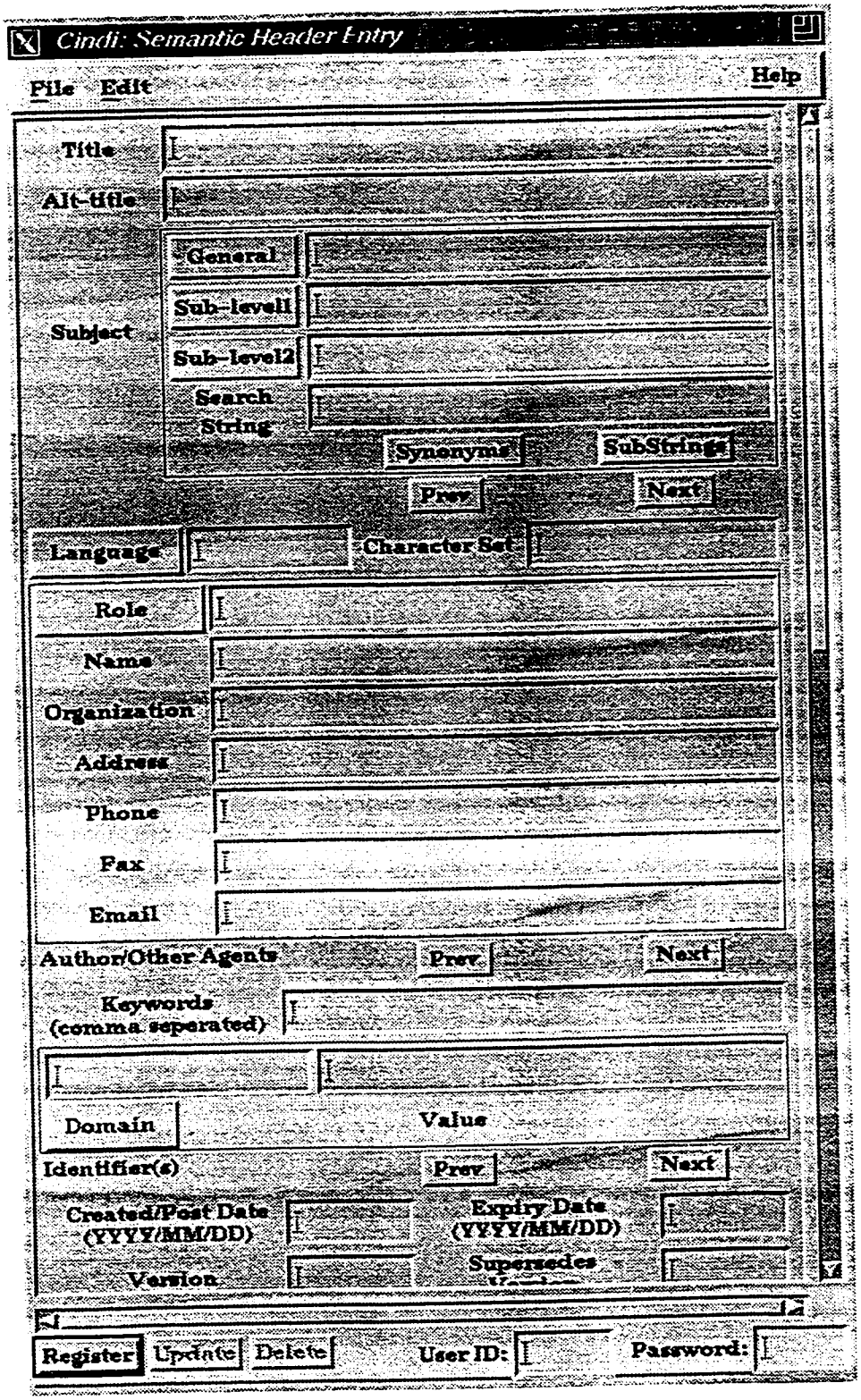

Figure 18: Semantic-Header Entry Interface: Page 1

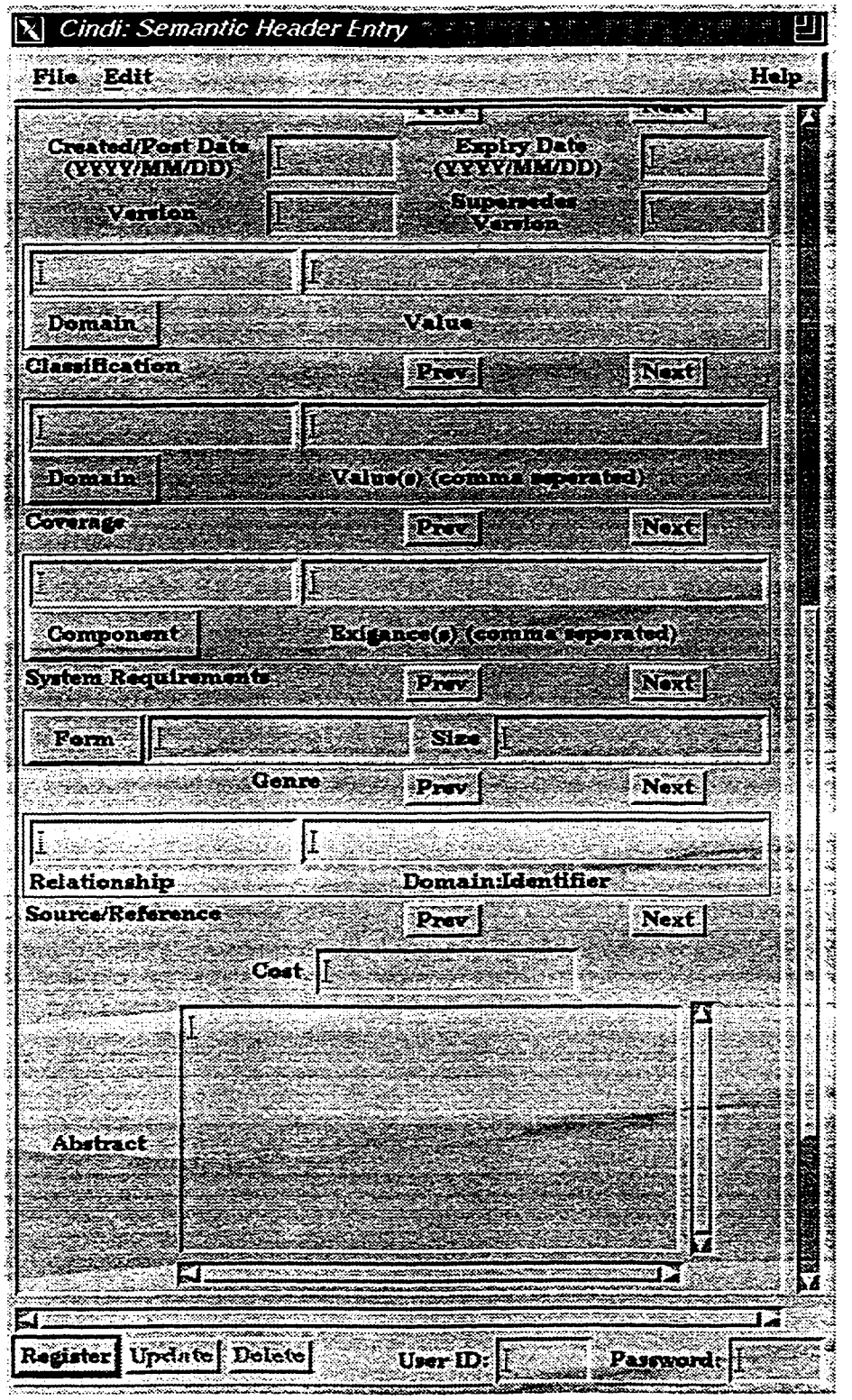

Figure 19: Semantic-Header Entry Interface: Page 2

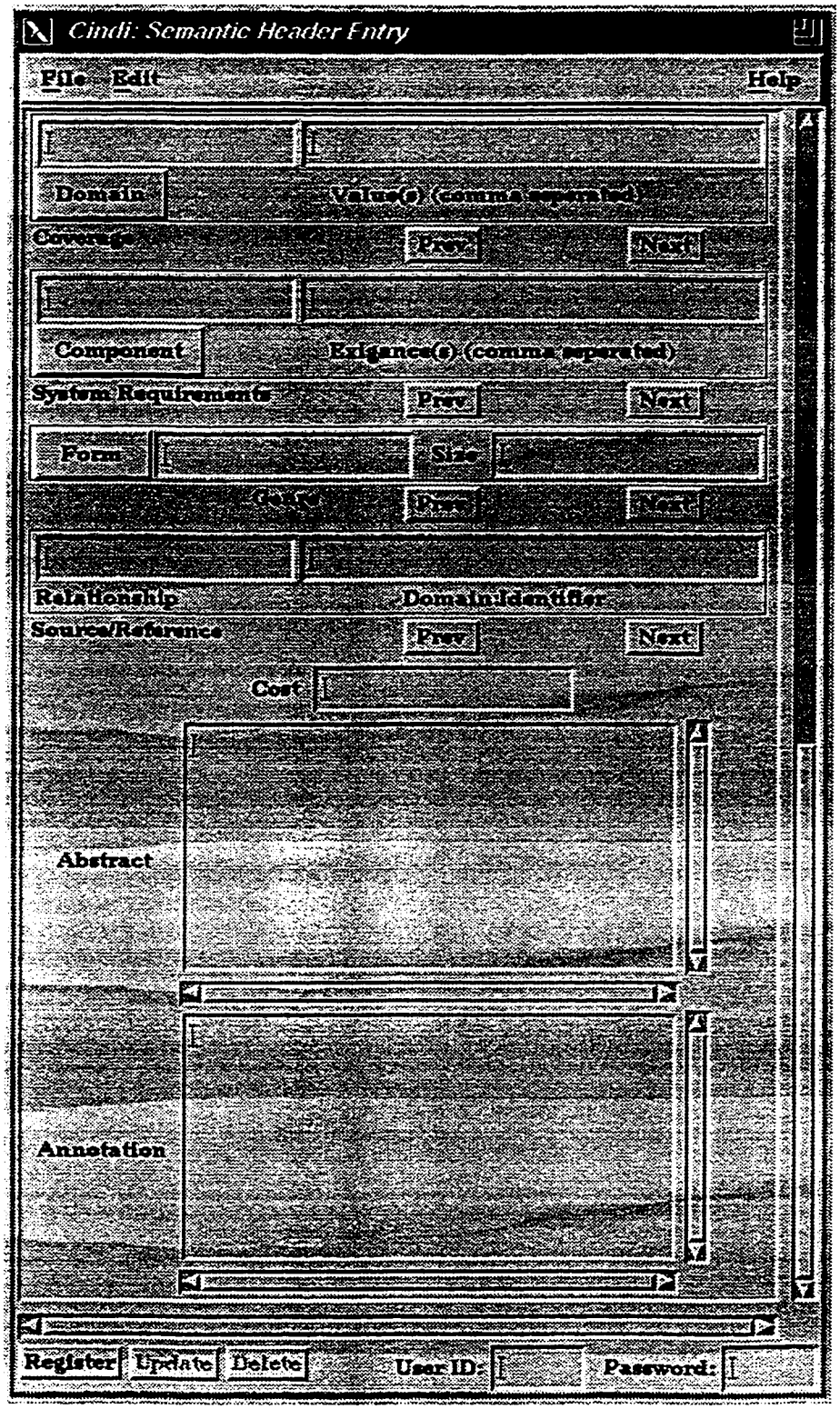

Figure 20: Semantic–Header Entry Interface: Page 3

- **Coverage Domain:** Audience coverage, Geographical coverage, Spatial coverage, Temporal coverage, and Epoch. These are standard predefined entry terms. The user cannot enter his own terms. The value for a chosen coverage domain must entered by the user. The value could be *Computer professional* for Audience, and *North America* for *Geographical* etc.

- **Component:** Components for system requirement are Hardware, Software, and Networks. This field is not editable. The corresponding exigance of a selected component must be entered.

- **Genre Form:** Currently available terms for selection are PS, Text, and Binary. The user can also enter his own values. The size for each selected Form must also be entered to complete the entry.

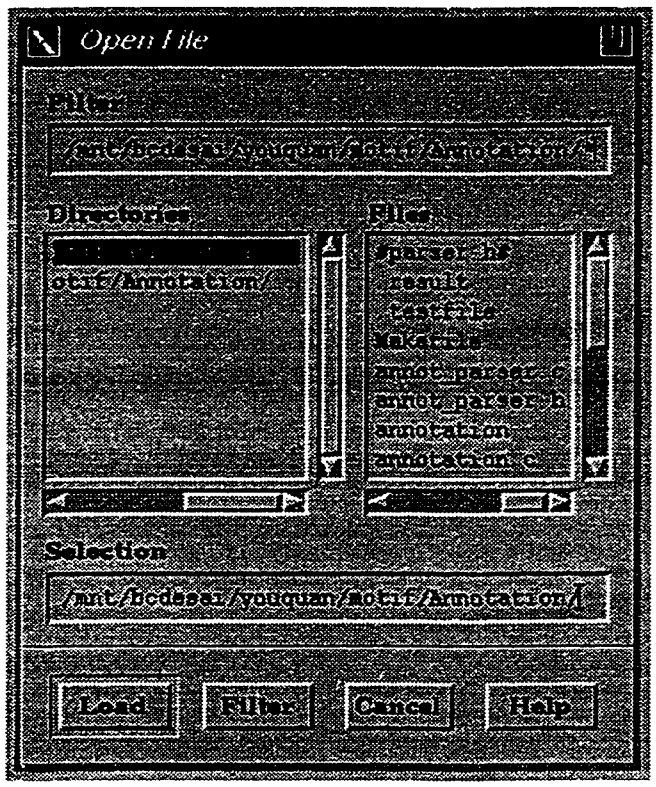

Figure 21: File selection popup: Open file

3.3.2 Functionalities of the Menu Bar

The menu bar of the graphical user interface contains the pull down menus **File**, **Edit**, and **Help**.

File Menu

The **File** pulldown menu contains the menu items **Open**, **Save**, and **Exit**. Selecting menu item **Open** will display a file selector (Figure 21) whereby a user can select a file containing an existing Semantic–Header. The file will be loaded into the Semantic–Header interface for further modification. Selecting menu item **Save**, invokes a file selector for file name entry (similar to Figure 21, with the title of *Save File*). After the file name is entered, the Semantic–Header on the entry screen is saved to disk by the local file system. Selecting **Exit** terminates the Semantic–Header entry application.

Edit Menu

The **Edit** pulldown menu contains the menu items **New Entry** and **Clear Field**. Selecting **New Entry** clears the Semantic–Header entry screen and allows the user to enter a new Semantic–Header. Menu item **Clear Field** is a cascade button which in turn contains menu items **Title, Alt-title, Subject, Author, Keyword, Identifier, Classification, Coverage, System requirement, Genre, Source/Reference, Abstract,** and **Annotation**. Selecting a menu item will clear the corresponding entry field or fields in the corresponding entry block. For example, selecting menu item **Title** will clear the title field, selecting **Subject** clears fields General, Sub–level1, Sub–level2 and Search String, and so on.

Help Menu

Currently, the **Help** pulldown menu contains the menu items **System Help, On Subject, On Author** and **On Keyword**. **System Help** provides online help for the whole Semantic–Header entry screen. **On Subject** provides online help concerning subject entry. **On Author** provides online help concerning author/other agents entry. **On Keyword** provides online help concerning keyword entry. An online help popup window example is shown in Figure 22.

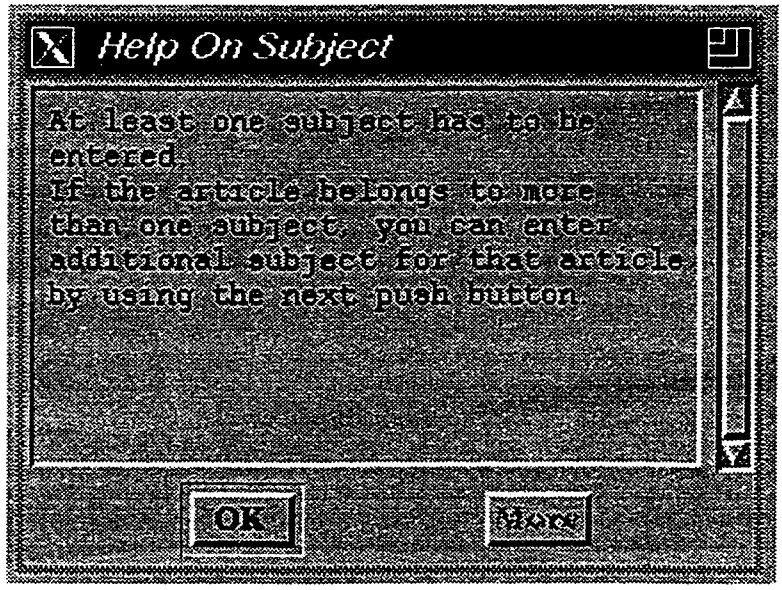

Figure 22: Online help popup window

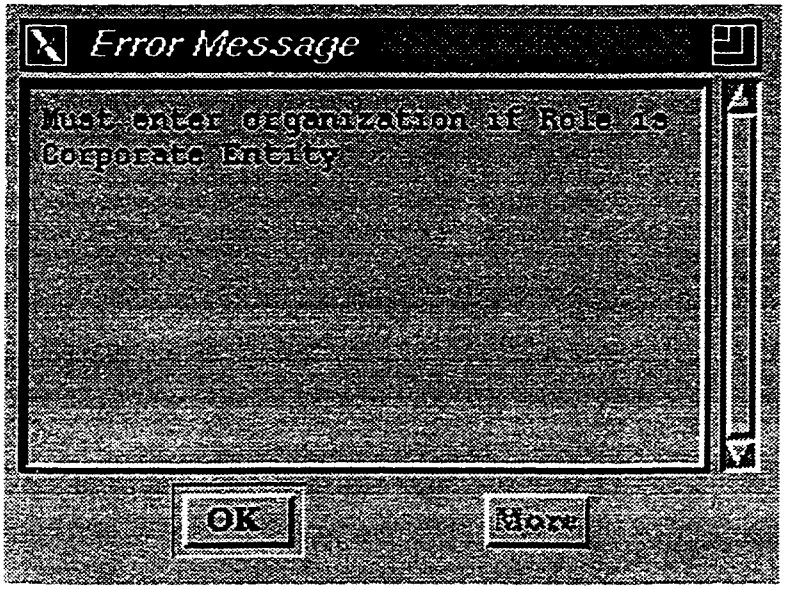

Figure 23: Error message: validate current entry

3.3.3 Repeatable Entry Fields

Some fields of the Semantic–Header may have multiple entries. They are called repeatable entry fields. The repeatable entry fields in the Semantic–Header are Subject, Author/Other Agents, Identifier, Classification, Coverage, System Requirements, Genre, and Source/Reference. For example, a document could be classified under more than one subjects, an article could be written by more than one authors, and the document can be identified by it's HTTP, FTP, and ISBN etc. We provide **Prev** and **Next** push buttons at the bottom of each block in the user interface to accommodate these entries.

The **Prev** push button allows the user to view or modify the previous entry of a block and The **Next** push button allows the user to view or modify the next entry of a block. To proceed to the previous or next entry, certain conditions applied:

1. If the current entry is the first entry and **Prev** is pressed, the system will generate an audio signal to remind the user.

2. The current entry must be a valid entry. Otherwise an error message appears to inform the user. For example, if the **Role** of an author is *Corporate Entity* and the **Organization** was not entered, an error message will pop up (Figure

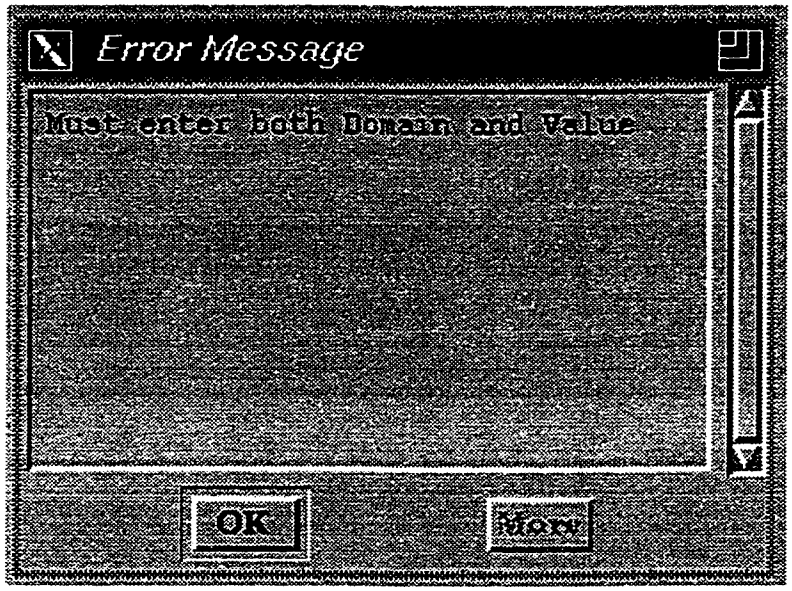

Figure 24: Error message: validate current entry

23).

3. If the current entry is the last entry, the user can clear all the fields in the block and proceed to the previous entry. Otherwise a validation will be done before proceeding to the previous entry.

4. The user cannot leave a set of blank fields for a block and proceed to the next or previous entry, except for case 3 above.

5. For Identifier, Classification, Coverage, System Requirements, and Genre entry, none of the fields in a block can be left blank prior moving to the next or previous entry, except for case 3 above. Otherwise an error message appears to inform the user (Figure 24).

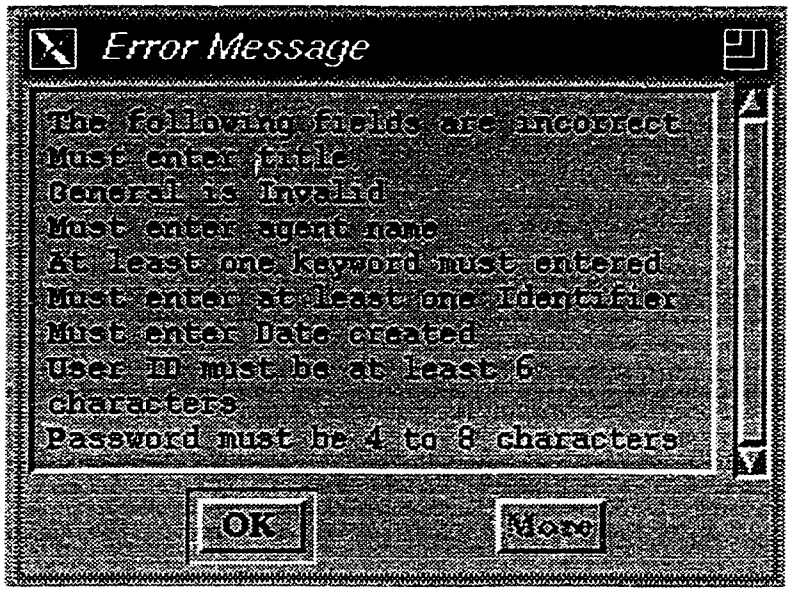

Figure 25: An error message popup for register

3.3.4 Functionalities of Action Bars

The action bar at the bottom of the entry interface contains three action buttons (**Register, Update** and **Delete**). The functionalities of these buttons are described below:

Register

The **Register** push button allows users to register the current Semantic–Header into Semantic–Header Database. To register a new Semantic–Header, an user ID. and password are required. When this button is pressed, the corresponding callback function will be involved. The callback function perform the following operations:

1. Validates the Semantic–Header entry: Before an actual registration is made, the callback function will check the semantic header entry to make sure all the required fields are entered. These fields are: title, at least one subject hierarchy, at least one keyword, at least one entry for an author, at least one identifier, the creation date , user ID, and password. If any of the required fields is missing, an error message is displayed (Figure 25). This validation ensures that the standard indexing scheme is enforced.

2. The client program sends the Semantic–Header to the server: If the semantic header entry passes validation, the callback function next involves the client program (at the user's workstation) which formats the Semantic–Header according to the standard scheme presented in section 3.1 and sends it to the Semantic Header Register function at the semantic header database (the server) through the TCP/IP protocol [Car93, San94].

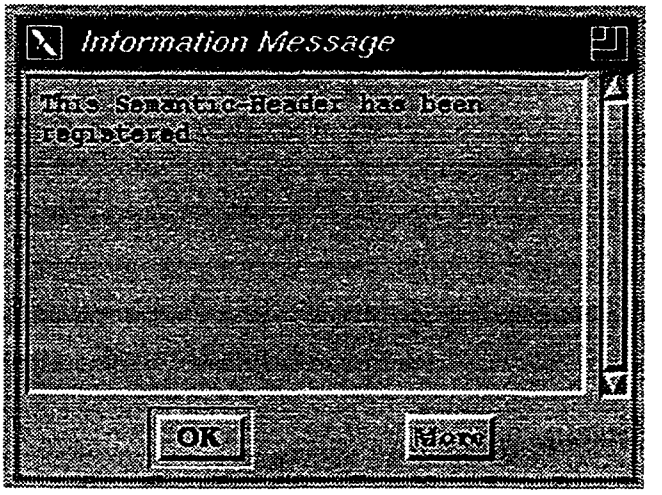

Figure 26: Information message: Register was done

3. The server registers the Semantic–Header: The Semantic Header Register function at the semantic header database decodes the formatted Semantic–Header and registers it into the database. A response code will be sent to the client indicating the status of the registration process.

4. The client informs the user by a popup window displaying appropriate message based on the response code received from the server. The response codes and the corresponding messages are as follows:

   - Code 1: This Semantic-Header has been registered (Figure 26).
   - Code 2: User ID already exist and Password does not match! The Semantic-Header cannot be registered.
   - Code 3: This Semantic-Header is in the database. It cannot be registered again.
   - Code 100: Data transmission error occurred. Please try later.

The **Register** push button will be disable when one open an existing semantic header for modification. If a Semantic–Header does not contain a UserID/Password pair, it is an existing Semantic–Header in the database. Otherwise, the Semantic–Header has not been registered and the **Register** button will be enable.

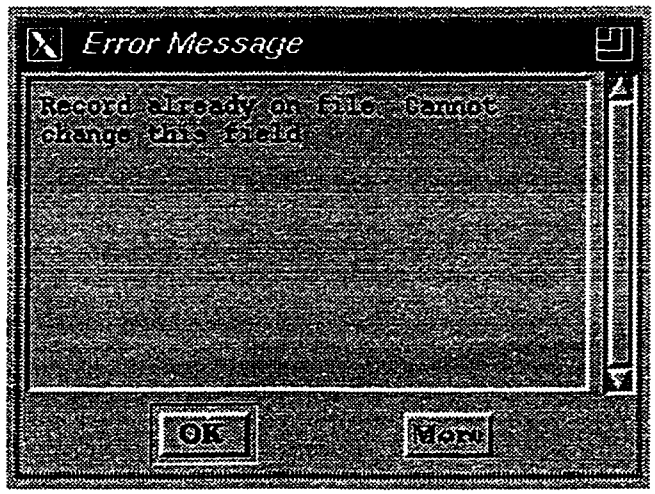

Figure 27: Error message: Cannot change SHN

Update

The **Update** push button allows users to update an existing Semantic–Header. This Semantic–Header is loaded into the GUI through the use of the **Open** menu item in the **File** menu. The Semantic–Header could have been retrieved from the Semantic–Header database through the Search sub–system and is saved through the display screen (As outlined in Chapter 4).

If a Semantic–Header has been registered, the user is not allowed to change entry fields corresponding to the Semantic Header Name (title, first author's name, first author's role, first subject, date created, and version number) and the annotation. The expert system will make these fields non–editable and provide appropriate warning messages if the user attempts to modify these fields (Figure 27).

To update a Semantic–Header, the user has to enter an UserID/Password pair that matches the pair entered when the Semantic–Header was registered. When this button is pressed, the corresponding callback function will be involved. The callback

Figure 28: Error message: Mismatch UserID/Password pair

function perform the following operations:

1. Validates the Semantic–Header entry: Before an actual update is made, the callback function will check the semantic header entry to make sure all the required fields that are not part of the Semantic Header Name are entered. These fields are: at least one keyword, user ID, and password. If any of the required fields is missing, an error message is displayed (Figure 25).

2. The client program sends the Semantic–Header to the server.

3. The server update the Semantic–Header: The Semantic Header Update function at the semantic header database uses the Semantic Header Name to locate Semantic–Header and update it. A response code will be sent to the client indicating the status of the update process.

4. The client informs the user by a popup window displaying appropriate message based on the response code received from the server. The response codes and the corresponding messages are as follows:

   - Code 11: This Semantic-Header has been updated.
   - Code 12: UserID/Password pair does not match. This Semantic-Header cannot be updated (Figure 28).
   - Code 100: Data transmission error occurred. Please try later.

The **Update** button is disabled when a new Semantic–Header is being entered.

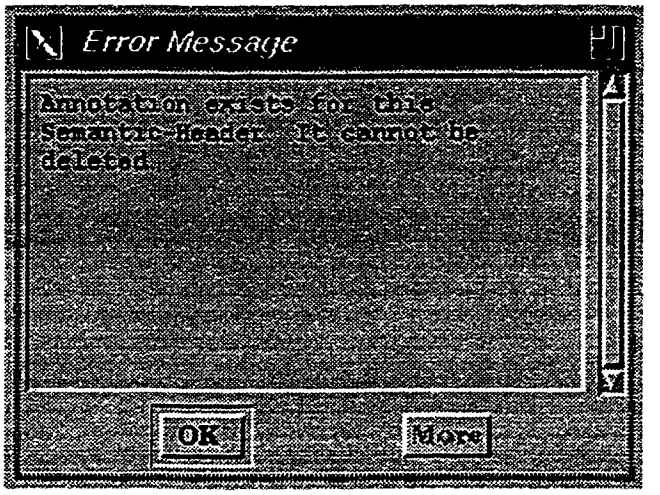

Figure 29: Error message: Cannot delete SH

Delete

The **Delete** push button is to remove a Semantic–Header from the system. A Semantic–Header cannot be deleted if it contains other viewer's annotations.

To delete a Semantic–Header, the user has to enter an UserID/Password pair that matches the pair entered when the Semantic–Header was registered. When this button is pressed, the corresponding callback function will be involved. The callback function perform the following operations:

1. The client program sends the Semantic–Header Name to the server.

2. The server delete the Semantic–Header: The Semantic Header Delete function at the semantic header database uses the Semantic Header Name to locate Semantic–Header and delete it. A response code will be sent to the client indicating the status of the deletion process.

3. The client informs the user by a popup window displaying appropriate message based on the response code received from the server. The response codes and the corresponding messages are as follows:

   - Code 21: This Semantic-Header has been deleted.
   - Code 22: UserID/Password pair does not match. This Semantic-Header cannot be deleted.

- Code 23: This Semantic-Header does not exist in the database.
- Code 24: Annotation exists for this Semantic-Header. It cannot be deleted (Figure 29).
- Code 100: Data transmission error occurred. Please try later.

The **Delete** button is disabled when the user enter a new Semantic–Header.

Chapter 4

Search Sub–system

The Search Sub–system incorporates the expertise used by the typical reference librarian. This expertise guides the user in entering the various search items in a graphical user interface. The query from the user could be *Known* or *Unknown*. *Known* implies that the user knows the exact title, author, or identifier of the resource. *Unknown* implies that the user may have at most a vague knowledge of the subject, title, author, or identifier of the resources.

The overall design of the graphical user interface of the search sub–system is shown in Figure 30. The search query entry area contains the following blocks/fields: title/alt–title, subject, author/other agents, identifier, keywords, date range, language, version, max hits, and words in abstract. The user has options to search by the following key terms: title, subject, author, keywords, or words in abstract. He can also search by the combination of two or more key terms. The user can select an associate radio button (*Exact, Substr/noncase*, or *Like*) for title, author, and identifier to focus the search. For repeatable entry blocks such as subject, author, identifier, and keyword, the user can express the query entry in mathematical *Infix* format by using radio buttons *And* and *Or* as well as push buttons *Prev, Next, open bracket '(',* and *close bracket ')'*. Detail discussion about these buttons will be given in Section 4.3. We provide context–sensitive help for subject entry and pulldown menu selection for identifier domain entry and language entry. The user can also retrieve document published within a period of time by specifying the posted date of the documents that are *After* and/or *Before* a specific date.

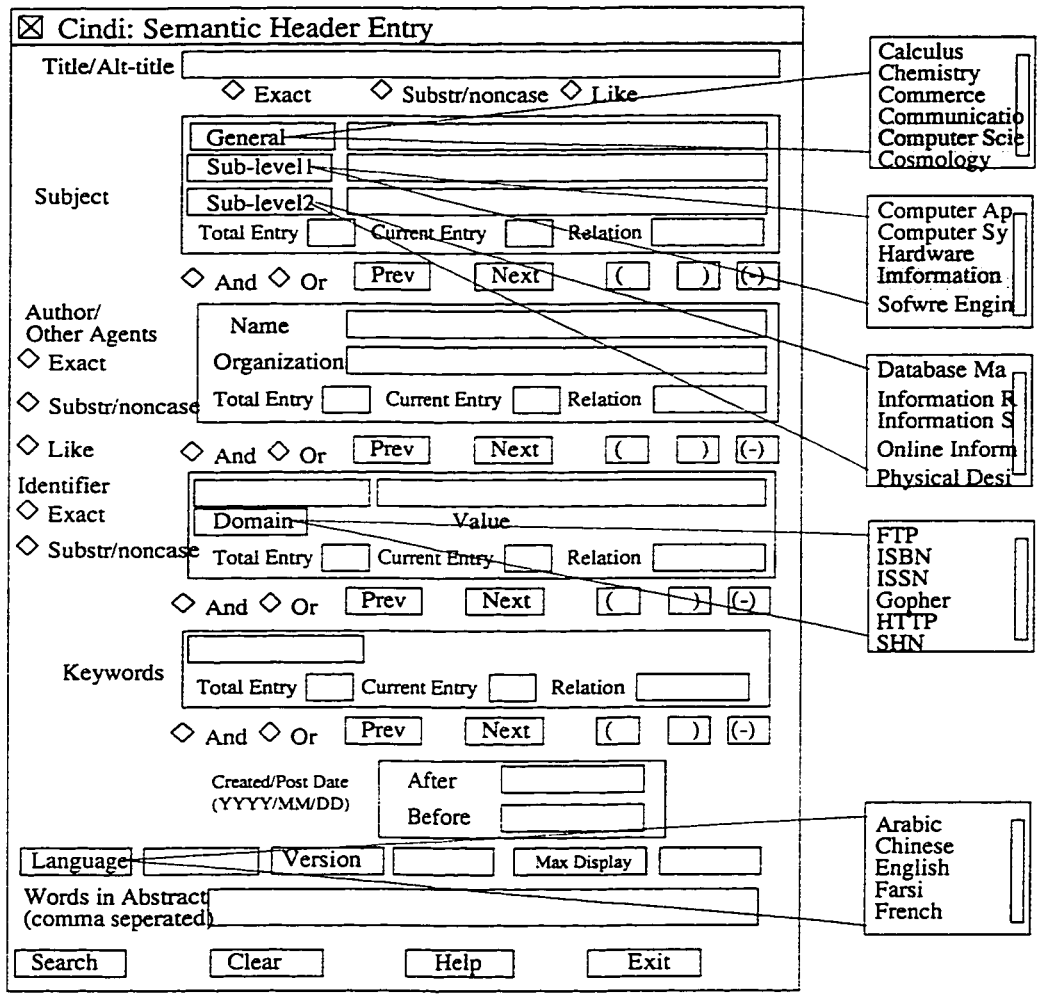

Figure 30: Search Interface: Design

Once the user has entered a search request, the client process communicates with the nearest SHDDB catalogue to determine the appropriate site of the SHDDB database. Subsequently, the client process communicates with this database and retrieves one or more Semantic–Headers. The results of the query can then be collected and sent to the user's workstation. The contents of these headers are displayed, on demand. If the user can access one or more of the actual on–line resources by connecting to a browser, currently connecting to Netscape browser is implemented. It may happen that the item in question may be available from a number of sources. In such a case the most cost effective choice may be selected. The client process attempts to use appropriate hardware/software to retrieve the selected resources.

4.1 Search Query Structure

4.1.1 Query Structure

After the user has entered the query via the GUI, the query information must be formatted, and sent through the client to the server. Consequently, both the client and the server must agree on the same grammar. Our search query format is similar to that of the Semantic-Header's. Each entry value or block begins and ends with a tag. The search query grammar written in BNF [GM86] is as follows:

search := "<search>" content "</search>"
content := current_block_no max_return_result title title_status
list_subject list_author list_identifier list_keyword
date_after date_before language version word_in_abstract
current_block_no := "<blockno>" 1|2|3|... "</blockno>"
max_return_result := "<nosh>" 10|20|50|100 "</nosh>"
title := "<title>" ... "</title>"
title_status := "<sts>" ... "</sts>"
list_subject := "<subject>" subject_item$^+$ "</subject>"
subject_item := general sublevel1 sublevel2 operator bracket num_of_bracket
general := "<general>" ... "</general>"
sublevel1 := "<sublevel1>" ... "</sublevel1>"
sublevel2 := "<sublevel2>" ... "</sublevel2>"
operator := "<oper>" ... "</oper>"
bracket := "<brck>" [...] "</brck>"
num_of_bracket := "<brckN>" ... "</brckN>"
list_author := "<author>" author_item$^+$ "</author>"
author_item := name organization operator bracket num_of_bracket
author_status
name := "<aname>" [...] "</aname>"
organization := "<aorg>" [...] "</aorg>"
operator := "<oper>" ... "</oper>"
bracket := "<brck>" [...] "</brck>"
num_of_bracket := "<brckN>" ... "</brckN>"

author_status := "<sts>" ... "</sts>"

identifier := "<identifier>" iden_item^{+}"</identifier>"

iden_item := domain value operator bracket num_of_bracket iden_status

domain := "<iden-domain>" [FTP|ISBN|ISSN |Gopher|HTTP|ULC|USA|ULN] "</iden-domain>"

value := "<iden-value>" [...] "</iden-value>"

operator := "<oper>" ... "</oper>"

bracket := "<brck>" [...] "</brck>"

num_of_bracket := "<brckN>" ... "</brckN>"

iden_status := "<sts>" ... "</sts>"

list_keyword := "<kw>" keyword_item^{+} "</kw>"

keyword_item := keyword operator bracket num_of_bracket

keyword := "<keyword>" [...] "</keyword>"

operator := "<oper>" ... "</oper>"

bracket := "<brck>" [...] "</brck>"

num_of_bracket := "<brckN>" ... "</brckN>"

date_after := "<dateAft>" [...] "</dateAft>"

date_before := "<dateBef>" [...] "</dateBef>"

language := "<languag>" ... "</language>"

version := "<version>" ... "</version>"

word_in_abstract := "<abstract>" ... "</abstract>"

4.1.2 Description of the Structure

A search query begins with the tag *<search>* and ends with the tag *</search>*. The content of the search query contains currently desired block, the maximum number of search hits returned, the title, a list of subjects, a list of author/other agents, a list of identifiers, a list of keywords, date after, date before, language, version, and word in abstract. For standardization purpose, we tag every field of the search query.

- Maximum number of hits to be returned is tagged by *<nosh>* and *</nosh>* which indicates the number of matching Semantic-Headers the user wants the

SHDB system to return. From the user interface, the user has the option of selecting 10, 20, 50, 100. The default number of hits to be returned is 10.

- The current block desired is tagged by *<blockno>* and *</blockno>*. This field is used by the server for searching purpose only. The user are not aware of the current block number, but he/she can view the previous or next block of Semantic–Headers by pressing the **Prev Block** or **Next Block** push buttons on the interface respectively. Based on the current block wanted and the number of hits to be returned, the search system at the SHDB will decide which matching Semantic–Headers should be returned to the user. The matching Semantic–Headers are sorted by their titles in ascending order and are numbered as 1, 2, 3 etc. The SHDB uses the following formula to determine which Semantic–Header should be sent back to the user. The Semantic–Headers returned would be number S to E, where

  $$S = (\text{current block } -1) * (\text{max number of SH to be displayed}) + 1$$

  $$E = (\text{current block}) * (\text{max number of SH to be displayed})$$

  Suppose the maximum number of hits is 10 and total number of search hits is 25. If the current block wanted is 1, the system will return Semantic–Header numbers 1 to 10; if the current block wanted is 2, the system will return Semantic–Header numbers 11 to 20 etc.

- The **Title** block contains two fields. The first field is the title itself, which is tagged by *<title>* and *</title>*. The second field is the status of the entered title, which is tagged by *<sts>* and *</sts>*. Possible values for status could be 'e', 's', or 'l'. The meaning of the these values are: 'e' – exact entry, the user is sure about the Semantic–Header's title; 's' – substring entry, the user is not sure about the title of the Semantic–Header and gives only a sub–string of the title; 'l' – vague entry, the user gives only something similar to the actual title.

- The **Subject** block is repeatable. It contains fields for general, sublevel1, sublevel2, operator, bracket, and number of brackets. Each subject level is also enclosed by tags and contains either an empty string or a standard subject

term, which would be looked-up from the Catalog Database. The operator is tagged by *<oper>* and *</oper>*. The value for the operator is either '|' (logical OR) or '&' (logical AND), which indicates the logical relationship between the current subject_item and the next subject_item if there is any. The bracket field is tagged by *<brck>* and *</brck>*. The bracket value could be an open bracket '(', close bracket ')' or space. The number of brackets is tagged by *<brckN>* and *</brckN>*. It indicates the number of open brackets or close brackets. In the case of space for bracket, this field is ignored.

- The **Author** block is repeatable. Each author_item contains fields for name, organization, operator, bracket, number of brackets and author status six fields. Tagged by *<aname>* and *</aname>*, the name field is the author's name. Organization, tagged by *<aorg>* and *</aorg>*, is the author's organization or the organization that posted the Semantic-Header sought. The author_status tagged by *<sts>* and *</sts>* could be 'e', 's', or 'l'.

- The **Identifier** block is also repeatable. Each identifier item contains fields for domain, value, operator, bracket, number of bracket, and identifier status six fields. *Domain* is the standard identifier of a Semantic-Header, which could be selected from a popup menu. *Value* is the value of the given domain. The iden_status tagged by *<sts>* and *</sts>* could be 'e', 's'.

- The **Keywords** block is repeatable. Each keyword item contains fields for the keyword, operator, bracket, and number of brackets.

- The field **Date after** indicates that the user seeks Semantic-Headers created or posted after the specified date.

- The field **Date before** indicates that the user seeks Semantic-Headers created or posted before the specified date.

- The field **language** indicates that the user seeks Semantic-Headers written in the specified language.

- The field **Version** indicates that the user wants the specified version of a Semantic-Headers.

- The field **Word in abstract** indicates that the user seeks Semantic–Headers with the specified word in their abstract.

4.1.3 An Example

Here is an example of the search query formulation.

<search>
<blockno> 1 </blockno>
<nosh> *10* </nosh>
<title> *search* </title>
<sts> s </sts>
<subject>
<general> *computer science* </general>
<sublevel1> *information systems* </sublevel1>
<sublevel2> *search process* </sublevel2>
<oper> </oper>
<brck> </brck>
<brckN> *0* </brckN>
</subject>
<author>
<sts> *s* </sts>
<aname> *desai* </aname>
<aorg> *concordia university* </aorg>
<oper> & </oper>
<brck> (</brck>
<brckN> 1 </brckN>
<sts> s </sts>
<aname> *shighal* </aname>
<aorg> *concordia university* </aorg>
<oper> </oper>
<brck>) </brck>
<brckN> 1 </brckN>
</author>

<identifier>
<sts> </sts>
<iden-domain> </iden-domain>
<iden-value> </iden-value>
<oper> </oper>
<brck> </brck>
<brckN> </brckN>
</identifier>
<kw>
<keyword> *information technology* </keyword>
<oper> | </oper>
<brck> </brck>
<brckN> </brckN>
<keyword> *hypermedia* </keyword>
<oper> </oper>
<brck> </brck>
<brckN> </brckN>
</kw>
<dateAft> *1994/01/01* </dateAft>
<dateBef> </dateBef>
<languag> *English* </language>
<version> </version>
<abstract> </abstract>
</search>

According to this query, the user wishes Semantic–Headers with **search** as a sub–string of their titles. These Semantic–Headers would in the subject of **computer science** as general level, **information systems** as sub–level1, and **search process** as sub–level2; having keywords: **information technology OR hypermedia**; the authors of the resources should be **desai** and **shinghal** at concordia university; posted after **1994/01/01** and written in the **English** language.

4.2 Expert Assistance for Searching

The guiding principle in the design of the search system is the model of a human reference librarian. S/he is called on to help in identifying the best sources of information for a given purpose and to aid in the selection of material to meet a particular interest or need. The reference librarian seeks a response to a query by using information derived from bibliographic search processed through the librarians own expertise and knowledge of the relevant subject. In addition, users of a library have access to the same bibliographic indices and many information databases and can use these to select relevant titles or weed out irrelevant ones.

The expert system should not be a significant overhead. Thus, we were considered to keep the expert system small. Though several expert system shells such as CLIPS [GR93] facilitate knowledge engineering and rule encoding, they incurred a significant overhead when integrated into our overall system owing to the following reasons [CSDR]:

- The expert system shell had to be embedded in the overall system because rule firings are controlled only by the inference engine. This increased the size of the overall system.
- The inference overhead was at times unacceptable. For example, for every rule firing, if a system such as CLIPS were used, its inference engine would recompute the set of rules that can fire.
- The working memory of an expert system directly affects its performance. Typically, several hundred objects can qualify as targets by a search query. These intermediate findings are needed since a user may wish to pursue the documents discovered by a subset of the query parameters. If this information accumulates in the working memory, the performance of the expert system slows down. This adds to the size of the rule base, because we also need additional "clean up" rules to periodically clean the working memory from unwanted or outdated information.
- Input entered by a user and stored in data structures should be converted into appropriate formats for insertion into the working memory before the rules

can fire. Later, rule inferences in the form of atoms should be parsed and mapped back into data structures. This increases the development effort, and the overhead of using the expert system.

- A rule application context will be exercised using a rule (or, a set of rules) instead of putting all rules in a rule base. This will improve the performance of the system.

Once the rules were designed, we encoded them directly as C functions, thus implicitly implementing the interface engine as part of the system. This also avoided the overhead of embedding an external shell into our overall system. Using this approach, the skill of a reference librarian was distributed throughout the system rather than centralized in one place. This improved performance, because only a small number of rules fired at any time.

A specific search and search type query may require the user to peruse a number of titles and select from among them. This type of query involves users who have fuzzy notions of their needs and whose questions are often vague [DS96]. They involve a certain amount of trial and error in the retrieval and browsing of documents.

In the current search system, we incorporate the elementary expertise used by a reference librarian. Reference librarians are aware of the conventions used by cataloging librarians. They are conversant with the classification schemes, terms, indices, structures, and resources available for a user's particular need. The expertise of the librarians should be replicated to assist the users of our application in discovery. This expertise will guide the user in entering the various search items in a graphical user interface similar to the one used by the registering sub–system (Figure 6). The appearance of the interface is shown in Figure 31. The expert search sub–system requires the expertise of a reference librarian to be built into help users formulate queries and launch these queries. The system was designed so that its query for document search facilitates efficient database access, and reduces the number of incorrect results generated. This requires several aspects to be considered as part of its function [CSDR].

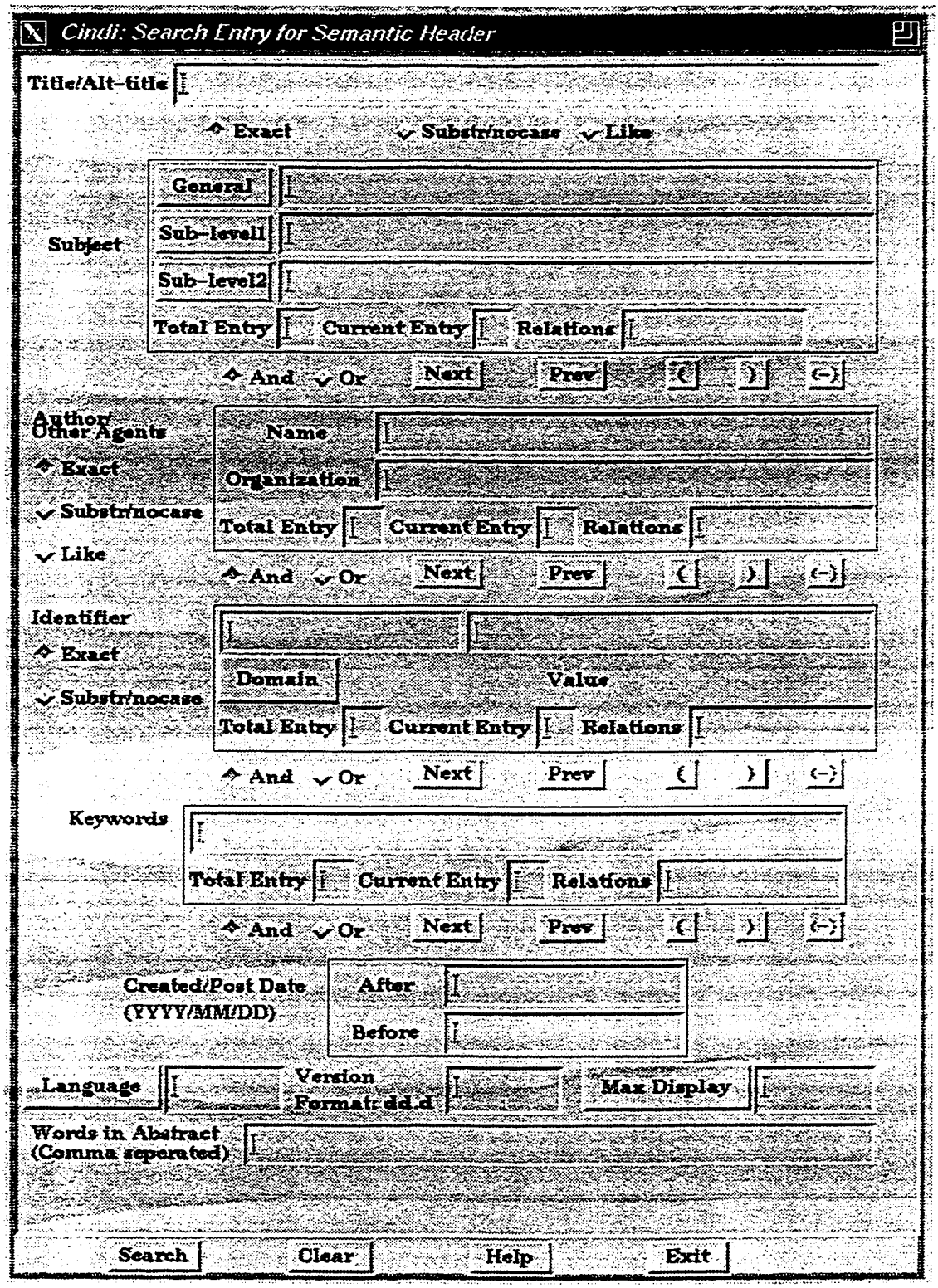

Figure 31: Search Interface: Screen dump

4.2.1 Subject Entry

The user can seek pertinent information source by subject. Since the user is often not familiar with subject classification, the expert system provides context–sensitive help for entering the subject hierarchy entry. The ideas are the same as those presented in Section 3.2. For simplicity, we also make the subject fields ineditable in our current version of implementation. The user must rely on the context–sensitive help to complete the subject entry.

Since the user may not know exactly which subject he is looking for, we provide multiple entry for this block. The user can enter more than one subject hierarchies for his search query. The relationship among the entries is connected by logical **AND** or **OR**. They are formatted in mathematical Infix format.

4.2.2 Vague Search Entry

When the user makes search by title, author, or identifier, he may not know the exact title, the full author name or identifier values. For example, the system aids the user in completing a given field entry based on the contents of the other search fields. We provide radio buttons for most of the search entry fields. They are **Exact, Substr/nocase** and **Like**. These provide disjoint options for users to specify their query precisely and hence to minimize irrelevant information returned to the user. These radio buttons allow the user to specify the level of confidence of the current entry. They take effects only when the user enters a non-empty string in the field corresponding to the buttons. The default value is **Exact**.

Exact

The **Exact** radio button specifies the current entry should match exactly in the Semantic–Header on the target set:

- Selecting **Exact** for **Title/Alt–title** entry means the user want to view Semantic–Headers having exactly the title entered.
- Selecting **Exact** for **Author/Other Agents** entry means the user wants to view Semantic–Headers having the author Name/Organization entered.

- Selecting **Exact** for **Identifier** entry means the user wants to view the Semantic–Header with matching identifier entered in the field.

Substr/noncase

The **Substr/noncase** radio button specifies the current entry is a sub–string (case non-sensitive) of the corresponding field of the target Semantic–Headers:

- Selecting **Substr/noncase** for **Title/Alt–title** entry means the system should return all Semantic–Headers with a title or an alt–title which contains the sub–string entered in the title/alt–title field.
- Selecting **Substr/noncase** for **Author/Other Agents** entry means the system should return all Semantic–Headers with the Name/Organization fields which has the entered sub–string.
- Selecting **Substr/noncase** for **Identifier** entry means the system should return all Semantic–Headers having the value entered as a sub–string for the domain identifier.

Like

The **Like** radio button specifies the current entry is similar to the corresponding field of the Semantic–Headers wanted. Rules for **Like** have not been implemented in the current version.

- Selecting **Like** for **Title/Alt–title** entry means the user wants the system to return all Semantic–Headers with a title/alt-title similar to the search string entered in the **Title/Alt–title** field.
- Selecting **Like** for **Author/Other Agents** entry means the user wants the system to return all Semantic–Headers with Authors' name/organization similar to the search string in this field.

4.2.3 Query Entry for Repeatable Fields

In the search interface, we provide multiple entries for the subject hierarchy entry, author/other agents entry, identifier entry, and keywords entry. To accommodate

multiple entries for a block, we provide radio buttons, **And** and **Or**, as well as push buttons **Prev, Next,** open bracket "(", close bracket ")", and delete bracket operator (-). The significant of each buttons is described as follows:

"And" and "Or" Radio Buttons

And and **Or** radio buttons represent logical operators **AND** and **OR** respectively. They are used to specify the logical relationship between the current entry of a block and the next entry of the same block. The operator **AND** has a higher priority than **OR**.

Prev and Next

The **Prev** push button allows the user to view or modify the previous entry of a block and The **Next** push button allows the user to view or modify the next entry of a block. To proceed to the previous or next entry, The same conditions as those stated in Section 3.3.3 applies.

"(" push button

The "(" push button allows the user to enter open bracket for the query expression. Each time this button is selected, an open bracket is added to the expression for the current block entry.

")" push button

The ")" push button allow the user to enter close brackets for the query expression. Each time the user select this button, a close bracket is added to the expression for the current block entry. If the current entry is the first entry, this button will be inactive.

(-) push button

The (-) push button allows the user to remove an open or a close bracket from the query expression. Each time this button is selected, a bracket is remove from the expression for the block unless there is no more bracket left for the current entry.

4.2.4 Infix Query Expression

The expert system interprets the user's multiple entries of a block into a mathematical Infix expression, so that the user can view the relationship among his entries. The Infix expression is displayed in the box labelled with **Relation** for each repeatable entry block in the form of "(A and B) or (C and D)". Where A, B, C, D represent the entries. "and" and "or" are the logical operators AND and OR respectively. The open and close brackets are added by pressing the "(" and ")" push buttons . The box labelled with **Total Entry** display the total number of entry for that block. The box labelled with **Current Entry** display the sequence number of the current entry.

How to Enter an Expression?

Here is an example showing how to enter a simple query for repeatable entry block.

> *Suppose the user wants to search for Semantic–Headers for resources written by "John Smith" or written by both "Larry Tom" and "Fled Boyal".*

The sequences of the entry could be:

1. Enter "John Smith" in the author **Name** field.
2. Select radio button **Or**.
3. Select push button **Next**. Now the Total Entry field will display "1"; the Current Entry field will display "2", and the Relation field will display "A".
4. Enter "Larry Tom" in the author **Name** field.
5. Select radio button **And**
6. Select push button **Next**. Now the Total Entry field will display "2"; the Current Entry field will display "3", and the Relation field will display "A or B".
7. Enter "Fled Boyal" in the author **Name** field.

If you select then **Next** or **Prev** push button, the Total Entry field will display "3" and the Relation field will display "A or B and C". the expression "A or B and C"

therefore represents "John Smith **OR** Larry Tom **AND** Fled Boyal".

To enter a more complicated query, Open/close brackets must be used. Here is another example.

> *Suppose the user want to search for Semantic–Headers for resources written by both "John Smith" and "Kim Chi" or written by both "Larry Tom" and "Fled Boyal".*

The sequences of the entry could be:

1. Select push button "(".
2. Enter "John Smith" in the author **Name** field.
3. Select radio button **And**
4. Select push button **Next**. Now the Total Entry field will display "1"; the Current Entry field will display "2", and the Relation field will display "(A".
5. Enter "Kim Chi" in the author **Name** field.
6. Select push button ")".
7. Select radio button **Or**.
8. Select push button **Next**. Now the Total Entry field will display "2"; the Current Entry field will display "3", and the Relation field will display "(A and B) or".
9. Select push button "(".
10. Enter "Larry Tom" in the author **Name** field.
11. Select radio button **And**
12. Select push button **Next**. Now the Total Entry field will display "3"; the Current Entry field will display "4", and the Relation field will display "(A and B) and (C".
13. Enter "Fled Boyal" in the author **Name** field.

14. Select push button ")".

To see the completed expression, just select **Next** or **Prev** push button. Once this is done the Total Entry field will display "4" and the Relation field will display "(A and B) and (C and D)". The expression "(A and B) and (C and D)" represents "(John Smith **AND** Kim Chi) **OR** (Larry Tom **AND** Fled Boyal)".

To change from open bracket to close bracket or vise versa, requires the removal of the current bracket until there is only one bracket left.

4.2.5 Validation of the Search Query

The composition of a search would involve formulating a search query based on the current value of the fields entered. The result of the query would be a set of semantic headers, possibly empty, matching the user search request. Before formulating a search query, however, additional checking is made. This is typical of the way a reference librarian would proceed to focus his/her search to identify a smaller number of documents; for example, asking the user to enter author information, title information, performing spelling correction, phonetic checks, and input checks such as mismatched brackets entry, etc.

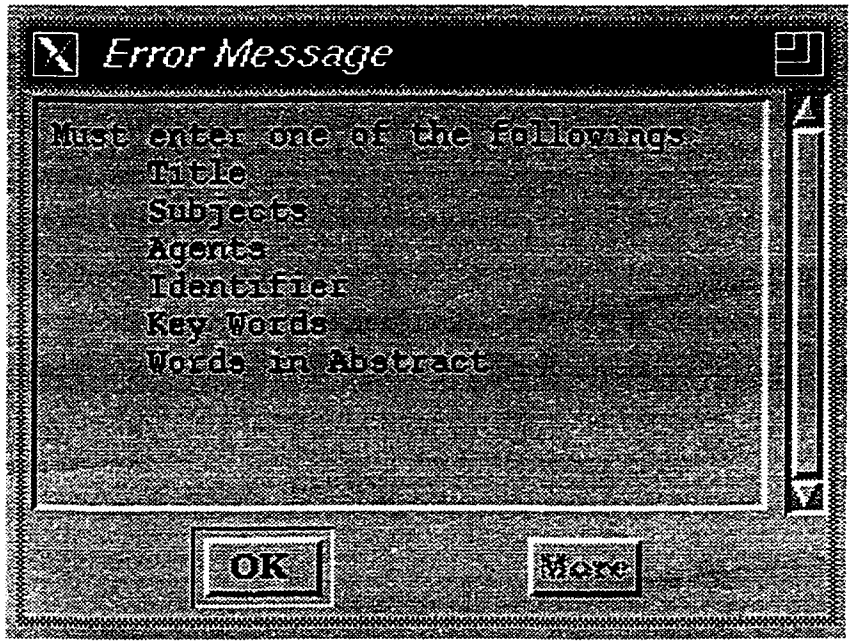

Figure 32: Error message: Search query checking - 1

When the **Search** push button is pressed, the system performs the following:

1. Validate the search query entry

   The query entered through the graphical user interface must be validated before it can be sent to the SHDB server for search. Two things are to be validated. The system check to make sure that at least one of the key terms has been entered (title, subject, author/other agent, keyword, identifier, and word in abstract). If none of the above is given, Error messages (Figure 32) is displayed. The system also checks to ensure that the brackets expressing a logical predicate for repeatable fields are properly matched. If there is a bracket mismatch, a

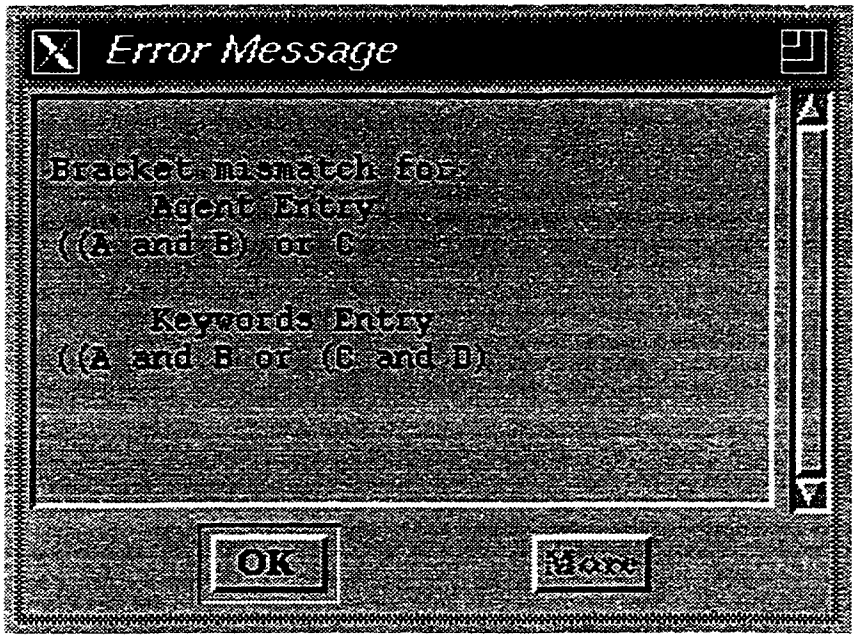

Figure 33: Error message: Search query checking - 2

popup window will appear (Figure 33). The user must correct the problem before the search process can proceed.

2. The client program sends the valid query string to the server
   As mentioned in Chapter 2, our system is a client–server application. Once the **Search** button is pressed and the search query has been verified, the client program will format the query and sends it to the server. The communication protocol is TCP/IP. The client will be initiated by the callback function of the **Search** button. The server is the Semantic–Header database server.

3. Checks the results returned by the server.
   If the result of the search query is 0 semantic header, then a a message , No document matches to your query were found, is displayed (Figure 34). If some Semantic–Headers are found for the query, a popup window appears displaying a list of the titles for the matching Semantic–Headers. The user is allowed to pick a subset of the retrieved documents for display of its full Semantic–Header one by one as outlined in Section 4.3.

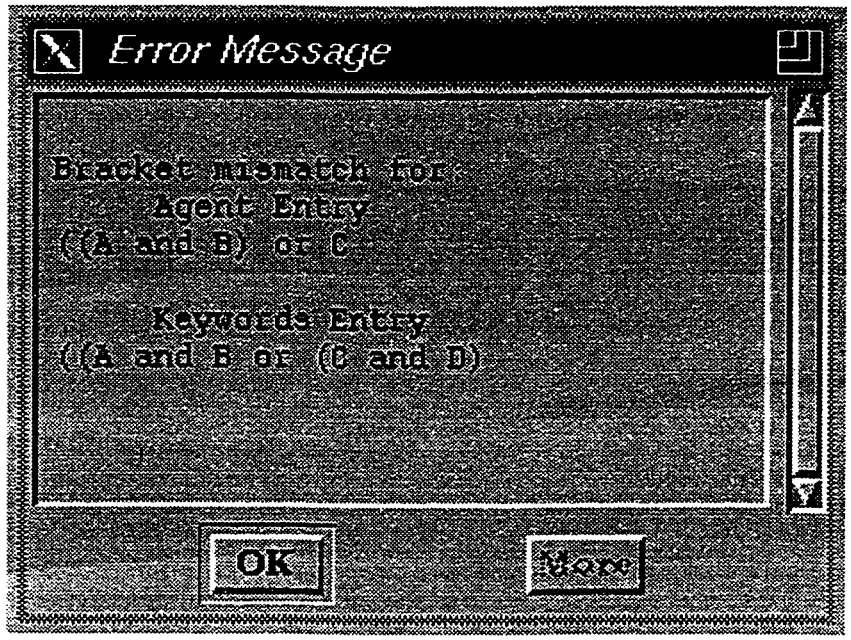

Figure 34: Error message: Search query checking - 2

4.3 Viewing the Semantic–Header

4.3.1 Overview

Depending on how many Semantic–Headers were matched for a search query, the system popups a window to display Semantic–Headers' titles for the first block of the matched Semantic–Headers (Figure 35).

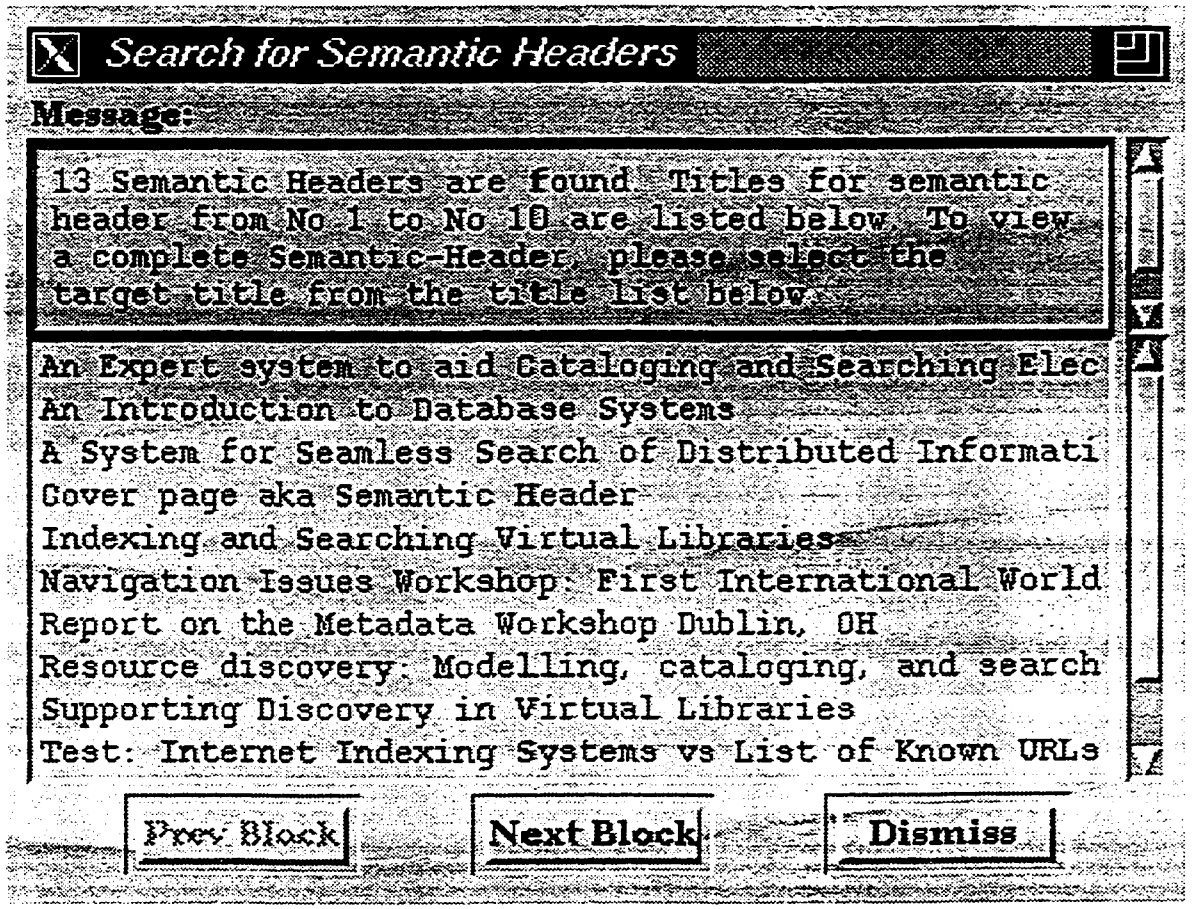

Figure 35: Search result popup: a list of titles

Figure 35 presents the result of a search by author name. The string entered for **Author Name** is *Desai* and **Substr/noncase** is selected. Since the block size is 10, only the titles for Semantic–Header numbers 1 to 10 is displayed. To view the titles of Semantic–Header numbers 11 to 20, just select the push button **Next Block**. And to view the titles for Semantic–Header numbers 1 to 10 again, just select push button **Prev Block**. Selecting **Dismiss** will remove the popup window. On Figure 35, the **Prev Block** button is inactive. This is because the current block is the first

block. Similarly, if the current block is the last block, the **Next Block** button will be inactive.

To view a complete Semantic–Header, simply click on the title for that Semantic–Header from the the title list in the pop up window. When this is done, the corresponding Semantic–Header will be displayed in the display window (Figures 36, 37,38, 39).

A Semantic Header Access Counter (SHAC) is kept along with the Semantic-Header to record the number of accesses to the Semantic-Header by users. Once the user views the complete semantic header, the SHAC will be increased by one.

4.3.2 Action Buttons of Displaying Windows

Next

The **Next** push button allows the user to view the next Semantic–Header. One can also view any other Semantic–Headers by clicking on the specific Semantic–Header title in Figure 35.

Prev

The **Prev** push button allows the user to view the previous Semantic–Header. One can also view any other Semantic–Headers by clicking on the specific Semantic–Header title in Figure 35.

Access

The **Access** push button allows the user to read the actual resource associate with the current Semantic–Header. When this button is pressed, Netscape browser is involved to display the actual resource. Please refer to section 4.3.3 for more details.

Annotate

The **Annotate** push button will tell the user how to add annotation to the current Semantic–Header. One have to save the current Semantic–Header then run the Annotation program to add annotations. Please refer to Chapter 5 for more details.

Save

The **Save** push button allows the user to save the current Semantic–Header in a file. A file selection window will popup for selecting a file name or accept a new file name (Figure 21, with *Save File* as it title).

Note: in order to modify the Semantic–Header or add annotation to the Semantic–Header, one has to save the Semantic–Header first.

Exit

Pressing **Exit** push button will exit the display window.

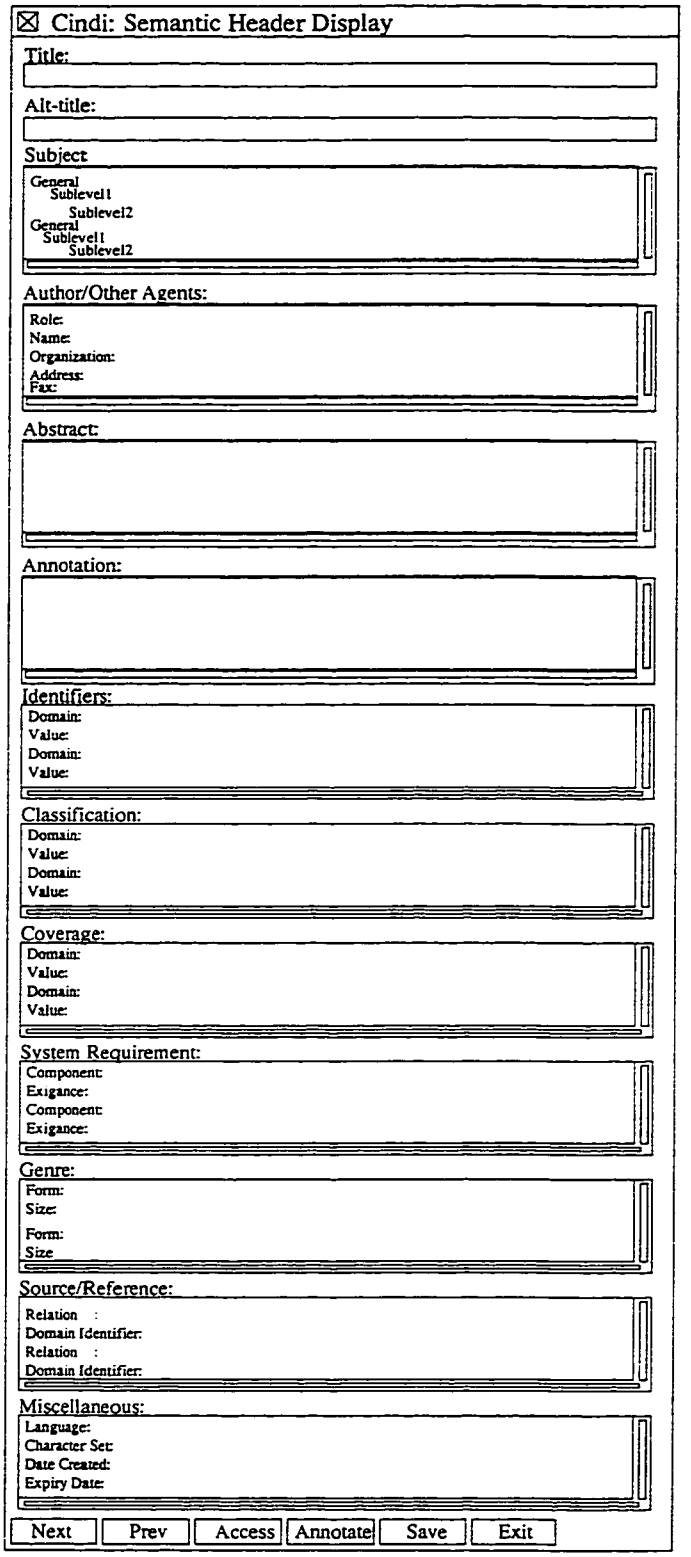

Figure 36: Display SH interface: Overall design

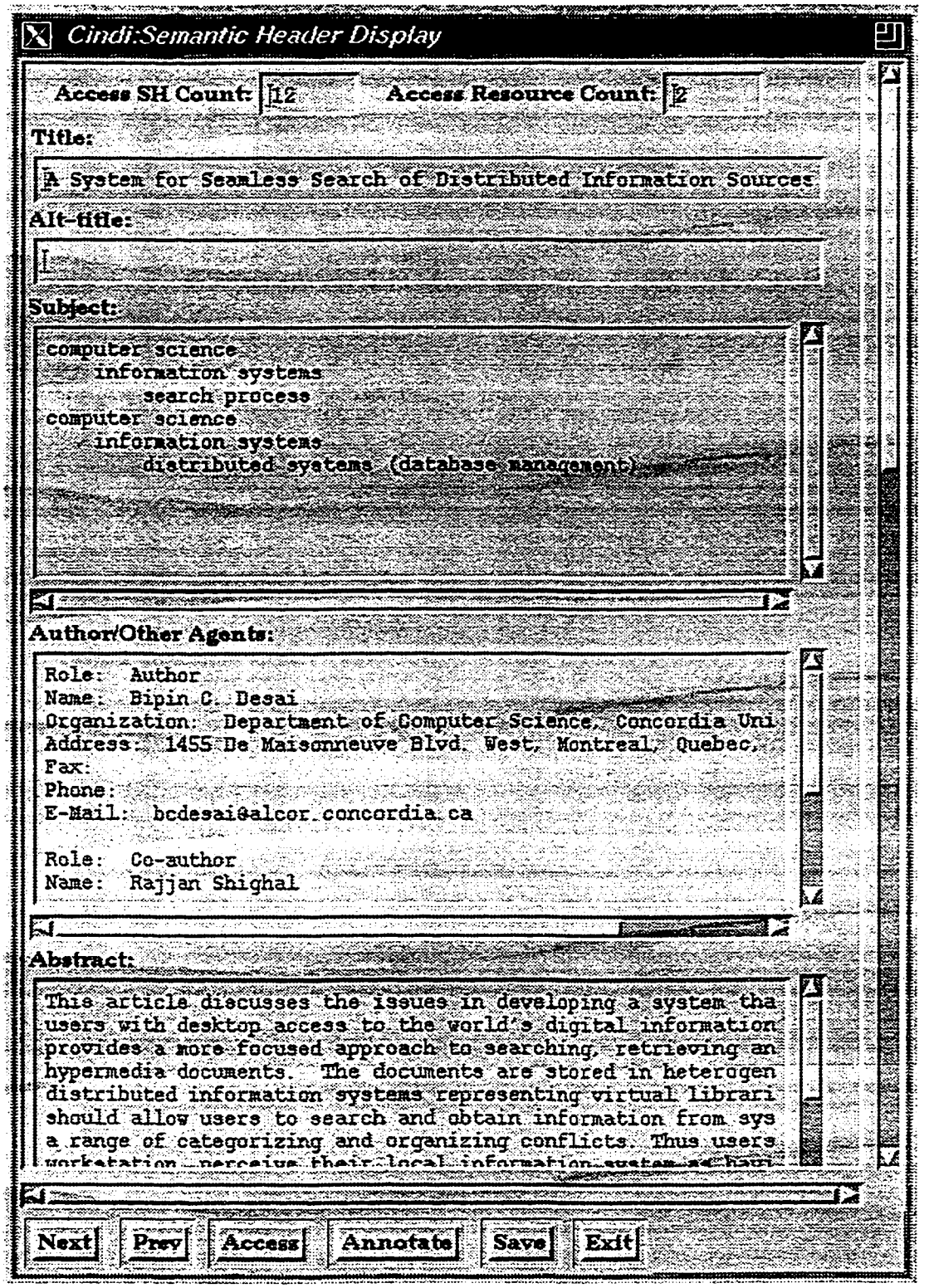

Figure 37: Viewing a Semantic-Header: Page 1

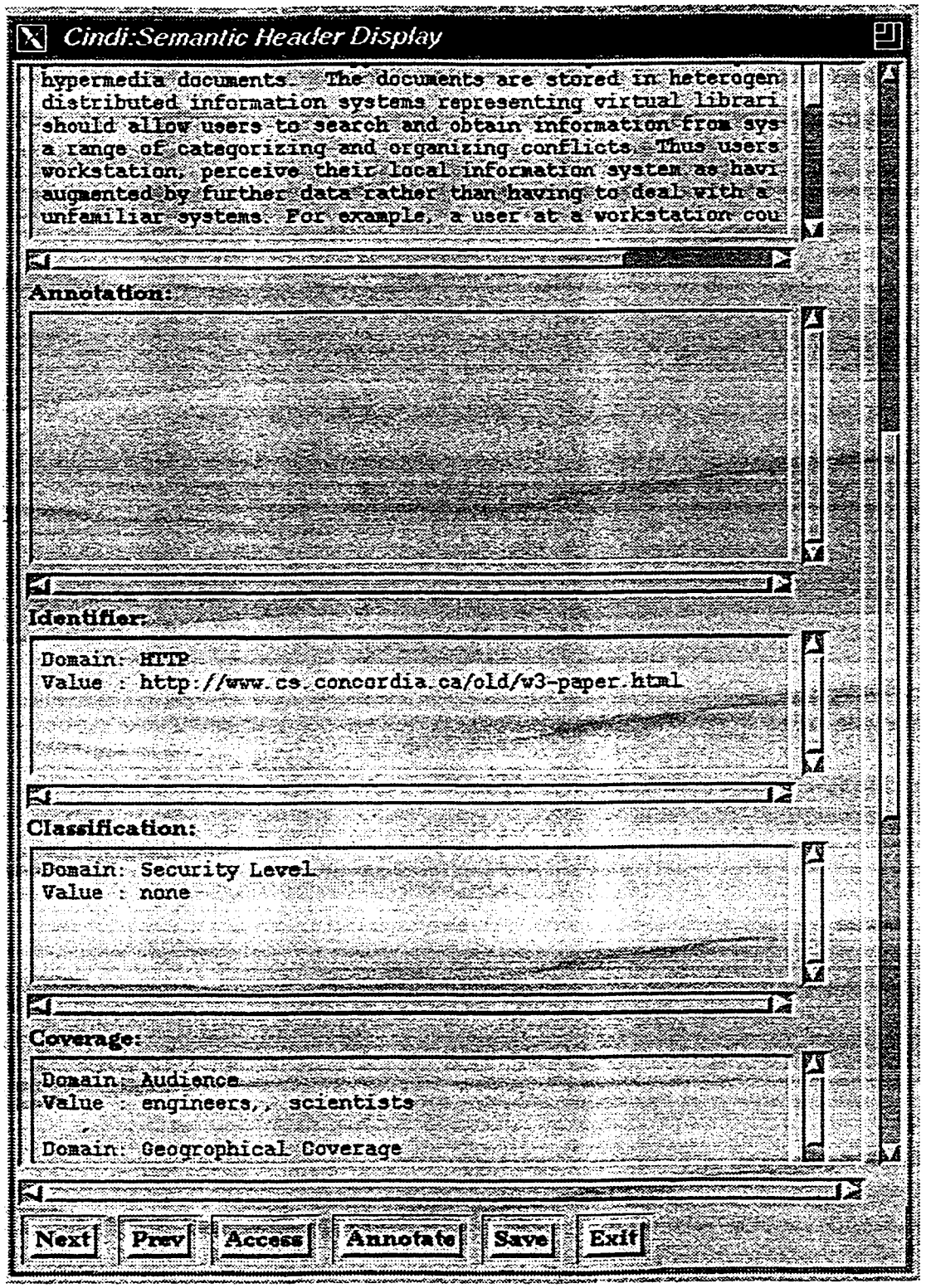

Figure 38: Viewing a Semantic–Header: Page 2

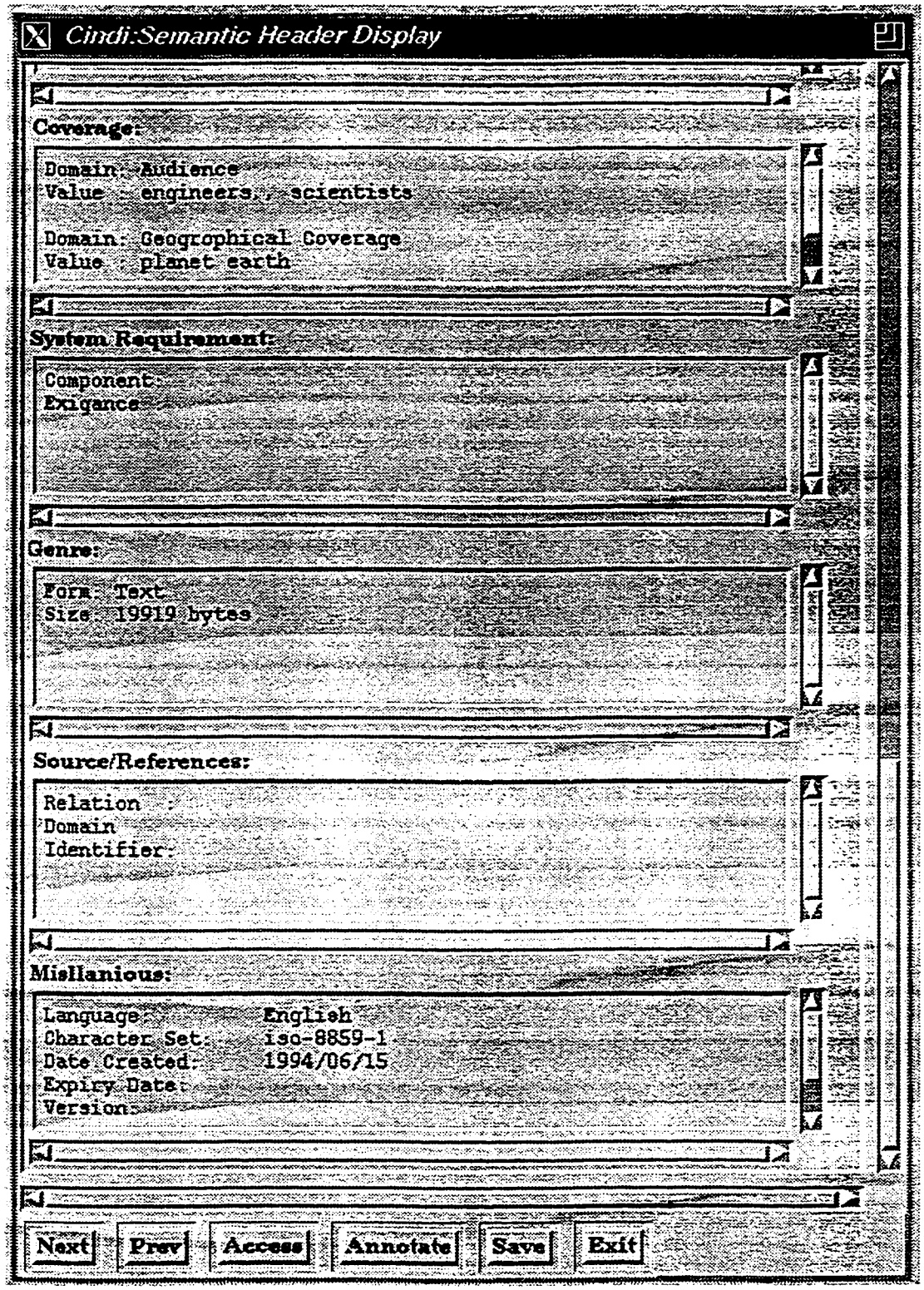

Figure 39: Viewing a Semantic-Header: Page 3

4.3.3 Viewing the Actual Resource

As mentioned above, to view the actual resource for the current Semantic–Header, one need to press the **Access** push button in the display window. A user can access the actual resource only if the selected item for access has an identifier which allows on–line access via a browser such as Netscape where it is used to display the resource.

Starting a Netscape Session

In the current version of our system, Netscape is used as the browser to display the actual resources. If a Netscape Navigator is not running on the user's machine, the callback function for **Access** will fork a process. There would be then two processes, the parent process and the child process. The parent process, the search system, can continue with what it is doing. The child process will start a Netscape session and pass along the identifier of the resource. This is accomplished by the statement.

(1) execlp("netscape","netscape", on–line–identifier–string, (char *) 0);

where "on–line–identifier–string" is the identifier of the resource.

Remote Control of a Netscape Session

Usually, the user does not want more that one Netscape process running on his machine. So remote control of the running Netscape process is necessary. In order to speak the remote control protocol directly, via X property changes, we adapted the protocol proposed by Jamie Zawinski [Zaw96]. We now can simply invoke Netscape by the following statement:

(2) netscape-remote -nonraise -remote 'openURL(on–line–identifier–string)'

When Netscape Navigator is invoked with the above statement, it does not open a Netscape window, but instead connects to and controls an already–existing process. The existing Netscape process accesses the resource using the identifier of the resource, "on–line–identifier–string", passed as the argument of openURL().

Statement (1) start a Netscape process. Statement (2) remote controls an existing Netscape process. This ensures that there is one and only one Netscape process is running at the user's workstation.

After Netscape is begun, the user may wish to exit from or kill the Netscape process and want to access it again later. The question is can another Netscape process start? The answer is yes! This is done by using "signal trap". The idea is as follows. We fork the main process, and in the child do an execlp to start Netscape and set "netscape_running" global variable to true in the parent process. We also setup a signal trap in the parent process that would catch any SIGCHLD from the child process. The signal trap function will set the "netscape_running" global variable to false. When the Netscape child process ends then a SIGCHLD signal will be sent to the parent process [Lea94]. The "netscape_running" global variable to false.

If the user presses **Access** push button, the associate callback function will check the "netscape_running" global variable, statement (1) is involved if "netscape_running" is false. Otherwise statement (2) will be involved.

Figure 40 is an example of calling netscape when the identifier is:

"http://www.cs.concordia.ca/w3-paper.html."

Increasing the Resource Access Counter

An Resource Access Counter (RAC) is kept along with the Semantic–Header to record the number of accesses to the associated actual source resource by users. Once a document is accessed, the RAC will be increased by one.

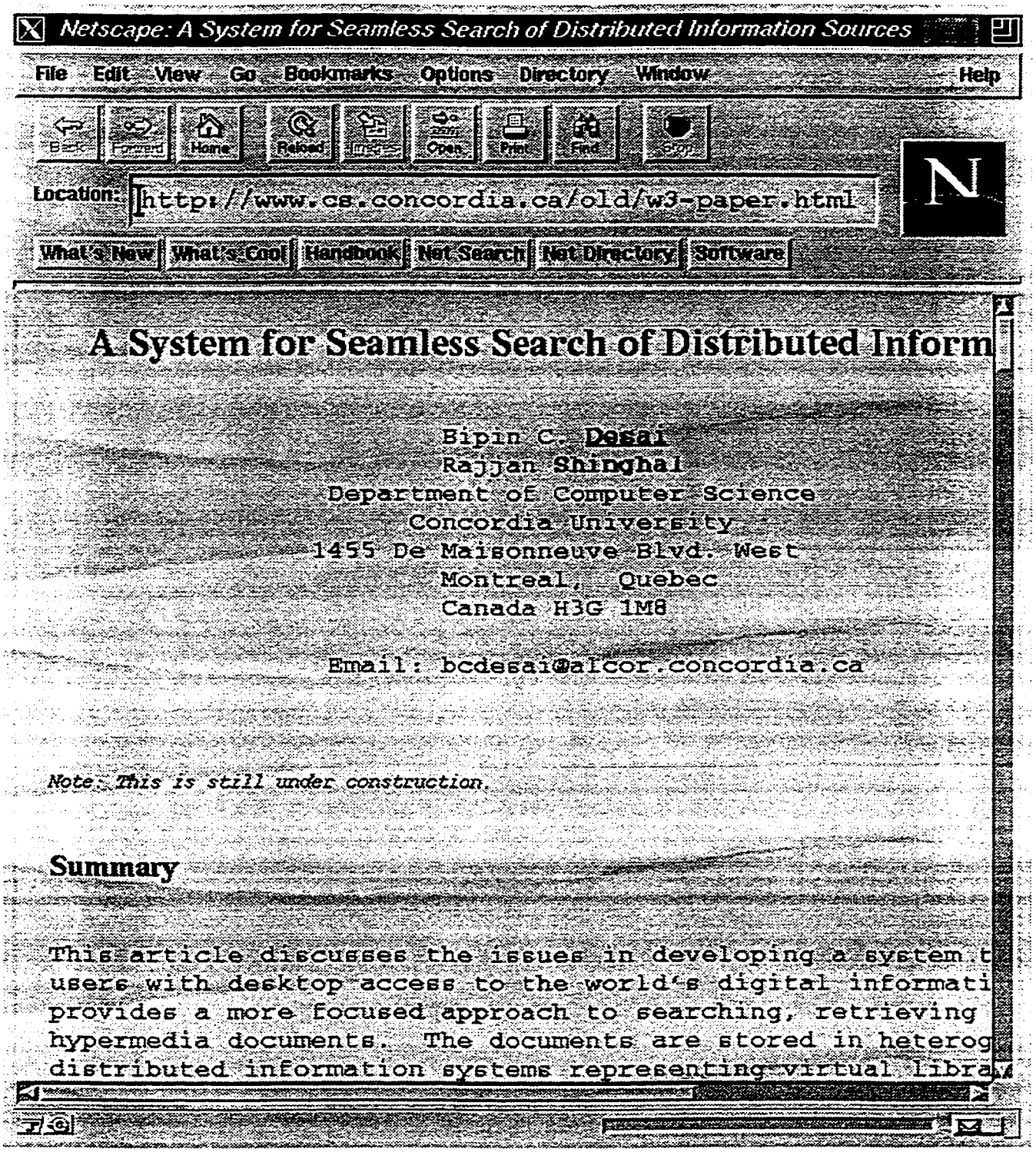

Figure 40: Access actual resource: An Example

Chapter 5

Annotation Sub–system

5.1 Overview

The research community depends on peer review of documents submitted for publication. Such review or annotation is often not published. However, comments to the editor made by readers of journals are usually published and are accessible to the community. Since many of the resources on the Internet tend not to be reviewed, it would be beneficial for a user to have access to annotations made by other users for a given resource. The proposed system allows users to add annotations to an existing resource. These annotations are stored along with the index in the SHDDB [DS96].

The annotation sub–system is similar to the indexing subsystem. However, only a few of the indexing entries that uniquely identify the resource in question are required. An annotation made by any user can be entered and would be registered with the identity of the user. Each annotation could be incorporated in the index entry and could be retrieved with the index. Such annotations by recognized persons would be a valuable guide for future users.

Peer reviews of electronically submitted papers could be implemented using such annotations. Authentication of reviews has to be done by an appropriate editorial board.

The graphical user interface of the annotation sub–system is shown in Figure 41.

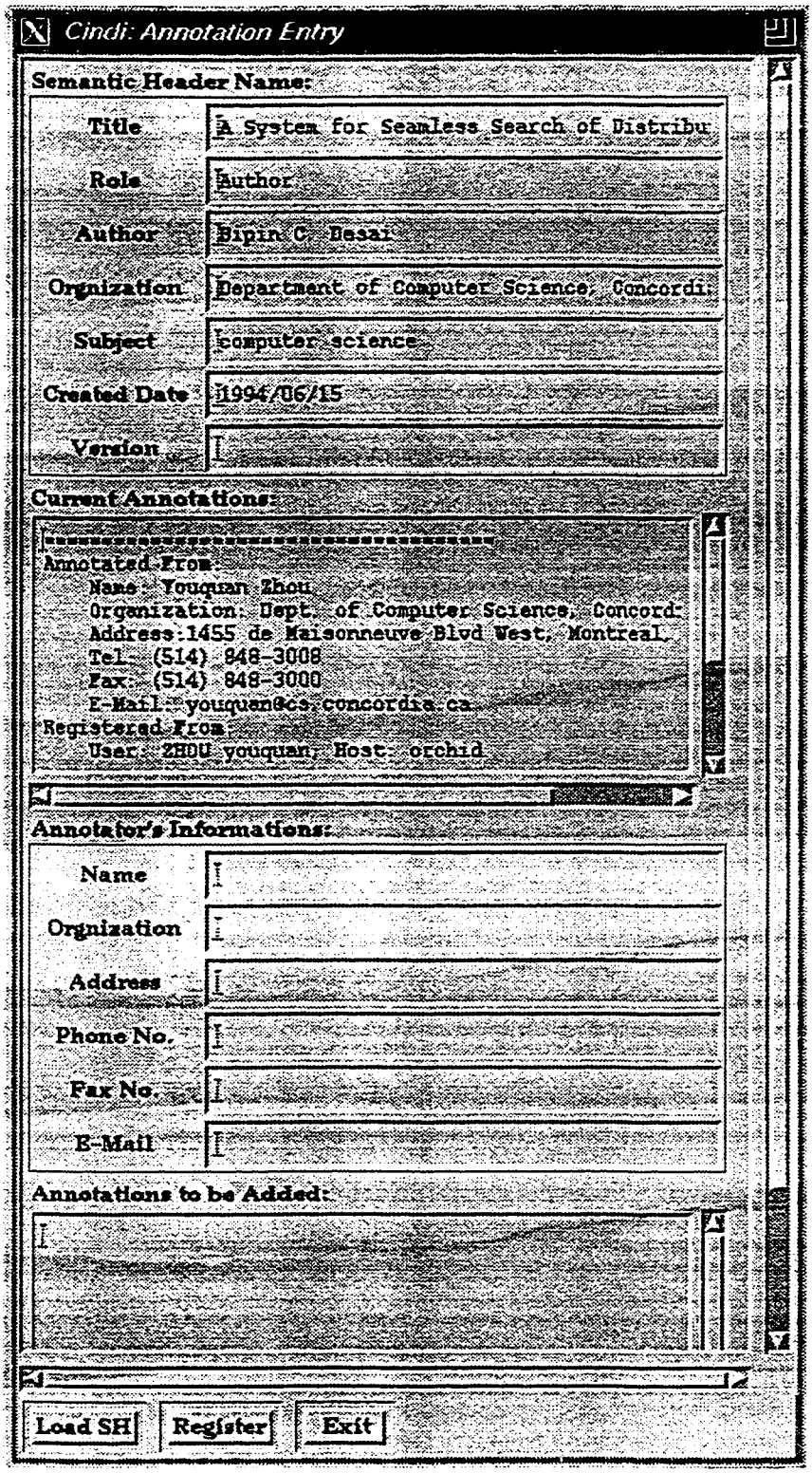

Figure 41: Graphical User Interface: Annotation

To avoid non–serious entries to the annotation by unscrupulous readers, we have separated the annotation entry from the Search Sub–system. In order to add annotation, the user has to save the Semantic–Header when viewing the Semantic–Header in the search sub–system. It is expected that the user actually access the resource corresponding to the Semantic–Header. If the user decides to make an annotation, he loads the saved Semantic–Header from local file system by pressing **Load SH** push button in the Annotation form.

The newly entered annotations together with the annotator's information as well as his login name and host name will be concatenated to the existing annotations when the annotator register his annotations. The components of Annotation user interface are described below.

5.1.1 Semantic–Header Name

The Semantic–Header Name block uniquely identifies a Semantic–Header. It is composed of the title, the first author's role, the first author's name, the first author's organization, the first subject, the date created, and the version number. The SHDB server will use the Semantic–Header Name to locate the Semantic–Header and append the new annotation to the existing one.

5.1.2 Annotator's Information

Annotator's information includes the annotator's name, organization, address, phone number, fax number, and email address.

For reference purpose and to associate a person with an annotation, we require annotator's information to be entered. Name and Address are require fields. The rest are optional.

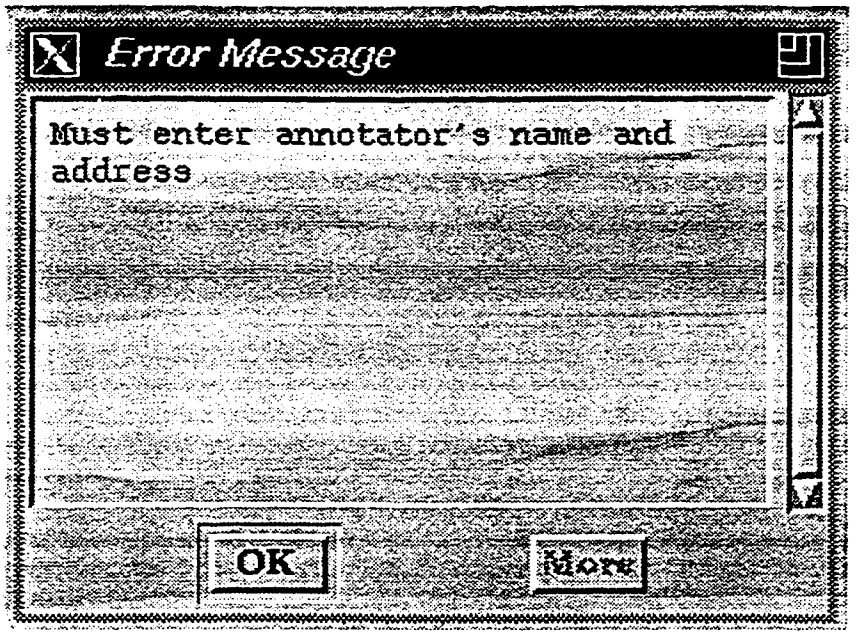

Figure 42: Annotation error checking popup1

5.2 Functionalities of Action Buttons

5.2.1 Loading a Semantic–Header

The **Load SH** push button allows the user to load a Semantic-Header into the annotation interface. A file selection window (Figure 21) will popup to allow the user to enter or select a file name. Only a Semantic–Header file can be loaded into the annotation interface. After loading the file, the Semantic–Header Name and the existing annotation is displayed. The **Register** button is enabled.

5.2.2 Register the Annotation

The client/server mechanism is used to register the annotation in the Semantic Header Database. When the **Register** button is pressed, two things are checked:

1. It ensure that the annotator's name and address are provided. If not, an error message is displayed (Figure 42).

2. The system verifies if the annotator's name entered is identical to the user's login full name. If not, a warning message will be given (Figure 43). There are two push buttons in this warning message window as shown in Figure 43:

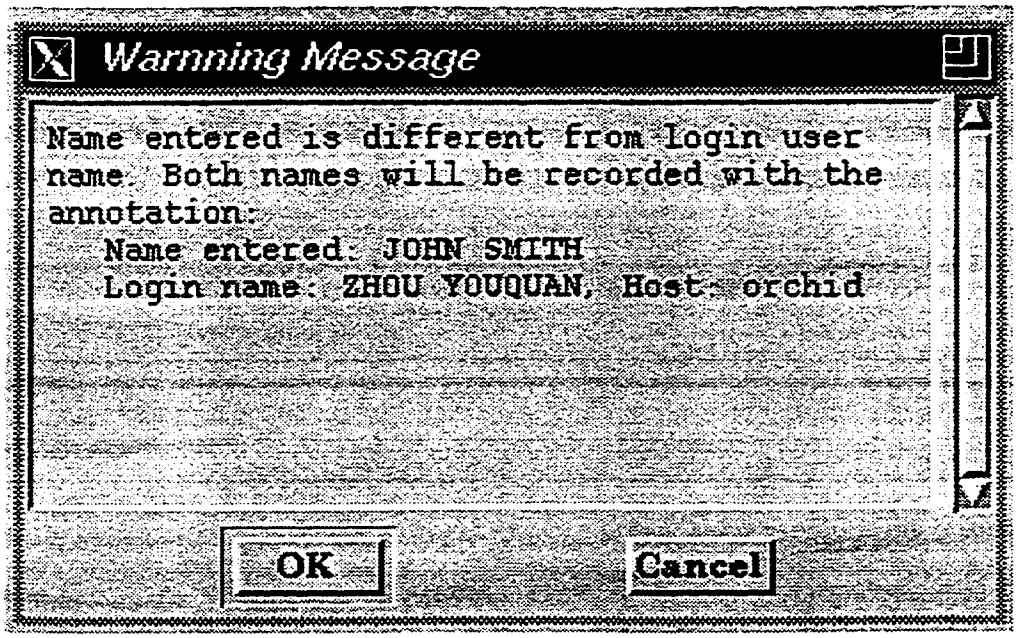

Figure 43: Annotation error checking popup2

Ok and **Cancel**. If one selects **Cancel**, the annotation will not be registered. Otherwise, it is registered. If the annotator's name entered is identical to the user's login full name, the annotation will be registered without warning.

Chapter 6

Summary and Future Work

6.1 Summary

Current indexing systems are based on harvesting a network of virtual libraries for new documents. Such documents are retrieved and their contents used to provide terms for the index. The big disadvantage with this scheme is the unreliability of the index entries produced and the lack of an authentic abstract for the item. Current index schemes are relevant for resources of limited protocols and are not applicable to other resources. Another problem with some of the robot-based approaches is the unnecessary traffic on the network and lack of cooperation and sharing among different systems. Finally, the in-feasibility of the existing approach become clear as more and more providers of information would require payment.

CINDI, an Indexing and Searching System On the Internet, is an ongoing project developed at Concordia University under the supervision of Dr. Desai. Its main goal is to develop a system that enables any user to find the location of any hypermedia document on the Internet.

To improve the efficacy of search, we have used a Semantic–Header: a data structure to record the metadata of network resources. Provided by the author/creator of the resource, it not only indicates the contents of the document and provides some vital extrinsic attributes, but also helps in indexing and locating the document.

The proposed system, CINDI, presented in this thesis is an attempt to tackle these problems from two viewpoints: from the point of view of the supplier of on-line information and from that of the "naive" user. The supplier of information must describe the information resource in a standard format. The user will use simple search terms to look for a particular resource and does not want to go from one catalog to another.

The supplier prepares the index information. Consequently, such index entry would be more reliable than the one derived by a third party or by simply scanning a document. The presence of an abstract in the Semantic–Header allows the contributor of the document to highlight the nature of the subject. Such a summary in the index allows users to make better decisions regarding the relevance of the resource. We have designed and implemented a graphical user interface for entering this information. Subject entry is guided by an expert system engine that controls the terms used to describe the subject. The expert system here mimics the expertise of a cataloguer. Terms such as Identifier domain, Language, Classification domain, Coverage domain etc. can be selected from popup menus.

For the user, we have designed and implemented a graphical user interface for the search sub–system. Subject entry is also guided by an expert system engine that controls the terms used to describe the subject. The expert system here would mimic the expertise of a reference librarian.

Users can also express their points of view in the form of annotation through the annotation sub–system.

6.2 Contribution of this Thesis to CINDI

The contributions made by this thesis fall into three categories: the architectual design of the CINDI system, the design and implementation of graphical user interface, and the design and implementation of the client part of the expert system. These contributions are listed as follows:

- Overall design: CINDI is divided into five sub–systems (Section 2.5).
- The design and implementation of the indexing/registering sub–system

- The design and implementation of the search sub–system, which includes the query entry interface, the Semantic–Header displaying interface, and the interface between CINDI and Netscape browser.
- The design and implementation of the annotation sub–system.
- Proposition and implementation of the rules for the context–sensitive help for subject hierarchy entry.
- Implementation of mathematical Infix expression for a repeatable query entry block.

6.3 Future Work

For the CINDI project, we have already built the basic components of our system. The following are suggestions for future work on the current implementation of the **CINDI** system.

6.3.1 Implementing the Fuzzy Search

As we notice from the search interface (Figure 31), we have the radio button **Like** for title entry and author/other agents entry. Two additional things need to be done for this feature.

1. Develop the rules for **Like**. When the user selects **Like**, he/she is not sure about the string entered. The user only enter something similar to what he/she seeks. It is necessary to clearly specify all rules for this or elicit the information from the user.
2. Once the rules are specified, the expert system can be implemented according to these rules. The user will be able to search relevant information even though he has at most vague idea about the title, author, and identifier about the target resources. This is called the fuzzy search.

6.3.2 Extension of the Search Query

In our current search query entry, the values entered in different blocks are **AND**ed together to execute the search. It would be a good idea to provide the user with

option of selecting **AND** or **OR** for the relationship of the values entered among the entry blocks. The job is to be done by both the GUI and the server:the GUI allows the user specify his options and the search engine at the semantic header database server site execute the search base on the query entered.

6.3.3 Internationalizing the Application

The resources on the Internet are written in many languages other than English. Internet users would no doubt like to have software applications in their native language. The next step for the CINDI System should be internationalization.

The purpose of internationalization is to have an application that can run anywhere in the world and yet interact with the user in the user's native language. The major goal of internationalization is not to change the binary application to run in a different country. The same executable program reconfigures itself under different languages, called *locales*. To do this, all text messages must be separated. For this, a separate set of *localization* for each target market is needed. The separate localization data could be stored in a file or files external to the program. For instance, there would be a set of localization files supporting the Chinese Language, a set of localization files supporting the French Language, and so on.

6.3.4 Speeding up the Semantic–Header entry process

Entering a Semantic–Header for a resource could be time consuming. To speed up this process, one of the future work could be to develop a system to scan over the resource and extract information such as the Title, Abstract, Keywords, Author info etc. from the Semantic–Header. The user can then load the auto–created Semantic–Header into the Semantic–Header entry interface to modify it as necessary. This project is actually underway.

6.3.5 Extension of Subject look–up

At the time of this writing, we limit our subject standard terms to computer science only. A more general scheme should be used; for instance, based on the classification of the *Library of Congress* to include all subjects. Since our subject hierarchy has

only three levels, a method should be introduced to suppress subjects with more than three levels of hierarchy.

Synonym resolution also needs to be implemented. This includes building a synonym database and implementing the look–up logic.

6.3.6 Building the SHDDB

CINDI was designed to have a Semantic–Header Distributed Database System (SHDDB) 1. At this time, we have built a centralized database. One of the major future work will be the construction of the SHDDB. The database must be horizontally partitioned [Des90] and partly redundant in order to allow *reliability* and *failure–tolerance*.

The distribution can be based on subject areas. The database on different subjects will be maintained at different nodes of the Internet. The end users need not know the distributed and replicated nature of the SHDDB. An expert system (central control system) should be constructed to track the locations of different nodes. Whenever there is a Semantic–Header to be registered or a search query to be processed, the expert system should decide where the Semantic–Header should be stored or where to find the answer for the search query.

The various sites of the database should work in a cooperating mode to maintain consistency of the replicated portion.

6.3.7 Enhancing the On–line Help

On–line help help users better understand the system. Our current on–line help could be extended for the following context:

- What is this and what is it for?
- Where am I ?
- Why did that happen?
- How do I do this?

- What can I do with this system?

The on–line help should be:

- Easy to use.
- Easy to find information.
- Allow easy switching between help and working context.
- Consist of a simple interface.
- Be laid out in a high–quality design.

Bibliography

[AGGL95] Arlein, R., Gava, J., Gehani, N., and Lieuwen, D. *Ode 4.1 User Manual.* AT&T Bell Laboratories, Murray Hill, New Jersey, 1995.

[Ass95] Association for Computing Machinery. The full computing classification system. *Computing Review, ACM Press*, page 6, January 1995.

[AW94] Antoniou, G. and Wachsmuth, I. Structuring and modules for knowledge bases: Motivation for a new modul. *Knowledge-based System*, 7(1):49–71, 1994.

[BCL+94] Berner–Lee, T., Cailliau, R., Luotonen, A., Nielsen, Frystyk H., and Secret, A. The world wide web. *Communication of the ACM*, Vol.37, No.8:76–83, August 1994.

[Ber96] Berghel, Hal. The client's side of the world–wide web. *Communication of the ACM*, Vol.39, No.1:32–40, January 1996.

[BGBG95] Baecker, Ronald, Grudin, Jonathan, Buxton, William A.S., and Greenberg, Saul. *Human–Computer Interaction Toward the year 2000.* Morgan Kaufman Publishers, 2nd edition, 1995.

[Blu96] Bluestone Inc. Uim/x. *http://www.bluestone.com/products/catalog/prodcat_uimx.html*, 1996.

[Boc94] Bochenski, B. . *Implementing production-quality client/server systems.* New York: John Wiley & Sons, 1994.

[Bro95] Brody, H. Internet@crossroad. *Technology Review*, May/June 1995.

[Byr91] Byrne, D.J. *MARC manual: understanding and using MARC record.* Libraries Unlimeted, Englewood, Colo, 1991.

[Car93] Carl-Mitchell, Smoot. *Practical internetworking with TCP/IP and UNIX.* Reading, Mass. : Addison-Wesley, 1993.

[Cho94] Chorafas, Dimitris N. *Beyond LANs Client/Server Computing.* McGraw–Hill, 1994.

[Cro94] Cromwell, W. A new bibliographic standard. *The Core Record: Library Resources and Technical Services*, 38(4):415–424, 1994.

[CSDR] Chander, P. G., Shinghal, R., Desai, Bipin C., and Radhakrishnan, T. An expert system to aid cataloging and searching electronic documents in digital libraries. *Expert Systems with Applications.* To appear in issue 12(4), June 1997.

[Des90] Desai, Bipin C. *An Introduction to Database Systems.* West, St. Paul, 1990.

[Des95a] Desai, Bipin C. Indexing and searching virtual libraries. *http://www.hpcc.gov/cic/forum/White_Papers.html*, July 1995.

[Des95b] Desai, Bipin C. Report on the metadata workshop, dublin, oh. *http://www.cs.concordia.ca/ faculty/bcdesai/metadata/metadata-workshop-report.html*, March 1995.

[Des95c] Desai, Bipin C. The semantic header and indexing and searching on the internet. *http://www.cs.concordia.ca/ faculty/bcdesai/cindi-system-1.0.html*, 1995.

[Des95d] Desai, Bipin C. Test: Internet indexing systems vs list of known urls. *http://www.cs.concordia.ca/ faculty/bcdesai/test-of-index-systems.html*, June 1995.

[DS94] Desai, Bipin C. and Shinghal, Rajjan. A system for seamless search of distributed information sources. *http://www.cs.concordia.ca/w3-paper.html*, May 1994.

[DS96] Desai, Bipin C. and Shinghal, R. Resource discovery: Modelling, cataloging, and searching. In *Proceeding of the Seventh International Conference and Workshop on Database and Expert Systems Applications (DEXA '96)*, pages 70–75. IEEE Press, Zurich, Switzerland, 1996.

[Dub96] Dube, Alan C. Discussion about the open look gui. *http://title.net/news/comp305.html*, 1996.

[FB93] Ferguson, Paula M. and Brennan, David. *Motif Reference Manual, Volume 6B.* O'Reilly & Associates, 1st edition, 1993.

[Gay94] Gaynor, E. Cataloging electronic texts: The university of virginia library experience. *Library Resources and Technical Services*, 38(4):403–413, 1994.

[GF95] Gettys, Jim and Frystyk, Henrik Nielsen. Http - hypertex transfer protocol. *http://www.w3.org/pub/WWW/Protocols/Overview.html*, 1995.

[GM86] Gehani,Narain and McGettrick, Andrew. *Software specification techniques.* Workingham, England ; Reading, Mass. : Addison-Wesley, 1986.

[Gol90] Goldfarb, Charles F. *The SGML Handbook.* Oxford University Press, 1990.

[GR93] Giarratano, J. and Riley, G. *Expert Systems: Principles and Programming.* PWS Publishing Company, Boston, MA. 2nd edition, 1993.

[HF94] Heller, Dan and Ferguson, Paula M. *Motif Programming Manual, Volume 6A.* O'Reilly & Associates, 2nd edition, 1994.

[Hor86] Horny, K.L. Minimal–level cataloging: A look at the issues a symposium. *Journal of Academic librarianship*, 11:332–334, 1986.

[JF90] Jacob, R.J.K. and Froscher, J.N. A software engineering methodology for rule–based systems. *IEEE Transaction on Knowledge and Data Engineering*, 2(2):173–189, 1990.

[JR93] Johnson, Eric F. and Reichard, Kevin. *Power Programming... Motif.* MIS Press, 2nd edition, 1993.

[Kat87] Katz, W. A. *Introduction to Reference Work.* McGraw–Hill, New York, 1987.

[Koc95] Kochmer, J. *The Internet Passport: NorthWestNet's Guide to Our World Online.* North West Net, 1995.

[Kosa] Koster, Martijn. Aliweb(archie like indexing the web. *http://web.nexor.co.uk/aliweb/doc/aliweb.html.*

[Kosb] Koster, Martijn. The web robots pages. *http://info.webcrawler.com/mak/projects/robots/robots.html.*

[Lan93] Lane, Elizabeth S. *An Internet primer for information professionals : a basic guide to Internet networking technology.* Westport, CT : Meckler, 1993.

[Lea94] Leach, Ronald. *Advanced topics in UNIX : Processes, files, and systems.* New York : J. Wiley, 1994.

[LS93] Laverna, M. and Saunders, I. *The Virtual Library: Visions and Realities.* Meckler, 1993.

[Mas94] Masinter, L. 1993 is the year the internet 'happened'. *Internet Society News*, Winter 1994.

[Mau95] Mauldin, M. L. (1995)measuring the web with lycos. In *Poster Proceeding of the Third International WWW Conf.*, pages 26–29, Darmstadt, April 1995.

[Nie95] Nielsen, Jakob. *Multimedia and Hypertext: The Internet and Beyond.* AP Professional, 1995.

[NO92] Nye, Adrian and O'Reilly, Tim. *X Toolkit Intrinsics Programming Manual, Volume 4M.* O'Reilly & Associates, 2nd edition, 1992.

[Ped92] Peddie, Jon. *Graphical User Interfaces And Graphic Standards.* McGraw–Hill, 1992.

[Pre95] Pree, Wolfgan. *Design Patterns for Object-Oriented Software Development.* Addison–Wesley, 1995.

[RBPEL91] Rumbaugh, James, Blaha, Michaeland Premerlani, William, Eddy, Frederick, and Lorensen, William . *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[San94] Santifaller, Michael. *TCP/IP and ONC/NFS : internetworking in a UNIX environment.* Reading, Mass. : Addison-Wesley, 2nd edition, 1994.

[Shi92] Shinghal, R. *Formal Concepts in Artificial Intelligence.* Chapman & Hall, London; New York, 1992.

[Thi90] Thimbleby, Harold. *User interface design.* Addison-Wesley Pub., 1990.

[Too96] Toon, John et al. The gvu center's sixth www user survey. *http://www.cc.gatech.edu/gvu/user_surveys/survey-10-1996*, November 1996.

[Web96] Web Mastery. The html language. *http://union.ncsa.uiuc.edu:80/HyperNews/get/www/html/lang.html*, 1996.

[Wil96] Wilder, CQuin, Liam et al. The money machine. *Information Week, Issue 561*, pages 25–28, 1996.

[Zaw96] Zawinski, Jamie. Remote control of unix netscape. *http://home.netscape.com/newsref/std/x-remote.html*, 1996.