CINDI: CONCORDIA INDEXING AND DISCOVERY SYSTEM

Nader Rajabieh Shayan

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

0-612-26019-4

Abstract

CINDI: Concordia Indexing and Discovery System

Nader Rajabieh Shayan

As the number of Internet users grows, the problem of indexing and retrieval of electronic information resources becomes more critical. A number of search systems are currently available for this purpose on the Internet; examples are Lycos, Yahoo, Web Crawler, etc. However, they offer uneven search results, namely, many of them produce mishits or miss existing resources. This is due to the fact that they attempt to match the specified search terms without context as to where the words appear in the target information resource. This calls for a proper cataloging to avoid such uneven results.

This thesis is concerned with a metadata-based indexing system proposed to describe the semantic content of information resources. The metadata description is called Semantic Header, and its main intent is to include those elements that are most often used in the search for an information resource. A variety of fields have been included in Semantic Header for the indexing and retrieval of resources. This will considerably reduce the aforementioned unpredictable results. The system also distributes the expertise of a librarian to help users choose appropriate subject terms, during indexing and searching, from an associated thesaurus.

A prototype has been developed based on this proposal for indexing and discovery of information resources on the Internet. The prototype is composed of two main subsystems viz. Graphical User Interface (the client) and Database (the server). This thesis presents the design and implementation of the Database subsystem. Object Modelling Technique (OMT) has been employed for the analysis and design of this subsystem. Object Database and Environment (ODE) is the database system used for indexing and retrieval of Semantic Headers. Communication between the two subsystems has also been implemented using the TCP/IP protocol.

To the memory of my brother,
Khosrow.

Acknowledgments

I would like to express my deepest gratitudes to my thesis supervisor Dr. Bipin C. Desai for his thoughtful guidances and financial support throughout my study. This thesis is the result of his patience, knowledge, and constructive critisisms which will never be forgotten.

I would also like to thank my friend Youquan Zhou for being a good companion by developing the GUI of the system in the best way possible and his many helps in integrating the GUI and the database subsystems.

The design of the expert system of the system has been based on the efforts of Dr. R. Shinghal and Dr. P. G. Chander. Thank you Gokul for your help and availability.

The second chapter of this document has been derived from my friend Yves Saintillant's work, under Dr. Desai's supervision, whose generosity is appreciated.

I am grateful to Stan Swiercz and Michael Assels, system administrators, who have been helpful in all phases of the implementation of the prototype.

The ODE support group, at AT&T Bell Laboratories, have provided many helps to eleminiate problems caused by systems incompatibiliy and minor bugs.

Many special thanks should go to my brother Dr. Yousef Shayan whose flawless guidances and encouragements have set the cornerstones of my success.

Last, but not least at all, I would like to thank my parents and my brother Nasser Shayan whose endless support and encouragements, far away from Canada, made this course of study easy to travel.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Discovery Problem

In recent years, the Internet has become well-publicized throughout the world. Computers, along with network facilities, have found their way through all aspects of our lives and the Internet is becoming a well-accepted repository of information.

As such, 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 [BLT90, BLT93] in particular. This is now becoming the accepted norm of disseminating and sharing information resources in hypermedia.

At the same time, as the number of the Internet users, be they professionals or the general public, grows, the discovery of information becomes more difficult since the information is no longer centralized. The purpose of indices and bibliographies (called secondary information) is to inventory the primary information and allow later discovery and access. A number of index generation systems and related search systems are currently available on the Internet[Kos, LYC, MBO, WebC, W3C, Arch, KB91, MML95, Thau]. Some of these are manually generated indices while others are generated by robots[Kos96]. Some of these robot-based systems also allow manual index entry. The search interface provides users very little flexibility and the results obtained are varied. This is illustrated in Table 1 (adapted from [BCD95b]) for a

Search System	Number of Hits	Number of Duplicates	Number of Mis-hits	Number of Items Missed
Aliweb	none	-	-	25
DACLOD	none	-	-	25
EINet	6	0	4	18
GNA Meta Lib.	none	-	-	25
Harvest	none	-	-	25
InfoSeek	7	0	0	17
Lycos	231	2	222	18
Nikos	none	-	-	25
RBSE	8	-	8	25
W3 Catalog	none	-	-	25
WebCrawler	7	3	0	21
WWWW	2	0	0	23
Yahoo	none	-	-	25

Table 1: Search statistics for using the search term Bipin (AND) Desai

query using the first and last names of an author as the search term. Even systems, such as Lycos, which claim to have indexed millions of documents have only partial success in locating all relevant documents [BCD95b][1]. The reason for the uneven result is due to the fact that many of these search systems attempt to match the specified search terms without context as to where the words appear in the target information resource.

This unpredictable retrieval of appropriate information resources, documented in Table 1, illustrates that there is a need for the development of a system which allows better control in searching for and accessing resources available on the Internet. Hence the need for proper cataloging.

[1] It should be noted that some of these systems are no longer accessible and newer systems have been introduced. If these tests, conducted in June 1995, were repeated today, the target items would most likely have been found by systems using robots. However, they would be hard to locate among the total number of entries found since the number of duplicates and the number of mishits would be in the hundreds and even thousands [CPG].

1.2 Solution

The problem with current automatically generated index databases is their inadequate semantic information. However, the ever-growing volume of information rules out any possibility of professionally cataloging the information resources, simply, because it would be extremely costly[2]. This precipitates a need for the design of metadata to provide a template for describing infromation about a resource. This, along with a registering system, which is the actual means of creating a metadata for a source document by the provider of the document, would establish a basis for later discovery. Also, an expert system should be provided to mimic the behavior of a reference librarian for cataloging and searching. Furthermore, there must be a mechanism to revise index information as the resource changes over time. Finally, annotation of a resource by independent users should be allowed.

1.2.1 Metadata for Indexing and Discovery

Metadata has become an intensely debated issue in the Internet community and many have come to the conclusion that the concept of catalog-based approach would be the most appropriate paradigm. Creating indices based on search robots (also called worms, spiders, or crawlers) have the following disadvantages:

- Repeated attempts by robots to find new resources will increase the traffic on the network. The number of these robots is increasing and system administrators will likely 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[BH95, BJN, Cr91, MMJ].
- Type of data gathered by 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 resource.

[2] A number of systems have addressed the problem of cataloging among which CORE [CRW], MARC system [BDJ, CrW, PTM90], MLC [HKL, Ross, Rhee], and TEI [Gyn, GIO] can be mentioned. However, these systems are mainly designed for professional catalogers.

- 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 nonmanageable. This trend for existing systems to exhibit poor selectivity is illustrated above.

Metadata should provide a means to describe the semantic content of a resource in order to better support discovery than the resource itself. This is because, in many cases, the resources themselves may not be able to provide the semantic dependencies, and in any case it would likely be computationally too expensive to do so. (For instance, how does one conclude that a given program code is used to provide computation of consumer loan payments without analyzing the program.) Metadata facilitates the cataloging of resources such as audio, computer programs, services, images and videos. This becomes important when the resource itself is not as easily accessible as the index.

Another reason for using metadata and extracting salient features of a resource is to support retrieval by content. Automatic processing of the contents of a source by extractors has been done on an ad-hoc basis but has been found to be unreliable. A case in point is the promise of natural language processing(NLP) which has not been quite realized. Approximations such as WAIS [KB91] have been useful but have also shown that relevancy measures derived using frequencies, proximity, etc. may not always be meaningful.

Metadata could also be used to express semantic dependencies which are inherent in a collection of objects. This means that the structure of the objects could be expressed using metadata as their surrogates and the actual sources could be separated from their metadata. This simplifies the storage of the resources and allows for the recognition of redundancies. Extracting such semantic dependencies in metadata allows for search based on the contents of multimedia resources. Initial query processing could be done on the metadata and thus avoid access to most of the resources and the possibility of their computationally bound interpretation. This becomes more advantageous when there are costs (time, money, network bandwidth and overloading) involved in accessing resources. The cost of accessing metadata would be much smaller than the cost of accessing the resource, because of small size of metadata description compared to that of the resource. Query processing would be supported by statistscs, as well as by an expert system to help formulate queries as is done by a reference librarian.

Appropriately constructed metadata could support query based on contents as well as traditional query based on items such as title, author, subject, etc. This means that actual sources could be separated from their metadata. This simplifies the storage of the resources. Professional catalogers have found the need for the above elements in indexing applications. The dependence on titles as the most commonly used search criterion suggests that they must indicate the contents of the document. Furthermore, the author or the cataloger has to add annotations, keywords, or key phrases to indicate its actual content. The accuracy or quality of a document can be indicated by including reviewers' opinions. However, such opinions are rarely accessible to the traditional cataloger. Another feature of importance is the presence of an accurate abstract. An abstract provides a summary of the material, and thus is more indicative of the contents than the title and keywords supplied by the author, or selected by scanning the text. Reference librarians and library users tend to use annotated bibliographies to help choose among competing sources.

1.2.2 Automated Reference Librarian

Expert Systems have been used quite extensively whenever human intelligence is to be modeled[Shin92, GJRG]. For example, a medical diagnosis system mimics doctors by capturing their knowledge that enable them to diagnose diseases 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 [Shin92]. Expert systems are also better suited as support tools in an application [LHL].

A reference librarian can guide the user in searching; this is what the 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 of the tasks of the expert system is to provide the required amount of help to complete the input. This supports the discovery of information in that the users will be guided to follow a well-established standardized method of index searching.

Similarly, in cataloging a new resource, the reference librarians use the knowledge of the 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 terms to describe the document. Employing an expert system to replicate professional catalogers' knowledge and expertise will assist in providing standard index structure and building a bibliographic system using standardized control definitions. These definitions could be built into the knowledge base of the expert system-based index entry and search interfaces. This will considerably reduce the inconsistency of information resources which, in turn, avoids the problem of retrieving irrelevant resources as well as not retrieving relevant ones.

1.3 Organization of the Thesis

This thesis is organized as follows. In chapter 2, we will introduce a number of current search engines on the Internet. For each search system, we will show the search results performed based on the name of an author. At the end, the drawbacks of these systems will be presented. Inspired from the two previous chapters, we will devote chapter 3 to the problem statement of our system. In this chapter we will introduce major components of the system as well as an overview of their interactions.

We will end this chapter by describing the strengths of the proposed system. In chapter 4, we will formally analyze the system by presenting the Object, Dynamic, and Functional Models based on the Object Modelling Technique. In chapter 5 the system architecture is presented. Chapter 6 extends the Object Model of the system and describes the implementation of the system in detail. We also give results obtained by actual verification of the application. Finally chapter 7 will mention extensions and further functionalities to be added to the system.

Chapter 2

Search Systems on the Internet

2.1 A concise description of Internet and World Wide Web

2.1.1 Internet

The Internet can be described as a federation of interconnected digital networks (LANs and WANs) that communicate with the TCP/IP protocol [Ste90]. A user connected to these networks can access any information stored electronically on the Internet. This information is either private or public. Private information is availabe for authorized users, who have a particular status or have paid a fee. Public information is available for any user of the Internet without any charge.

2.1.2 World Wide Web

The World Wide Web, whose development began at CERN (Geneva) in 1989, is defined as an information service on the Internet that has the following properties [BLT94]:

- It uses a common addressing syntax currently in the form of a Unique Resource Locator (URL).

- It uses Hypertext Markup Language (HTML), a document formatting language. HTML is a language used to describe hypertext resources, in which links with other resources can be defined. It can also describe hypermedia resources, wherein the links are not associated to textual information, but to other resources such as images or sounds.
- It uses the HTTP protocol (HyperText Transfer Protocol) in order to transfer information resources between two computers (a client and a server) of a network. These resources could be texts, menus, hypertexts, images, etc.

A number of WWW browsers have been developed concurrently with the expansion of powerful workstations:

- Lynx, a general purpose text-based Web browser.
- (NCSA) Mosaic
  Developped by the NCSA (National Centor for Supercomputing Applications), Mosaic is a freeware browser that retrieves hypertext documents, accessible on the Internet with the HTTP protocol, or available in Gopher or WAIS databases. The HTML language is used to write hypertext documents whose components can be displayed directly in the window of Mosaic. Each document can have multiple hypertext links to other documents. Mosaic was first developped for UNIX, and it has been extended to Microsoft Windows and Macintosh. 10% of the WWW users navigate with Mosaic (statistics in [Net]).
- Spyglass Mosaic (also called 'Enhanced NCSA Mosaic').
  1.7% of World Wide Web users navigate with this browser.
- Netscape Navigator (for Windows, XWindow and Macintosh).
  56% of the users navigate with Netscape, which is the most popular browser.
- Netcom Netcruiser (1.6% of the users).
- I.B.M. WebExplorer (1.5% of the users).
- Some other browsers exist, but they represent together only 1.4% of the users.

As the amount of the information available on the Internet grows, the search for a particular resource becomes more difficult. Hence, there is a need for software that helps the user locate and access rapidly information resources on the Internet. Such systems already exist and some of them are very popular. Moreover, their searches provide limited information and are not optimal. The following section will be devoted to brief descriptions of some of these systems.

2.2 Search Systems on the Internet

2.2.1 WAIS (Wide Area Information Servers)

- WAIS [KJ93] is a retrieval tool accessing different kinds of databases (traditional databases, ftp archives, library catalogs, usenet archives, etc...).
- WAIS provides a search based on the full content of the document (e.g., keywords), and not only on the titles of objects.
- WAIS provides **relevance feedback**: a document which has been retrieved can be considered as a source of keywords for a future search.
- WAIS supports only text objects, which is a limitation of the system.

2.2.2 Archie/XArchie

Archie is a service which helps users to locate files and directories on anonymous FTP servers anywhere on the Internet.

Administrators all over the world register anonymous FTP servers with the archie service; once a month the archie service runs a program which scans the directories and filenames contained in each of the registered FTP servers, and generates a merged list of all the files and directories contained in these servers. More than 1000 anonymous FTP sites are now represented in this list, which is referred to as the archie database. The archie database currently contains more than 2,100,000 filenames [Arch].

The archie database is made available on several archie servers, all of which contain the same information. The archie database contains both the directory path and the file names.

To retrieve a file, the user can enter a *Search Term*, which is either the name of the file or a sub-string of the name of the file. The system retrieves a list of ftp sites and their directories which contains files whose names correspond to the search term. The user can hence retrieve the files from one of the retrieved hosts indicated using the access path.

Xarchie is an X11 browser interface to the Archie Internet Archives database using the Prospero virtual filesystem protocol.
Drawbacks:

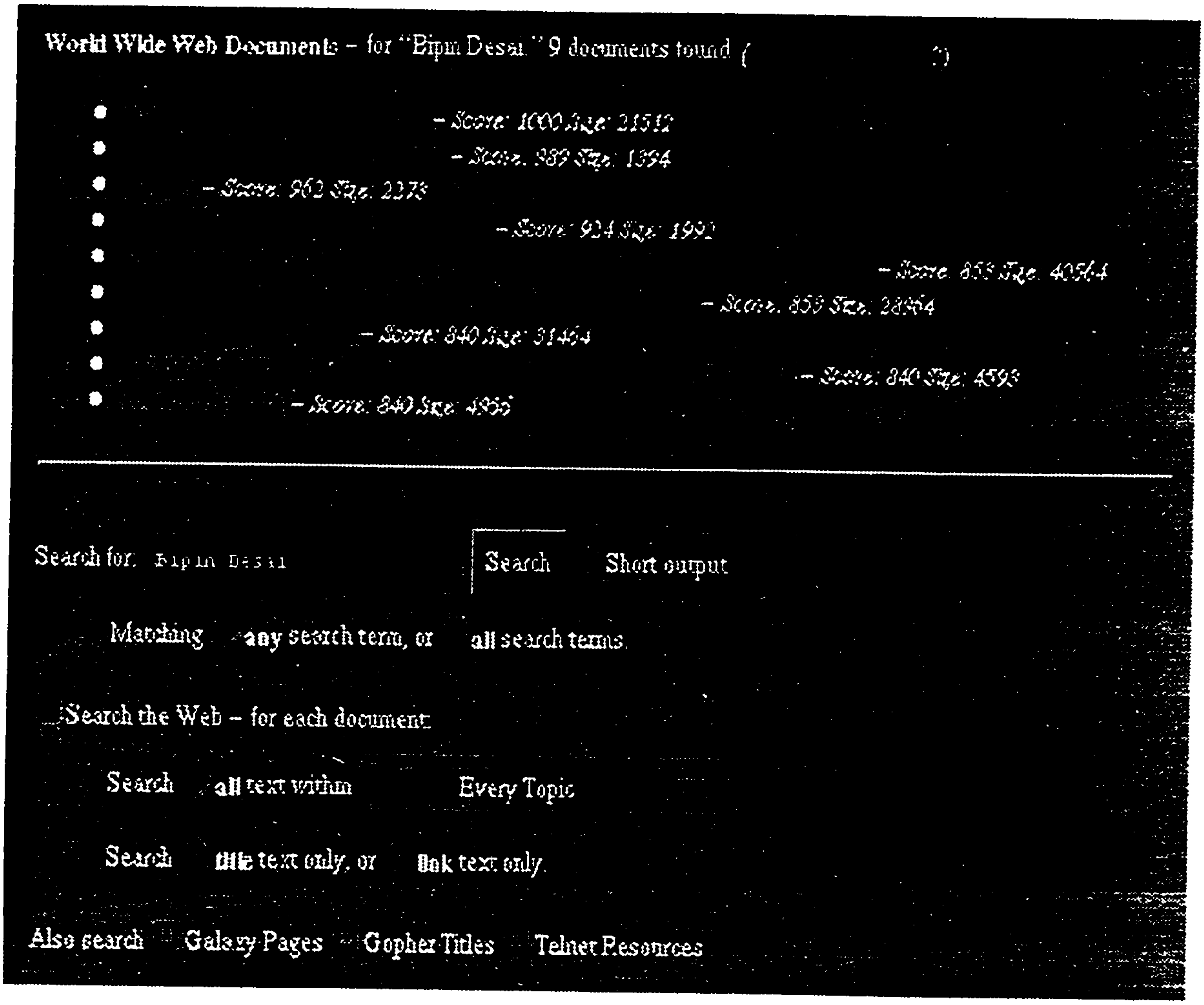
- The system is not easy to use for a non-initiated user. For instance, if the user does not enter a judicious string, s/he will not find a file and hence the sites.
- The system does not give any information about the content of the file.
  The user has to access the file from one of the hosts and open it to peruse it in order to judge if it is interesting or not.

2.2.3 Veronica (Very Easy Rodent-Oriented Net-Wide Index to Computerized Archives)

Veronica [VER] is an index and retrieval system which can locate items on most of the gopher servers in the Internet. The veronica index contains about 10 million items from approximately 5500 gopher servers (June 1994). Veronica finds resources by searching for *words* in *titles.* It does not do a full-text search of the contents of the resources; it finds resources whose titles contain the specified search word(s). The *title* is the title of the resource as it appears on the menu of its *home* gopher server. Veronica is used with a gopher client. One chooses *veronica* from the menu of a gopher server, and enter a set of query words or special directives. When the search is finished, the results will be presented as a normal gopher menu. The result resources can be browsed in this menu, as any other gopher menu.

2.2.4 EINet Galaxy

EINet Galaxy is a tree organized by subjects, which are used by the search engine. The User Interface of the search system along with the result of the search term 'Bipin Desai' is shown in Figure 1 [EIN].



World Wide Web Documents – for "Bipin Desai." 9 documents found ( )

- – Score: 1000 Size: 21512
- – Score: 989 Size: 1394
- – Score: 962 Size: 2273
- – Score: 924 Size: 1992
- – Score: 855 Size: 40564
- – Score: 859 Size: 28964
- – Score: 840 Size: 31464
- – Score: 840 Size: 4593
- – Score: 840 Size: 4955

Search for: Bipin Desai

Matching any search term, or all search terms.

Search the Web – for each document:

Search all text within Every Topic

Search title text only, or link text only.

Also search Galaxy Pages Gopher Titles Telnet Resources

Figure 1: EINet Galaxy Search Interface

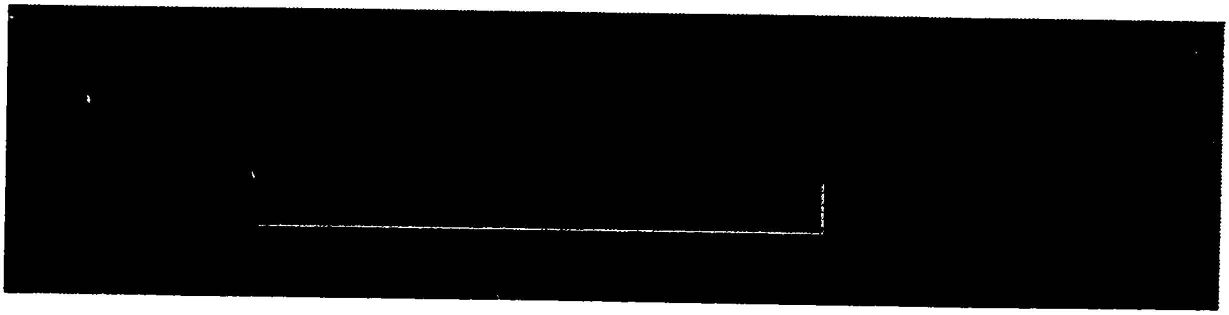
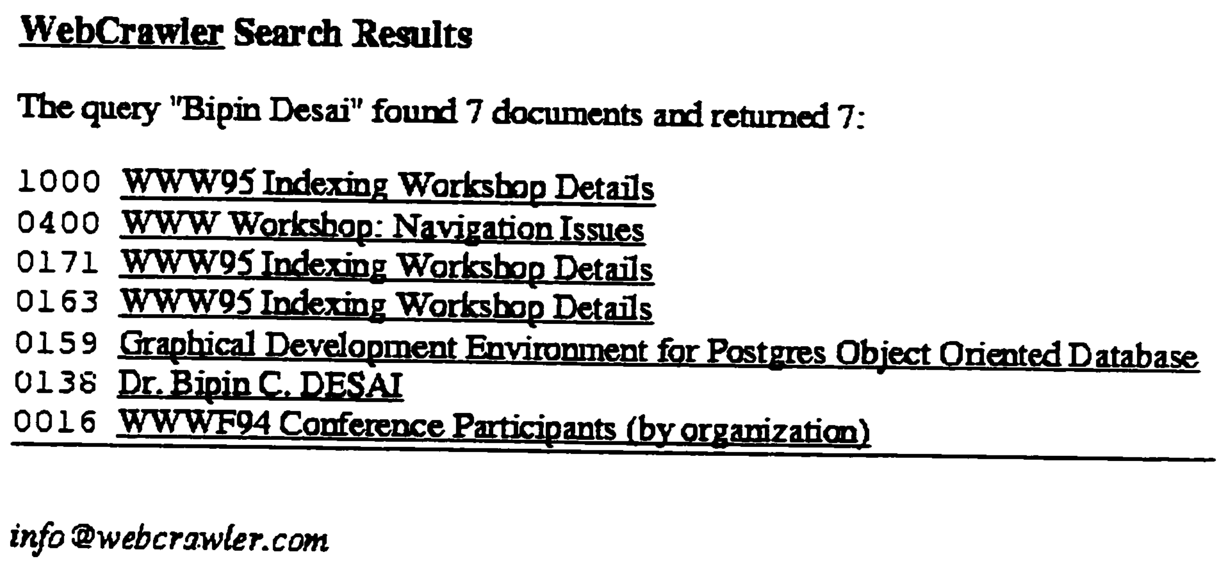


Figure 2: Web Crawler Search Interface



WebCrawler Search Results

The query "Bipin Desai" found 7 documents and returned 7:

1000 WWW95 Indexing Workshop Details
0400 WWW Workshop: Navigation Issues
0171 WWW95 Indexing Workshop Details
0163 WWW95 Indexing Workshop Details
0159 Graphical Development Environment for Postgres Object Oriented Database
0138 Dr. Bipin C. DESAI
0016 WWWF94 Conference Participants (by organization)

info@webcrawler.com

Figure 3: Results of Web Crawler Search System

2.2.5 Web Crawler

In this search system, the documents are indexed by keywords in a single database [WebC]. The system uses links referenced by a document to organize its search (breadth-first search). The documents are retrieved by 'agents' using the CERN's WWW library. The user can add new references to documents into the database. The indexing is updated about every week. The User Interface of this search engine is shown in Figure 2. The results obtained with the terms 'Bipin Desai' is given in Figure 3.

2.2.6 WWWW (World Wide Web Worm)

The search is done only in one database as well, by keywords, or by regular expressions similar to those used with the UNIX 'egrep' command. The user can retrieve documents based on one of the search types shown in Table 2 [MBO].

The database is built by a robot that locates WWW resources and constructs corresponding indexes. Nevertheless, the user can directly submit a URL to the robot, and can also associate keywords to this URL. Hence, indexes in the WWWW database are created manually and automatically. Since the database is centralized in only one server, the risk of overloading is high when multiple users make requests to the search system.

Search Type
HTML Title Strings.
URL Hypertext References.
HTML Title string of HTML containing a URL.
Any component of the URL name of a URL.
Unions of components of the URL name.

Table 2: WWWW Search Types

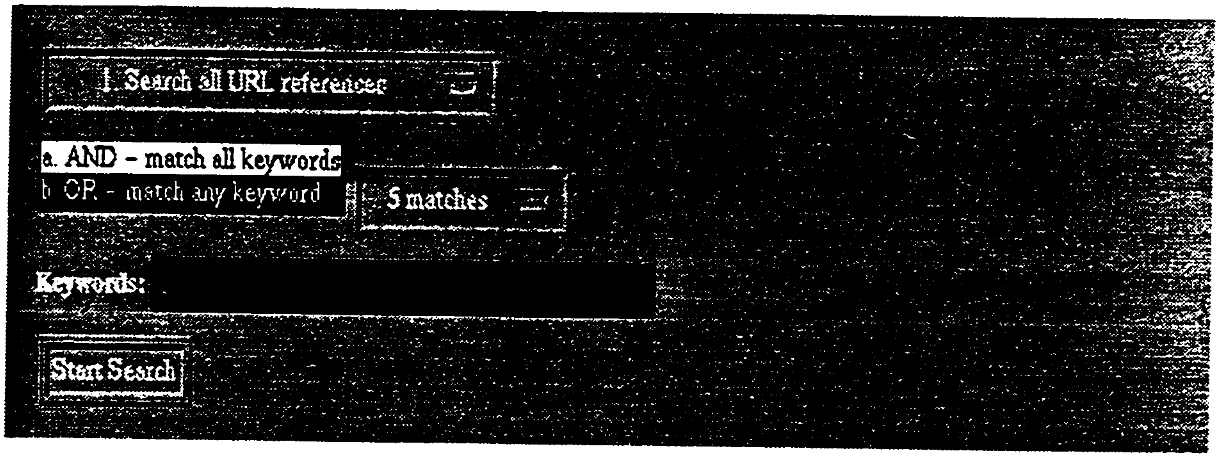

1. Search all URL references
a. AND - match all keywords
b. OR - match any keyword
5 matches
Keywords:
Start Search

Figure 4: WWWW Search Interface

The User Interface of this search engine is shown in Figure 4. We tried to do a search with the terms 'Bipin Desai'. No result is obtained based on network or title names. But the search in the hypertext citations gave one result, shown in Figure 5. The comments about this result follow:

- The World Wide Web Worm indicates that the string 'Bipin C. Desai' is the name of an anchor which belongs to the document located at the URL: 'http://www.cs.concordia.ca/people/People.html'.

  Moreover, WWWW allows the user to retrieve both the document referenced by the anchor, and the document that contains the anchor.
- An important drawback of the given result is that WWWW does not mention anything about the name of the document referenced by the anchor, or the name of the document containing the anchor. Hence, the user has to retrieve the document in order to access its title!

Actual keywords used: help and deal

1. [illegible]

Cited in: [illegible]

Figure 5: Results of WWWW Search System

2.2.7 ALIWEB

ALIWEB [Kos] is a search system that operates in a database containing about 7473 indexes. The system provides a way of duplicating entirely this database on other sites, which is called 'Mirroring the ALIWEB Data'. When an update occurs, all sites containing a copy have to be updated. The indexes give information about different types of objects: documents, services, organizations, and users. Each index has a title, a type, a network address (URL), an informal description, and a set of keywords. The index can contain more information depending on the value of the type. For instance, the index relative to an organization contains a phone number, a fax number, electronic and postal address, etc. To register in the database the reference of a document or a service, the user has to write the fields of the index in a file with the standard IAFA (Internet Anonymous FTP Archives) format. The details of this format can be found at [DPE]. An example of a record is given below:

Template-Type:	DOCUMENT
Title:	Aviation Related Items
URI:	/users/jpo/aviation.html
Description:	Various Aviation related information, including A summary of aviation teminology; information about pilots and ratings; differences between US and UK flying; a summary of the UK PPL Flight training and General Flight test examinations. Some links to other WWW/gopher sites.
Keywords:	aviation, flying, pilot, training, terminology
Author-Email:	j.onions@nexor.co.uk

When users write their descriptions in a standard format into a file, they inform

ALIWEB of this file. Subsequently, ALIWEB retrieves all these files, and combines them into a searchable database.

To do a search, the user has to specify first the database in which s/he wants to do the search. Then s/he enters a search term, specifies a search type and the field(s) s/he likes to retrieve in addition to the title (viz. Description, keywords, URL's, ...). The User Interface of this search engine is shown in Figure 6. The search result on 'Bipin Desai' is shown in Figure 7.

Advantages: The system provides an informal description of the content of each document.
Drawbacks: The size of the ALIWEB database is still less that 1 megabytes. The information contained in this system is relatively small compared to the huge amount of documents available on the Internet. Obviously, the larger the database becomes, the more difficult it will be to have enough disk space, in order to manage this centralized database and the updating of multiple copies.

2.2.8 RBSE Spider

Available on Mosaic, the Repository Based Software Engineering (RBSE) Spider [RBSE], given keywords, explores not only the title, but the full content of the documents, using the WAIS protocol. Currently the RBSE's database contains 179116 indexed documents. All these documents are in the HTML format and accessible via the http protocol. The User Interface of this search engine is shown in Figure 8. The results obtained with the search terms 'Bipin Desai' is in Figure 9.

2.2.9 Harvest

The Harvest architecture consists of four parts: the Gatherer, the Broker, the Object Cache and the Replication Manager [HarS]. The Gatherer summarizes information at archive sites and creates object summaries called Summary Object Interchange Format (SOIF) objects. The Broker provides a network-accessible object database, formed by collecting SOIF objects from Gatherers and other Brokers. This maintains a loose consistency between the Broker databases and the files found on gathered archives. Harvest users search for archived objects through a query interface to the Brokers database of object summaries. To decrease network and server load, the

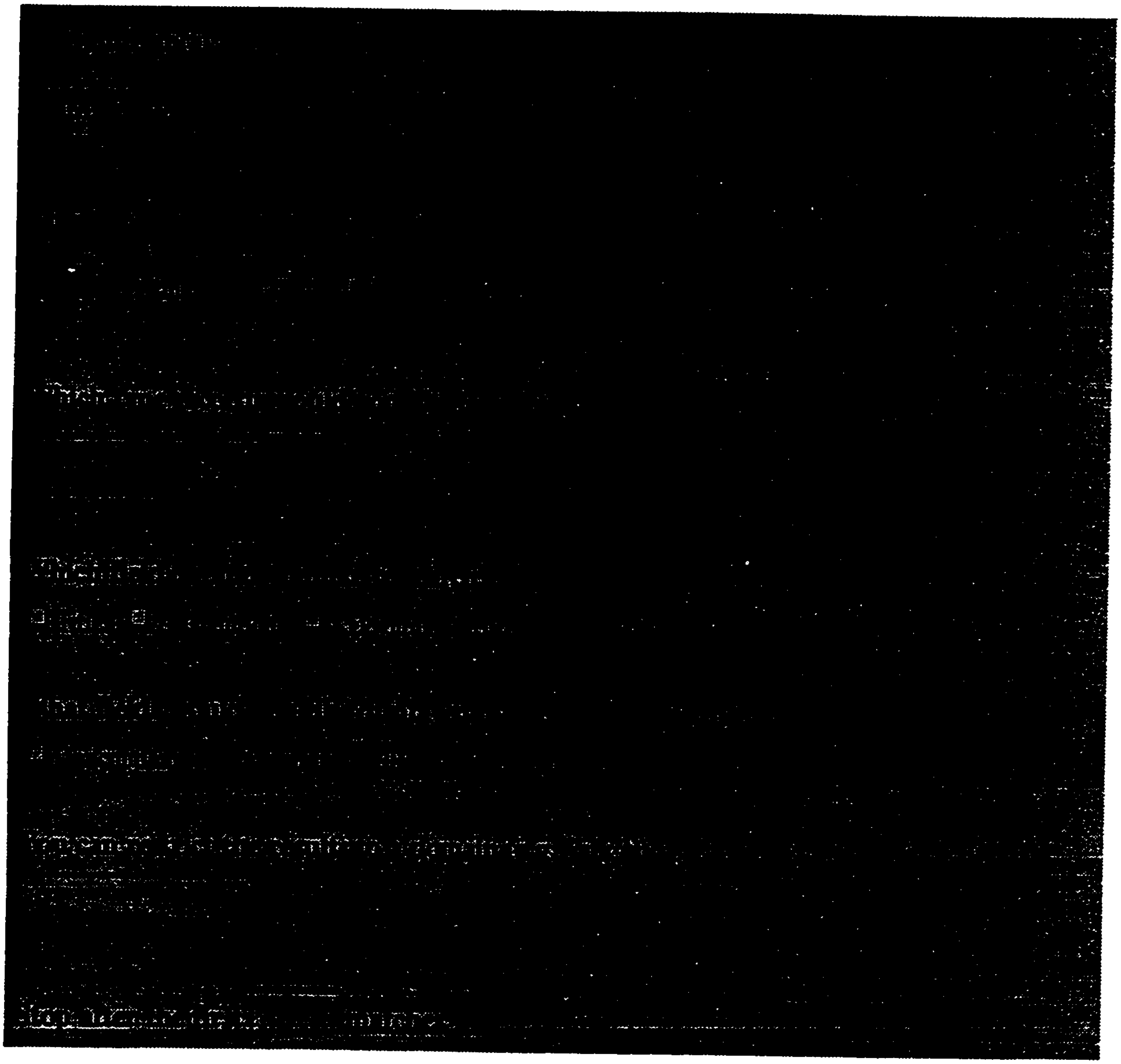
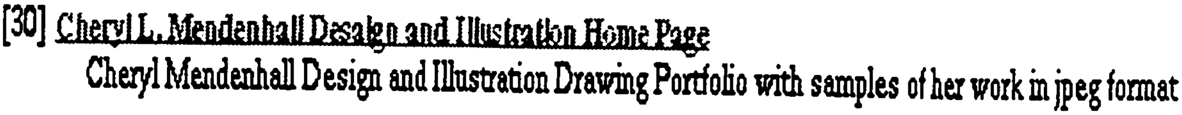


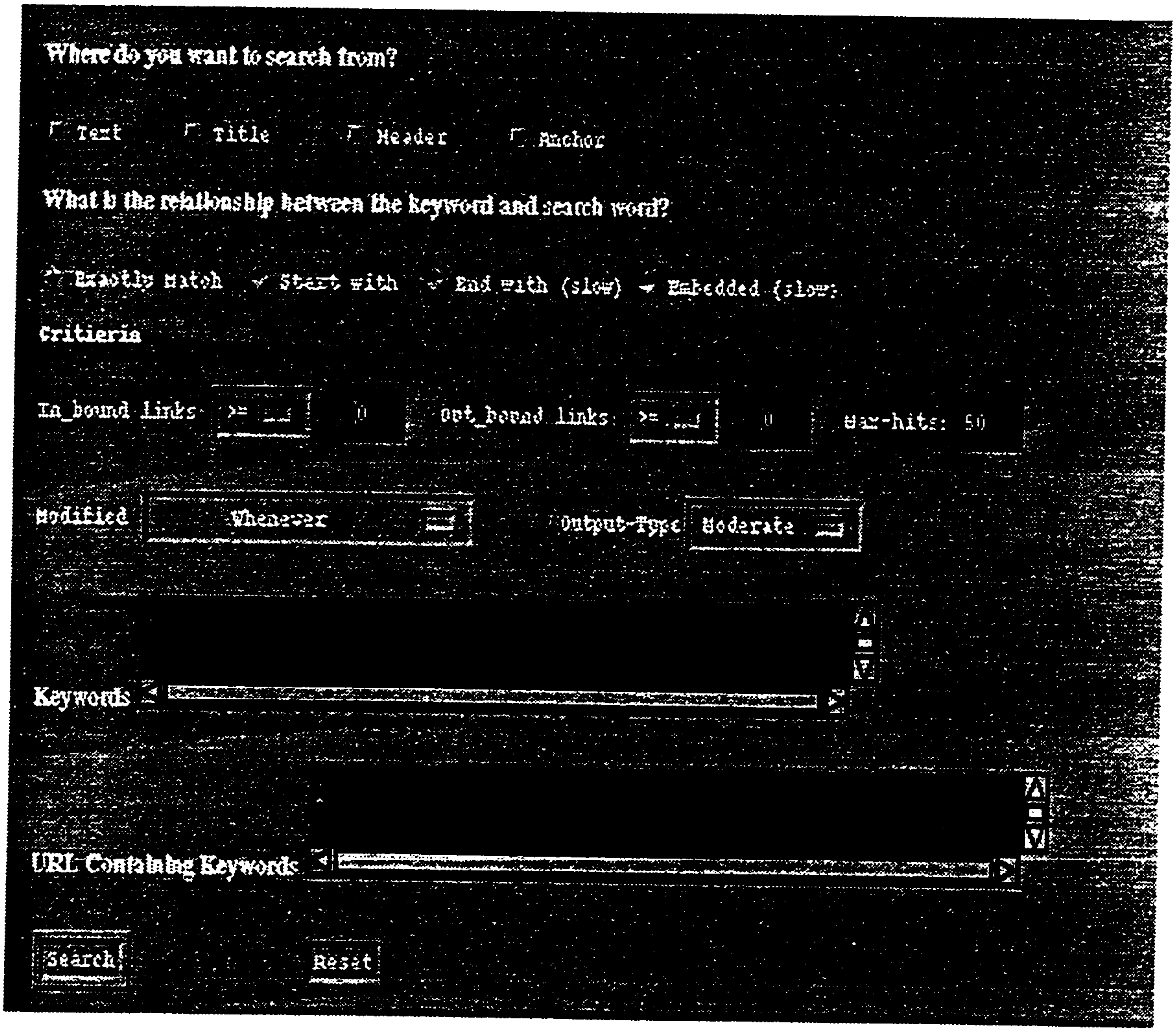
Figure 6: ALIWEB Search Interface



[30] Cheryl L. Mendenhall Design and Illustration Home Page

Cheryl Mendenhall Design and Illustration Drawing Portfolio with samples of her work in jpeg format

Figure 7: Result of ALIWEB Search System



Where do you want to search from?

□ Text □ Title □ Header □ Anchor

What is the relationship between the keyword and search word?

○ Exactly match ✓ Start with ○ End with (slow) ○ Embedded (slow)

Critieria

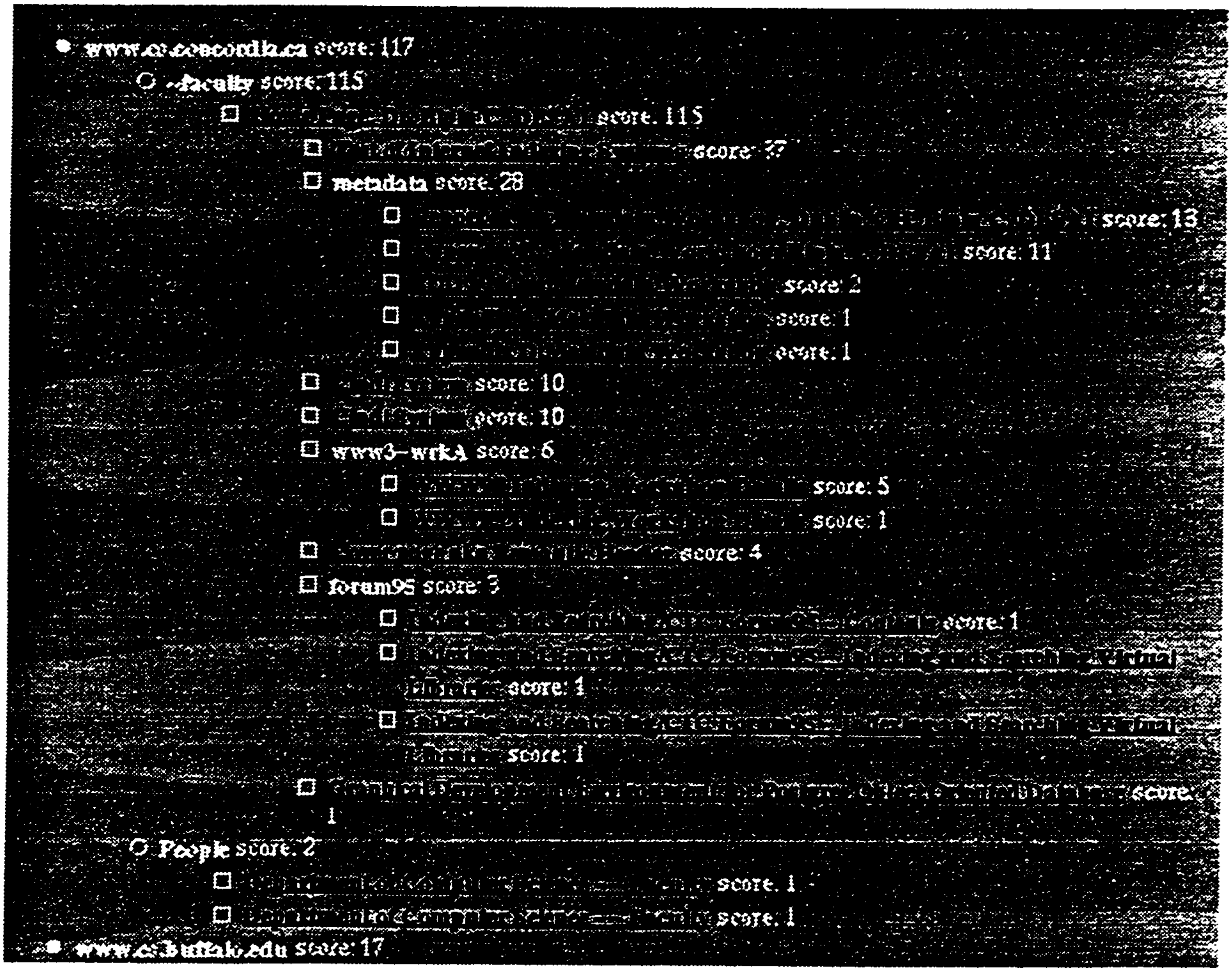
In_bound Links >= 0 Out_bound links >= 0 Max-hits: 50

Modified Whenever Output-Type Moderate

Keywords

URL Containing Keywords

Figure 8: RBSE Spider Search Interface



```
● www.cs.concordia.ca score: 117
    ○ ~faculty score: 115
        □ [illegible] score: 115
            □ [illegible] score: 37
            □ metadata score: 28
                □ [illegible] score: 13
                □ [illegible] score: 11
                □ [illegible] score: 2
                □ [illegible] score: 1
                □ [illegible] score: 1
            □ [illegible] score: 10
            □ [illegible] score: 10
            □ www3-wrkA score: 6
                □ [illegible] score: 5
                □ [illegible] score: 1
            □ [illegible] score: 4
            □ forum95 score: 3
                □ [illegible] score: 1
                □ [illegible]
                  [illegible] score: 1
                □ [illegible]
                  [illegible] score: 1
            □ [illegible] score:
              1
    ○ People score: 2
        □ [illegible] score: 1
        □ [illegible] score: 1
● www.cs.buffalo.edu score: 17
```

Figure 9: RBSE Spider Search Results

Replication Manager maintains several identical copies of a Broker. The Object cache maintains local copies of popular objects to improve access performance. The search interface of Harvest is illustrated in Figure 10. No result was obtained with the search term 'Bipin Desai' in this search system.

2.2.10 Lycos

Developed at Carnegie-Mellon University, the Lycos search system allows the user to make search requests into a catalog containing over 66 million Web pages (Jan. 1997), with an elaborate search language. The database is updated once a week. The user can do a search by object names, keywords and full content. The search gives priority to the documents that have the maximum occurences of the keywords. The User Interface of Lycos is shown in Figure 11. An example of the results obtained with the search terms 'Bipin Desai' is shown in Figure 12 [LYC].

2.2.11 System for Navigational Search on the Internet

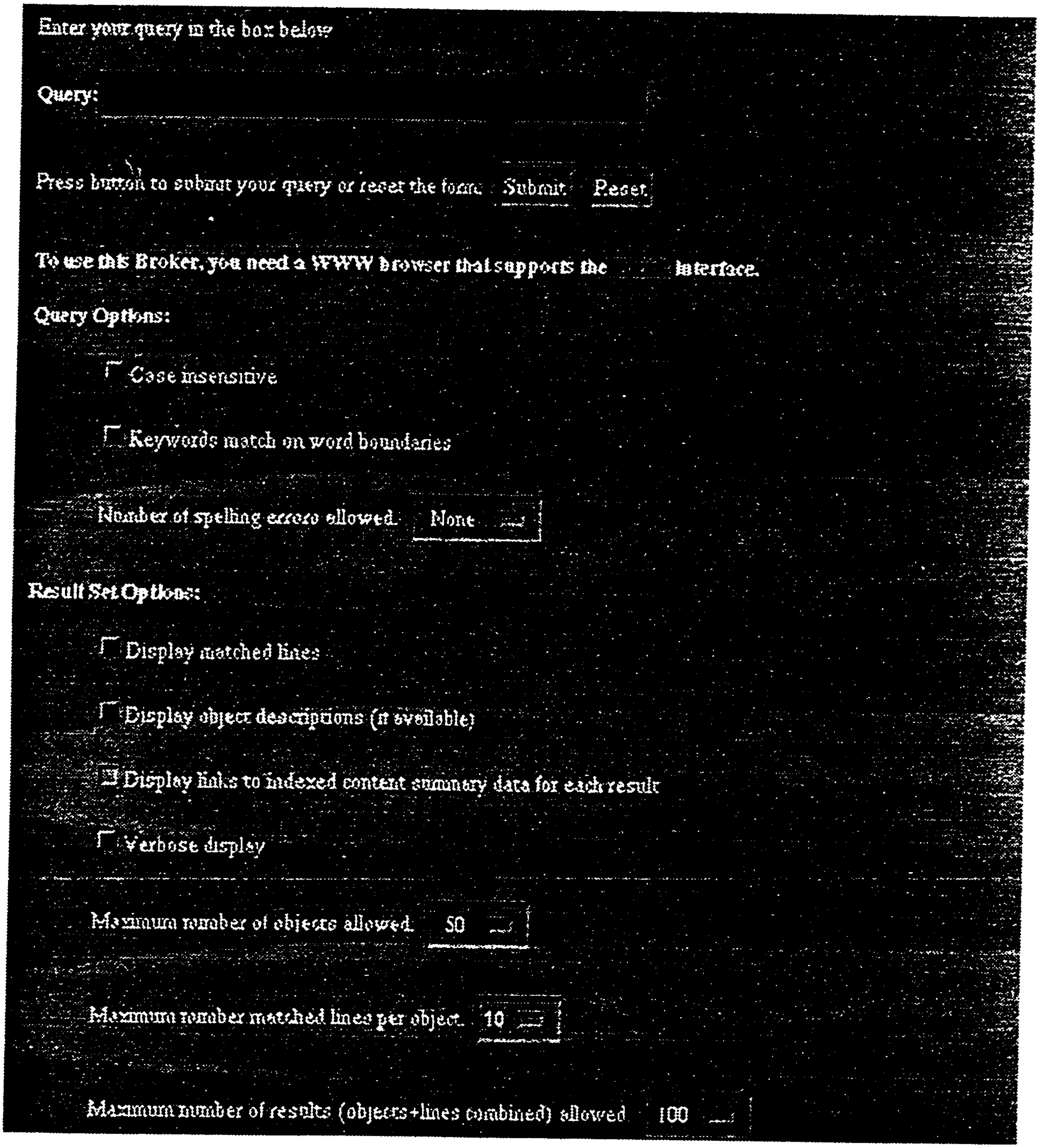
This system is currently in development. It will provide an addition to Mosaic for finding a set of HTTP nodes that match one or more keywords. This will be done by adding a keyword search field at the bottom of the Mosaic User Interface [DBP].

2.2.12 Yahoo

Yahoo is a tree hierarchically organized by subjects, that is quite similar to EINet Galaxy [Yah].

2.2.13 InfoSeek

InfoSeek is a search system on World Wide Web that has two versions: a commercial one, and a free one. The free search facility is limited because the maximum number of hits returned is equal to 10. InfoSeek claims to be 'the most comprehensive and accurate WWW search on the Internet'! Unfortunately, we couldn't test the veracity of this claim because the commercial InfoSeek was not available at Concordia university. The User Interface of the system and the results obtained by the search terms 'Bipin Desai' are given in Figures 13 and 14, respectively [InS].



Enter your query in the box below

Query:

Press button to submit your query or reset the form. Submit Reset

To use this Broker, you need a WWW browser that supports the interface.

Query Options:

☐ Case insensitive

☐ Keywords match on word boundaries

Number of spelling errors allowed: None

Result Set Options:

☐ Display matched lines

☐ Display object descriptions (if available)

☒ Display links to indexed content summary data for each result

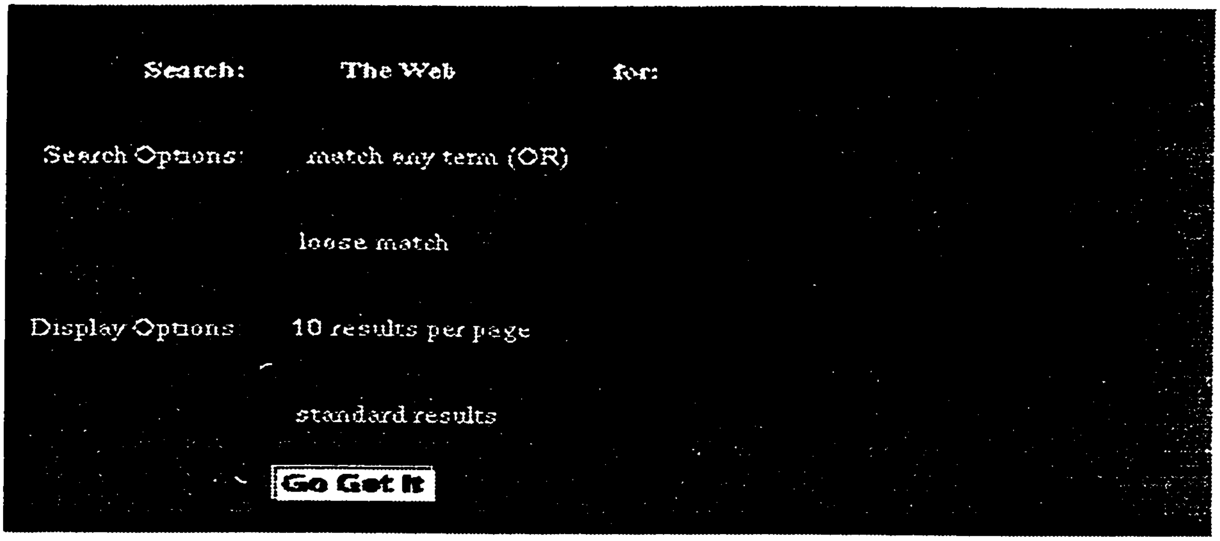
☐ Verbose display

Maximum number of objects allowed: 50

Maximum number matched lines per object: 10

Maximum number of results (objects+lines combined) allowed: 100

Figure 10: Harvest Search Interface



Search: The Web for:

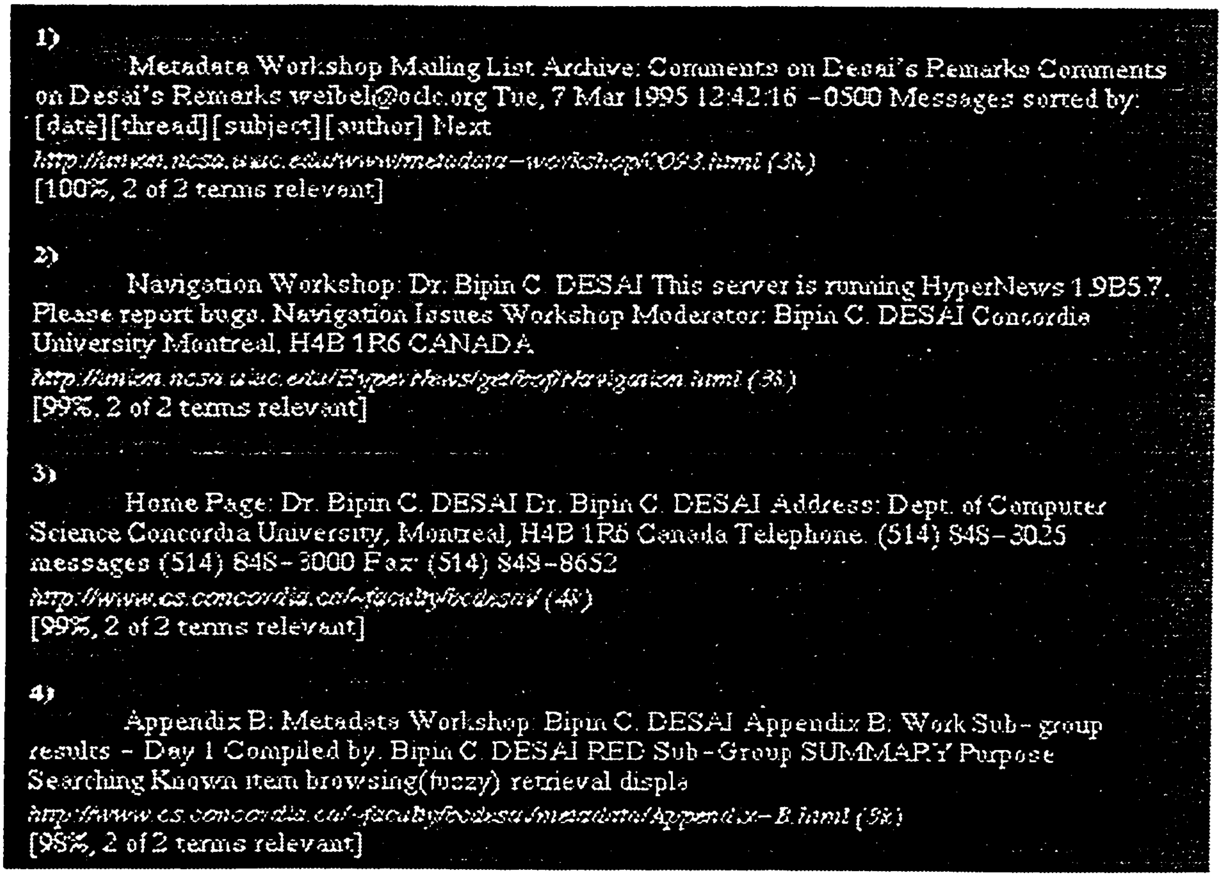
Search Options: match any term (OR)

loose match

Display Options: 10 results per page

standard results

Figure 11: Lycos Search Interface



1)

Metadata Workshop Mailing List Archive: Comments on Desai's Remarks Comments on Desai's Remarks weibel@oclc.org Tue, 7 Mar 1995 12:42:16 -0500 Messages sorted by: [date][thread][subject][author] Next

http://union.ncsa.uiuc.edu/www/metadata-workshop/0093.html (3k)

[100%, 2 of 2 terms relevant]

2)

Navigation Workshop: Dr. Bipin C. DESAI This server is running HyperNews 1.9B5.7. Please report bugs. Navigation Issues Workshop Moderator: Bipin C. DESAI Concordia University Montreal, H4B 1R6 CANADA

http://union.ncsa.uiuc.edu/HyperNews/get/www/navigation.html (3k)

[99%, 2 of 2 terms relevant]

3)

Home Page: Dr. Bipin C. DESAI Dr. Bipin C. DESAI Address: Dept. of Computer Science Concordia University, Montreal, H4B 1R6 Canada Telephone. (514) 848-3025 messages (514) 848-3000 Fax: (514) 848-8652

http://www.cs.concordia.ca/~faculty/bcdesai/ (4k)

[99%, 2 of 2 terms relevant]

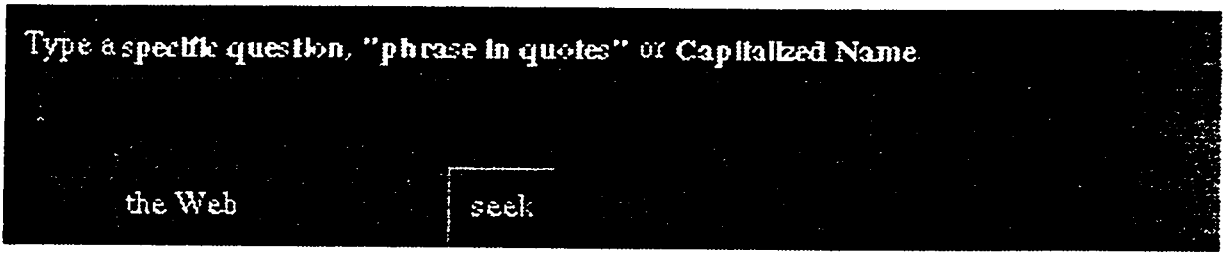
4)

Appendix B: Metadata Workshop Bipin C. DESAI Appendix B: Work Sub-group results - Day 1 Compiled by. Bipin C. DESAI RED Sub-Group SUMMARY Purpose Searching Known item browsing(fuzzy) retrieval displa

http://www.cs.concordia.ca/~faculty/bcdesai/metadata/Appendix-B.html (9k)

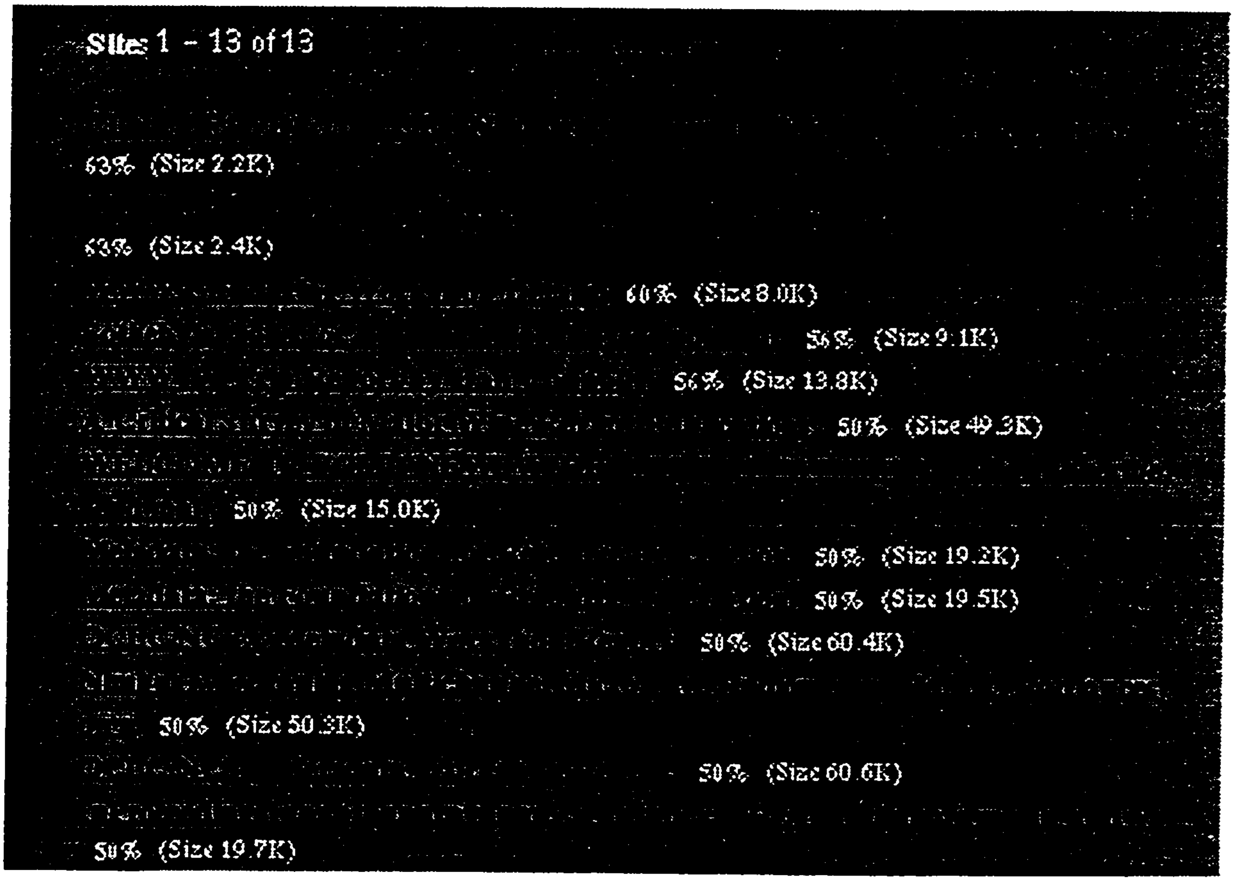
[98%, 2 of 2 terms relevant]

Figure 12: Lycos Search Results



Type a specific question, "phrase in quotes" or Capitalized Name

Figure 13: User Interface of InfoSeek



Sites 1 - 13 of 13

63% (Size 2.2K)

63% (Size 2.4K)

60% (Size 8.0K)

56% (Size 9.1K)

56% (Size 13.8K)

50% (Size 49.3K)

50% (Size 15.0K)

50% (Size 19.2K)

50% (Size 19.5K)

50% (Size 60.4K)

50% (Size 50.3K)

50% (Size 60.6K)

50% (Size 19.7K)

Figure 14: Results of the InfoSeek System

2.3 Drawbacks of aforementioned systems

Searching in these systems can be very cumbersome, because of the following reasons:

- The data is not organized and different indexing systems can classify the same document in two different manners. *Hence the need for a standard index scheme that ensures homogeneity of the terms and organizations.*
- Non-initiated users can navigate a long time through menus if they do not know how to express perfectly their query, and if they do not put the judicious keyword. *Hence the need for an expert system that guides the users and helps them to specify standard keywords or subjects.*

In the next chapter we will describe the requirements of the CINDI System and point out the advantages of the system.

Chapter 3

CINDI, the Concordia INdexing and DIscovery System

In chapter 1, we discussed the major concerns regarding the cataloging and discovery of hypermedia information resources. This problem will be exacerbated due to the exponential growth of the Internet. We also uncovered some of the shortcomings of current search systems on the Internet and came to the conclusion that these systems are not flexible enough to meet the expectations of ever-growing number of providers and users of such resources.

In this project we will use the proposed solutions (outlined in section 1.2) to design and implement a metadata based system, called the Concordia INdexing and Discovery (CINDI) System, for indexing and searching hypermedia documents available on the Internet. To do so, we will start by introducing the concept of Semantic Header (Section 3.1). Then we will describe its major components as well as their interactions.

3.1 Metadata Revisited

In this section we present two recent initiatives for allowing suppliers of resources to prepare well thought out catalog information for their resources.

3.1.1 Dublin Metadata Core Elements

The Metadata Invitational Workshop was held in March 1995 in Dublin, OH [BCD95a]. The main objective was to address the problem of cataloging network resources with adoption, extension or modification of current standards and protocols 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 following assumptions were made to develop a consensus to arrive at a minimum set of core data elements.

- The elements are intended to describe a Document-Like Object.
- Common (or core) element set: These are metadata elements that apply to most/many DLOs.
- The elements of the core are chosen to support resource discovery.
- All elements of the metadata are repeatable.
- All elements are optional.
- All elements describe the DLO itself with the exception of the SOURCE element, which can be thought of as a recursive instance of the entire record, except that it applies to an object from which the electronic record is derived.
- The core elements are intended to describe intrinsic characteristics of the DLO...thus, transactional data, archival status, and copyright characteristics (as well as others) are not included in this set.
- No assumption is made concerning whether the DLOs are network accessible or specifically electronic.
- The core element set assumes an arbitrarily complex hierarchy.
- Elements not included in the core set are not specifically excluded.

The following elements emerged as the ones required in a minimum set. They were dubbed the *Dublin Metadata Element List (DMEL).*

- **subject:** words are phrases indicative of the information content.
- **title:** the title, name or short description of the object.
- **author:** the name of the creator of the content.
- **other-agent:** the name of any other entity responsible for the content of the object.
- **publisher:** the name of the entity responsible for making the object available.
- **date:** the date of publication.
- **identifier:** a character string or a number used to distinguish this object from other objects.
- **object-type:** conceptual description of object.
- **form:** physical, logical, or encoding characteristics.
- **relation:** important relationship to other objects.
- **language:** natural language of the content of the object.
- **source:** object from which derived; contains a nested object description.
- **coverage:** characterizes parameters to specify the audience, or the time or space.

The following are some of the concerns [BCD97]:

- Designing all elements of the DMEL as optional may create a problem; a minimum set should be required.
- The user may need some extrinsic characteristics of the resource such as its cost and hence should be included (though optional). All documents of any permanence should be archived and accessible for an indefinite period of time.
- Subject element should be made up of two sub-fields, a schema and a hierarchical subject field which includes sub-subject and sub-sub-subject.
- For non-titled objects, we need an algorithm to insert a title or an alternate title.
- Relation and source have similar semantic implications and could be described using a relationship sub-field with the identifier of the object to which it is related. An optional sub-field may be used to provide annotation or useful information.

- In addition to the language of the DLO, the (natural) language used to specify the elements of the metadata and its character set should be specified. Furthermore, elements such as an abstract and annotations are *missing*. The former, generated by a human or other agent gives a capsule idea of the DLO. The latter is a placeholder for additional details regarding the DLO by responsible agents or other users.

In the next section, we propose a new set of metadata elements which is predated by DMEL but was modified to include the concerns raised by it and hence is better suited to the users' needs.

3.1.2 Semantic Header

For cataloging and searching, we use a metadata description called a Semantic Header to describe an information resource[1]. The intent 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, name of the authors (70%), subject and sub-subject (50%) [Katz], we have made the entry of these elements mandatory in the Semantic Header. The abstract and annotations are, as well, relevant in deciding whether or not the resource would be useful; these items are also included. The elements of the Semantic Header are described briefly below:

Title, Alt-title:

The first field of the Semantic Header is the title[2] of the resource. It is a name given to the resource by its creator(s) and is a required field. The alternate title field is used to indicate an 'official' secondary title or an alternate title of the resource. The alternate title is an optional element.

Subject:

The subject and sub-subjects of the resource are indicated in the next field which

[1] This section is from [BCD97].

[2] Title for non document like resources may require some creativity. For instance, the title of a satellite image could be generated from the name of the satellite, its location, date, time, camers, frequencies, filters, etc.

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 for this field.

Language, Character set:

The character set used and the language of the resource is given in the next two optional fields.

Author and other responsible agents:

The details about the author(s) and/or other agent(s) responsible for the resource is given in the next repeating group[3]. The sub-fields are: role[4] of the agent, name, organization, address, phone and fax numbers, and e-mail address. All sub-fields save the name are optional, except in the instances of corporate entities in which case the organization must be given. By using the role sub-field and giving it appropriate value, semantics for agents such as editor or publisher are incorporated in this repeating group.

Keyword:

The list of keywords is included in this field. It is required to include at least one keyword.

Identifier:

The next element is 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. Each resource should possess at least one identifier. The domain could be an accepted or standardized coding scheme issued by an appropriate authority such as ISBN, ISSN, URL(FTP, GOPHER, HTTP)[5] [BLTC], or

[3] For resources such as satellite image, the agent may be the agency controlling the satelite or the satellite itself.

[4] Typical values for role of the agent could be author, co-author, designer, editor, programmer, creator, artist, corporate entity, publisher, etc.

[5] Universal Resource Locator, File Transfer Protocol, Hypertext Transfer Protocol.

URN[6] [Soll] etc., and the value contains the corresponding coded identifier. Since a resource in electronic form may be accessible from one or more sites there could be one or more entries for the same domain such as URL. The URN field gives the unique name of the resource, if any. This name may be used instead of a location (URL) if the item is likely to move or is accessible from multiple locations[7]. The identifier(s) can be used to locate the resource.

In the absence of an accepted standard for URN, we use an alternate name, called Semantic Header Name (SHN). The SHN is derived by concatenating the following required elements in the Semantic Header: title, name of first author (or name of organization, if the resource is attributable to a corporate or organizational entity), first subject, creation date, and version. With this scheme, the user supplied elements in the SHN, with a very small probability, may map to more than one resource. If multiple hits are encountered during a search based on user supplied elements of the SHN, the system would inform the user of the 'collision'. The user could then select the appropriate resource index entry by perusing the other elements recorded in the Semantic Header. The SHN can be used in a basic index for simple default search system.

The identifier entry in the Semantic Header may also contain an entry for an archive site. The domain value UAS (Universal Archive Site) is used to indicate the archive site for the resource. It is expected that the resource will exist at this site beyond its expiry date, if any. Of course, the site itself is guaranteed to exist beyond the life of any resource. It is envisaged that the archive site could be an independent resource provider. Examples of such traditional resource providers that would be feasible archive sites for the resource are the national libraries such as the Library of Congress in U.S., British Library, National Library and CISTI in Canada. However, private, for profit, corporations could be alternate sites for archiving resources. Archiving would provide an anchor for the otherwise ephemeral nature of some resources on the network. Since the archive site may not be known when the Semantic

[6] Universal Resource Name.

[7] The idea of the Semantic Header is to provide bibliographic information about resources and by including SHN and/or URN and a list of URLs it also provides a mapping from SHN and/or URN to URLs.

Header is first registered, the system would support update operations in which existing entries could be modified. Other update operations such as modification of addressed, URLs etc., would also be supported.

Date:

The dates of creation(required) and expiry, if any, are given next.

Version:

The version number, and the version number bieng 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 optional repeating group. It consists of a domain (target audience, coverage in a spatial and/or temporal term, etc.) and the corresponding value.

System Requirements:

A list of system requirements such as hardware and 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 values are: hardware, software, network) and the corresponding exigency.

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 its 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 the 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 resouce and an opportunity to registerit for resources of other formats.

Cost:

In the case of a resource accessible for a fee, the cost of accessing it[8] is given next. This element is optional.

Abstract and Annotations:

The abstract and annotations are two optional elements 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. Once registered, the annotations *cannot* be modified.

User ID, Password:

The last two items in the Semantic Header are the user ID and the password. Any change to the update-able part of the Semantic Header requires these fields to be filled in. Each user ID is assigned one and only one password.

[8] Such cost could change over time and requires updating.

A sample of a Semantic Header is shown in Appendix A.

3.2 Expert System

In cataloging and searching digital libraries, it is essential to employ the knowledge and expertise of reference librarians. However, employing professional librarians may require a great deal of time and cost. Thus, an expert system is required to model librarians expertise to guide users in cataloging and searching. The expert system will help the users choose correct subject terms. It will also guide them to register, update, delete, and annotate Semantic Headers. The expert system is designed so that its query for resource search facilitates efficient database access, and reduces the number of incorrect results generated. This requires the following aspects to be considered as part of its function:

1. *Choosing appropriate subject terms from the Thesaursus for cataloging and searching.* The expert system guides the user in selecting the most appropriate (three-level) subject hierarchies from an on-line Thesaurus. In addition, the system guides the user to convert terms entered by the controlled vocabulary drawn from the established subject heading and descriptions. This is done by allowing the user to enter synonyms or sub-strings of standard subjects. This facility is provided for both indexing and searching a Semantic Header.
2. *Providing context sensitive help for indexing, updating, and annotating a Semantic Header.* For example, the system aids the user in completing compulsory fields of a Semantic Header to be indexed.

The expert system designed for CINDI is divided into two components[9]:
Indexing Component: A Semantic Header is to be completed by the author of a resource. In addition to aiding the user to choose appropriate subject hierarchies from the Thesaurus, the expert system will also help the user in entering those fields required for indexing or updating a Semantic Header. As for adding annotation to an existing Semantic Header, the user will also be guided for entering compulsory fields.

[9] An earlier version of such an expert system is described in [CPG]. The current system has simplified the approach while reducing the number of rules

Retrieval Component: A graphical interface allows the user to enter search requirements. In searching for a given set of resources, often the users enter vague, partial, and incorrect information in identifying the terms of the index for the documents they are searching. In other words, the user search specification is often 'ill-structured' [SIM]; hence, expertise is needed to help users articulate their needs. In addition, the total number of input field combinations for document search to be handled is large. Only some of these input field combinations occur at any given time. For example, in a given search, the user may be interested only in the documents of a particular subject hierarchy (AI.ExpertSystems.Verification) without caring about the title of the document. In this case, it is impractical to consider search situations that also explicitly require the title of documents. Thus, by isolating and encoding the input combinations and handling them in rules, only a subset of the rules need be active to process the user input and provide appropriate help[10]. This also improves the system's modularity and understandability [JRJK, AGW].

It should be pointed out that this prototype version of the CINDI system will deal with indexing and searching resources related only with the *Computer Science* discipline. Hence, the standard terms used for expressing subjects and sub-subjects will be based on a standard classification of computer science areas. More precisely, the terms will be taken from the classification system of the *Computing Reviews* [CR91]. This classification can be compared to a tree, where the root is the general area Computer Science, the first level represents the subject, and the four lower levels represent the hierarchy of subsubjects. In order to include all subjects of the five levels of this classification system in our three-level subject hierarchy, the lowest two levels of the classification system have been merged with the third level of the hierarchy. For instance, the five-level hierarchy

Computer Science
  Software
    Programming Techniques
      Concurrent Programming
        Distributed programming

[10] A direct encoding of knowledge to check for all input field combinations one after another had been attempted earlier, but such a system was inefficient [CPG].

is divided into three three-level subject hierarchies such that each hierarchy is composed of Computer Science and Software as the first two subjects and one of the remaining three subjects (of the five-level hierarchy) as the third level subject.

3.3 User Interface

3.3.1 Index Entry Graphical Interface

The index entry and registering sub-system provides a graphical interface (Figure 15) to facilitate the provider (author/creator) of a resource to register the bibliographic information about the resource. The interface allows the provider to enter the information and it offers help by means of pop-up selection windows and an expert engine to suggest controlled terms. This expert engine is intended to bring some of the expertise of a catalog librarian to the ordinary user [CPG]. Many of the elements in the Semantic Header can be extracted directly from the resource document, if they are properly tagged[11]. Once the information is correctly entered, the author can decide to register the Semantic Header entry in the database. When the header information is accepted by the database, the author/creator is notified. A user ID and the associated password is to be provided when the Semantic Header is first registered and for all changes made to it. Since the user ID and the password are 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. Changes that may be made could be due to changes made in the resource or its migration from one system to another. A copy of the Semantic Header is stored at the site of the resource for convenience in later updates.

3.3.2 Search Graphical Interface

The search interface shown in Figure 16 allows a user to enter search requirements consisting of sub-strings and synonyms. The user is not aware of the underlying expert

[11] This work is underway in another project at Concordia.

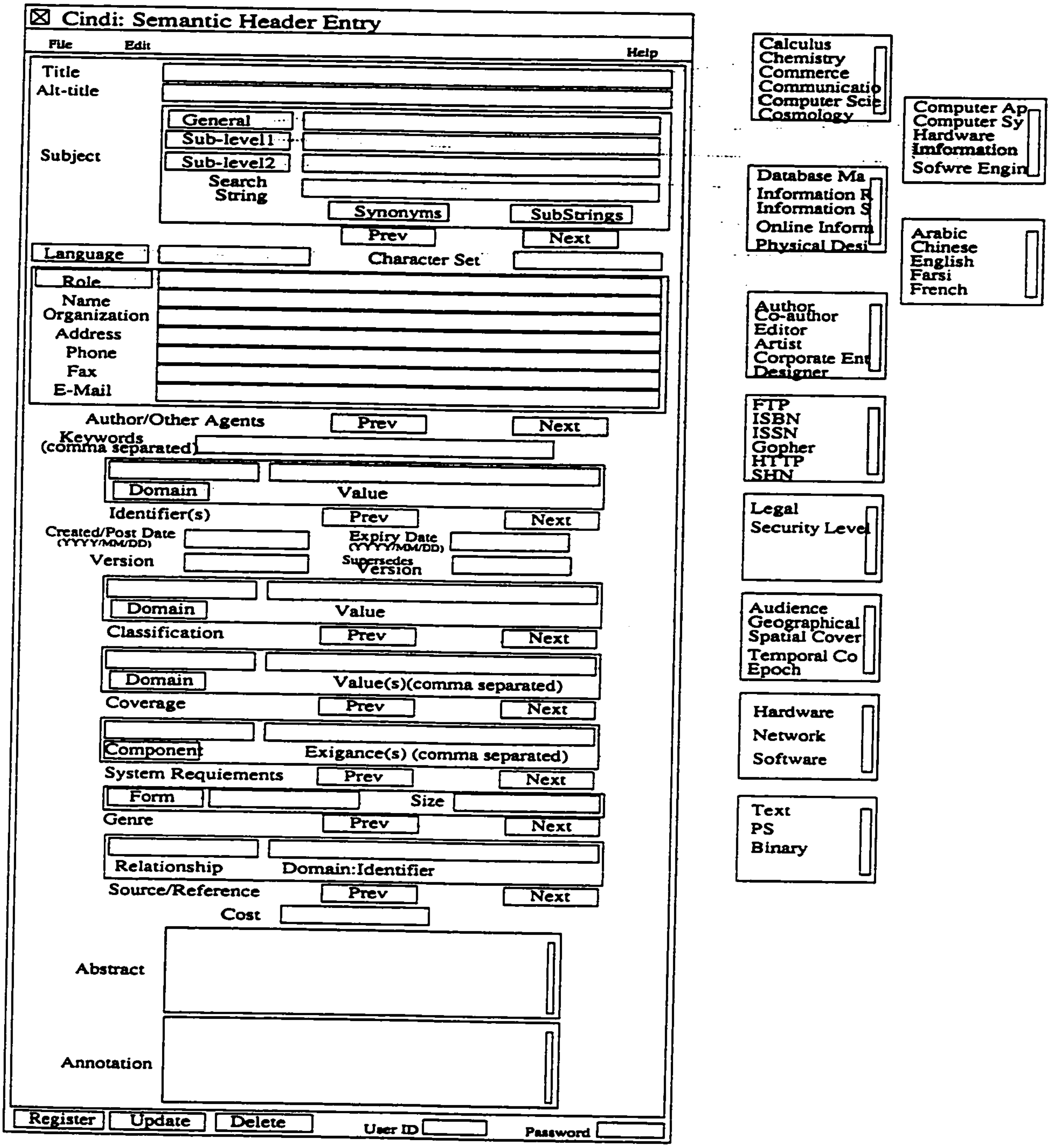


Figure 15: The Index Entry User Interface

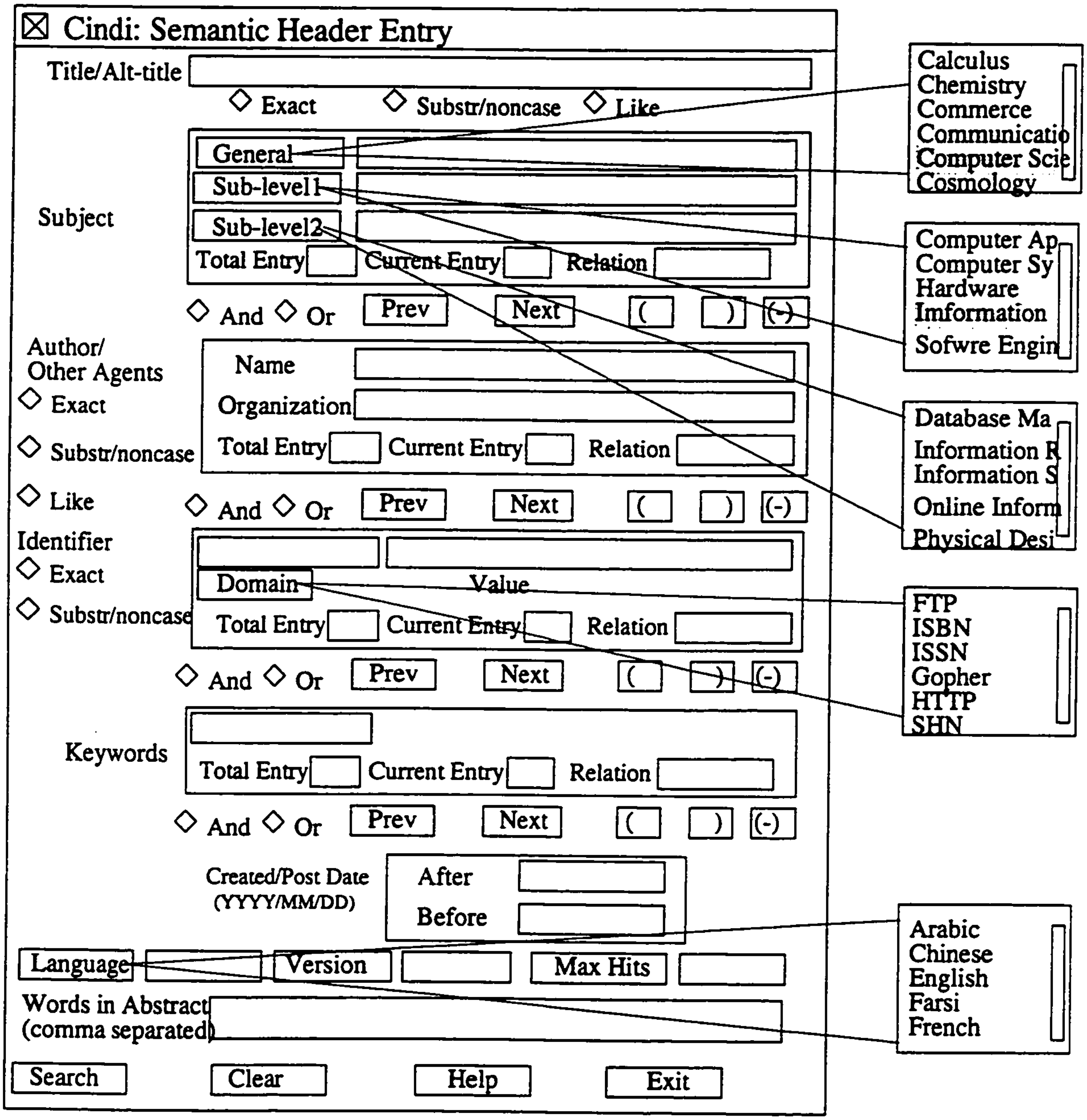


Figure 16: The Search User Interface

system that provides the help as outlined earlier. The search interface provides for the precise statement of a user query by allowing complex predicates based on search items such as author, subject, keywords, dates, and words appearing in the abstract.

3.3.3 Annotation Graphical Interface

The annotation subsystem is similar to the indexing subsystem. However, only a few of the indexing entries, to uniquely identify the resource in question, are required (Figure 17). An annotation made by any user can be entered, by means of the annotation interface, and would be registered with the identity of the user. Each annotation could then 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.

3.4 Database

The index entries registered by a provider of a resource is stored in a Semantic Header Distributed Database system (SHDDB). From the point of view of the users 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 [BCD90].

It is envisaged that the 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 the thesaurus for all subjects as well as the information concerning the locations of

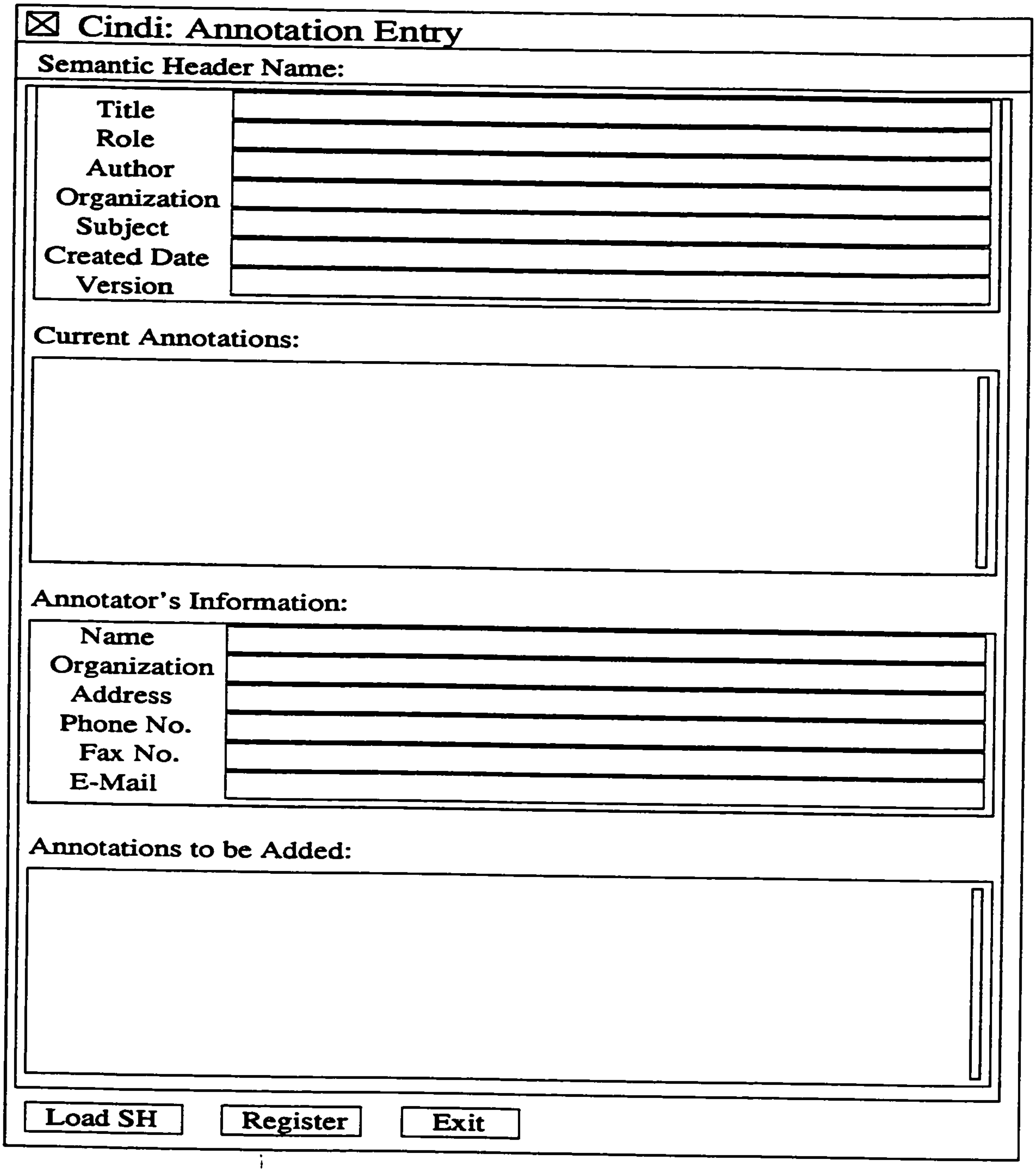


Figure 17: The Annotation User Interface

Semantic Headers, pertaining to a subject, in the distributed system.

The overall structure of the CINDI System is illustrated in Figure 18. The Semantic Header information entered by the provider of the resource using 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 on 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 forwards the replicated information to appropriate nodes. For example, the Semantic Header entry shown in Appendix A would be part of the SHDDB for subjects Computer Science and Library Studies.

Similarly 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.

On 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 user appropriate hardware/software to retrieve the selected resources.

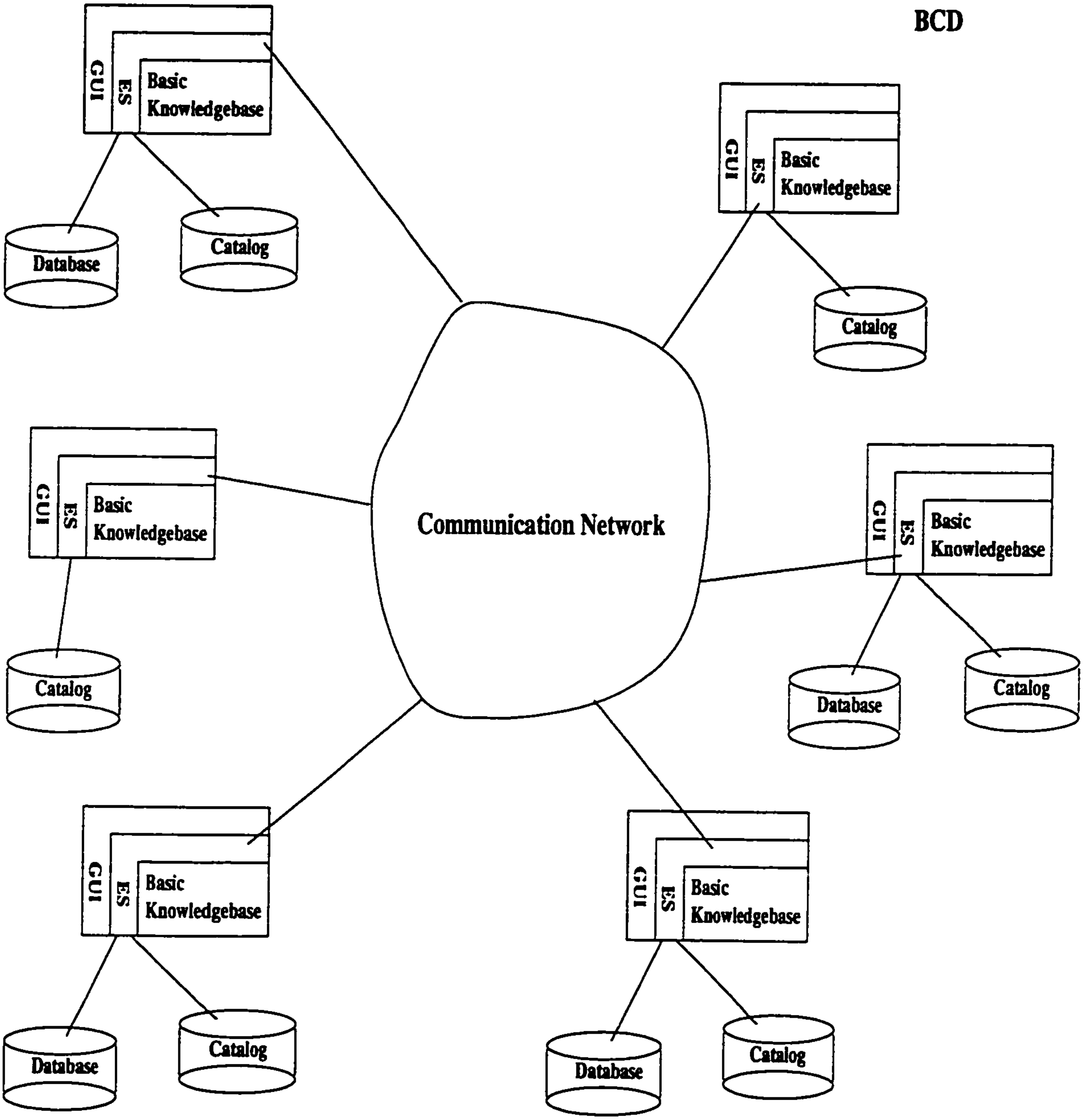


Figure 18: Overall Structure of the CINDI System

3.5 Advantages of the CINDI System

Comparing the proposed system with existing search systems, the following advantages are easily recognizable:

- The Semantic Header allows the indexation of resources accessible on-line or not on-line.
- A majority of the existing search systems do not provide the possibility for the user to make comments about the resource and to read annotations of other people.

  The Semantic Header will contain annotations of reviewers.
- The existing search systems often employ imprecise indices (title only, full content, non standardized keywords, etc.).

  The Semantic Header syntax provides a way to register standardized keywords, and these are chosen by the provider of the resource.
- Many search engines do not give, in their search results, an informal description of the resource; the additional information provided by them is an extraction from the first part of a resource.

  The Semantic Header has an 'abstract' field.
- The size of the index database is not limited since the database is distributed amongst different sites.
- The Semantic Header may become a part of each document. The format used to write the content of the header allows its display by the Internet browsers.
- In existing self indexing systems, one of the limitations is the low number of indexed resources. The problem is to convince people to submit to the system the resources they put on the Web. This problem is solved in the CINDI System by means of an enforcement procedure which forbids one author to submit his/her resource to the Web if the Semantic Header has not been created[12].
- The registration of the Semantic Header in the database is performed by the provider of the resource, which improves cost, accuracy and efficiency.

[12] Furthermore, automatic generation of Semantic Headers from resources is one of the works in progress.

Chapter 4

Analysis of the CINDI System

4.1 Choosing a Design Methodology

There are three important aspects in modeling a system: static, dynamic, and functional. These related aspects can be well described and differentiated by the Object Modeling Technique (OMT). In OMT, the *object model* represents the static, structural, 'data' aspects of a system. The *dynamic model* represents the temporal, behavioral, 'control' aspects of a system. The *functional model* represents the transformational, 'function' aspects of a system [Rum91]. In this chapter, we will model the CINDI System using OMT by elaborating on these three models.

4.2 Object Model

The object model describes the structure of objects in a system, namely, their identity, their relationships to other objects, their attributes and their operations. The object model provides the essential framework into which the dynamic and functional models can be placed.

4.2.1 Object Model of the system

The system is divided into five subsystems: *User Interface*, *Semantic Header*, *Thesaurus*, *DB Server Interface*, and finally *File System*. The *Thesaurus* subsystem itself, contains a subsystem called *Subject Hierarchy*. The *User Interface* subsystem resides in the user site (client site). Other subsystems reside in the database site (server

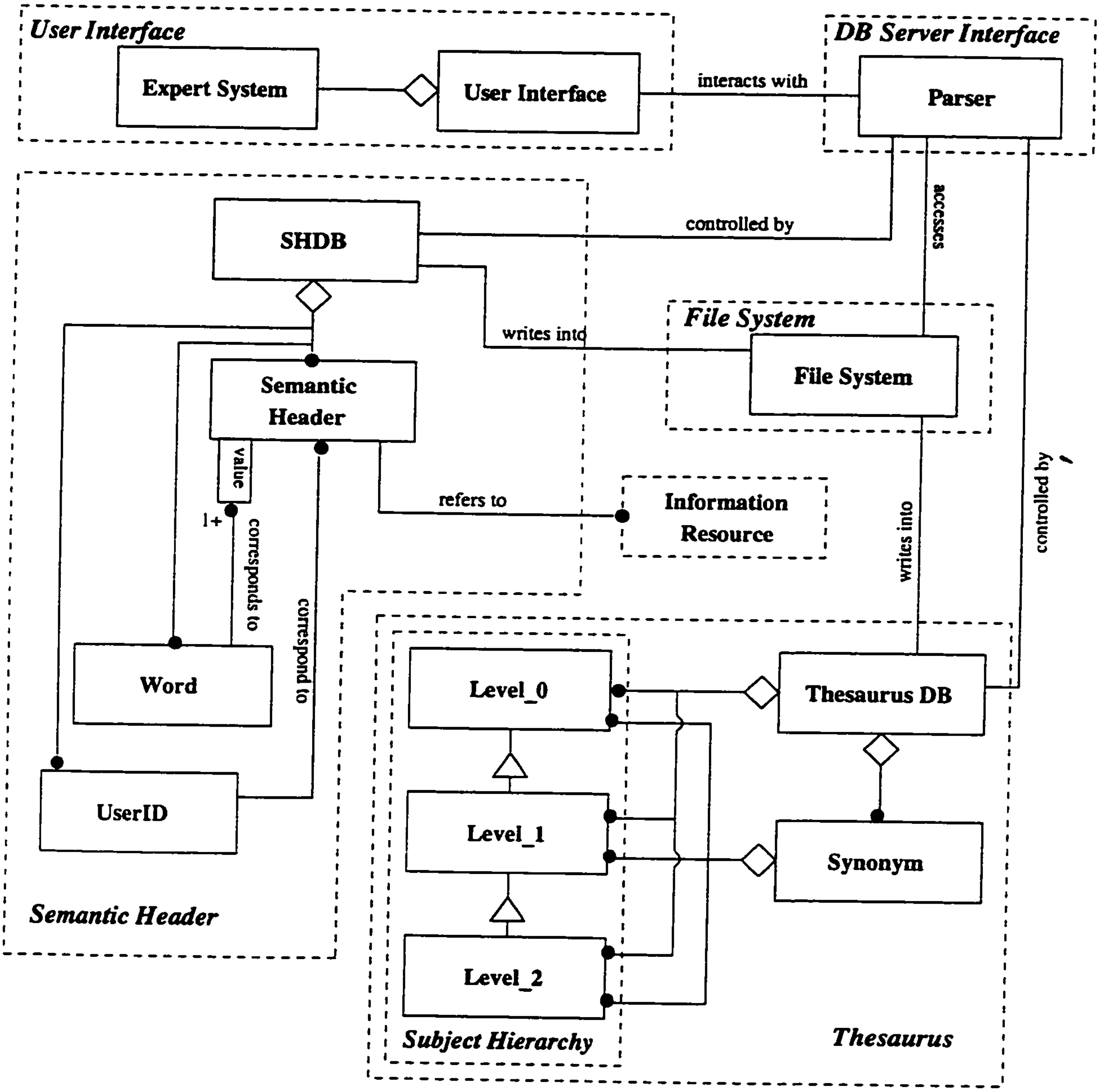


Figure 19: Object Model of the CINDI System

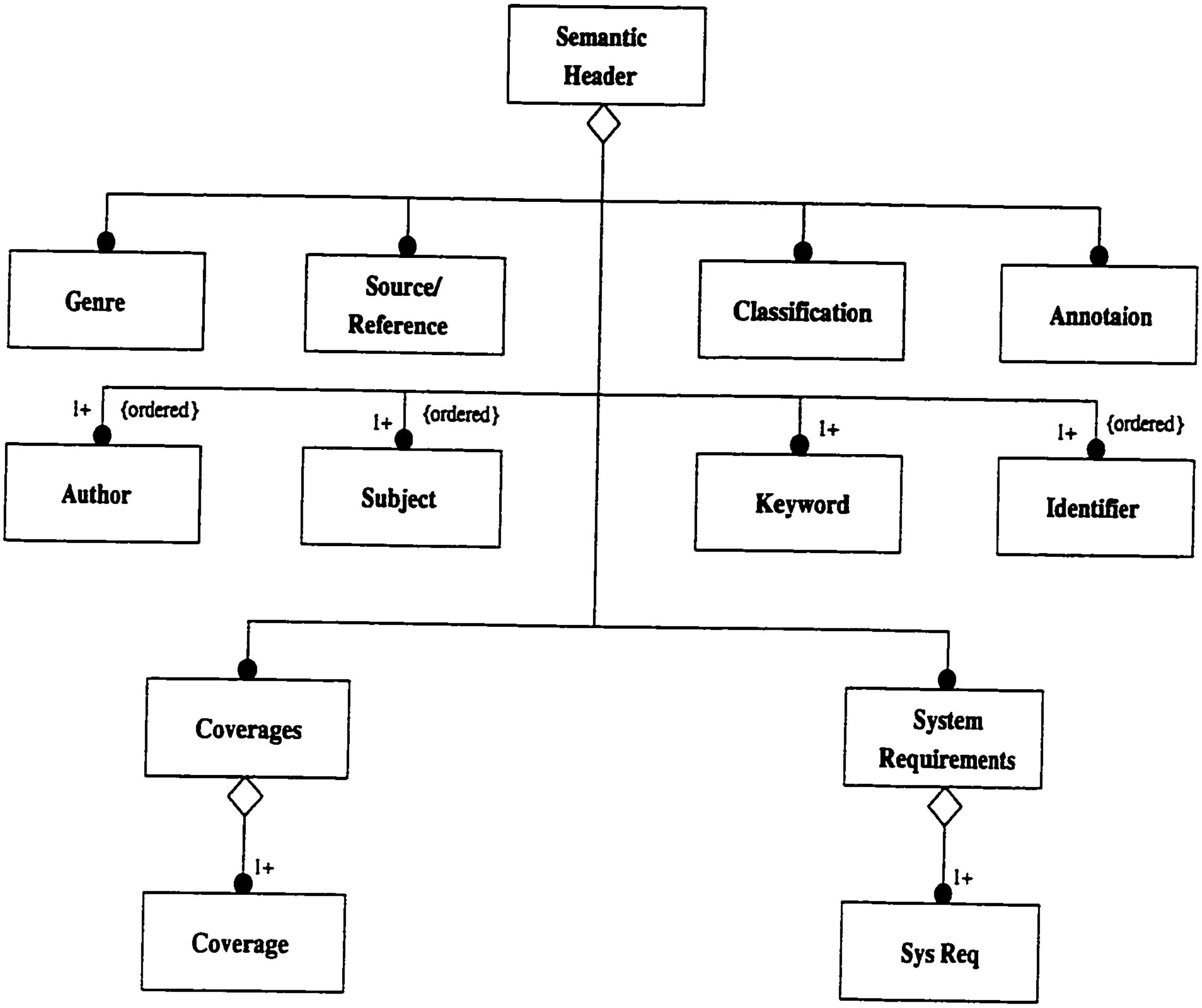


Figure 20: Object Model of Semantic Header

site)[1]. The object model of the system is depicted in Figure 19[2]. The object models of the Semantic Header object class and the Word object class (explained at the end of this section) are given in Figures 20 and 21, respectively. In this schema, the problem is viewed as if all data is centralized in a single database. The description of the object model, the subsystems, the object classes and their associations, is given below.

The *DB Server Interface* subsystem contains the object class Parser and has two main responsibilities.

[1] The *User Interface* subsystem can be divided into other subsystems as well. However, since this document concerns with the server site only, we will not consider issues related to the client site.

[2] The object model notation is shown in Appendix B.

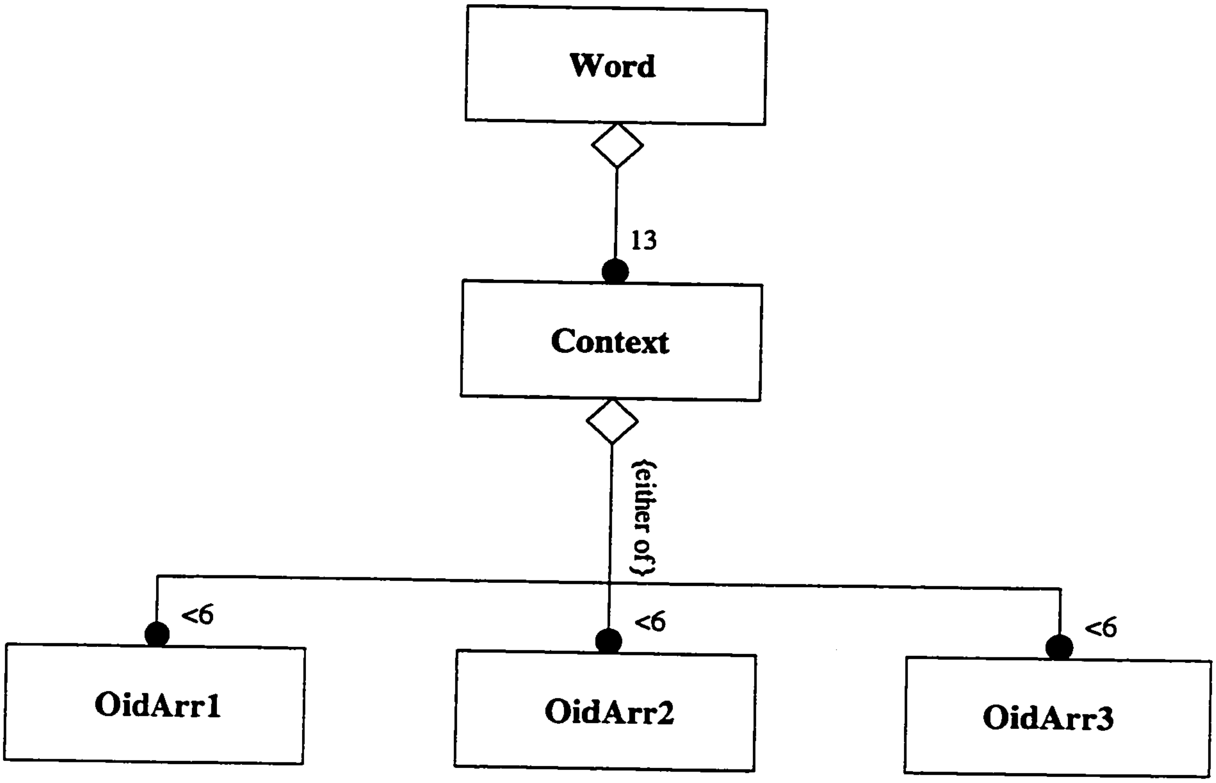


Figure 21: Object Model of Word

1. It responds to the data transmitted via network. Namely, it receives the user request in a file, and returns the result of the request to the client site, again in a file. It should be noted that this part of the subsystem is not included in the analysis of the system. This is because it has no effect on issues concerning the design of the system.

2. It analyzes the user request by scanning through the input file, extracting and storing data items, and finally specifying the flow of control through other subsystems, in order to process the request.

The *Thesaurus* subsystem is responsible for transactions related to indexing and searching standard subject hierarchies. This subsystem contains a database area, called ThesaurusDB, which stores the standard subjects and their synonyms. ThesaurusDB contains a three-level subject hierarchy and the Synonym object. The subject hierarchy is composed of three classes Level_0, Level_1, and Level_2. The hierarchy is refined from top to bottom using the inheritance relationship. There is an aggregation relationship between ThesaurusDB and each one of hierarchy objects.

Each synonym consists of its contolled subject terms. Thus, a Synonym object could contain either of the objects of a hierarchy.

The *Semantic Header* subsystem is responsible for registering, deleting, updating, and annotating Semantic Headers. This subsystem contains a database area (SHDB) which stores the Semantic Headers and other related objects (to be explained). The Semantic Header Database is an aggregation of three objects, Semantic Header, UserID, and Word. Semantic Header contains all fields in a Semantic Header document. Some of these fields are included in the Semantic Header object as attributes (or data members); the others are objects which are components of the Semantic Header object (Figure 20). In this figure, the notation {*ordered*} indicates that the elements of the 'many' end of the aggregation have an explicit order that must be preserved. As an example, the Author object is a part of the Semantic Header object which should be (partially) ordered. This is because, in the GUI, the first author field entered by the user will take part in the construction of the Semantic Header Name. The order of other author entries is immaterial. The objects in this figure correspond to the fields of the Semantic Header metadata described in previous chapters; except for Coverage and SysReq objects, each of which is a part of Coverages and System Requirements, respectively. This is due to the fact that, say, the coverage field of the Semantic Header can be entered repeatedly and that the value of coverage can have more than one value as well (please see section 6.5.2). Semantic Header also includes information in the Identifier object to access an Information Resource on-line. We have used dashed line to illustrate the Information Resource object because it is outside the scope of this system. The UserID object contains a pair of user ID and password entered by the user in the GUI. Each Semantic Header and user ID uniquely identifies a UserID object.

The *File System* is part of the Operating System responsible for file management. It is used for storing local copies of the files received from the client site as well as those which contain the result of the user request to be sent to the client site.

The SHDB object also contains the Word object. It stores non-noise words[3] appearing in those fields of a Semantic Header which may be used during the search operation[4]. The Word object corresponds to the Semantic Header objects where the

[3] Noise words are common invariant words which usually occur repeatedly in a text and do not relate to any specific topic or concept. Some examples of noise words are: am, all, also, about, across, already, everyone, different, etc.

[4] It should be noted that some of the fields of Semantic Headers in register GUI have been

value of the Word object appears in the Semantic Headers. This object contains a fixed number of Context objects equal to the number of search GUI fields. Each Context object, at any given time, has either of the three objects OidArr1, OidArr2, or OidArr3. Context contains a fixed number of these objects (please see section 6.5.3). Each one of OidArr objects contains a fixed number of OIDs or Semantic Header Names (SHN). The Semantic Headers, where the user-entered words in the search GUI occur, are retrieved by means of these SHNs.

4.3 Dynamic Model

The dynamic model describes those aspects of a system concerned with the sequencing of operations over time. In other words, it describes events that mark changes, sequences of events, states that define the context for events, and the organization of events and states. The dynamic model captures control. Control is that aspect of a system that describes the sequences of operations that occur, without regard for what the operations do, what they operate on, or how they are implemented. We separate our dynamic model into scenarios and event traces, and state charts given in the following sections.

We will begin by presenting a typical scenario that gives a general overview of the dynamic behavior of the whole system:
The user in the client (or user interface) site makes a request by filling in some of the fields in a GUI. The user interface calls the client program. The client connects, through the network, to the server and sends the user's request in a file which has a special format. At the server (or database) site, the server receives this file and calls the parser to validate the special syntax of the file (the purpose of having a special format is to identify the content of various fields of the client request, and to transform user-defined queries to database queries). At this point, the parser, based on the information observed in the file, calls appropriate database functions to process the query. The result supplied by the database, is stored in a file (again, with a special format). Finally, the server sends the file through the network to the client

eliminated from those of the search GUI. This is due to the fact that they do not manifest suitable search criteria for effective information retrieval. Some examples are: system requirements, coverage, character set, and address.

site. In the client site, the file is parsed and depending on the information stored in the file, the user interface displays the result of the query or an error message which may have occurred either in the database or in the network.

4.3.1 Scenarios and Event Traces

A scenario is a sequence of events that occurs during one particular execution of a system. An event is something that happens at a point in time. Each event transmits information from one object to another. The sequece of events and the objects exchanging events can both be shown in a pictorial form called event trace diagram. Herein, each object is shown as a vertical line and each event as a horizontal arrow from the sender object to the receiver object. Time increases from top to bottom, however the spacing is irrelevant.

In the following scenarios, the distributed and replicated nature of the Semantic Header and Thesaurus DataBases are hidden.

1) Registering a new Semantic Header (Figure 22):

- The user enters a set of elements of the Semantic Header, such as title, author, and subjects.
- The user interface asks the **Expert System** for standard terms to help the user choose subject hierarchies. This can be done by pressing on subject hierarchy buttons or by entering a synonym or a substring to search for the corresponding subjects.
- The **Parser** receives the request from the **User Interface.**
- The **Parser** parses the file received from the **User Interface** (not shown in the Figure).
- The **Parser** sends the request to **ThesaurusDB.**
- The **ThesaurusDB** searches for matching **Thesaurus** objects and sends the result to the **Parser** (not included in the Figure).
- The **Parser** receives the result from the **Thesaurus.**
- The **Parser** returns the result to the **User interface.**

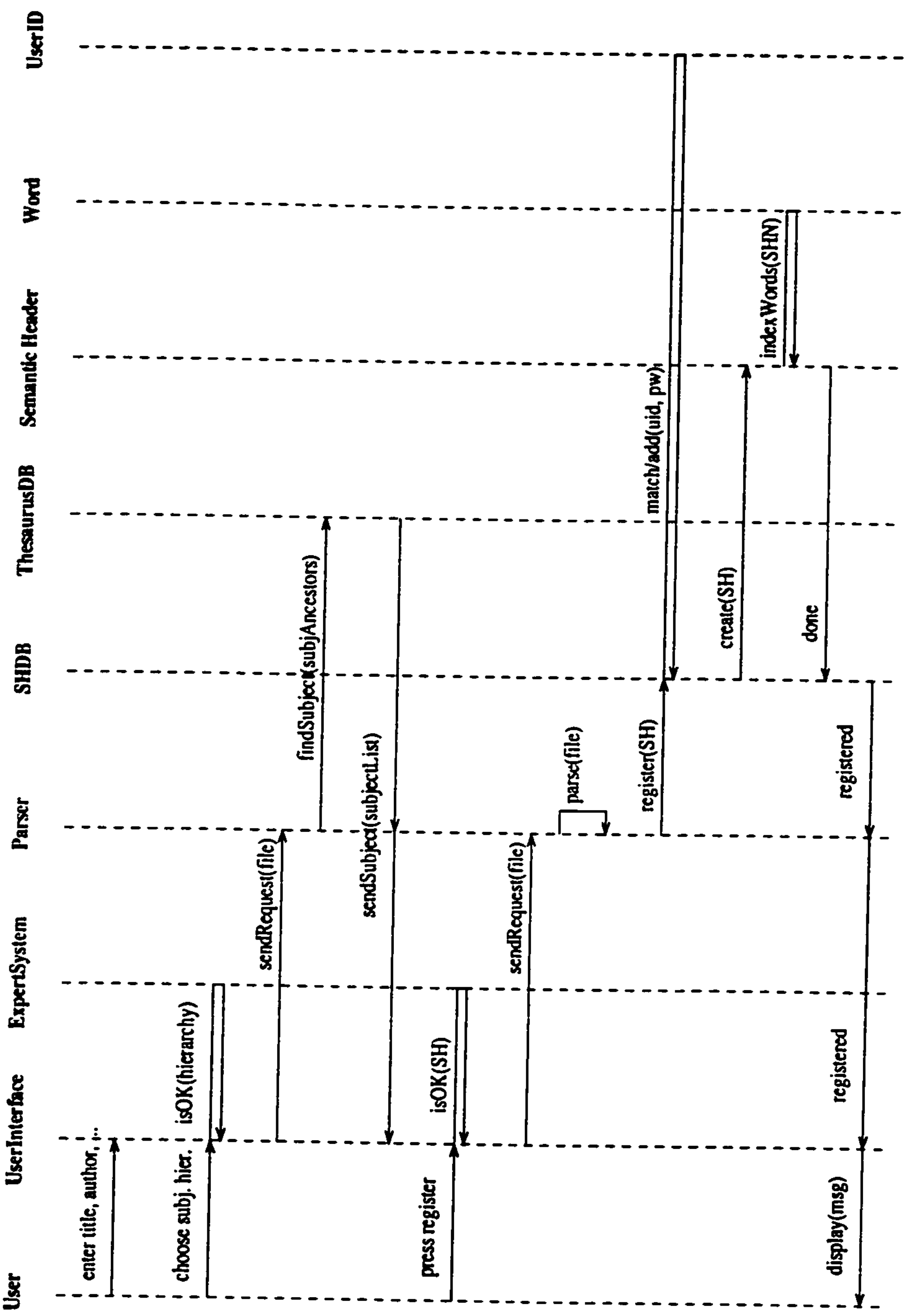


Figure 22: Event Trace for 'Register Semantic Header'

- The **User Interface** displays the standard items and asks the user to make his/her choice.
- The user presses the register button and the **Expert System** makes sure that mandatory fields have been entered by the user.
- The **User Interface** sends the register request to the **Parser.**
- The **Parser** parses the file received from the **User Interface.**
- The **Parser** sends the request to the **Semantic Header Database.**
- The **SHDB** validates the user ID and the password. Namely, it will create a new **User_ID** object if such user ID does not exist; if it does, it will make sure the user ID and the password match.
- The **Semantic Header Database** creates a new **Semantic Header.**
- The **Semantic Header** adds the non-noise words of the Semantic Header to **Word** so that each word points to the Semantic Header. If the word already exists in the database, only a pointer to the Semantic Header will be added to the **Word** object. Subsequently, it sends an acknowledgement to the **SHDB.**
- The **Parser** sends an acknowledgement to the **User Interface.**
- The **User Interface** displays appropriate message to the user.

2) Deleteing a **Semantic Header (Figure 23):**

- The user opens a file containing a Semantic Header (not shown in the Figure).
- The user enters the user ID and the password and presses the delete button.
- The **Expert System** makes sure the annotation field is empty, otherwise the Semantic Header cannot be deleted.
- The **User Interface** sends the request to the **Parser.**
- The **Parser** sends the delete request to the **SHDB.**
- The **SHDB** validates the user ID and the password. Namely, it checks if such a user ID exists. If it does, it requests the **UserID** object to verify that the user ID and the password match. If the user ID does not exist, it terminates the transaction and returns an error code to the **Parser** (this case is not considered here).

- The **SHDB** makes sure that no annotation is added after the registration of the Semantic Header.
- The **SHDB** makes a delete request from the **Semantic Header.**
- The **SH** deletes the Semantic Header.
- The **SH** deletes the Semantic Header Name index from all words in the Semantic Header and sends an acknowldgement to the **SHDB.**
- The **Parser** sends an acknowledgement to **User Interface.**
- The **User Interface** displays appropriate message to the user.

3) **Updating a Semantic Header (Figure 24):**[5]

- The user opens the file containing a Semantic Header (not shown in the Figure).
- The user makes appropriate modifications with the help of the **Expert System.** A number of items cannot be changed and the **Expert System** will verify this and inform the user.
- The user presses the update button.
- The **User Interface** sends the request to the **Parser.**
- The **Parser** parses the file received from the **User Interface.**
- The **Parser** sends the update request to the **Semantic Header Database.**
- The **SHDB** validates the user ID and the password. Namely, it checks if such a user ID exists. If it does, it requests the **UserID** object to verify that the user ID and the password match. If the user ID does not exist, it terminates the transaction and returns an error code to the **Parser** (this case is not considered here).
- The **SHDB** makes sure that the user has not modified the annotation field.
- The **SHDB** makes an update request from the **Semantic Header.**
- The **Semantic Header** updates the modified fields of the Semantic Header.
- The **Semantic Header** updates the modified words.
- The modified fields are exhausted. The **Semantic Header** sends an acknowledgement to the **SHDB.**

[5] For brevity, we will not mention some steps repeated in the registration scenario.

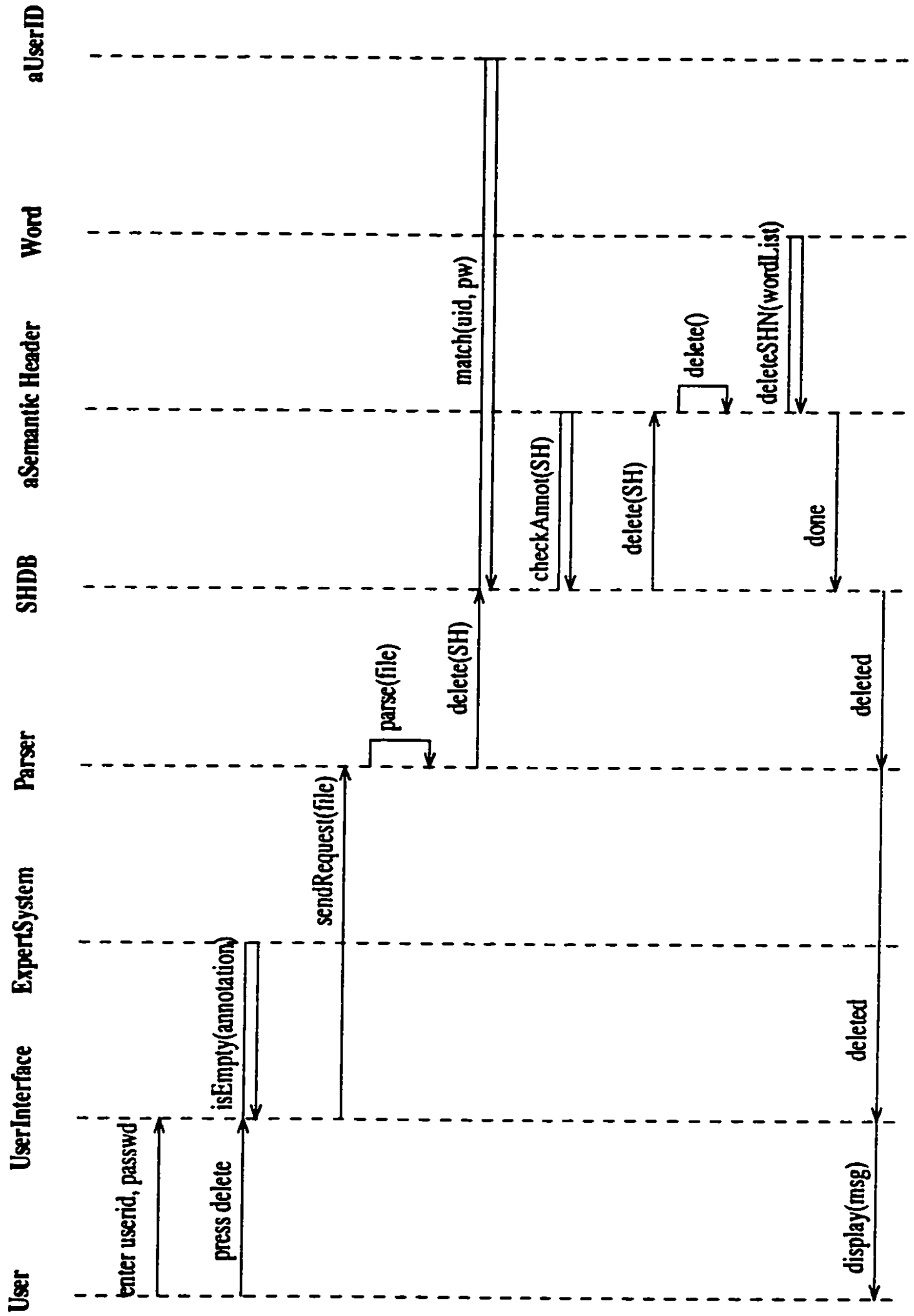


Figure 23: Event Trace for 'Delete Semantic Header'

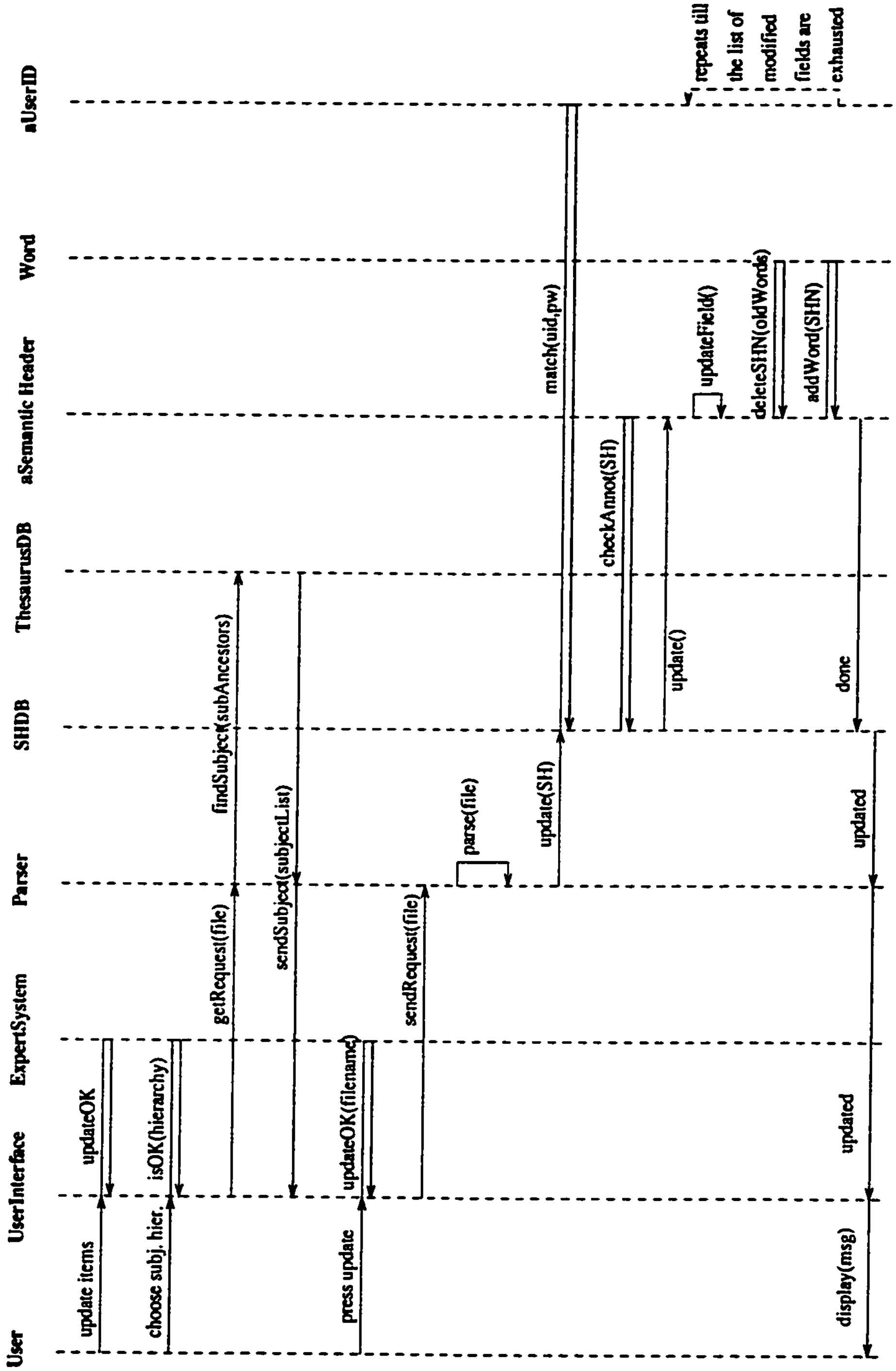


Figure 24: Event Trace for 'Update Semantic Header'

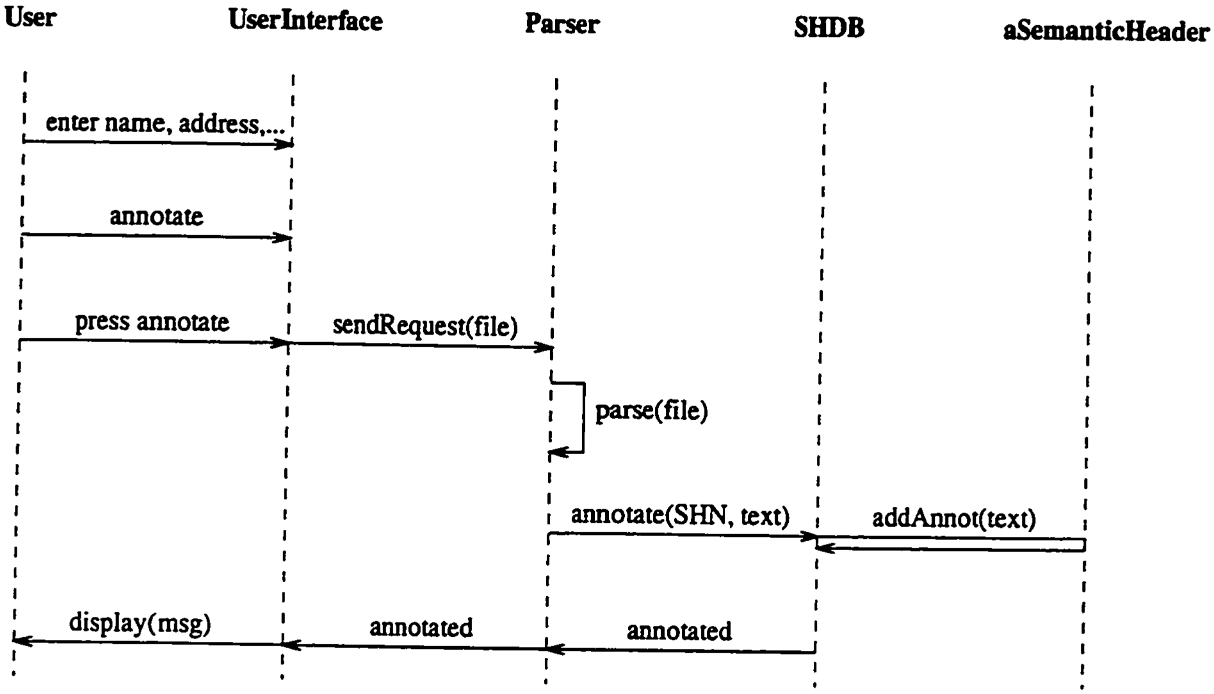


Figure 25: Event Trace for 'Annotate Semantic Header'

- The **Parser** sends an acknowledgement to the **User Interface.**
- The **User Interface** displays appropriate message to the user.

4) **The user annotates a Semantic Header (Figure 25):**

- The user opens the file containing a Semantic Header (not shown in the Figure).
- The user enters personal information such as name, address, and email.
- The user annotates the Semantic Header.
- The user presses the annotate button.
- The **User Interface** sends the request to the **Parser.**
- The **Parser** parses the file received from the **User Interface.**
- The **Parser** sends the request to the **Semantic Header Database.**
- The **SHDB** adds annotation to the list of annotations in the **Semantic Header** object.
- The **Parser** sends an acknowledgement to the **User Interface.**
- The **User Interface** displays appropriate message to the user.

5) **The user makes a search request (Figure 26):**

- The user enters a synonym to find corresponding subject hierarchies.
- The user fills in other search items.
- The user presses the search button.
- The **User Interface** sends the request to the **Parser**.
- The **Parser** parses the file received from the **User Interface**.
- The **Parser** sends the request to the **SHDB**.
- The **SHDB** repeats the following three steps untill the search fields are exhausted.
- The **SHDB** requests a search from the **Word** object.
- The **Word** object finds appropriate indexes characterized by Semantic Header Names and sends them to **SHDB**.
- The **SHDB** performs *and* or *or* operation on two previously found results.
- When the search fields are exhausted, the **SHDB** retrieves Semantic Headers corresponding to SHNs.
- The **SHDB** increments the number of hits of each Semantic Header retrieved.
- The **Parser** sends the result received from **SHDB** to the **User Interface**.
- The **User Interface** displays the result to the user.

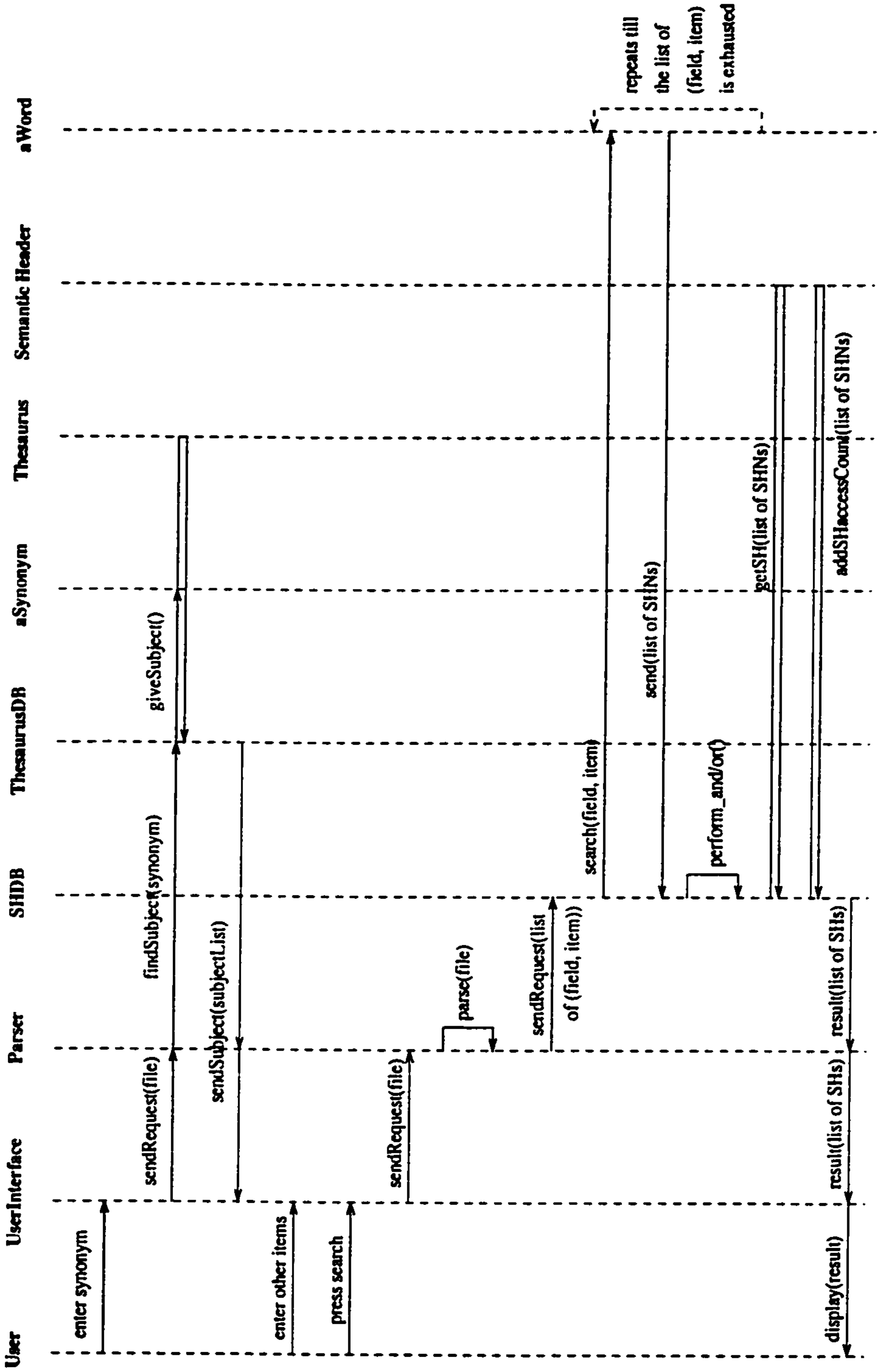


Figure 26: Event Trace for 'Search for Semantic Headers'

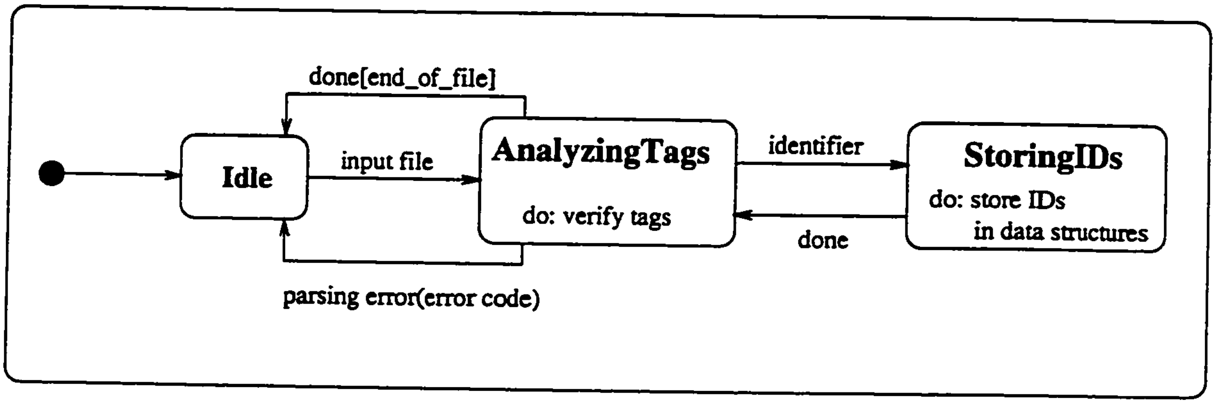


Figure 27: State Chart for Parser

4.3.2 State Diagrams

A state is an abstraction of the attribute values and links of an object. Sets of values are grouped together into a state according to properties that affect the gross behavior of the object. A state chart relates events and states. When an event is received, the next state depends on the current state as well as the event. A state chart is a graph whose nodes are states and whose directed arcs are transitions labeled by event names. In this section we will describe the state diagrams of those objects that exhibit interesting behavior. The dynamic model notation is shown in Appendix B.

The state chart of the Parser is shown in Figure 27. When the Parser receives user request in a file, it analyzes the syntax of the tags in the file. The tags are used to surround the data items (or identifiers) entered by the user as well as to specify the flow of control to other subsystems. Subsequent to analyzing each pair of tags and extracting identifiers they are stored in variables and data structures for later use by the database related subsystems.

The state chart of Synonym is shown in Figure 28. We consider the case when the user has entered a synonym to retrieve the subject hierarchies pertinent to the synonym's controlled terms. In the state of Synonym Matching, Synonym verifies the existence of such Synonym object. If it exists, for each level of the hierarchy, it retrieves the list of controlled terms. Subsequently, for each of these controlled terms, it retrieves the subject sub-hierarchy of the controlled term[6]. If no such synonym

[6] We define a sub-hierarchy of a subject term to be a subset of the hierarchy starting with its parent general (or level_0) subject and ending with the subject term itself. Thus the sub-hierarchy of a general subject would be *(level_0)*; the sub-hierarchy of a level_1 subject would be *(level_0, level_1)*; and the sub- hierarchy of level_2 would be the hierarchy *(level_0, level_1, level_2)*.

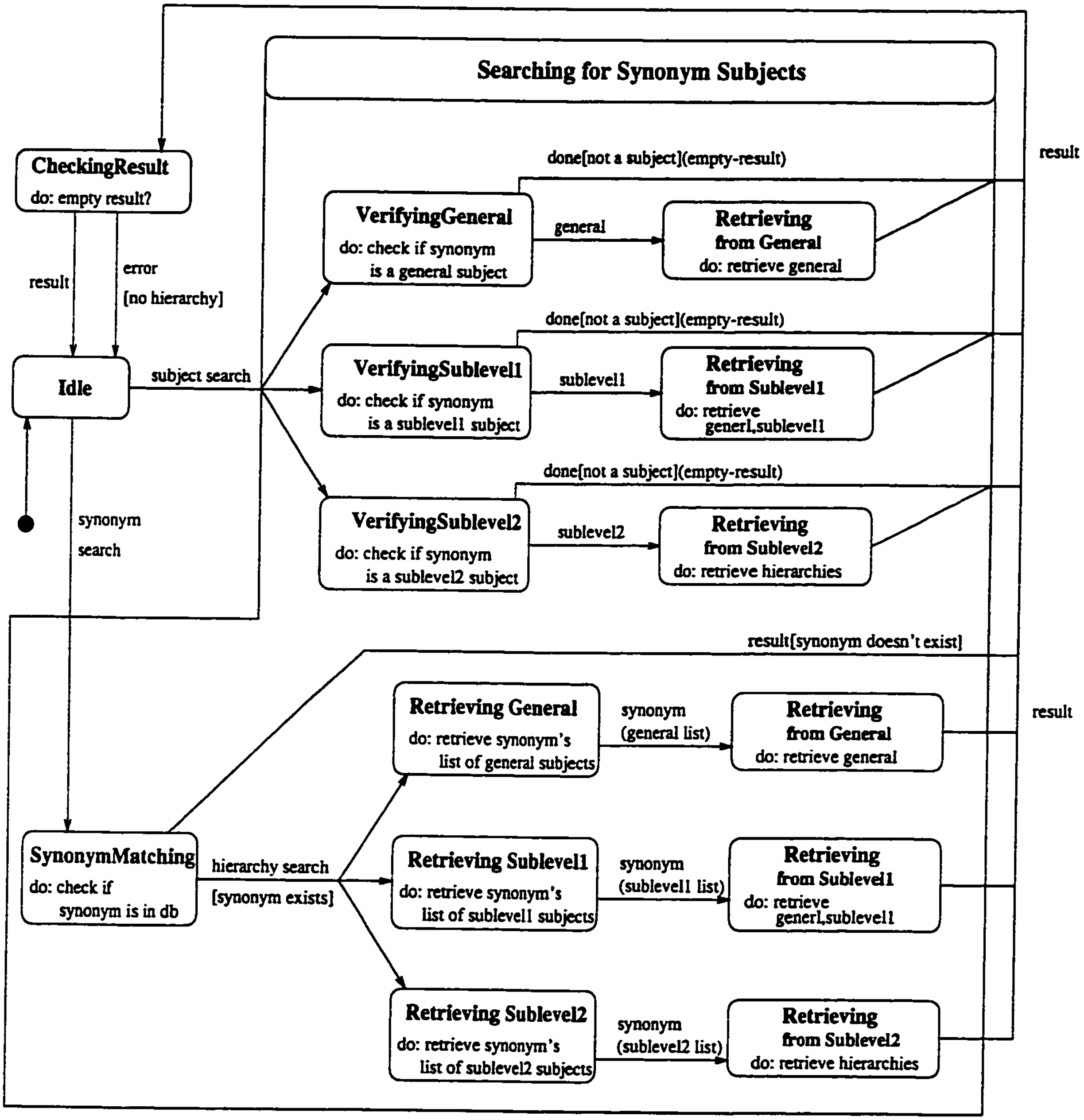


Figure 28: State Chart for Synonym

exists in the database, an empty result will be returned. Besides retrieving a list of controlled terms, Synonym, will also match the synonym value with subject terms in all three levels. Subsequently, for each subject term found, the parent subject term(s), if any, will be retrieved. In case there was no match whatsoever, an error will be returned. It should be noted that the user can make a hierarchy search request based on a substring (not shown). In this case, ThesaurusDB will look for all subject terms in which the substring occurrs. Subsequently, it will retrieve all parent subject term(s), if any, pertinent to those subject terms.

The state chart of the SHDB object class for registering and deleting a Semantic Header is illustrated in Figure 29. When the register event occurs, the SHDB object verifies the existence of the Semantic Header to be registered. If the Semantic Header exists it will terminate the registration by returning an error message. If it doesn't, it will verify the existence of the user ID. If the user ID doesn't exist, it will be added to the database. If it exists, the password for the user ID will be compared with the password entered. If they match (assuming that the Semantic Header does not exist), a transition will be made to the Registering state. The deletion of a Semantic Header takes similar steps as that of registering one. However, prior to deletion, SHDB will verify that no new annotation has been added to the Semantic Header in question after it was registered. If new annotations are made the Semantic Header may not be deleted. The state chart of SHDB for searching for Semantic Headers is illustrated in Figure 30. The SHDB object receives the search criteria. For each criterion, it finds corresponding Semantic Headers and merges the new result with the previous (intermediate) result by means of performing logical *and* or *or* operations. At the end, when the final result is found, the count of each Semantic Header in the result will be incremented by one. This count indicates the number of times a Semantic Header has been accessed. The update of a Semantic Header is analogous to that of a delete operation; however, in the Verifying state, the SHDB will allow the update operation only if the annotation field is not being modified by the user.

Figure 31 illustrates the state chart of the Semantic Header object. In the state of registering a Semantic Header, two main activities will be performed. One is adding the Semantic Header to the database. This is done by creating objects which are part of the Semantic Header (such as Author, Keyword, Identifier, ...). These objects are illustrated in Figure 20. The other activity to be performed is adding non-noise

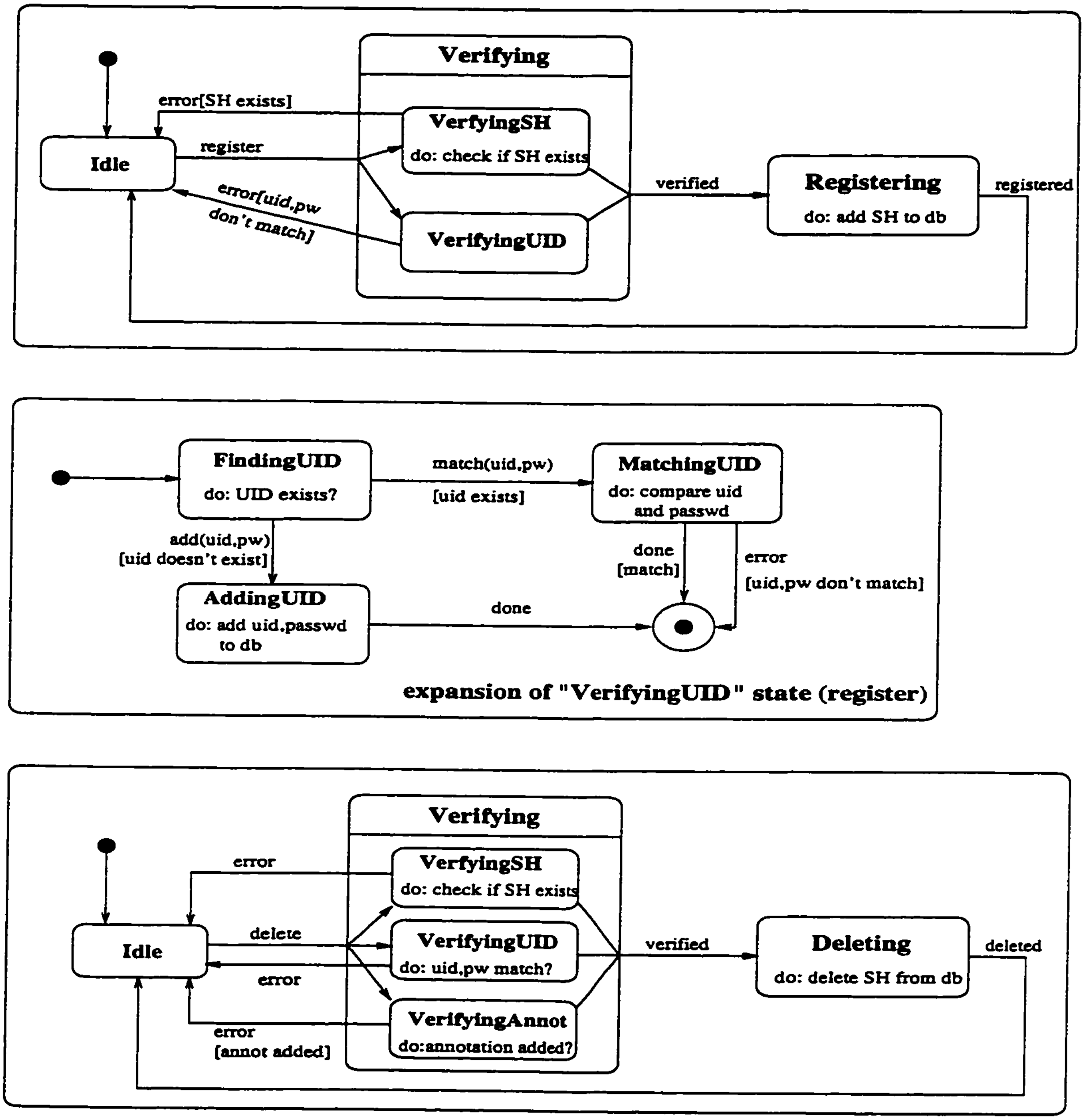


Figure 29: State Chart for SHDB (Register and Delete)

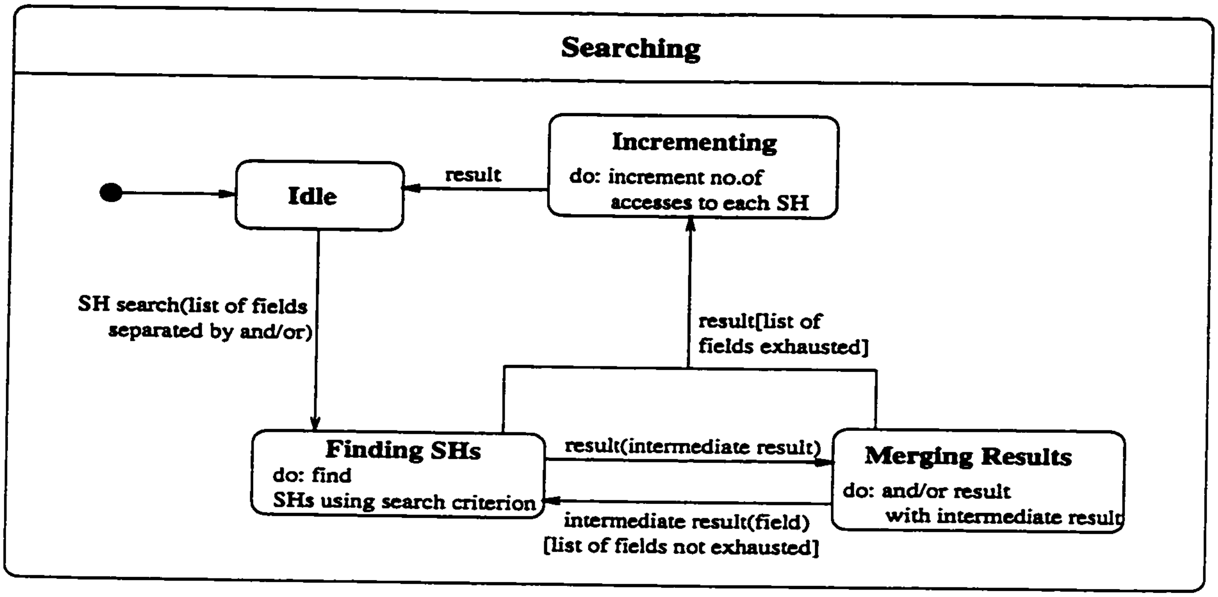

Searching
Incrementing
do: increment no.of accesses to each SH
result
Idle
SH search(list of fields separated by and/or)
result[list of fields exhausted]
Finding SHs
do: find SHs using search criterion
result(intermediate result)
Merging Results
do: and/or result with intermediate result
intermediate result(field)
[list of fields not exhausted]

Figure 30: State Chart for SHDB (Search)

words of the Semantic Header document to the database[7]. To add the non-noise words, first the existence of the Word objects corresponding to the words values is verified. The SHN will be added to the list of SHNs of those objects which exist in the database. Those Word objects which do not exist in the database will be added to the database prior to adding the SHN to their list of SHNs. A similar scenario holds for deleting state. In the state of updating a Semantic Header (not shown), there will be three states. The UpdatingSH state will update those fields of the Semantic Header modified by the user. The other two states are DeletingWords and AddingWords as discussed before (Figure 31). In these states the old words modified by the user will be deleted from the database and the new words will be added to the database.

The state chart of the Word object is depicted in Figure 32. Two states have been taken into consideration, AddingSHN and DeletingSHN. The Word object may be in either of these states when a Semantic Header is being registered, deleted, or updated. For the sake of understandibility[8], suppose each Word object has a list of SHNs through which Semantic Header retrieval becomes possible. Also assume that we have three such lists with small, medium, and large capacities. Each Word object,

[7] Note that these non-noise words belong to those fields of the Semantic Header which are included in the search GUI.

[8] Since we are still in analysis stage, we will avoid elaborating on technical aspects of the Word object while keeping its dynamic behavior intact. For technical details, see chapter 6.

at any given time, can contain only one of such lists. In the state of AddingSHN, an SHN is added to the current list of the object's SHN list. If the list reaches its maximum capacity, its content will be copied to the next larger list. Subsequently, the smaller list will be deleted from the object. In the state of DeletingSHN, an SHN will be deleted from the current list of the object. If the capacity of the current list becomes equal to the capacity of the list smaller than the current one, the content of the current list will be copied to the smaller list and the larger list will be deleted.

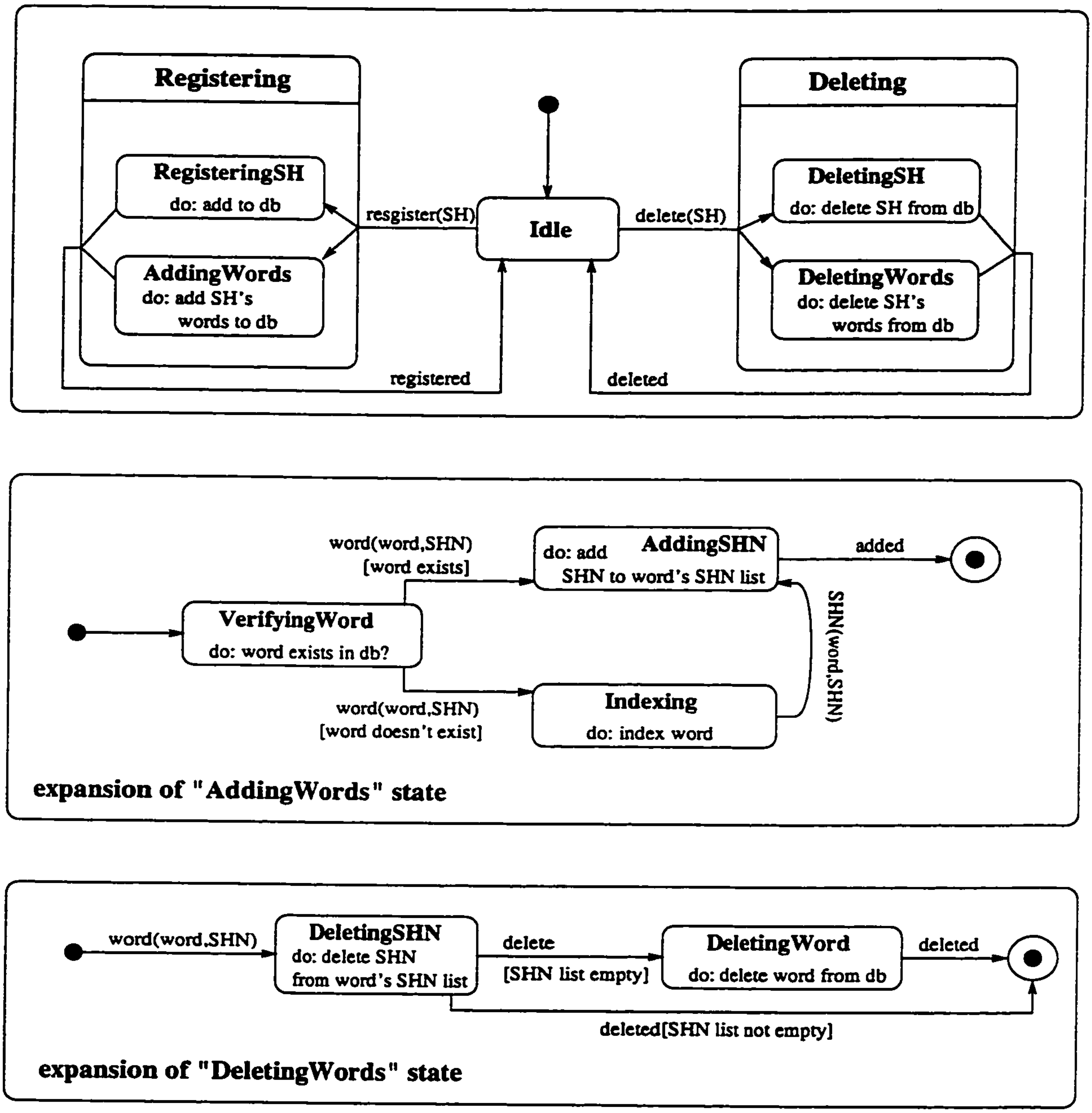


Figure 31: State Chart for SH

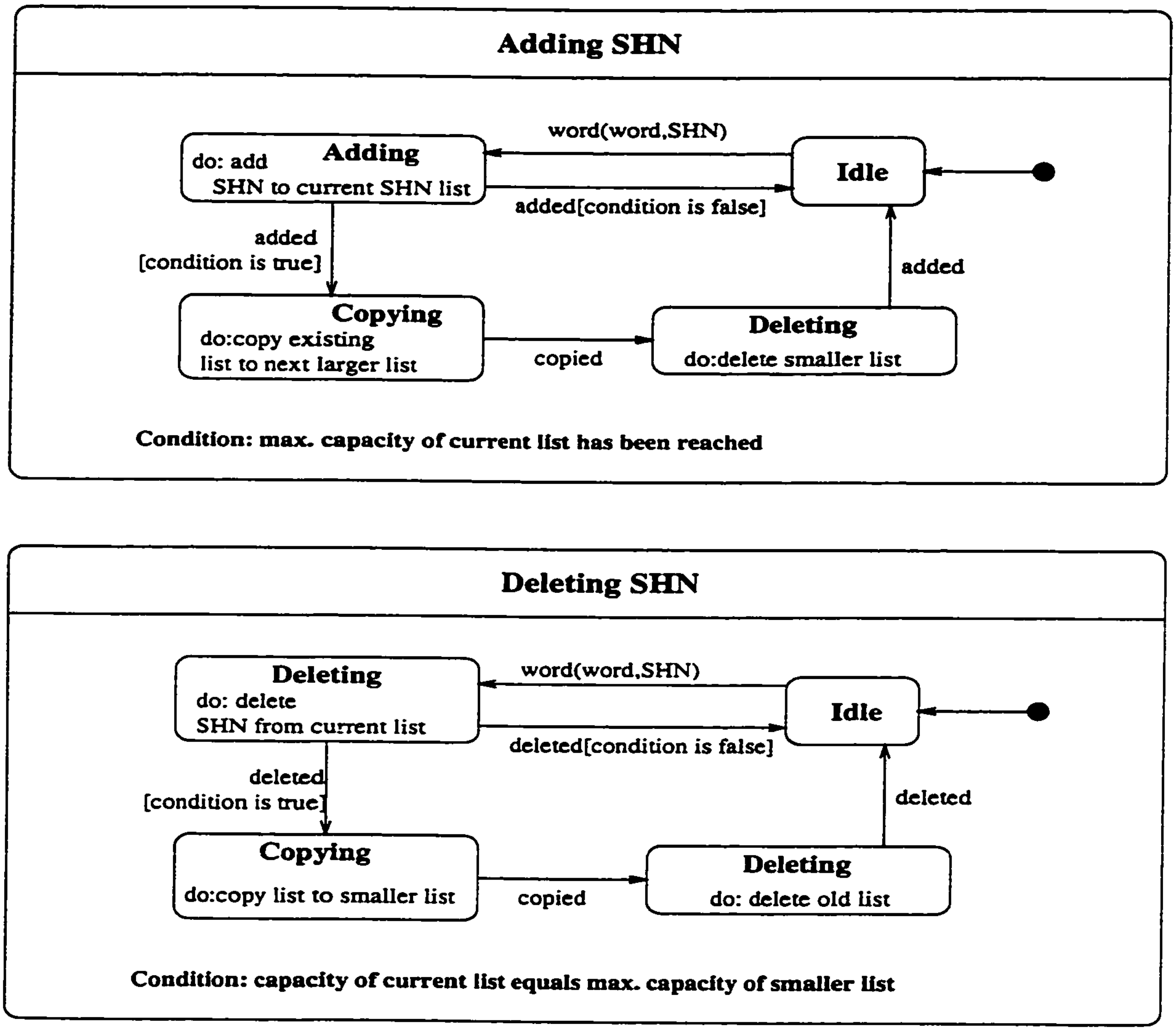

Adding SHN
word(word,SHN)
Adding
do: add SHN to current SHN list
Idle
added[condition is false]
added [condition is true]
added
Copying
do:copy existing list to next larger list
copied
Deleting
do:delete smaller list
Condition: max. capacity of current list has been reached
Deleting SHN
Deleting
do: delete SHN from current list
word(word,SHN)
Idle
deleted[condition is false]
deleted [condition is true]
deleted
Copying
do:copy list to smaller list
copied
Deleting
do: delete old list
Condition: capacity of current list equals max. capacity of smaller list

Figure 32: State Chart for Word

4.4 Functional Model

The functional model describes those aspects of a system concered with transformations of values; namely, functions, mappings, constraints, and functional dependencies. The functional model captures what a system does, without regard to how or when it is done. The funtional model is represented with data flow diagrams. The functional model notation of OMT is shown in Appendix B.

4.4.1 Data Flow Diagrams

A data flow diagram (DFD) is a graph showing the flow of data values from their sources in objects through processes that transform them to their destinations in other objects. A DFD contains processes that transform data, data flows that moves data, actor objects that produce and consume data, and data store objects that store data passively.

The data flow diagram for parsing a file, received from the GUI, is shown in Figure 33. The Scan process reads each token from the file and determines whether the token is a reserved word or an identifier[9]. The Parse process receives tokens from the scanner and sends identifiers to other processes for further refinements. As illustrated in this figure, the Parse process sends identifiers to four different processes. However, depending on the status, the Parse process will send identifiers to some of these processes. In case of register or update, 'Parse' will send 'id' to 'Extract words'. The 'Extract words' process extracts words from the identifier and sends them to 'Extract non-noise words'. This process, using the 'Noise words' data store, extracts non-noise words and sends them to the 'Store in data structures' process to store them in data structures. In case of annotate or delete, 'Parse' will send 'id' to 'Store in data structures' process to store it in data structures. There is no need to extract words from identifiers for annotation and deletion operations. Finally, in case of search, 'Parse' will send 'id' to either 'Extract operations' or 'Extract search words'. The 'Extract operations' process extracts *and* or *or* logical operations. The 'Extract search words' process extracts the search words belonging to Semantic Header metadata fields. In all cases identifiers will be sent to 'Store in data structures' process. This process will store data items in appropriate data structures by means of predefined

[9] For instance, the tag <EOF> is a reserved word which causes the parser to stop reading the file. An identifier may be any word or phrase entered by the user in the GUI.

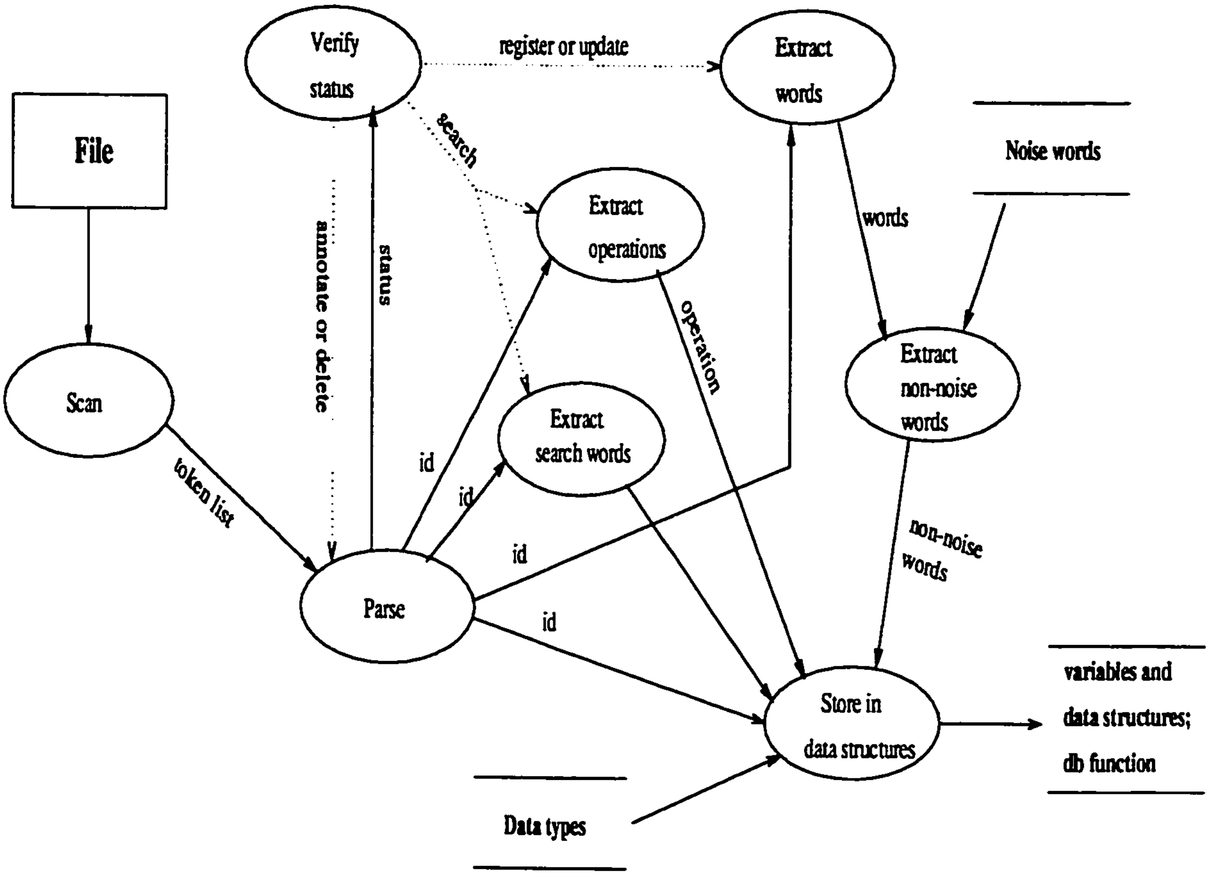


Figure 33: Data Flow Diagram for Parsing an input file

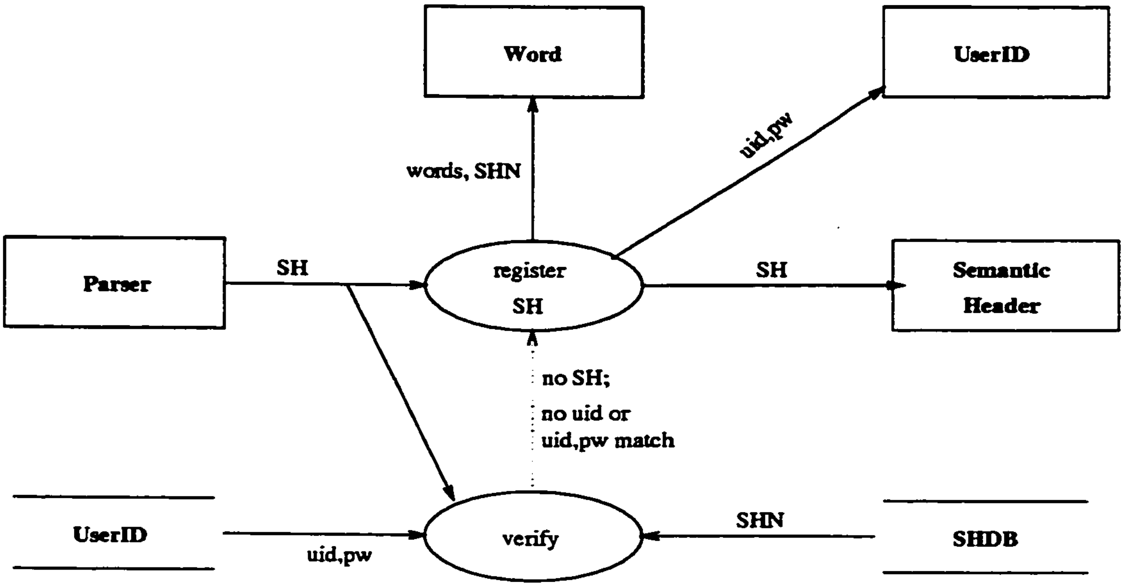


Figure 34: Data Flow Diagram for Registering a SH

data types. The result of parsing will be a collection of variables, data structures, and database function to be called. It should be noted that the database function is determined by the Parse process which is not shown in the diagram to improve readability.

Figure 34, depicts data flow diagram of the register operation of a Semantic Header. For registering a Semantic Header, the RegisterSH process receives the content of a Semantic Header from the Parser object. Subsequently, this process creates a Semantic Header object, a Word object (if needed) and a UserID object (if needed). It also adds the SHN to the Word object. This is performed assuming that the conditions of registration are met. The verify process manages this by retrieving necessary data items from UserID and SHDB data stores. The data flow diagram for deleting a Semantic Header is illustrated in Figure 35 which is similar to that of registering one.

Figure 36 illustrates the data flow diagram of the update operation of a Semantic Header. The UpdateSH process receives the content of the modified Semantic Header from the Parser object. This process sends modified Semantic Header fields to 'delete SH fields' and 'add SH fields' processes. The 'delete SH fields' process deletes old fields from the Semantic Header. The SHN will also be deleted from the words which make these fields. If the deleted SHN is the last SHN in a Word object, the Word object will also be deleted. The 'add SH fields' process adds new fields to the Semantic

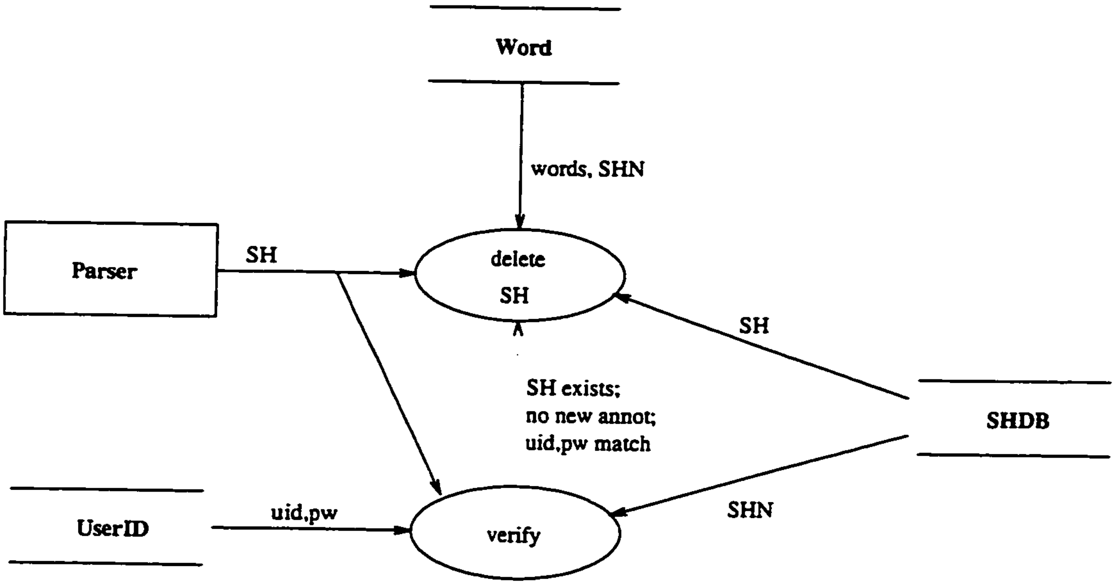


Figure 35: Data Flow Diagram for Deleting a SH

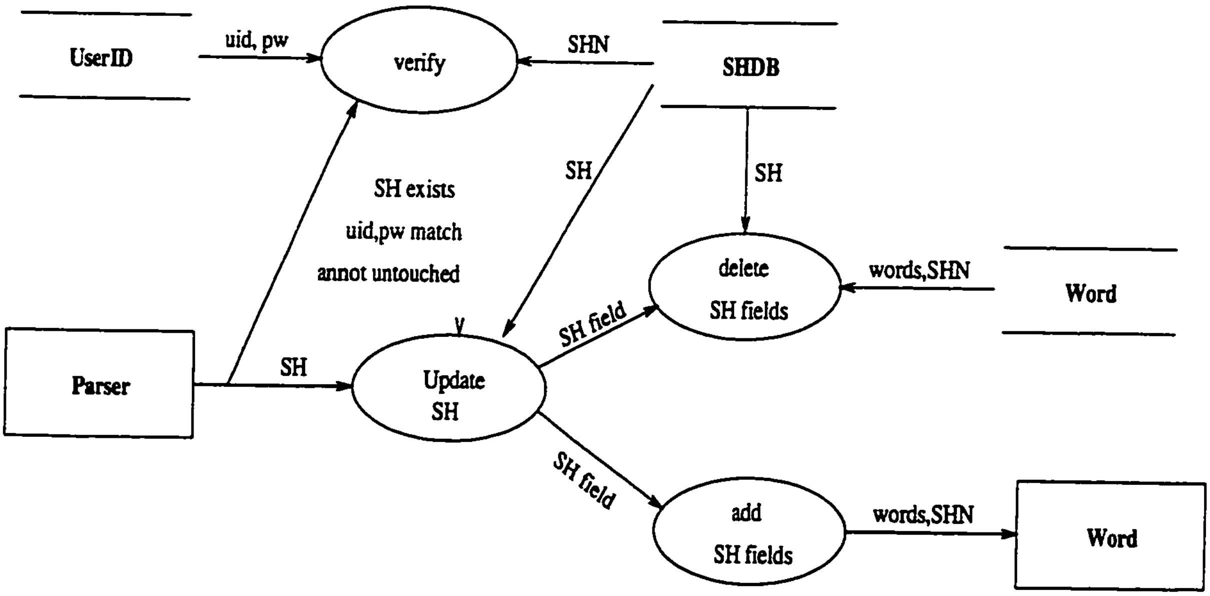


Figure 36: Data Flow Diagram for Updating a SH

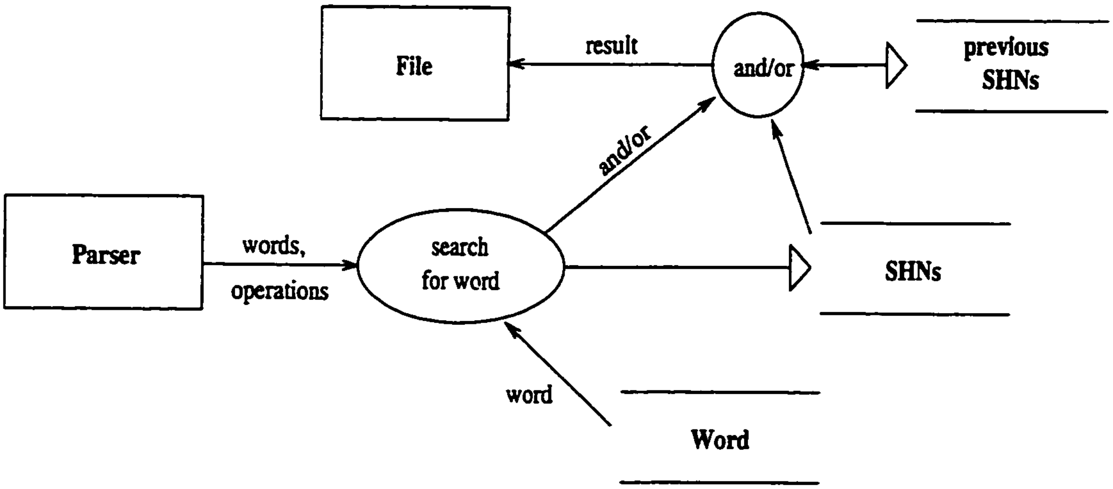


Figure 37: Data Flow Diagram for Searching for SHs

Header. It will also create new Word objects or add SHN to existing ones.

Figure 37 illustrates the data flow diagram of the search operation for Semantic Headers. The process 'search for word' receives words and operations from Parser. By referring to Word data store, it accesses the Semantic Header Names in each Word object. This leads to the creation of SHN data store[10]. These SHNs are *and*ed or *or*ed with previous search result until the list of words and operations are exhausted when the final result is produced in a file.

Data flow diagram for retrieving subject hierarchies using a synonym is shown in Figure 38. The 'find synonym' process receives the synonym and looks for the corresponding Synonym object using the Synonym data store and relays the controlled terms to the 'retrieve sub-hierarchies' process. This process, retrieves the subject sub-hierarchies pertinent to the controlled terms as well as those with one of whose levels the synonym matches. The final result will be produced in a file.

[10] A data flow that generates an object used as the target of another operation is indicated by a hollow triangle at the end of the data flow

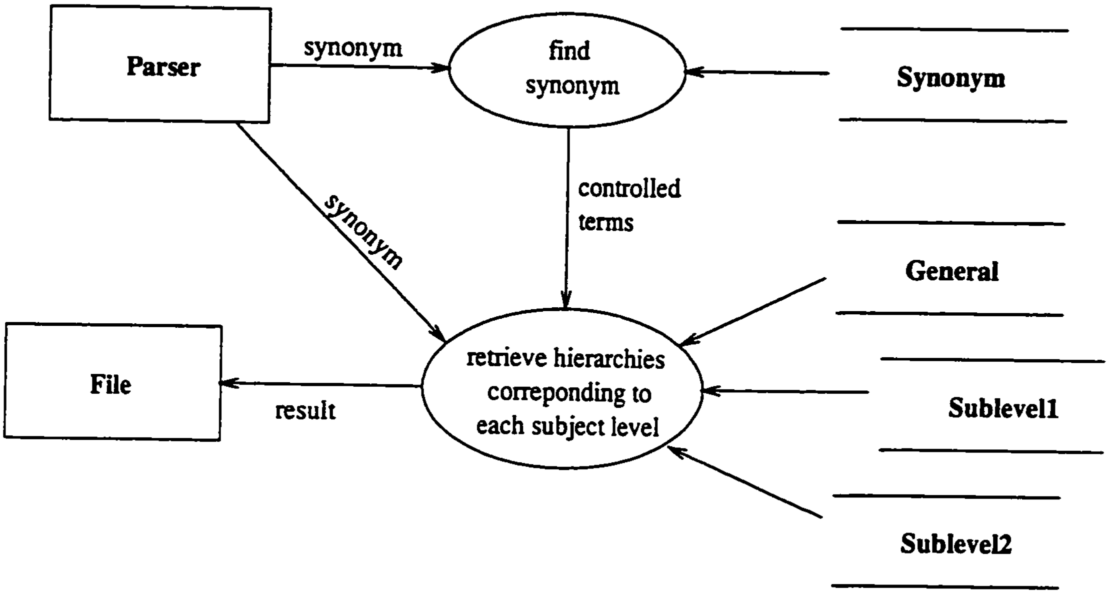


Figure 38: Data Flow Diagram for Retrieving Subject Hierarchies by Synonym

Chapter 5

System Architecture

5.1 Decomposition of the system into subsystems

The system architecture of CINDI is shown in Figure 39. Following the principle of high cohesion and low coupling, the system has been decomposed into five subsystems viz the User Interface Subsystem, the DB Server Interface subsystem, the Semantic Header Subsystem, the Thesaurus Subsystem, and the File System. The architecture is based on *Client-Server* relationship, namely, each subsystem knows about the layers below it, but has no knowledge of the layers above it. Figure 40 illustrates the block diagram of the system. In this diagram the decomposition of the system is organized as a sequence of horizontal layers and vertical partitions. As depicted, the third layer of the diagram is divided into three vertical partitions *Semantic Header*, *File System*, and *Thesaurus*. This indicates that these partitions are weakly coupled subsystems. The diagram also illustrates the client-server relationship between lower layers (providers of services) and upper layers (users of services). All of the subsystems are *event-driven* systems. The responsibilities of each subsystem has been described earlier in section 4.2.1.

The system is a hybrid of an *interactive interface*, a *transaction management system* and a *batch transformer*. The User Interface subsystem is dominated by interactions between users and the graphical user interfaces. The Semantic Header and Thesaurus subsystems are a transaction management system whose main function is to store and access information. The DB Server Interface sequentially transforms input file to output file with no interaction with outside world.

5.2 Concurrency

Semantic Header and Thesaurus subsystems are inherently concurrent, because they can receive events at the same time without interacting with one another.

5.3 Management of data stores

Data will be stored in homogeneous distributed databases. Each physical database has to be managed by the same kind of DBMS, to satisfy the requirement of homogeneity.
The system uses the ODE Database Management System for database transactions. General characteristics of ODE will be described in section 6.3.

There are two database objects in the system (implemented in ODE). One is the Thesaurus Database which contains the three-level subject hierarchies and their synonyms, if any; and the other is the Semantic Header Database which contains the Semantic Headers and non-noise words of the Semantic Headers along with Semantic Header Names (SHN). The words are indexed to be used for searching Semantic Headers (please see chapter 6 for more detail).

5.4 Allocation of subsystems to processors and tasks

The User Interface subsystem has client-server relationship with the Event Handler subsystem. The client-server connection is made by means of the TCP/IP network protocol.

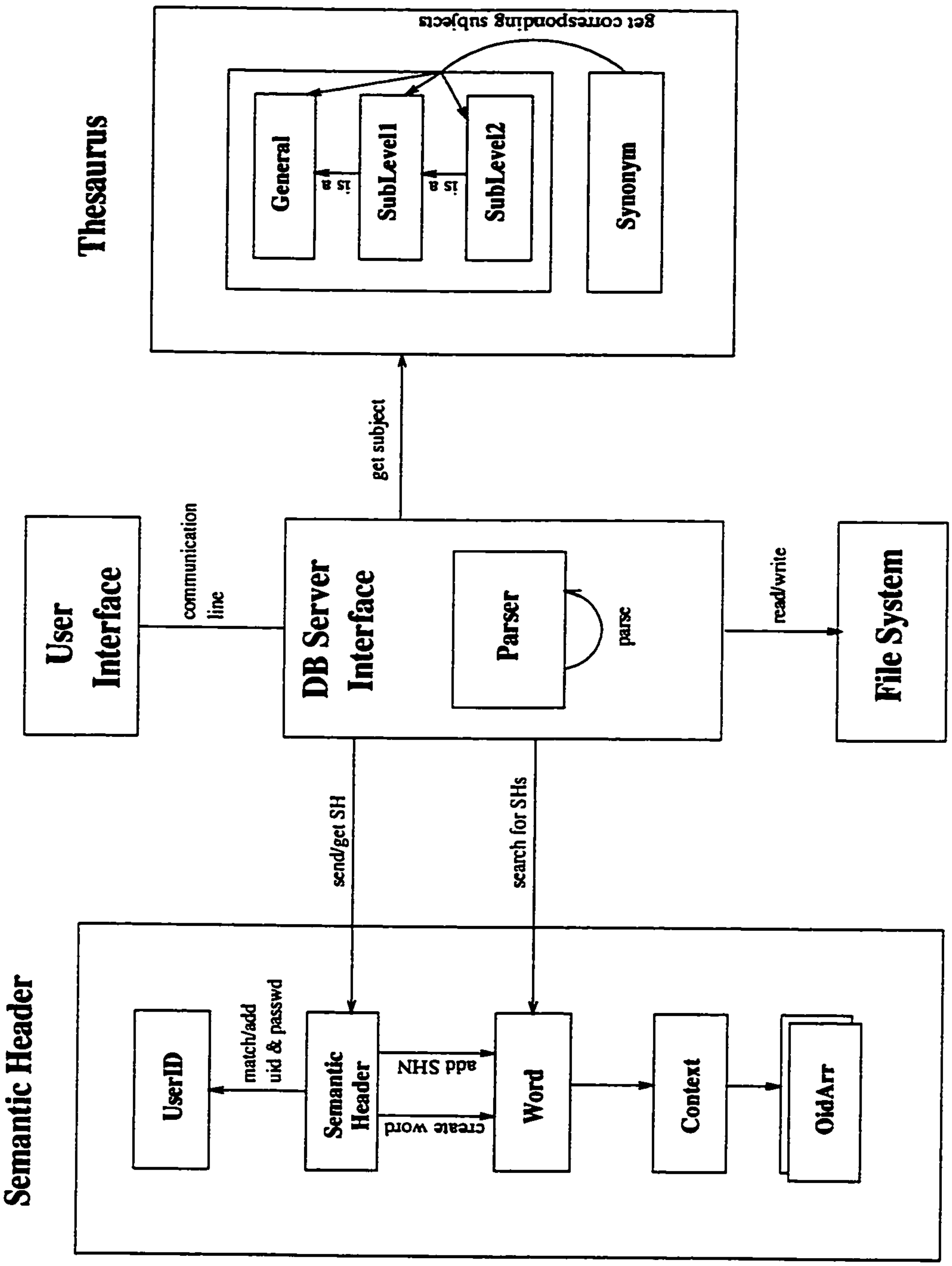


Figure 39: Architecture of the CINDI System

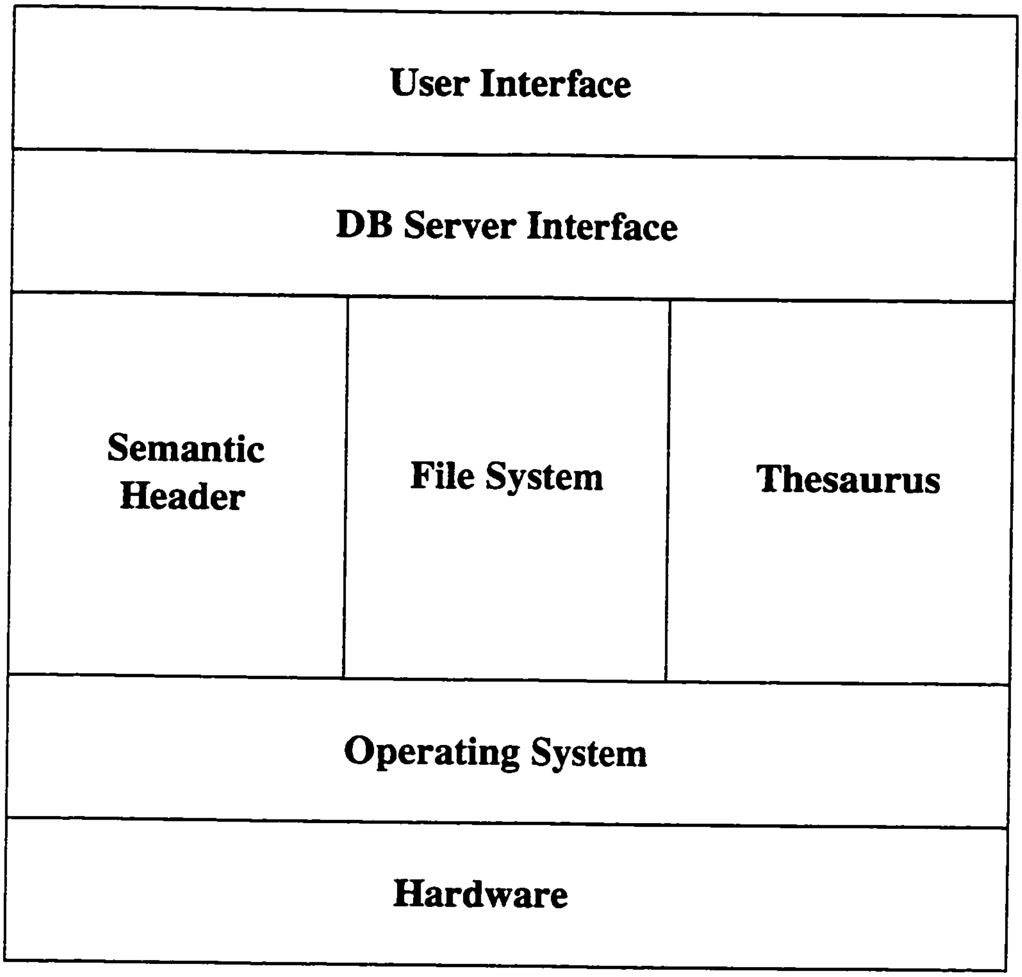


Figure 40: Block Diagram of the System

Chapter 6

Implementation

In this chapter, we will present the full definitions of the classes and associations used in the implementation by combining three models mentioned in the analysis phase. We will also describe the interfaces and algorithms of the methods used to implement the operations. Finally, we will close this chapter by illustrating some results obtained from dry-test and actual runs.

6.1 Client/Server Communication

The communication between the user interface and the database is made using the TCP/IP protocol suite using Berkeley socket interface and it is written in the C language. The server runs continuously at the server site, waiting for a client to connect. When a client, at the user site, is called by the user interface, it connects to the server and sends data in a file, containing the query request, to the server. The server calls appropriate functions for parsing the file and transforming it to database specific query (or queries). This query is sent to the database for processing. Finally, the server receives the result of the query in a file created by the database module and sends it back to the client using the TCP/IP protocol.

Since a server may provide services to more than one client at a time, the server assigns a unique filename to each file received from the client site. Each filename is a concatenation of three fields. The first field is a fixed string used in all files. The second field is the value of time in seconds since 00:00:00 UTC, Jan. 1, 1970. The third one is the process ID of the child process responsible to serve a specified client.

Thus possible filename collisions at the server site are avoided.

For the case where a network problem prevents data transmision, the server program provides a timeout mechanism to avoid the GUI, at the user site, from waiting for the server response indefinitely. Namely, if after a specified period of time the server fails to complete the process of transmitting data from/to the client, the server will send an appropriate message to the client process and disconnect from the client process. Subsequently, the user interface will receive the error code from the client process and display an error message to the user. This way, the GUI will not freeze, and the user can carry on making other requests.

6.2 Parser

To process the data and queries received by the server (from the client), we have provided a parser which follows a special grammar to extract the data items and to call appropriate database functions to manipulate the data and make queries if needed. The parser is written in C language. It uses the *recursive descent parsing technique* to analyze the syntax of the input file[Aho88]. Recursive descent parsing is a top-down method of syntax analysis in which we execute a set of recursive functions to process the input. A function is associated with each nonterminal of a grammar. The grammar of Semantic Header registration written in BNF, is illustrated below. In BNF, brackets ([]) indicate optional parts. An element followed by a '*' may repeat 0 or more times. And an element followed by a '+' should occur at least once. Braces ({}) surround more than one element and they indicate that the elements inside can repeat 0 or more times. We will use the word 'ID' to represent the data entry located between each pair of tags. It should be pointed out that if the field separated by <arole> and </arole> is any string but 'Corporate Entity', then the field separated by <aname> and </aname> is required but that separated by <aorg> and </aorg> is optional. In case of 'Corporate Entity', however, the reverse of the above mentioned rule holds.

```
sem_hdr          ::= '<semhdrR>' content '</semhdr>' '<EOF>'
content          ::= title alt_title list_subjects language character_set
```

```
                        list_resp_agents list_keywords list_ids dates version
                        supersede_version list_classification list_coverage list_sysreq
                        list_genre list_reference cost abstract annotation userid
                        password
title                ::= '<title>' ID '</title>'
alt_title            ::= '<alt_title>' [ID] '</alt_title>'
list_subjects        ::= '<subject>' hierarchy+ '</subject>'
hierarchy            ::= general sublevel1 sublevel2
general              ::= '<general>' ID '</general>'
sublevel1            ::= '<sublevel1>' [ID] '</sublevel1>'
sublevel2            ::= '<sublevel2>' [ID] '</sublevel2>'
language             ::= '<language>' [ID] '</language>'
character_set        ::= '<char-set>' [ID] '</char-set>'
list_resp_agents     ::= '<agent> resp_agent_item+ </agent>
resp_agent_item      ::= role name organization address phone fax email
role                 ::= '<arole> ID '</arole>'
name                 ::= '<aname> ID '</aname>'
organization         ::= '<aorg> ID '</aorg>'
address              ::= '<aaddress>' [ID] '</aaddress>'
phone                ::= '<aphone>' [ID] '</aphone>'
fax                  ::= '<afax>' [ID] '</afax>'
email                ::= '<aemail>' [ID] '</aemail>'
list_keywords        ::= '<keyword>' ID {,ID} '<keyword>'
list_ids             ::= '<identifier> id_item+ '</identifier>'
id_item              ::= domain1 value1
domain1              ::= '<domain1>' ID '</domain1>'
value1               ::= '<value1>' ID '</value1>'
dates                ::= '<dates>' created expiry '</dates>'
created              ::= '<created>' ID '</created>'
expiry               ::= '<expiry>' [ID] '</expriy>'
version              ::= '<version>' [ID] '</version>'
supersede_version    ::= '<spversion>' [ID] '</spversion>'
list_classification  ::= '<classification>' classification_item* '</classification>'
classification_item  ::= domain2 value2
```

```
domain2            ::= '<domain2>' ID '</domain2>'
value2             ::= '<value2>' ID '</value2>'
list_coverage      ::= '<coverage>' coverage_item* '</coverage>'
coverage_item      ::= domain3 value3
domain3            ::= '<domain3>' ID '</domain3>'
value3             ::= '<value3>' ID {,ID} '</value3>'
list_sysreq        ::= '<system-requirements>' sysreq_item* '</system-requirements>'
sysreq_item        ::= component exigency
component          ::= '<component>' ID '<component>'
exigency           ::= '<exigency>' ID {,ID} '</exigency>'
list_genre         ::= '<genre>' genre_item* '</genre>'
genre_item         ::= form size
form               ::= '<form>' ID '</form>'
size               ::= '<size>' ID '</size>'
list_references    ::= '<source-reference>' references_item* '</source-refernece>'
references_item    ::= relationship domain-identifier
relationship       ::= '<relationship>' ID '</relationship>'
domain-identifier  ::= '<domain-identifier>' ID '<domain-identifer>'
cost               ::= '<cost>' [ID] '</cost>'
abstract           ::= '<abstract>' [ID] '</abstract>'
annotation         ::= '<annotation>' [ID] '</annotation>'
userid             ::= '<userid>' ID '</userid>'
password           ::= '<password>' ID '</password>'
```

As the grammar shows, each data item of the input file is surrounded by a pair of tags. A pair of tags is also provided in the input file that specifies the type of action to be taken by the database (in this case <semhdrR> and </semhdr> which indicates the operation of registering a SH). While extracting the data items from a file, depending on the type of tags encountered, the parser will store them in variables and data structures for later use by the database module[1]. For instance, the following linked list stores the three subject terms of a subject hierarchy in a node.

[1] In case of a parsing error, the parser returns an error code. This code is included in a file and sent to the client site by the server process. At the client site, an error message will be displayed on the GUI.

```
struct listptrl {
        char val0[MAX1];
        char val1[MAX1];
        char val2[MAX1];
        struct listptrl *next;
};
typedef struct listptrl LIST1;
```

When the parser sees the tag <general> it extracts the string between this tag and </general> and stores it in the *val0 array of char type* of a newly allocated node of the list, for later use by the database. In the meantime, it will extract the non-noise words of this string and store them in another data structure for later manipulation by the database. In order to find the noise words fast, they have been inserted in a binary search tree. The parser compares each word against the noise words and it ignores it if a match occurs. Thus common invariant words can be kept out of the index. Over two hundred noise words have been provided [Cans].

The parser also plays an important role when a search request is to be processed, as the CINDI System offers a wide range of search criteria to allow effective resource retrieval. Many search fields are included to better suit the users' needs. In most of these fields, the search can be specified to be performed on exact match or substrings of the word entered by the user. Most of these fields allow the logical operations *and* and *or* to refine the search result. Parantheses are also provided to allow nested logical statements. To transform the user-defined queries, received from the client, into database queries, we employ the reverse Polish (or postfix) notation. The parser scans through the input file, checks for syntax errors, and converts it to an infix expression kept in a stack. Subsequently, prior to a database function call, the infix expression is converted to postfix expression which is again kept in a stack. The main data structures used during this process are mentioned below.

```
typedef struct element {
        char status; //takes 'e', or 's' (or 'e', or 'b') indicating
                //exact/substr (or before/after in case of date)
        char string[MAX1]; //word string
        char oper; //takes either of ’&’, ’|’, ’(’, or ’)’
        int field; //title==0, subject==1, ...
} ELEM;
```

The structure *ELEM* stores information about search field, logical operation, or parentheses. It contains four fields: *status* indicates whether the field will be searched for exact matches or substrings; or in case of a search based on ’date’, it will indicate the period before or after the specified date. *string* stores the content of a search field. *oper* indicates the type of logical operation to be performed on search fields. This field may also take opening or closing parentheses. *field* indicates a search field (such as title, subject, abstract, ...). Note that, at any given time, *ELEM* either contains a search field along with its field and status, or a logical operation, or a parenthesis. In each case the other unnecessary fields are not used.

Infix and postfix expressions are of the type DLIST, mentioned below, which is a doubly linked list each node of which contains an *ELEM* structure.

```
typedef struct dlist {
        ELEM* elem;
        struct dlist* left;
        struct dlist* right;
} DLIST;
```

6.3 The ODE Database System

6.3.1 ODE: The Object Database and Environment

Ode<EOS>, also known as Ode, is a database system and environment based on the object paradigm [AGG]. It is built on top of the EOS storage manager. The database is defined, queried and manipulated using the database programming language O++, an extension of C++. A few facilities have been added to C++ to make it suitable for database applications. O++ provides facilities for creating persistent objects which are stored in the database and for querying the database. It also has support for both large objects and versioned objects. It also allows triggers to be associated with objects.

The Ode database is based on a client-server architecture. Each application runs as a client of the Ode database. Multiple Ode applications, running as clients of the database server, can concurrently access the database.

6.3.2 The EOS Storage Manager

EOS is a generic object manager providing key kernel facilities for the fast development of high-performance database management systems and persistent languages. EOS is based on the client-server architecture with support for concurrency control and recovery [BAP93].

The EOS server is a multi-threaded daemon process that mediates all the accesses made to the database, i.e. the storage, update, and reading of database pages. Figure 41 sketches the architecture used (adapted from [BAP93]. When the server starts up it launches the disk daemon **eos_diskd** which is responsible for all the existing storage areas and for providing asynchronous I/O. In the current release the **eos_diskd** is also responsible for the creation and deletion of storage areas (it plays the role of the *area manager* process). In addition, the server spawns the checkpoint and the global log processes, and allocates a number of UNIX system V shared memory segments and semaphores[Ker84, Ste90].

The shared memory segments are used by the shared buffer pool and the concurrency control module to allow all the processes spawned by the server, both during start up and when a new client is connected, to access the same structures. Semaphores are used to provide mutual exclusion among the spawned processes. The communication among the server's children is done by means of message queues.

When an area is attached for the first time by an application program, the disk daemon creates a separate disk process to handle the I/O requests dealing with this area. The communication between the EOS server and the disk processes is done through a UNIX domain socket and a number of shared memory segments. The current release can virtually support up to 6000 different areas [BAP93].

6.3.3 The O++ Database Programming Language

The O++ object model is based on the C++ object model as defined by the *class* facility. Classes support data encapsulation and multiple inheritance. O++ extends

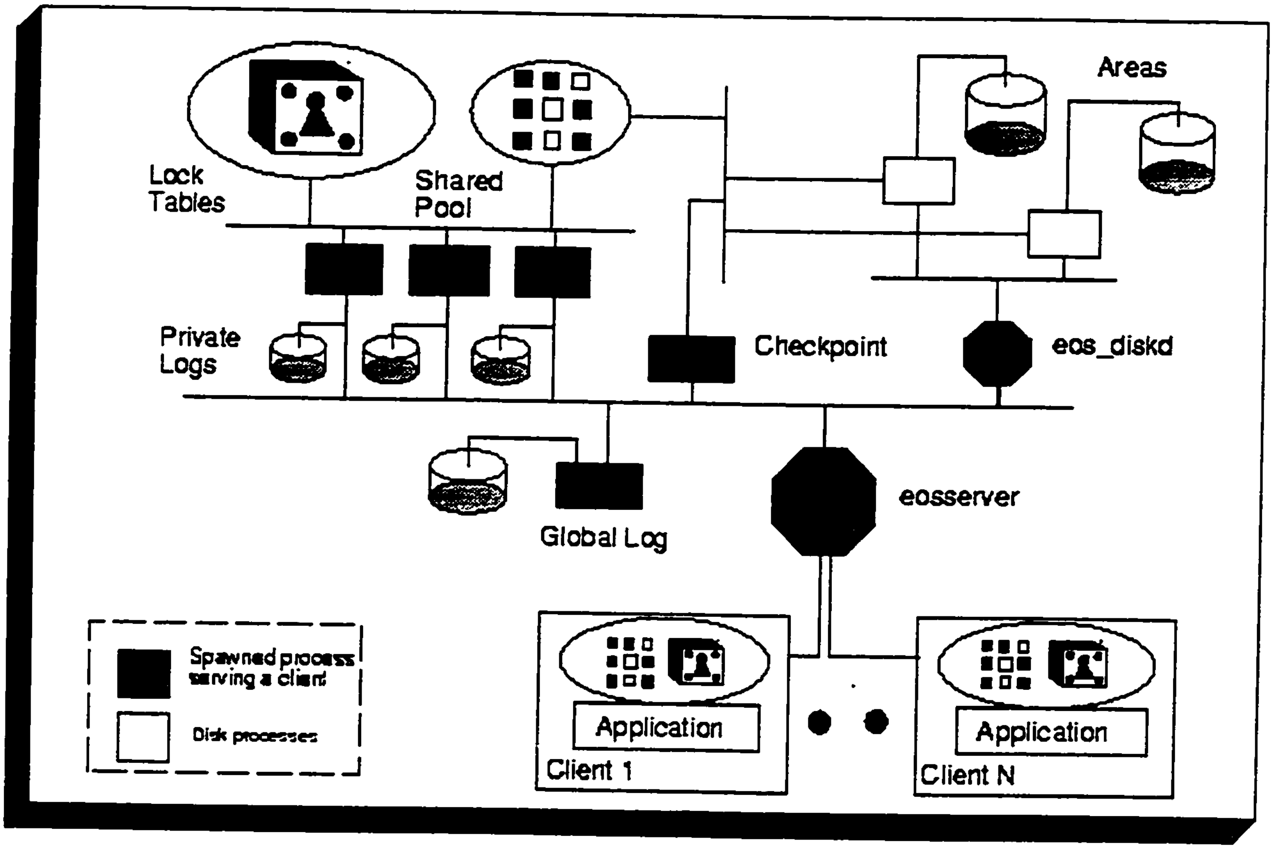


Figure 41: The client-server architecture of EOS

C++ classes by providing facilities for creating and manipulating persistent objects and their versions, and associating constraints and triggers with objects [Agr89].
In this section, we will give a brief overview of the O++ programming language. The O++ user manual provides a brief summary of O++ syntax [AGG].

Databases:
The built-in class *database* provides functions for manipulating (closing, opening, etc.) the database and naming persistent objects.

Transactions:
All code interacting with the database (except database opening and closing) or manipulating persistent objects must be within a transaction block. In general, transactions have the form *trans* { ... }. Transactions can be explicitly aborted by using the *tabort* statement. In such a case, control simply flows to the statement following the transaction block.

Persistent objects:
O++ visualizes memory as consisting of two parts: volatile and persistent. *Volatile* objects are allocated in volatile memory and are the same as those created in ordinary C++ programs. *Persistent* objects are allocated in persistent store and they continue to exist after the program that created them has terminated [Agr93].
Objects of any class type and of the primitive types can be made persistent.
An example of a persistent class is shown below:

```
persistent class employee {
    char name[40];
    char address[40];
    ...
};
```

Object Clusters:
Objects within a database are stored within clusters. By default, all objects of a type are stored in a default cluster associated with the type. Users with special performance need can allocate objects in user-defined clusters.

Queries:

Objects of class types can be accessed using the associative for loop as illustrated below:

```
persistent employee* pe;
for (pe in employee)
    cout << "Name == " << pe->name << endl;
```

The *suchthat* clause can be used to restrict a search to those objects that satisfy a Boolean expression. For instance, the following code accesses only employees older than 25 years:

```
for (pe in employee) suchthat (pe->age > 25)
    cout << "Name == " << pe->name << ", Age == " pe->age << endl;
```

Joins can be performed using nested for loops or a loop with multiple loop variables.

Indexes:

Indexes can be built on arithmetic type data members and statically allocated character array data members (but not dynamically allocated arrays). These data members must be declared as *indexable* in the class definition. Hash indexes are used to speed up queries that retrieve all objects such that some field assumes a particular value. B-tree indexes may also be used in the same way, as well as to retrieve all objects whose member values fall within a specified range.

Named Persistent Objects:

Persistent objects can be named. The names enable fast access to these objects. For such objects, it is not necessary to use the general *for* query statement to access them. The relationship between names and objects is one-to-one, i.e., an object may have at most one name, and a name may correspond to at most one object.

Persistent Arrays:

O++ allows the user to allocate and access persistent arrays, in a manner analogous to C++. A persistent array is allocated (dynamically) by specifying its size. As in C++, a default constructor must exist for the type, and it will be used to initialize each object in the array. Multi-dimensional arrays are also supported.

Large Objects:

An object larger than a ”page” is classified as large. The current page size is 4K bytes. Large objects are, by default, handled transparently. However, applications may find it more efficient to manipulate large objects by explicitly accessing portions of such objects. Class *large* provides functions for efficiently manipulating large objects.

Versions:
Object versioning in O++ is orthogonal to type, that is, versioning is an object property and not a type property. Version of an object can be created without requiring any change in the corresponding object type definition, all objects can be versioned, and different objects of the same type can have a different number of versions.

Triggers:
Triggers are event-action pairs. Events can be ”basic” or ”composite”, the latter being composed from both basic and other composite events. Triggers are specified in class specifications.

6.4 Thesaurus Database

The Thesaurus Database contains four object classes: *Level_0* which represents the general subject of the subject hierarchy, *Level_1* which represents the sub-subject of and is derived from *Level_0*; *Level_2* which represents the sub-subject of and is derived from *Level_1*, and finally *Synonym* which contains those subject terms (at any level) synonymous to the *Synonym*'s value. The classes of the subject hierarchy are defined below.

```
persistent class Level_0 {
protected:
        indexable char lev_0[MAX1];
public:
        Level_0(char * lv_0);
        virtual void print();
        persistent Level_0* isLevel_0(char *str);
        void list_all_level_0();
        int list_substr_0(char* str, int count);
};
persistent class Level_1 : public Level_0 {
protected:
        indexable char lev_1[MAX1];
public:
        Level_1 (char *lv_1, char *lv_0);
        virtual void print();
        virtual void print_all();
        persistent Level_1* isLevel_1(char *str);
        void list_all_level_1(char* lev_0);
        int list_substr_1(char* str, int count);
};
persistent class Level_2 : public Level_1 {
        indexable char lev_2[MAX1];
public:
        Level_2 (char *lv_2, char *lv_1, char *lv_0);
        virtual void print();
        virtual void print_all();
        persistent Level_2* isLevel_2(char *str);
        void list_all_level_2(char* lev_0, char* lev_1);
        int list_substr_2(char* str, int count);
};
```

Since the subject hierarchy classes have similar characteristics, we will describe only the methods of, say, the *Level_2* class. The constructor initializes the *Level_2*'s and its parents' values. The method *print()* prints *Level_2* object into the result file. The method *print_all()* prints *Level_2* object as well as its parent objects *Level_0* and *Level_1*. The method *isLevel_2* returns a *Level_2* object whose value is 'str'. It returns 0 (NULL pointer) if no such object exists in the database. The method *list_all_level_2* prints (into the result file) all *Level_2* objects where their level_0 and level_1 subject values correspond to 'lev_0' and 'lev_1', respectively. The definition of this method which contains a query follows:

```
persistent Level_2* Lev_2;
for (Lev_2 in Level_2) suchthat (strcmp(Lev_2->lev_0, lev_0) == 0
                                   && strcmp(Lev_2->lev_1, lev_1) == 0)
    Lev_2->print();
```

The method *list_substr_2* prints all objects each of whose value starts with the string 'str'; and it returns the number of such objects found. The definition of this method which contains a query follows:

```
persistent Level_2* Lev_2;
for (Lev_2 in Level_2) suchthat (starts_with(Lev_2->lev_2, str) == 1) {
    Lev_2->print_all();
    ++count;
}
```

The class *Synonym* contains three linked lists, each of which contains pointers to one of subject hierarchy classes[2]. This is because a phrase could be a synonym to a number of subjects belonging to different levels of subject hierarchies. The following illustrates the definition of this class:

```
persistent class Synonym {
        indexable char syn[MAX1];
        persistent Level_0_list *Lev_0_list;
        persistent Level_1_list *Lev_1_list;
        persistent Level_2_list *Lev_2_list;
public:
        Synonym (char *s);
        persistent Synonym* isSynonym(char *str);
        void find_ctrl(char* str);
};
```

The constructor of *Synonym* initializes the Synonym value and the linked lists. The

[2] In general, *sets* are good candidates to implement aggregation relationship. However, since Ode does not support *sets* we had to employ linked lists instead.

method *find_ctrl* retrieves all controlled terms of the synonym value 'str'. It also retrieves all subject terms which happen to be same as 'str'. Subsequently, it retrieves all subject sub-hierarchies pertinent to these subject terms and prints them into the result file. As an example the query for retrieving controlled terms of a synonym with value 'str' from level_1 is illustrated below:

```
persistent Synonym* Syn;
if (Syn = Syn->isSynonym(str))
    while (Syn->Lev_1_list->NonEmpty())
        Lev_1_list->Top()->print_all();
```

In this query, the existence of such syonoym is verified. If the correponding object is found, the while loop traverses through the level_1 list and prints all sub-hierarchies of level_1 objects found in the the list.

To retrieve all level_1 subject terms which match the synonym string 'str' the following query is used:

```
persistent Level_1* Lev_1;
if (Lev_1 = Lev_1->isLevel_1(str)) Lev_1->print_all();
```

The query checks if a level_1 object with value 'str' exists. If it does, it is printed into the result file.

It should be pointed out that the expert system rules, used to help users retrieve subject terms, are implicitly included in the Thesaurus subsystem. In fact, the inheritance relationship used in the Thesaurus subsystem automatically applies these rules in subject retrieval.

6.5 Semantic Header Database

6.5.1 User ID

To control unauthorized access to update an existing Semantic Header we define the class UserID below:

```
persistent class UserID {
        indexable char userid[MAX3];
        char passwd[MAX3];
public:
        char* get_userid();
        char* get_passwd();
        UserID(char *uid, char *pw);
        persistent UserID* isUserID(char* uid);
        int uid_pw_match(char* pw);
        virtual void print();
};
```

On registering a Semantic Header, the methods of this class will be called to check if the user ID exists in the database. If so, the user ID and the password must match, otherwise an error code will be relayed to the client. In case the user ID does not exist, a new UserID object will be created in the database; the password will also be allocated by the constructor of this object. On deleting and updating a Semantic Header, two conditions must hold. The user ID should exist, and the user ID and the password should match. Otherwise an error message will be forwarded to the client site. The deletion of a Semantic Header will not affect its corresponding UserID object.

6.5.2 Semantic Header

The class *SemHdr* contains persistent linked lists of classes *Author*, *Subject*, *Keyword*, *Ident*, *Clsf*, *Covrg*, *SysReq*, *Genre*, and *SrcRef*. This is because Semantic Headers allow more than one entry for the fields author, subject, keyword, identifier, classification, coverage, system requirements, genre, and source/reference. For annotations, we also need to include a persistent linked list of class *Annot*, since a Semantic Header may be annotated by any reader of the corresponding information resource.

In some cases the object could be the node of a linked list and this node could

be the head of another linked list. For instance, the field System Requirements is composed of two fields Component and Exigance. To each component more than one Exigance can be assigned. As an example, the component 'Software' may have three (comma separated) exigencies as 'Borland C++, Turbo C++, Standard C++'. Hence the need to include another linked list to manage this kind of situation. This is done by i) introducing a new class *PlistReq* which contains the value of the Exigance field and by ii) including a persistent linked list of *PlistReq* in the class *SysReq*.
The definition of these classes are given below:

```
persistent class Author {
        unsigned long count;
        indexable char role[MAX1];
        indexable char name[MAX1];
        indexable char org[MAX1];
        indexable char addr[MAX1];
        indexable char tel[MAX1];
        indexable char fax[MAX1];
        indexable char email[MAX1];
public:
        void add_count();
        void sub_count();
        int count_is_zero();
        Author(char* r, char* n, char* o, char* a, char* t, char* f, char* e);
        virtual void print();
        persistent Author* isAuthor(char* r, char* n, char* o, char* a,
                                    char* t, char* f, char* e);
};
persistent class Subject {
        unsigned long count;
        indexable char lev_0[MAX1];
        indexable char lev_1[MAX1];
        indexable char lev_2[MAX1];
public:
        void add_count();
        void sub_count();
        int count_is_zero();
        Subject(char* l0, char* l1, char* l2);
        virtual void print();
        persistent Subject* isSubject(char* l0, char* l1, char* l2);
};
persistent class Keyword {
        unsigned long count;
        indexable char kw[MAX1];
public:
        void add_count();
        void sub_count();
        int count_is_zero();
        Keyword(char* kw);
        virtual void print();
        persistent Keyword* isKeyword(char *str);
};
persistent class Ident { //short form of Identifier in Object Model
        unsigned long count;
        indexable char d3[MAX1];
        indexable char v3[MAX1];
public:
        void add_count();
        void sub_count();
        int count_is_zero();
        Ident(char* dom3, char* val3);
        virtual void print();
        persistent Ident* isIdent(char* s0, char* s1);
};
persistent class Clsf { //short form of Classification in Object Model
        unsigned long count;
        indexable char d4[MAX1];
        indexable char v4[MAX1];
```

```
public:
        void add_count();
        void sub_count();
        int count_is_zero();
        Clsf(char* dom4, char* val4);
        virtual void print();
        persistent Clsf* isClsf(char* s0, char* s1);
};
persistent class PlistCov { //correponds to Coverage in Object Model
        unsigned long count;
        indexable char val[MAX1];
public:
        char* get_val();
        void add_count();
        void sub_count();
        int count_is_zero();
        PlistCov(char* value);
        virtual void print();
        persistent PlistCov* isPlistCov(char* s);
};
persistent class Covrg { //short form of Coverages in Object Model
        unsigned long count;
        indexable char d5[MAX1];
        indexable char sh_oid[MAX1];
        persistent PlistCov_list* Val_list;
        friend persistent class SemHdr;
public:
        void add_count();
        void sub_count();
        int count_is_zero();
        Covrg(char* dom5, char* oid, LIST* list);
        virtual void print();
        persistent Covrg* isCovrg(char* s, char* oid, LIST* list);
};
```

```
persistent class PlistReq { //corresponds to SysReq in Object Model
        unsigned long count;
        indexable char val[MAX1];
public:
        char* get_val();
        void add_count();
        void sub_count();
        int count_is_zero();
        PlistReq(char* value);
        virtual void print();
        persistent PlistReq* isPlistReq(char* s);
};
persistent class SysReq { //short form of SystemRequirements in Object Model
        unsigned long count;
        indexable char comp[MAX1];
        indexable char sh_oid[MAX1];
        persistent PlistReq_list* Exg_list;
        friend persistent class SemHdr;
public:
        void add_count();
        void sub_count();
        int count_is_zero();
        SysReq(char* cmp, char* oid, LIST* list);
        virtual void print();
        persistent SysReq* isSysReq(char* s, char* oid, LIST* list);
};
persistent class Genre {
        unsigned long count;
        indexable char frm[MAX1];
        indexable char sz[MAX1];
public:
        void add_count();
        void sub_count();
        int count_is_zero();
        Genre(char* form, char* size);
        virtual void print();
        persistent Genre* isGenre(char* s0, char* s1);
};
persistent class SrcRef { //short form of Source/Reference in Object Model
        unsigned long count;
        indexable char rel[MAX1];
        indexable char dom_id[MAX1];
public:
        void add_count();
        void sub_count();
        int count_is_zero();
        SrcRef(char* rl, char* d_id);
        virtual void print();
        persistent SrcRef* isSrcRef(char* s0, char* s1);
};
persistent class Annot { //short form of Annotation in Object Model
        char annt[ANNT_MAX];
public:
        Annot(char* ant);
        virtual void print();
};
persistent class SemHdr { //short form of SemanticHeader in Object Model
        unsigned long sh_accesses;//no of accesses to semantic header by GUI
        unsigned long res_accesses; //no of accesses to resource by GUI
        indexable char title[MAX1], alt_title[MAX1], lang[MAX1], char_set[MAX1],
                cr_date[MAX1], exp_date[MAX1], vers[MAX1];
        char spvers[MAX1], cost[MAX1], abs[ABS_MAX];
        short ant_status; //0 if no annt added after registration; 1 otherwise
        persistent UserID* Uid;
        persistent Subject_list* Sub_list;
        persistent Keyword_list* Kw_list;
        persistent Author_list* Aut_list;
        persistent Ident_list* Id_list;
        persistent Clsf_list* Cl_list;
        persistent Covrg_list* Cov_list;
        persistent SysReq_list* Req_list;
        persistent Genre_list* Gnr_list;
        persistent SrcRef_list* Ref_list;
        persistent Annot_list* Ant_list;
  public:
        SemHdr(char* t, char* alt_t, char* lan, char* ch_set, char* c_date,
                 char* x_date, char* ver, char* sver, char* cst, char* abst,
               persistent UserID* Userid,
               LIST2* aut_list, LIST1* sub_list, LIST* kw_list, LIST3* id_list,
               LIST3* cls_list, LIST4* cov_list, LIST4* req_list,
               LIST3* gnr_list, LIST3* ref_list, LIST5* ant_list);
        void add_res_accesses();
        short get_ant_stat();
        unsigned long get_res_accesses();
        void set_ant_stat();
        int delete_sh();
        int update_sh();
        int add_annot();
        virtual void print();
        persistent SemHdr* isSemHdr(char* str);
};
```

As illustrated above, the attributes of the SemHdr class corresponds to those of a Semantic Header document. Some of these attributes are simple strings such as title and alt_title. These attributes take only a single value in a Semantic Header document. Some are persistent linked lists pertinent to those Semantic Header fields that accept more than one value such as author and keyword.

Semantic Header Registration:

The system will perform a number of steps in order to register a Semantic Header. At the beginning, the parser will verify the syntax of the input file and make sure that the mandatory fields of the Semantic Header have been entered. These fields include those needed to assign a Semantic Header Name which are: title, first general subject, name of first author (or name of organization if the value of the field role is 'Corporate Entity'), date of creation, and finally version[3]; as well as first identifier and one keyword. In the meantime, the non-noise words of the Semantic Header will be stored in temporary variables and data structures for later use. The next step would be for the database module to verify the status of the user ID and the password, as explained in section 6.5.1. Subsequently, it will be assured that such Semantic Header does not exist in the database. This is managed by means of a built-in database function *get_obj()*[4]. This function returns 0 if no Semantic Header exists under this name. Finally, the words and the Semantic Header are indexed into the database. Subsequently, the constructor of SemHdr will be called by Ode's *pnew* operator. In this method, the non-noise words will be added to the database and all attributes of the SemHdr object will be initialized, be it single strings or linked lists. Finally, using the database function *set_name()* the SHN will be assigned to the newly added Semantic Header object.

It should be pointed out that in each case where an error occurs, an error code will be sent to the client site and the program will exit.

Semantic Header Deletetion:

A Semantic Header can be deleted by the user who registered the Semantic Header only if no annotation after the SH was registered; the status of annotation is stored in the data member ant_status. The procedure of deleting a Semantic Header is similar to that of registering it. The Semantic Header and the SHN maintained in the word object with the value corresponding to each non-noise word will be deleted from the database. If a word object happens to contain no SHNs, it will be deleted from the database as well. To access the SHN of a Semantic Header we use the database function *get_name()* which returns a pointer to the SHN character string. The deletion procedure is managed by the method delete_sh in the SemHdr class.

Semantic Header Update:

A Semantic Header can be updated by the user who registered the Semantic Header and requires the user ID and the password during the initial registration. However, a number of the fields of a Semantic Header cannot be modified. These fields are

[3] The Semantic Header Name is the concatenation of these five fields.

[4] Note: ODE allows fast access to objects by Object IDs.

those comprising the Semantic Header Name as well as the annotation field added after the SH was registered. The method update_sh() is responsible for updating a Semantic Header. It updates those fields which are modified by the user. For each modified field, the deletion and addition of words are also managed by this method. These operations on words have been explained in registering and deleting a Semantic Header.

Semantic Header Annotation:
A Semantic Header can be annotated by a third party who wishes to make comments about an indexed information resource. The annotation of a Semantic Header is a straightforward procedure managed by the method add_annot(). A number of fields are needed to be entered by the user. These fields include those required to build the Semantic Header Name of the Semantic Header in question, the annotator's personal information such as name and email, and the annotation text. The annotation text will be added to the Semantic Header object by accessing the SHN and including another node to the list of annotations. This operation will also set the value of the data member ant_status to 1, indicating that an annotation text has been added to the Semantic Header after registration. In case the SHN does not exist in the database or the user has not entered the required fields (name and address), an error code will be sent to the user.

Semantic Header Retrieval:
In section 6.2, we described the role of the parser in transforming the user-defined search queries into database queries. We mentioned that this transformation is performed by converting infix to postfix notation. Using this postfix expression from the parser, we will continue the search process in the database module.

In general, to evaluate the postfix expression it is required to pop two elements of the stack, perform the required boolean operation *and* or *or* on them, and finally push the result of the calculation back onto the stack. In this application, the calculation will involve querying the database on these two elements and applying the boolean operation on the results of these queries, and pushing the new result back onto the stack. Handling large size of intermediate results requires care to avoid sluggish query processing. The result of each intermediate query is stored in a balanced binary search tree (adapted from [TRE84]). More precisely, the intermediate and final results should be stored in a linked list of balanced binary search trees. The structures of the tree and the result stack are shown below:

```
typedef struct btree {
        char bi; // balance indicator
        char name[MAX2]; // oids are stored here
        struct btree *left;
        struct btree *right;
} BTREE;

typedef struct result_stack {
        char status; //takes 'e' or 's' (or 'e', 'b', or 'a') indicating
                    //exact/substring (or exact/before/after in case of date)
        char string[MAX1]; // the search word
        int field; //title==0, subject==1, ..., abstract==12
        int flag; //1 if content of bt is to be considered as operand,0 otherwise
        BTREE* bt;
        struct result_stack* next;
} STACK;
```

The search result of each word contains the Semantic Header Names of the Semantic Headers in which the word (or a substring of it) occurs[5]. These SHNs are inserted into a balanced tree. Subsequently, the result of a boolean operation will be achieved by *and*ing or *or*ing two such trees. We will give an example to clarify this procedure. Let us assume that the expression '(OOA **or** OOD) **and** OOP' has been converted to the postfix expression 'OOA OOD **or** OOP **and**'. This expression is stored in the following list mentioned before in section 6.2.

```
typedef struct dlist {
        ELEM* elem;
        struct dlist* left;
        struct dlist* right;
} DLIST;
```

[5] For more detail, see section 6.5.3

Therefore, DLIST (of type struct dlist) will contain five nodes (elements), three operands and two operations. The system starts the search by popping 'OOA' from DLIST and retrieving its SHNs and pushing the resulting BTREE into STACK (of type struct result_stack). Same action is taken for 'OOD'. The next element is the *or* operation. Thus, the last two nodes of STACK will be *or*ed after having them poped from STACK. The result of the operation will be pushed into STACK. Subsequently, OOP will be popped from DLIST and its SHNs, in a BTREE, will be pushed into STACK. Finally, *and* will be popped from DLIST and the last two BTREEs of STACK will be *and*ed. The result is ready to be sent to the client site.

To perform an *and* operation on two trees, we start from the root of one of the trees and check whether the node's value occurs in the other tree. If it does occur, the node is inserted in the result tree. If it doesn't, it is ignored. Subsequently, the function is called recursively to perform the operation on the left and the right branch of the tree. Similar procedure holds for *or*ing two trees. The result tree is initialized to one of the trees (call it t1). The root node of the other tree (call it t2) is added to the result tree if it doesn't occur in t1. The addition of the nodes of t2 to the result tree is carried on recursively on its left and right branches.

It should be noted that depending on the search status of each word, we may have to perform the search based on exact match or substring of the word. In the case of substring search, the matches found will be *or*ed together. To find the substrings of a string, we use two regular expression handlers. These library functions are re_comp() and re_exec(). re_comp() compiles a string into an internal form for pattern matching. re_exec() checks the argument string against the last string passed to re_comp(). In our case, assuming that we need to find the substrings of the string "DBMS", the string passed to re_comp() should look like ".*DBMS.*". Thus, re_exec("DBMS") will return 1 if the string "DBMS" matches the last compiled regular expression.

6.5.3 Word

The class word is needed to support the system's search facility. Therefore, a word object should mainly contain information about Semantic Headers in which the word occurs. However, given that the number of resources is rapidly growing, a word object may be part of millions of Semantic Headers. Thus, the use of efficient information

retrieval and disk space become critical. Having these concerns in mind (and knowing that ODE does not support set-valued attributes), we describe the design of the structure for the class Word:
The class Word contains an array of type *struct context.* Each member of this array corresponds to one of the fields in the search GUI (e.g., title, subject, abstract, ...). Each array member contains three persistent linked lists of class types *OidArr1*, *OidArr2*, and *OidArr3*. Each one of these three classes has an array of Semantic Header Object IDs, or in this context Semantic Header Names (SHN). The main difference among these classes lies in the size of OID arrays. *OidArr1* contains an OID array of 1000. *OidArr2* contains an OID array of 10000. *OidArr3* contains an OID array of 100000. The number of nodes in each of the three classes can grow up to 5. Thus the capacity of each linked list is equal to the size of the OID array multiplied by the constant 5. The definitions of these classes are illustrated below:

```
#define C_MAX 13 // number of search contexts (fields)
#define OID_MAX1 1000 // maximum array size in OidArr1
#define OID_MAX2 10000 // maximum array size in OidArr2
#define OID_MAX3 100000 // maximum array size in OidArr3
#define COEF 5 // indicates max. no. of nodes allowed in each list of OidArr's
// NOTE: The formulas OID_MAX1 * COEF * 2 == OID_MAX2
// and OID_MAX2 * COEF * 2 == OID_MAX3 must hold

persistent class OidArr1 {
        int id; //ranges from 0 to COEF-1, each corresponding to a list node
        char value[MAX1];
        int c_ind;
        int oid_ind; //keeps track of oid array's index (defined below)
        char oid[OID_MAX1][MAX2];
public:
        OidArr1(char* str, int ident, int ind);
        void add_oid(char* str);
        void sub_ind();
        char* get_oid(int i);
        char* get_value();
        int get_oid_ind();
        virtual void print();
        void print2();
};
persistent class OidArr2 {
        int id; //ranges from 0 to COEF-1, each corresponding to a list node
        char value[MAX1];
        int c_ind;
        int oid_ind; //keeps track of oid array's index (defined below)
        char oid[OID_MAX2][MAX2];
public:
        OidArr2(char* str, int ident, int ind);
        void add_oid(char* str);
        void sub_ind();
        char* get_oid(int i);
        char* get_value();
        int get_oid_ind();
        virtual void print();
        void print2();
};
persistent class OidArr3 {
        int id; //ranges from 0 to COEF-1, each corresponding to a list node
        char value[MAX1];
        int c_ind;
        int oid_ind; //keeps track of oid array's index (defined below)
        char oid[OID_MAX3][MAX2];
public:
        OidArr3(char* str, int ident, int ind);
        void add_oid(char* str);
        void sub_ind();
        char* get_oid(int i);
        char* get_value();
        int get_oid_ind();
        virtual void print();
        void print2();
};
struct context {
friend persistent class Word;
private:
        persistent OidArr1_list* Oid1_list;
        persistent OidArr2_list* Oid2_list;
        persistent OidArr3_list* Oid3_list;
};
typedef struct context CTXT; persistent class Word {
        int count[C_MAX];
        indexable char value[MAX1];
        CTXT c[C_MAX];
        void pnew_list1(int c_ind);
        void pnew_list2(int c_ind);
        void pnew_list3(int c_ind);
        void move1_2(int c_ind);
        void move2_3(int c_ind);
        void move2_1(int c_ind);
        void move3_2(int c_ind);
public:
        char* get_value();
        Word(char* w);
        int Word1(char* oid, int c_ind);
        virtual void print();
        int remove_oid(int c_ind, char* oid);
        BTREE* find_oids(BTREE* t, int c_ind);
        int isContext(int c_ind);
        persistent Word* isWord(char* str);
};
```

At the beginning, when a Word object is being indexed, the list of *OidArr1* objects is created in the database. This is done by the method *Word::pnew_list()*. Once the number of OIDs in this list reaches its capacity, the linked list is deleted from the database after it is flushed out into the list of *OidArr2*. This is managed by *Word::move1_2()*. The same scenario holds for the case when *OidArr2* is filled up when it has to be moved to the list of *OidArr3* using the method *Word::move2_3()*. Reverse actions will be taken when the number of OIDs in a linked list becomes equal to the maximum capacity of the next smaller linked list. These are managed by the methods *Word::move2_1()* and *Word::move3_2()*. For example, if the number of OIDs in *OidArr3* equals the capacity of *OidArr2*, the OIDs will be placed in a linked list of *OidArr2* with 5 nodes and finally the list of *OidArr3* will be deleted.

It can be observed that by restricting the number of nodes in each linked list to a small constant, we are reducing the number of disk accesses; and by performing the aforementioned actions to the linked lists as the number of OIDs decreases or increases, we are allocating a reasonable amount of disk space.

There are three methods in the Word class which play important roles in registering, deleting, and retrieving Semantic Headers. The method find_oids() in the Word class is used in the search system. This method is called to retrieve SHNs of a Word object and store them in a BTREE discussed earlier. The method remove_oid() in this class is used for deleting a SHN from a Word object. After deleting the SHN, this method will check the capacity of the current list of SHNs. Subsequently, it will call either of *move2_1* or *move3_2* to make appropriate changes to satisfy the design requirements mentioned above. On registering a Semantic Header, the method *Word1()* is called to index a SHN into a Word object. This method uses the *move1_2()* and *move2_3()* methods to make sure that the design requirements are met. *Word1* returns an integer indicating whether an overflow has occurred; namely, if the maximum capacity of *OidArr3* has been reached. The method *isWord()* is used to retrieve a Word object from the database. It returns a pointer to the object. If no object was found, it would return 0. The definition of this method and including the retrieval query follows:

```
persistent Word* Word::isWord(char* str) {
persistent Word* Wrd;
for (Wrd in Word) suchthat (strcmp(Wrd->value, str) == 0)
    return Wrd;
return 0;
}
```

6.6 Results

We will close this chapter by illustrating some results produced by execution of the system. It should be noted that, although the execution is in a client-server environment, we will only consider the input files and output files transmitted to/from the database module, regardless of the Graphical User Interface module. To familiarize the reader with the actual interaction between the database and the GUI modules, however, we will include actual query and result in the first example.

Example 1:
The following example shows a level_2 subject search based on given level_0 and level_1 subjects. Figure 42 illustrates the pertinent part of the GUI for this subject search.

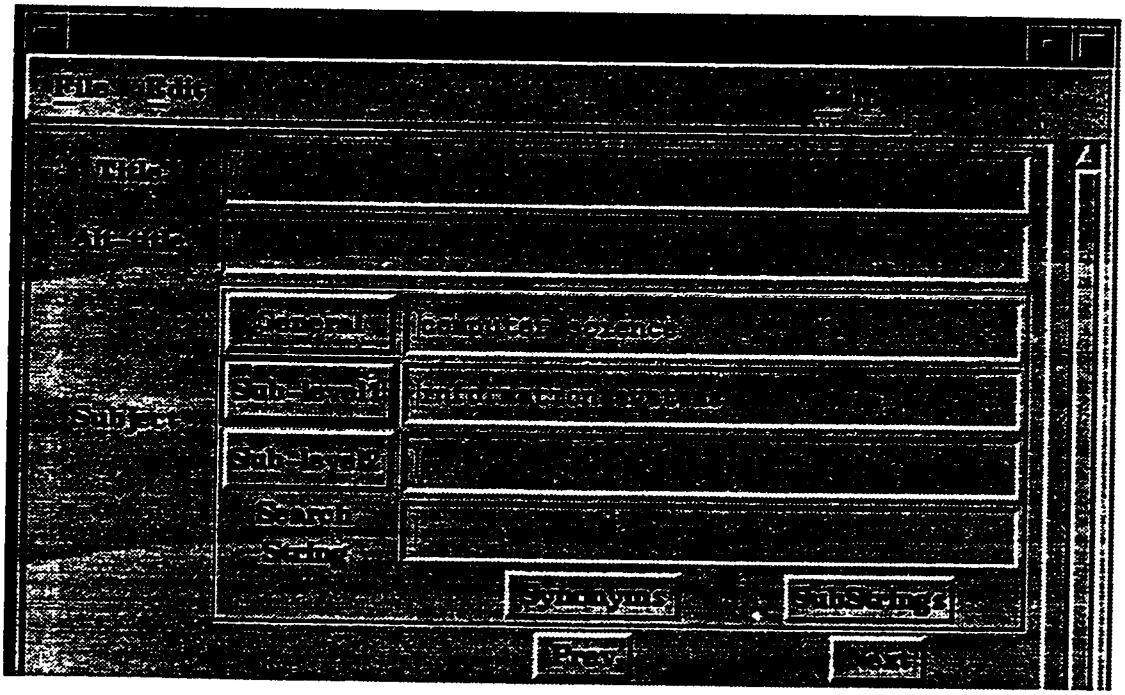
Query 1:

```
2
<level0> computer science </level0>
<level1> information systems </level1>
<EOF>
```

The number 2 indicates that the search should be done on level_2 and <EOF> indicates the end of the file. In return, database will provide the following level_2 subjects. Figure 43 illustrates the result of this search displayed in the GUI.

Result 1:

```
0 2
<level2> abstracting methods </level>
<level2> animations </level>
<level2> artificial realities </level>
<level2> asynchronous interaction </level>
<level2> audio input/output </level>
```



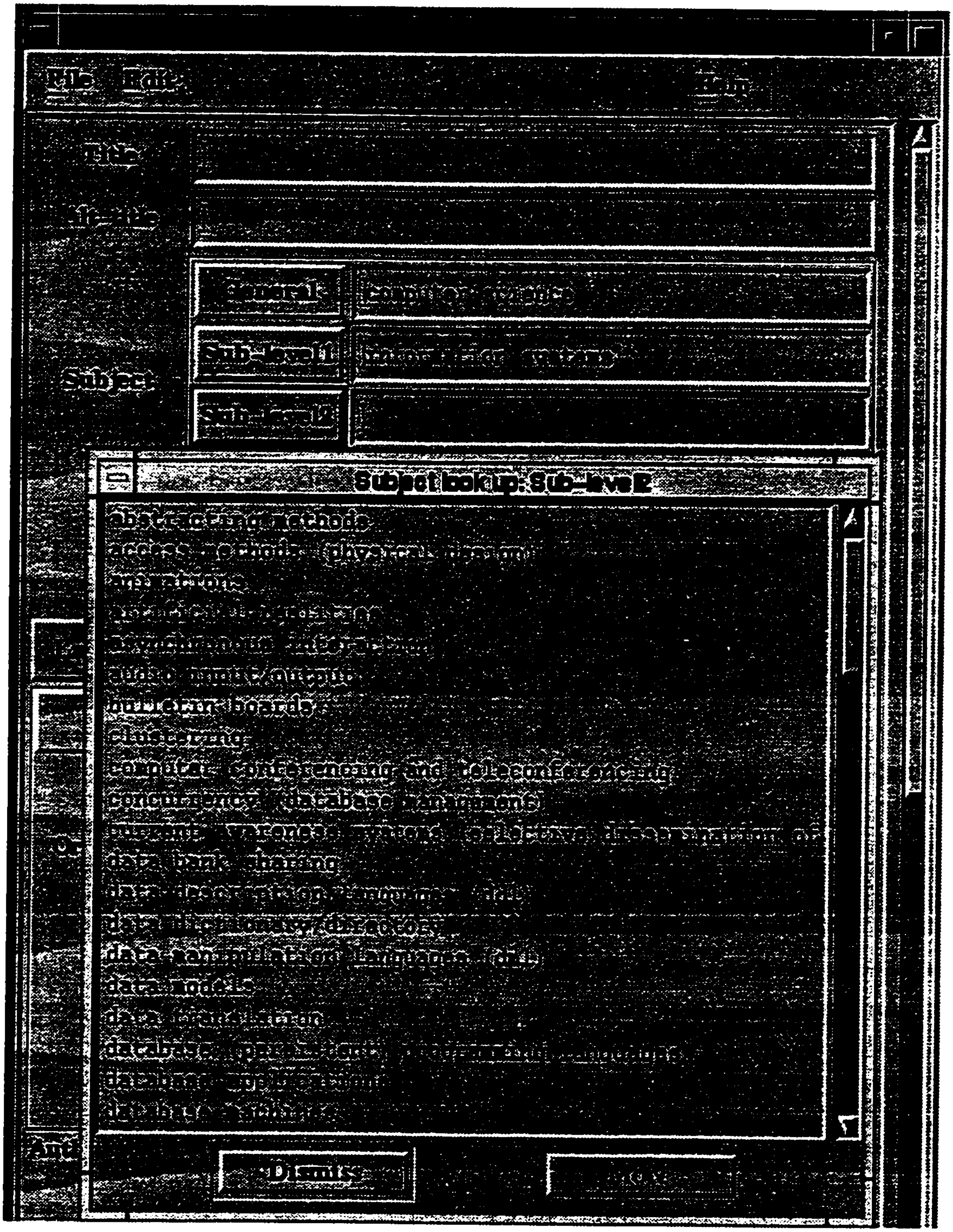
computer science

information systems

Figure 42: GUI: a level_2 subject search

```
<level2> bulletin boards </level>
<level2> clustering </level>
<level2> selection process </level>
<level2> spreadsheets </level>
<level2> synchronous interaction </level>
<level2> theory and methods </level>
<level2> theory and models </level>
<level2> thesauruses </level>
<level2> transaction processing </level>
<level2> value of information </level>
<level2> videotex </level>
<level2> windowing systems </level>
<level2> word processing </level>
<EOF>
```

The first number in the result file indicates an error code. The number 0 states that there has been no errors. The number 2 corresponds to the subject level, which is level_2.



computer science

information systems

abstracting methods
access methods (physical design)
animations
artificial realities
asynchronous interaction
audio input/output
bulletin boards
clustering
computer conferencing and teleconferencing
concurrency (database management)
current awareness systems (selective dissemination of
data bank sharing
data description languages (ddl)
data dictionary/directory
data manipulation languages (dml)
data models
data translation
database (persistent) programming languages
database applications
database machines

Figure 43: GUI: Result of a level_2 subject search

Example 2:

The following input requests a subject search on the synonym string 'comu'; the number 3 indicates a search based on synonym.

Query 2:

```
3 <string> comu </string>
<EOF>
```

The following is the result of the synonym search. The last three numbers on the first line indicate the number of subject terms found in each hierarchy level. Since no subject terms are found, the numbers are all zeros.

Result 2:

```
0 3 0 0 0
<Level0>
<EOL>
<Level1>
<EOL>
<Level2>
<EOL>
<EOF>
```

Example 3:

In the following example, a substring search on subject hierarchies is to be performed. The number 4 indicates a search on substrings.

Query 3:

```
4 <string> ba </string>
<EOF>
```

As illustrated below, 1 general subject, 0 level_1 subject, and, 4 level_2 subjects are found for the above query.

Result 3:

```
0 3 1 0 4
<Level0>
<level0> balkan peninsula </level>
<EOL>
<Level1>
<EOL>
<Level2>
<level0> computer science </level> <level1> computing methodologies </level>
<level2> backtracking </level>
<level0> computer science </level> <level1> data </level> <level2> backup/recovery
</level>
<level0> computer science </level> <level1> software </level> <level2> backup
procedures </level>
<level0> computer science </level> <level1> software </level> <level2> batch
processing systems </level>
<EOL>
<EOF>
```

Example 4:

The following input requests a Semantic Header registration. The tag <semhdrR> indicates that a Semantic Header is to be registered.

Query 4:

```
<semhdrR>
<userid> shayan </userid>
<password> passwd </password>
<titl> The CINDI System </title>
<alttitl> </alttitle>
<subject>
<general> computer science </general>
<sublevel1> information systems </sublevel1>
<sublevel2> search process </sublevel2>
</subject>
```

```
<languag> English </language>
<char-set> </char-set>
<author>
<arole> Corporate Entity </arole>
<aname> Concordia University </aname>
<aorg> </aorg>
<aaddress> </aaddress>
<aphone> </aphone>
<afax> </afax>
<aemail> </aemail>
</author>
<keyword> metadata, Semantic Header </keyword>
<identifier>
<domain3> HTTP </domain3>
<value3> to be specified </value3>
</identifier>
<dates>
<created> 1997/03/14 </created>
<expiry> </expiry>
</dates>
<version> </version>
<spversion> </spversion>
<classification> <domain4> </domain4>
<value4> </value4>
</classification>
<coverage>
<domain5> </domain5>
<value5> </value5>
</coverage>
<system-requirements>
<component> </component>
<exigency> </exigency>
</system-requirements>
<genre>
```

```
<form> </form>
<size> </size>
</genre>
<source-reference>
<relation> </relation>
<domain-identifier> </domain-identifier>
</source-reference>
<cost> </cost>
<abstract>
The document discusses issues about
electronic information indexing and
retrieval.
</abstract>
<annotation>
</annotation>
</semhdr>
<EOF>
```

The result of the query sent to the client is a 0 code indicating that the registration has been done successfully.

Result 4:

```
0
<EOF>
```

Example 5:
When adding an annotation to an existing Semantic Header, those fields of a Semantic Header which make up the SHN are sent to the server site, with the annotator's personal information and the text of the annotation.

Query 5:

```
<annotation>
<titl> A System for Seamless Search of Distributed Information Sources </title>
<arole> Author </arole>
```

```
<aname> Bipin C. Desai </aname>
<aorg> Department of Computer Science, Concordia University </aorg>
<general> computer science </general>
<created> 1994/06/15 </created>
<version> </version>
========================================
name: nader shayan
organization: concordia
address: montreal
phone:
fax:
email: shayan@cs
========================================
This section is used for annotation.
</annotation>
<EOF>
```

The result of the query sent to the client is a 0 code indicating that the annotation has been done successfully.

Result 5:

```
0
<EOF>
```

Example 6:

Finally, the last example will illustrate Semantic Header search request. The infix expression for this search request is:
(computer science **or** electrical engineering) **and** (desai **and** shing), where 'desai' is based on exact match (indicated with letter e between the tages <sts> and </sts>), and 'shing' is based on substring matches (indicated with letter s). The letters & and |, surrounded by the tags <oper> and </oper>, correspond to *and* and *or* operations, respectively. The third line of the input, the number 10, indicates the number of Semantic Headers to be retrieved per block. In fact, this number stands for the size of each block of Semantic Headers to be sent to the client site. The second line contains

the block number. We will clarify this with an example. Suppose the user makes a search request and asks for the retrieval of 10 Semantic Headers. Let us assume that the corresponding search operation finds 16 Semantic Headers. However, only 10 Semantic Headers will be sent to the user. The user can ask for the remaining 6 Semantic Headers. In this case the block number will be 2, indicating the retrieval of the second 10 (or less) Semantic Headers.

Query 6:

```
<search>
<blockno> 1 </blockno>
<nosh> 10 </nosh>
<titl> </title>
<sts> </sts>
<subject>
<general> computer science </general>
<sublevel1> </sublevel1>
<sublevel2> </sublevel2>
<oper> | </oper>
<brck> ( </brck>
<brckN> 1 </brckN>
<general> electrical engineering </general>
<sublevel1> </sublevel1>
<sublevel2> </sublevel2>
<oper> </oper>
<brck> ) </brck>
<brckN> 1 </brckN>
</subject>
<author>
<sts> e </sts>
<aname> desai </aname>
<aorg> </aorg>
<oper> & </oper>
<brck> ( </brck>
<brckN> 1 </brckN>
<sts> s </sts>
<aname> shing </aname>
<aorg> </aorg>
<oper> </oper>
<brck> ) </brck>
<brckN> 1 </brckN>
</author>
<identifier>
<sts> </sts>
<domain3> </domain3>
<value3> </value3>
<oper> </oper>
<brck> </brck>
<brckN> </brckN>
</identifier>
<kw>
<keyword> </keyword>
<oper> </oper>
<brck> </brck>
<brckN> </brckN>
</kw>
<dateAft> </dateAft>
<dateBef> </dateBef>
<languag> </language>
<version> </version>
<abstract> </abstract>
</search>
<EOF>
```

The result of this search is the number of Semantic Headers found. This is sent along with the Semantic Headers to the client site. Obviously, the number of Semantic Headers sent will be less than or equal to that the user has asked for.

Result 6:

```
0 20
<semhdr>
content of an SH
</semhdr>
...
...
<semhdr>
content of an SH
</semhdr>
<EOF>
```

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Current index systems are based on harvesting the network 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. The current Dublin Metadata Element list also suffers from the absence of the abstract. Furthermore, 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 infeasibility of the existing approaches becomes clear as more and more providers of information would require payment.

In the system based on the Semantic Header, the provider of the resource is the one who prepares the index information. Consequently, such an index entry would be more reliable than one derived by a third party or by simply scanning a document. The inclusion of an abstract in the index entry enables the provider of the resource to highlight the nature of the subject. If asked to pay a fee, users would not be inclined to retrieve resources with irrelevant titles. The Semantic Header provides additional details about the resource and allows users to make better informed decisions regarding the relevance of the source resource.

The index database contains a number of control entries for resource. Control entries are items such as the size of the resource, the password for authenticating subsequent updates of the index entry, and a list of annotations made about the resource by independent users. The Semantic Header based cataloging and discovery system has two major components for indexing and searching.

The system supports an expert system-driven graphical interface for the provider of the resource to produce an index entry, and have this entry entered in the index database. The expert system provides help in choosing appropriate terms for index entries such as subject, sub-subject, keywords etc. It is also responsible for verifying the consistency of the index entry and the accessibiblity of the resource and then for posting the index entry to the index database.

While searching, a user's search requirement is often vague; hence, the system guides the users in articulating their needs. The search component provides better search syntax and offers help to the user in the terminology used. This supports context sensitivity missing in many existing search systems which are evolving towards our model. The selectivity of the resulting search is thus improved. SHN provides an element for a basic index and simpler search mechanism for non-graphical use.

The Semantic Header based system proposed meets the challenge of the coming information age by defining: a metadata structure which allows automatic and semi-automatic (human assisted) extracting of metadata from resources; building a distributed indexing system; providing an expert based resource registering and searching systems with an intuitive graphical user interface to interact in the registering and discovery process. The distributed and replicated nature of the Semantic Header database provides reliability and scalability.

7.2 Contribution of this thesis

The contributions made by this thesis to the CINDI project are listed below:

- Design and implementation of the Database subsystem for indexing and retrieval of Semantic Headers. Furthermore, for the DBA of the system, automatic indexing and deleting of subject terms as well as Semantic Headers have been provided.

- Employment of the recursive descent parsing technique for the implementation of the parser, mainly, to extract data items, received from the client site, and send them to the database subsystem.

- Implementation of the Expert System of the Database subsystem to help users in subject retrieval. This module is distributed in the database subsystem.
- Implementation of the client-server communication using TCP/IP.

7.3 Future Work

The current prototype of the CINDI System satisfies the needs of the Internet users for effective retrieval of electronic information resources. However, in the near future, other functionalities could be built upon the prototype to better serve the Internet community. Furthermore, the performance and the tune-up of the database module could be studied.

- A major extension to the system would be to build a distributed system based on this prototype centralized system.
- The subject areas of the Thesaurus Database could be expanded using the classification of the Library of Congress Subject Headings.
- To facilitate the Semantic Header entry, a new functionality will be added to automatically create a Semantic Header for an information resource. This project is already in progress at Concordia as a part of the theses of other graduate students.
- In the search subsystem the need for additional *and/or* operations for some of the search fields should be taken into consideration.
- A new feature will be added to those search fields that can be searched based on exact match and substrings. With this feature, the system will look for words analogous to the search word entered by the user.
- The performance of the Word structure should be examined as to the use of linked lists instead of sets. In case of the availability of sets in ODE, the Word structure should be adapted to function based on sets.
- The query processing strategies should be studied to verify the amount of overhead caused by large intermediate results, kept in balanced trees, in searching for Semantic Headers.

- The current on-line help of the GUI could be extended to better help users understand the system.

- The application could be internationalized to support various languages.

Bibliography

[AGG] Arlein, R., Gava J., Gehani, N., Lieuwen, D., Ode 4.1 (Ode<EOS>) User Manual, AT&T Bell Laboratories.

[AGW] Antoniou, G. and Wachsmuth, I., Structuring and Modules for Knowledge Bases: Motivation for a New Model. *Knowledge-based Systems*, Vol. 7-1, pp. 49-51.

[Agr89] Agrawal, R., Gehani, H., Ode (object database and environment): the language and the data model. *Proc. ACM-SIGMOD 1989 Int'l Conf. Management of Data*, pages 36-45, May 1989.

[Agr93] Agrawal, R., Dar, S., Gehani, N., The O++ Database Programming Language: Implementation and Experience, *Proc. IEEE 9th Int'l Conf. Data Engineering*, pp. 61-70, 1993.

[Aho88] Aho, V. A., Sethi, R., Ullman, J. D., Compilers: Principles, Techniques, and Tools, Addison-Wesely, 1988.

[Arch] *Guide to Network Resource Tools - archie*
http://kelim.jct.ac.il/science/online/archie.html

[BAP93] Biliris, A., Panagos, E., EOS User's Guide (Release 2.0), AT&T Bell Labs, 1993.

[BCD90] Desai, B. C., Introduction to Database Systems, West, St. Paul, MN, 1990.

[BCD92] Desai, B. C., Pollock, R., MDAS: A Heterogeneous Distributed Database Management System, *Information and Software Technology*, Jan. 1992, Vol. 34-1, pp. 28-41.

[BCD94] Desai, Bipin C., Shinghal, R. 'A system for Seamless Search of Distributed Information Sources', May 1994. http://www.cs.concordia.ca/old/w3-paper.html

[BCD95a] Desai, Bipin C. 'Report of the Metadata Workshop, Dublen, OH (March 1995)', http://www.cs.concordia.ca/faculty/bcdesai/metadata/metadata-workshop-report.html

[BCD95b] Desai, B. C., *Internet Indexing Systems vs List of Known URLs*, June 1995. http://www.cs.concordia.cs/ faculty/bcdesa/test-of index-systems.html

[BCD97] Desai, Bipin C. 'Supporting Discovery in Virtual Libraries, Jan. 1997', http://www.cs.concordia.ca/ faculty/bcdesai/

[BDJ] Byrne, D. J., *MARC manual: understanding and using MARC record*, Libraries Unlimited, Englewood, Colo.

[BH95] Brody, H., Internet@crossroad, *Technology Review*, May/June, also URL http://web.mit.edu/afs/athena/org/t/techreview/www/articles/may95/ Brody.html

[BJN] Brownlee, J. N., *New Zealand Expreiences with Network Traffic Charging*, http://www.auckland.ac.nz/net/Accounting/nze.html

[BLTC] Berners-Lee, T., Connolly, D., *UR* and The Names and Addresses of WWW objects*,
http://www.w3.ch/hypertext/WWW/Addressing/Addressing.html

[BLT90] Berners-Lee, T., Cailliau, R., *WorldWideWeb: Proposal for a HyperText Project*, 1990. http://www.w3.org/hypertext/WWW/Proposal.html

[BLT93] Berners-Lee, T., *Wide Web Initiative: The Project*, 1993.
http://info.cern.ch/hypertext/WWW/TheProject

[BLT94] Berners-Lee, T., Cailliau, R., Luotonen, A., Frystyk Nielsen, H., and Secret, A., 'The World Wide Web'. In *Communications of the ACM*, August 1994, vol.37-8, pp. 76-82.

[BPE] Deutsch, P., Emtage, A., Koster, M., 'Publishing Information on the Internet with Anonymous FTP', http://src.doc.ic.ac.uk/computing/internet/internet-drafts/draft-ietf-iiir-publishing-03.txt.

[CPG] Chander, P. G., Shinghal, R., Desai, B. C., Radhakrishnan T., An Expert System to Aid Cataloging and Searching Electronic Documents on Digital Libraries. *Expert Systems with Applications.* To appear in issue 12(4), June 1997.

[CPG95] Chander, P. G., Shinghal, R. and Radhakrishnan, T., Goal Supported Knowledge Base Restructuring for Verification of Rule Bases. In *Notes of the IJCAI'95 Workshop on Verfication and Validation of Knowledge-Based Systems*, Montreal, pp. 15-21.

[CRW] Cromwell, W. The Core Record: A New Bibliographic Standard. *Library Resources and Technical Services*, 38(4), pp. 415-424.

[CR91] 'The Full Computing Reviews Classification System (1991 version)'. In *Computing Reviews*, January 1995, p.6.

[Cans] *Canonical stems and noise words for the Peregrinator*, http://www.maths.usyd.edu.au:8000/jimr/pe/Stems.html

[CrW] Crawford, W., *MARC for Library Use: Understanding USMARC*, G. K. Hall, Boston, MA, 1984.

[Cr91] Cocchi, R., Estrin, D., Shenker, S. and Zhang, L., A Study of Priority Pricing in Multiple Service Class Networks, *Proc of SIGCOMM*, Sept. 1991 revised version at: ftp://parcftp.xerox.com/pub/net-reserch/pricing1.ps.Z

[DBP] De Bra, P., Houben, G-J., and Kornatzki, U. 'Navigational Search in the World-Wide Web', http://www.win.tue.nl/help/doc/demo.ps

[DPE] Deutsch, P., Emtage, A., Koster, M. 'Publishing Information on the Internet with Anonymous FTP', http://src.doc.ic.ac.uk/computing/internet/internet-drafts/draft-ietf-iiir-publi shing-03.txt

[EIN] *EINet Galaxy*, http://www.einet.net/

[GIO] Giordano, R., The Documentation of Electronic Texts Using Text Encoding Initiative Headers: An Introduction, *Library Resources and Technical Services*, Vol. 38-4, pp. 389-401.

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

[Gyn] Gaynor, E., Cataloging Electronic Texts: The University of Virginia Library Experience, *Library Resources and Technical Services*, Vol. 38-4, pp. 403-413.

[HKL] Horny, K. L., Minimal-level cataloging: A look at the issues
A symposium, *Journal of Academic librarianship*, Vol. 11, pp. 332-334.

[HTML] *HyperText Markup Language (HTML)*,
http://www.w3.org/pub/WWW/MarkUp/

[HarI] *Query Interface to the Derma Harvest Server Broker*,
http://www.uni-erlangen.de/Harvest/brokers/DERMA/

[HarS] *The Harvest Broker Searching System*,
http://www.okbmei.msk.su/FAQ/Harvest.html

[InS] *InfoSeek*, http://www.infoseek.com:80/

[JRJK] Jacob, R. J. K. and Froscher, J. N., A Software Engineering Methodology for Rule-Based Systems. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2-2, pp. 173-189.

[KB91] Kahle, B., *An Information System for Corporate Users: Wide Area Information Servers*, Thinking Machines Technical Report TMC-199, April 1991.

[KJ93] Kochmer, J., NorthWestNet. *The Internet Passport: NorthWestNet's Guide to Our World Online*, published by NorthWestNet, Bellevue, WE, 1993.

[Katz] Katz, W. A., *Introduction to Reference Work*, Vol. 1-2 McGraw-Hill, New York, NY.

[Ker84] Kernighan, B., Pike, R., The UNIX Programming Environment. Prentice-Hall Sofware Series, 1984.

[Kos] Koster, M., *ALIWEB* http://www.nexor.co.uk/public/aliweb/aliweb.html

[Kos96] Koster, M., *The Web Robots Pages*, 1996. http://info.webcrawler.com/mak/projects/robots/robots.html

[LHL] Lieberman, H., Letizia: An Agent that Assists Web Browsing. In *Proceedings of the Fourteenth International Joint Conference in Artificial Intelligence*, Montreal, pp. 924-929.

[LYC] *Lycos-The Catalog of the Internet*, http://lycos.cs.cmu.edu/

[MBO] McBryan, Olever A., *World Wide Web Worm*, http://www.cs.colorado.edu/home/mcbryan/WWWW.html

[MML95] Mauldin, M. L., Meassuring the Web with Lycos, *Poster Proceeding of the Third International WWW Conf.*, Darmstadt, April 1995, pp. 26-29.

[MMJ] MacKie-Mason, J., Varian, H., *Usage-Based Pricing: Analyses of Various Pricing Mechanism*, http://gopher.econ.lsa.unich.edu/EconInternet/Pricing.html

[Net] 'Netscape Knows Fame and Aspires to Fortune'. In *The New York Times*, March 1, 1995.

[PTM90] Petersen, T., Molholt, P. (ed), *Beyound the book: extending MARC for subject acess*, G. K. Hall, Boston, MA, 1990.

[RBSE] *RBSE Spider*, http://rbse.jsc.nasa.gov/Spider/

[Rhee] Rhee, S., Minimal-level cataloging: Is it the best local solution to a national problem?, *Journal of Academic librarianship*, Vol. 11, pp. 336-337.

[Ross] Ross, R. M., West, L., MLC: A contrary viewpoint, Journal of Academic librarianship, Vol. 11, pp. 334-336.

[Rum91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenson, W., *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[SIM] Simon, H. A., The Structure of Ill-Structured Problems. *Artificial Intelligence*, 4, pp. 181-201.

[Shin92] Shinghal, R., *Formal Concepts in Artificial Intelligence*, Chapman & Hall, London, U.K., co-published in the U.S. with Van Nostrand, New York, 1992.

[Soll] Sollins, K. Masinter, L. *Functional Requirements for Uniform Resource Name*, RFC1737, ftp://ds.internic.net/rfc/rfc1737.txt

[Ste90] Stevens, R., UNIX Network Programming. Prentice-Hall Software Series, 1990.

[TEI] *TEI Guidelines for Electronic Text Encoding and Interchange*, http://etext.virginia.edu/bin/tei-tocs?div=DIV1&id=SG

[TRE84] Tremblay, J. P., Sorenson, P. G., 'An Introduction to Data Structures with Applications', McGraw-Hill, 2nd edition, 1984.

[Thau] Thau, R., 'SiteIndex Transducer'. http://www.ai.mit.edu/tools/site-index.html

[VER] *How to compose veronica queries.* gopher://gopher.scs.unr.edu/hh/veronica/About/how-to-query-veronica

[WSG] Weibel, S., Godby, J., Miller, E., Daniel, R., *OCLC/NCSA Metadata Workshop Report*, http://www.oclc.org:5046/conferences/metadata/dublin_core_report.html

[WebC] *WebCrawler*, http://webcrawler.cs.washington.edu/WebCrawler/Home.html

[W3C] *W3 Catalog*

[Yah] *Yahoo Search*, http://www.yahoo.com/search.html

Appendix A

Example of a Semantic Header

```
<semhdrR>
<userid> bcdesai </userid>
<password> confidential </password>
<title> Semantic Header and Indexing and Searching on the Internet </title>
<alttitle> Sailing the Internet with a navigational System </alttitle>
<subject>
<general> computer science </general>
<sublevel1> information storage and retrieval </sublevel1>
<sublevel2> indexing </sublevel2>
<general> library studies </general>
<sublevel1> cataloging </sublevel1>
<sublevel2> semantic header </sublevel2>
<general> computer science </general>
<sublevel1> database management </sublevel1>
<sublevel2> distributed databases </sublevel2>
</subject>
<language> English </language>
<char-set> ISO-8879 </char-set>
<author>
<arole> Author </arole>
<aname> Desai, Bipin C. </aname>
<aorg> Concordia University </aorg>
<aaddress> 7141 Sherbrooke Street West </aaddress>
```

```
<aphone> (514) 848-3025 </aphone>
<afax> (514) 848-8652 </afax>
<aemail> bcdesai@cs.concordia.ca </aemail>
<keyword> Bibliographic Record, Content Description </keyword>
<identifier>
<domain3> HTTP </domain3>
<value3> http://www.cs.concordia.ca/doc.html </value3>
</identifier>
<dates>
<created> 1994/07/11 </created>
<expiry> 1996/01/11 </expiry>
</dates>
<version> 1.1 </version>
<spversion> 1.0 </spversion>
<classification>
<domain4> Legal </domain4>
<value4> Copyright for free </value4>
<domain4> Security Level </domain4>
<value4> Public </value4>
</classification>
<coverage>
<domain5> Audience </domain5>
<value5> Computer Science, Library Science, Inter Types </value5>
</coverage>
<system-requirements>
<component> Hardware </component>
<exigency> Workstation, Mainframe </exigency>
<component> Software </component>
<exigency> Browser </exigency>
<component> Network </component>
<exigency> Internet </exigency>
</system-requirements>
<genre>
<form> Text </form>
```

```
<size> 45000 bytes </size>
</genre>
<source-reference>
<relation> Derived from tex file </relation>
<domain-identifier> cindi3.tex </domain-identifier>
</source-reference>
<cost> 0.31$ Cnd. </cost>
<abstract>
This paper describes an indexing system based on a data set called Semantic Header
for Internet resources...
</abstract>
<annotation>
The Semantic Header was first conceived as a cataloging Internet Resource...
</annotation>
</semhdr>
```

Appendix B

OMT Notations

The notations of the Object Modeling Technique for Object, Dynamic, and Functional Models are illustrated in Figures 44, 45, and 46, respectively.

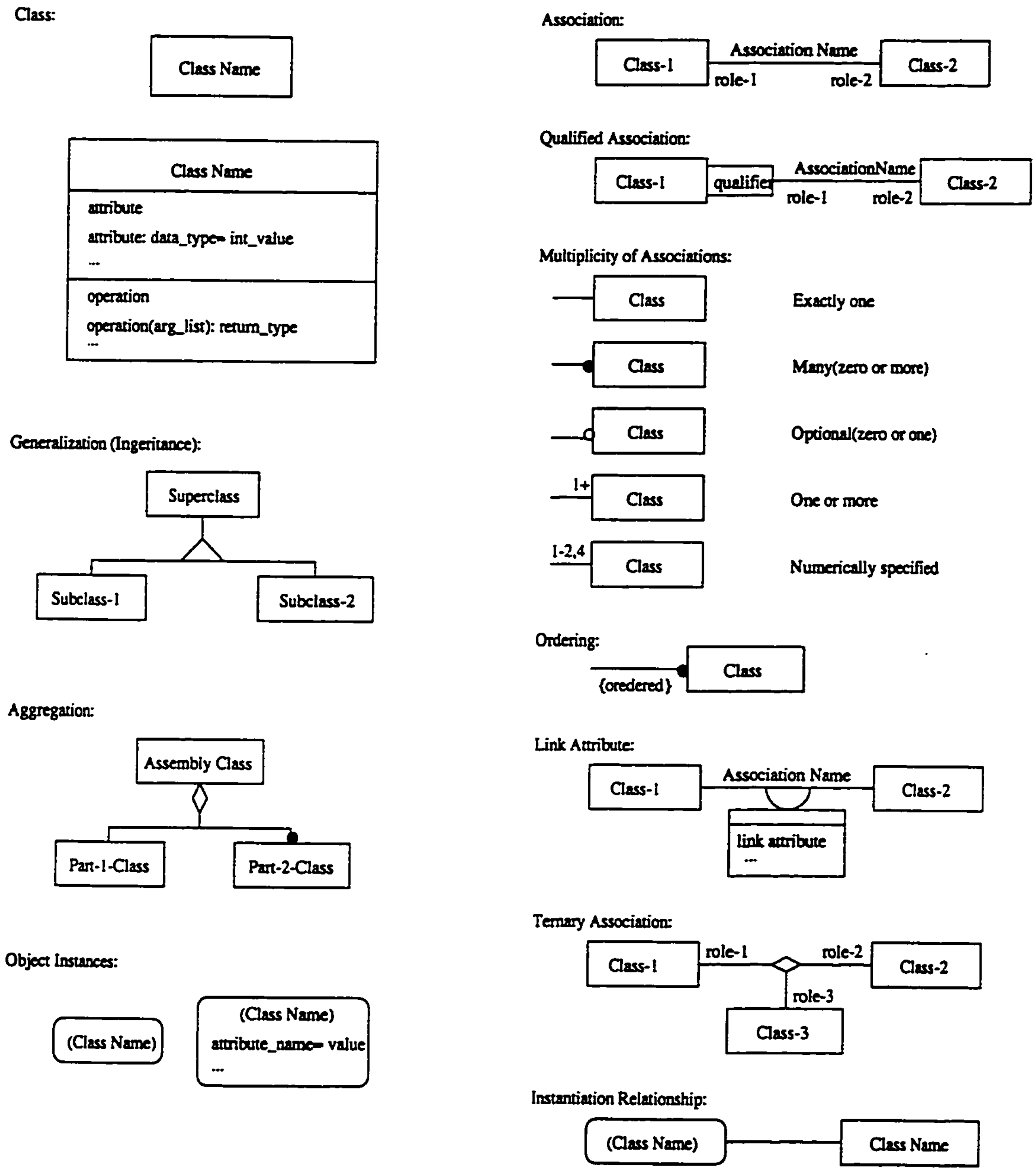


Figure 44: Object Model Notation

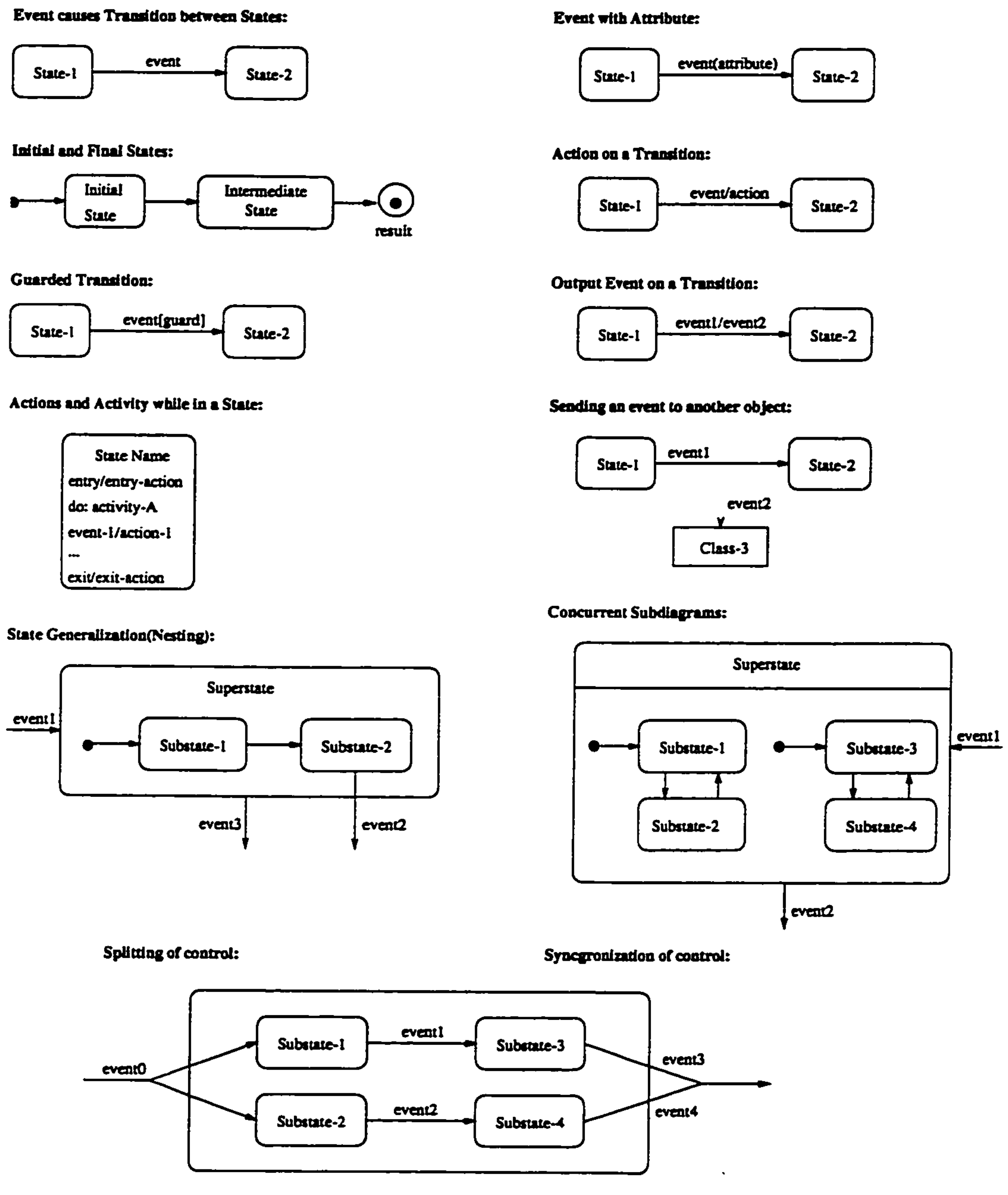


Figure 45: Dynamic Model Notation

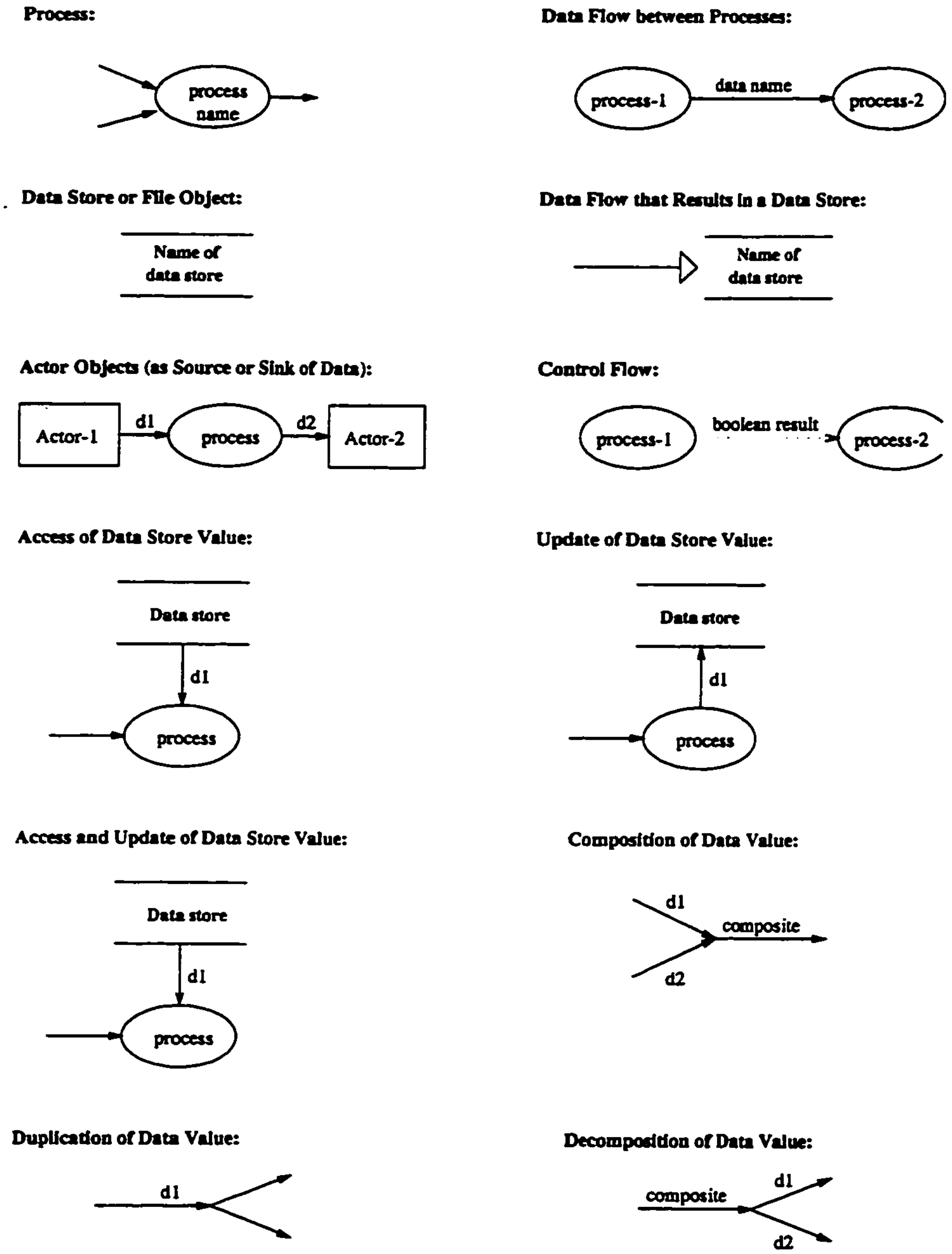
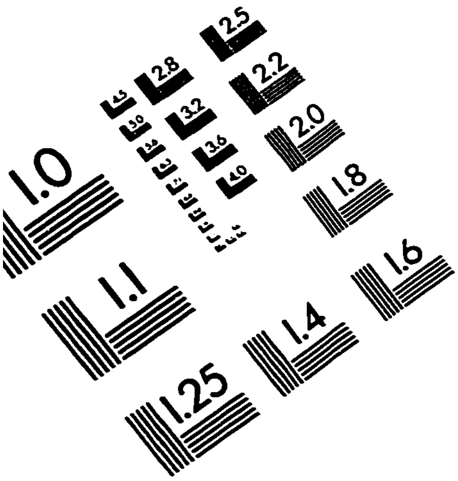
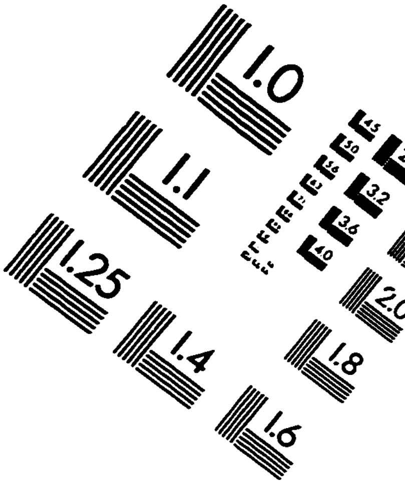
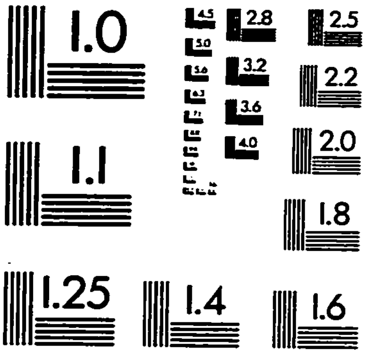

Process:
process name
Data Flow between Processes:
process-1
data name
process-2
Data Store or File Object:
Name of data store
Data Flow that Results in a Data Store:
Name of data store
Actor Objects (as Source or Sink of Data):
Actor-1
d1
process
d2
Actor-2
Control Flow:
process-1
boolean result
process-2
Access of Data Store Value:
Data store
d1
process
Update of Data Store Value:
Data store
d1
process
Access and Update of Data Store Value:
Data store
d1
process
Composition of Data Value:
d1
composite
d2
Duplication of Data Value:
d1
Decomposition of Data Value:
d1
composite
d2

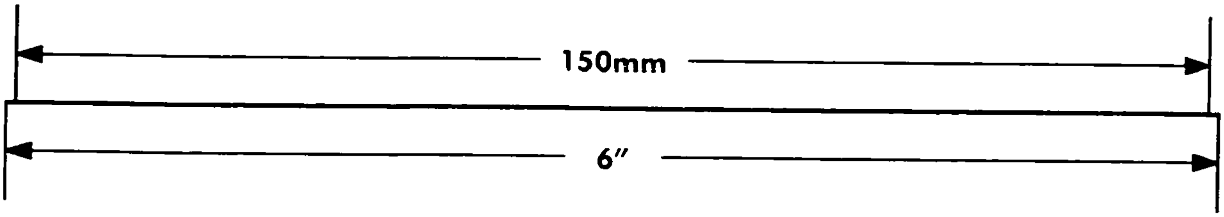
Figure 46: Functional Model Notation

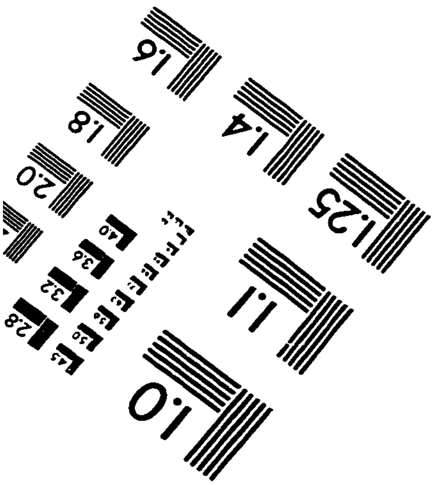
IMAGE EVALUATION
TEST TARGET (QA-3)

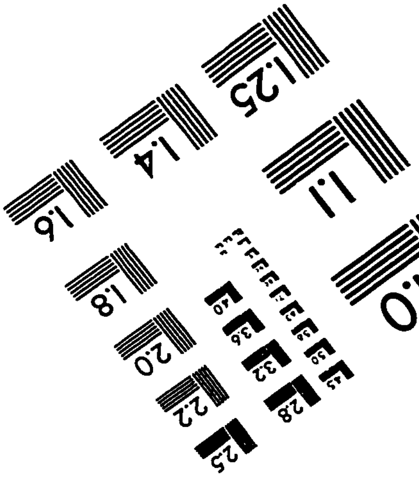












APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989