INFORMATION TO USERS

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

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

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

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

WEB BASED CINDI SYSTEM

Mohamed Amokrane Mechouet

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 2001

Abstract

Web Based CINDI System

Mohamed Amokrane Mechouet

This thesis presents the design and implementation of the CINDI System based on the World Wide Web. The CINDI (Concordia INdexing and DIscovery) is a system which enables a user to index and discover information resources on the Internet. The information resource is described using a metadata called a Semantic Header, which is stored in a distributed databases management system; it is distributed for reliability and availability. The Semantic-Header is replicated at different nodes of the Internet. The locations of such nodes are stored in a database catalog.

A prototype has been developed based on this proposal. The prototype is composed of three main subsystems: the Web Server, which also hosts the catalog database, a primary site for the Semantic Header and subject databases, and a replicated site. In the prototype, UML (Unified Modeling Language) has been employed for the analysis and design of the CINDI system using Oracle Database Management System for its implementation. The Apache Web Server is used to communicate between the Browser (the client) and the Databases (the server) subsystems using the HTTP protocol. This communication is implemented using Java servlets which have two main functions. They are used as server to retrieve the HTML form information from the browser, and as client to send requests to Oracle databases.

Acknowledgments

I would like to express my deepest gratitude to my supervisor Dr. Bipin C. Desai.
His guidance and encouragement made my thesis work a pleasant and extremely educational experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Discovery Problem

In recent years, the Internet has become extremely popular throughout most of the world. Computers, along with network facilities, have found their way into many 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 using the WWW (World Wide Web) ([BLT90], [BLT93]). The Web has become the accepted norm of disseminating and sharing information resources in hyper-media.

There is a need for the development of a system which allows easy "search and access" to resources available on the Internet. A system for cataloging and indexing data has three constituents: a method of collecting information about the data, a database for storing the information and a method of selectively accessing data using the information. In practice, the complicated part of creating a Web-based cataloging and indexing system is getting the information into the database. Once the information has been stored, it becomes relatively straightforward to search and manipulate using standard database techniques. With servlets and browsers available, the user interface design is also straightforward. The prerequisites for operation of such a system are a Java compiler and virtual machine, access to an SQL conformant database and access to a Web server capable of running servlets.

Existing Search Systems on the Internet

The vast amount of information available today on the Internet has great potential to improve the quality of life. To make effective use of the wealth of information on the Internet, users need ways to locate pertinent information. In the past few years, many second and third generation search systems have been developed and some of them have gained wide acceptance on the Internet. These include AltaVista, Google, HotBot and Lycos. 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. Moreover, some of them are specialized for the WWW, while others are designed to locate files on Anonymous FTP (File Transfer Protocol) sites. The search interface of these systems provides users with little flexibility and the search results are not always pertinent.

Table 1 summarizes the test results of some of the existing search systems using keywords Bipin (AND) Desai. There were 329 URLs (Universal Resource Locator) containing Bipin (AND) Desai on the WWW at the time the test was taken. The test was carried out in February 2001. The terminology used in the table is as follows:

- Search System: System used to conduct the search
- Number of Hits: The number of documents found containing Bipin (AND) Desai.
- Number of Duplicates: The number of times the same document was retrieved.
- Number of Miss-Hits: The number of irrelevant documents found.
- Number of Missed: The number of relevant documents not found even though they exist on WWW.

Table 1 shows that none of these systems was successful in retrieving all documents sought. The reason for these unexpected results is that many of these systems attempt to match the specified search terms without regard for the context in which the words appear in the target information resource.

Search System	Number of Hits	Number of Duplicates	Number of Miss-Hits	Number of Items missed
AltaVista	99	24	67	230
Google	155	10	403	174
HotBot	62	21	121	267
Lycos	239	37	711	90

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

1.1.1 AltaVista search engine

Table 2 shows the pages distribution of the test results of the AltaVista search system using keywords Bipin (AND) Desai.

1.1.2 Google search engine

Table 3 shows the pages distribution of the test results of the Google search system using keywords Bipin (AND) Desai.

1.1.3 HotBot search engine

Table 4 shows the pages distribution of the test results of the HotBot search system using keywords Bipin (AND) Desai.

1.1.4 Lycos search engine

Table 5 shows the pages distribution of the test results of the Lycos search system using keywords Bipin (AND) Desai.

Pages Number	Number of Hits	Number of Duplicates	Number of Miss-Hits
page 1	2	6	2
page 2	7	1	2
page 3	7	0	3
page 4	7	1	2
page 5	3	0	7
page 6	3	0	7
page 7	5	0	5
page 8	5	0	5
page 9	3	0	7
page 10	4	0	6
page 11	6	0	4
page 12	5	2	3
page 13	5	1	4
page 14	6	2	2
page 15	6	2	2
page 16	3	7	0
page 17	5	4	1
page 18	4	4	2
page 19	7	3	0
page 20	6	1	3

Table 2: AltaVista Pages Distribution for searching on Bipin (AND) Desai

Pages Number	Number of Hits	Number of Duplicates	Number of Miss-Hits
page 1	8	1	1
page 2	8	1	1
page 3	5	1	4
page 4	7	0	3
page 5	4	0	6
page 6	5	1	4
page 7	8	0	2
page 8	5	1	4
page 9	8	0	2
page 10	8	1	1
page 11	9	0	1
page 12	7	0	3
page 13	6	0	4
page 14	6	0	4
page 15	2	1	7
page 16	7	0	3
page 17	5	0	5
page 18	1	1	8
page 19	6	1	3
page 20	9	1	0
page 21	6	0	4
page 22	3	0	7
page 23	0	0	10
page 24	0	0	10
page 25	0	0	10
page 26	0	0	10
page 27	5	0	5
page 28	2	0	8
page 29	0	0	10
page 30	1	0	9
page 31	1	0	9
page 32	0	0	10
page 33	0	0	10
page 34	0	0	10
page 35	0	0	10
page 36	0	0	10
page 37	1	0	9
page 38	0	0	10
page 39	1	0	9
page 40	1	0	9

Pages Number	Number of Hits	Number of Duplicates	Number of Miss-Hits
page 41	0	0	10
page 42	1	0	9
page 43	1	0	9
page 44	1	0	9
page 45	0	0	10
page 46	1	0	9
page 47	0	0	10
page 48	1	0	9
page 49	6	0	4
page 50	4	0	6
page 51	3	0	7
page 52	2	0	8
page 53	1	0	9
page 54	3	1	6
page 55	3	0	7
page 56	3	0	7
page 57	2	0	6

Table 3: Google Pages Distribution for searching on Bipin (AND) Desai

Pages Number	Number of Hits	Number of Duplicates	Number of Miss-Hits
page 1	8	1	1
page 2	3	2	5
page 3	2	0	8
page 4	2	1	7
page 5	3	1	6
page 6	5	1	4
page 7	3	0	7
page 8	6	1	3
page 9	3	1	6
page 10	4	0	6
page 11	3	0	7
page 12	2	0	8
page 13	2	0	8
page 14	3	1	6
page 15	4	5	1
page 16	5	2	3
page 17	0	0	10
page 18	1	5	4
page 19	3	0	7
page 20	0	0	10
page 21	0	0	4

Table 4: HotBot Pages Distribution for searching on Bipin (AND) Desai

Pages Number	Number of Hits	Number of Duplicates	Number of Miss-Hits
page 1	7	0	3
page 2	3	0	7
page 3	4	0	6
page 4	3	0	7
page 5	3	0	7
page 6	1	0	9
page 7	3	0	7
page 8	2	0	8
page 9	3	0	7
page 10	4	0	6
page 11	6	0	4
page 12	4	0	6
page 13	4	3	3
page 14	4	0	6
page 15	3	0	7
page 16	3	0	7
page 17	1	0	9
page 18	3	0	7
page 19	2	0	8
page 20	1	0	9
page 21	2	0	8
page 22	3	0	7
page 23	2	0	8
page 24	1	0	9
page 25	1	0	9
page 26	3	0	7
page 27	1	0	9
page 28	2	0	8
page 29	4	0	6
page 30	0	0	10
page 31	1	1	8
page 32	3	1	6
page 33	3	0	7
page 34	1	0	9
page 35	3	0	7
page 36	1	0	9
page 37	0	0	10
page 38	1	0	9
page 39	2	0	8
page 40	3	0	7

Pages Number	Number of Hits	Number of Duplicates	Number of Miss-Hits
page 41	1	0	9
page 42	1	0	9
page 43	1	0	9
page 44	0	0	10
page 45	3	1	6
page 46	2	2	6
page 47	1	0	9
page 48	2	0	8
page 49	3	1	6
page 50	4	0	6
page 51	0	0	10
page 52	2	2	6
page 53	3	0	7
page 54	2	0	8
page 55	1	0	9
page 56	0	0	10
page 57	1	0	9
page 58	1	0	9
page 59	0	0	10
page 60	1	0	9
page 61	1	1	8
page 62	0	0	10
page 63	1	1	8
page 64	0	0	10
page 65	0	0	10
page 66	1	0	9
page 67	5	3	2
page 68	5	3	2
page 69	4	4	2
page 70	9	1	0
page 71	2	0	8
page 72	4	0	6
page 73	5	0	5
page 74	1	3	6
page 75	1	1	8
page 76	4	2	4
page 77	4	0	6
page 78	6	0	4
page 79	1	0	9
page 80	4	1	5

Pages Number	Number of Hits	Number of Duplicates	Number of Miss-Hits
page 81	2	1	7
page 82	4	2	4
page 83	4	0	6
page 84	4	0	6
page 85	5	1	4
page 86	2	0	8
page 87	2	0	8
page 88	3	0	7
page 89	3	1	6
page 90	3	0	7
page 91	0	0	10
page 92	0	0	10
page 93	4	1	5
page 94	4	0	6
page 95	7	0	3
page 96	2	0	8
page 97	1	0	9
page 98	0	0	10
page 99	1	0	6

Table 5: Lycos Pages Distribution for searching on Bipin (AND) Desai

1.2 Proposed Solution

The problem with the current automatically generated index databases is their inadequate semantic information. This inadequacy precipitates a need for the design of meta-data to provide a template for describing information about resources. This meta-information is described in Chapter 2, Section 2.1.2.

CINDI (Concordia INdexing and DIscovery System), a system proposed by Desai et al [BS94], enables any resource provider to catalog his/her own resource and any user to search for hyper-media documents using typical search items such as Author, Title, Subject, etc. The system will offer a bibliographic database that provides information about documents available on the Internet. A standardized index scheme will be used to ensure homogeneity of the syntax and semantics of such an index. These index entries are stored in a database system (Semantic Header Database System). The CINDI system is based on a set of nodes connected to the Internet.

The proposed CINDI system requires a two-step process for its implementation. First a Semantic Header design is required to describe each information resource [BCD95], as well as, the design of distributed databases that store the Semantic Headers. Then, a registering system is also required to register the Semantic Header into the distributed Semantic Header databases, and a search system that allows users to enter a query based on multiple fields. Since the registering and search systems are to be carried out on the WWW, an Apache Web server is required.

1.3 Organization of the Thesis

This thesis describes the architecture of the CINDI system, the design and implementation of the distributed database system and the implementation of the interface between the distributed CINDI database system and the Web Browser. The formal analysis and implementation of the Semantic Header database is presented in Chapter 3.The Semantic Header is described in Section 3.2. The design of the Semantic Header database is given in figure 7.

In Chapter 4, we introduce the CINDI distributed database system. The implementation of the Oracle replicated database system is presented in Section 4.1. The administration of the replicated database system is discussed in Section 4.2. The web applications interface between the browser and the distributed databases are discussed in Chapter 4. The data flow and control flow of the CINDI system are covered in Chapter 5 for both main tasks of registration and search. Finally in Chapter 6 we draw our conclusion and one suggestion for future work.

Chapter 2

Background

2.1 Information Retrieval

Information retrieval is concerned with the representation, storage, organization and accessing of information. The first step in the retrieval process is for the user to state the information needed. This has to be done in a format that enables the information retrieval system to understand it and to act on it [FD95]. Indexing is the basis for retrieving documents that are relevant to the user's need [LJ96]. Building an accurate representation of a document, which would increase precision, is one of CINDI's main objectives.

2.1.1 SemiStructured Data

SemiStructured Data is often defined as "schemaless" or "self-describing", terms which indicate that there is no separate description of the type or structure of data.

XML

XML (eXtended Markup Language) is a new standard adopted by the World Wide Web Consortium (W3C) to complement HTML for data exchange on the Web [ABS00].

HTML (Hyper-Text Markup Language) was designed specifically to describe the presentation of data, not the content. XML was designed specifically to describe content rather than presentation. It differs from HTML in three major respects. One, new tags may be defined, and two, structures of the data contained in the document can be nested to arbitrary depth. The third major difference is that an XML document can contain an optional description of its grammar.

An example of an HTML file is shown in Figure 1.

```
<h1> Students on project </h1>
<p> <b> Smith </b>, 23 years, <i> smith@cs.concordia.ca </i> </p>
<p> <b> Robert </b>, 24 years, <i> robert@cs.concordia.ca </i> </p>
```

Figure 1: HTML presentation

In this particular HTML file, "Students on project" has been identified as title by enclosing it between <h1> and </h1> tags. For each individual student, the paragraph is introduced with a <p> tag. Each student name is put in bold between and tags. For the student's age, we have a default presentation. The email address is presented in italics using <i> tag.

An example of an XML file is given in figure 2. In the XML example, the title "Students on project" is described with a <description> tag, which is a new tag. For each student we enclose the data with a <person> tag. Each student name is described with a <name> tag, the age with an <age> tag and the email with an <email> tag.

The information in the HTML file is essentially the same as that in the XML file. Both describe two students working on some project, but while HTML describes the presentation, XML describes the content. From the previous example we can see that the basic XML syntax is perfectly suited for describing semistructured data.

```
<description> Students on project </description>
<student>
        <person>
                <name> Smith </name>
                <age>  23 </age>
                <email> smith@cs.concordia.ca </email>
        </person>
        <person>
                <name> Robert </name>
                <age>  24 </age>
                <email> robert@cs.concordia.ca </email>
        </person>
</student>
```

Figure 2: XML presentation

RDF

The Resource Description Framework (RDF) is a proposal for representing metadata in XML [ABS00]. Its intended application will provide better search engine capabilities in resource discovery, and enhance cataloging for describing the content and content relationships available at a particular Web site. It also allows intelligent software agents to share and exchange knowledge.

RDF consists of a data model and syntax. RDF's syntax is a convention for representing this in XML. For instance, the previous example would be represented in RDF as shown in Figure 3.

```
<rdf :description ID=  "&o1" >
     <person>
           <rdf: description ID ="&o2" >
                 <name> Smith </name>
                 <age>  23 </age>
                 <email> smith@cs.concordia.ca </email>
           </rdf :description>
     </person>
</rdf: description>
```

Figure 3: RDF presentation

The <rdf: description> element describes a resource. In this example we create two new resources and give them unique IDs (the ID attributes). Resource &o1 has property person whose value is resource &o2; Resource &o2 has property name with value Smith and property age with value 23 and property email with value smith@cs.concordia.ca

2.1.2 Meta-Data

Meta-Data is the information that records the characteristics and relationships of the source data. It helps provide succinct information about the source data which may not be recorded in the source itself due to its nature or an oversight [BCD90]. The first Meta-Data Invitational Workshop was held in March 1995 in Dublin. On September 9th, 1999, the Dublin Core Element Set, Version 1.1 was released [BCD95]. 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 to achieve a consensus on a set of core data elements for Document-Like Objects (DLO), and to map these and related elements to accepted standards, as well as devising an extension scheme for registering other types of network objects.

A number of assumptions were made to develop a consensus to arrive at a minimum set of core data elements. It was assumed that the elements are intended to describe a DLO (Document-Like Object) and that Common (or core) element sets are Meta-Data elements that apply to most/many DLOs. It was also understood that the elements of the core are chosen to support resource discovery, and that all elements are repeatable and optional, with the exception of the source element which can be thought of as a recursive instance of the entire record as it applies to an object from which an 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. The core element set assumes an arbitrary complex hierarchy, but elements not included in the core set are not specifically excluded. No assumption is made concerning whether the DLOs are networking accessible or specifically electronic.

From this workshop, numerous elements emerged as the ones required in a minimum set.

Element	Description
Subject	words or phrases indicative of the information content.
Title	name or short description of the DLO object.
Author	the name of the creator of the content.
Other-agent	the name of any other entity responsible for making the DLO object available.
Date	the date of publication.
Identifier	a character string or a number used to distinguish this DLO object from other objects.
Object-Type	conceptual description of the DLO object.
Form	physical, logical, or encoding characteristics.
Relation	important relationship to other DLO objects.
Language	natural language of the content of the DLO object.
Source	object from which derived; contains a nested DLO object description.
Coverage	characterizes parameters to specify the audience, or the time or space.

Table 6: Dublin Meta-Data Element List

They were named the Dublin Meta-Data Element List (DMEL) and are referred to in Table 6.

Semantic Header

A Meta-Data description called a Semantic Header is used to describe an information resource and is presented in Section 3.2.1.

2.2 Networked Databases

The proliferation of computer networks has enabled users to access a large number of data sources. This increased access to databases is likely to have a great practical impact; data and services can now be offered directly to customers in ways that were impossible until recently. This unprecedented access will lead to increased and novel demands upon DBMS (DataBase Management System) technology.

2.2.1 Distributed Databases

A distributed database is a set of databases stored on multiple sites that typically appears to applications as a single database. Consequently, an application can transparently access and modify the data in several databases in a network.

2.2.2 Database Replication

Replication is the process of copying and maintaining database objects in multiple databases that make up a distributed database system. While replication relies on distributed database technology to function, database replication can offer applications benefits that are not possible within a pure distributed database environment. Most commonly, replication is useful to improve the performance and protect the availability of applications because alternate data access options exist; so, if a site that contains a replica becomes inaccessible, we can find the same data at another site [RGR98]

2.2.3 Oracle's Distributed Database Architecture

Oracle offers the implementation of a distributed database. Each Oracle database in a distributed database system is controlled by its local Oracle server but cooperates to maintain the consistency of the global distributed database.

Clients and Servers

A database server is the Oracle software managing a database, and a client is an application that requests information from a server. Each computer in a system is a node. A node in a distributed database system acts as a client, a server, or both, depending on the situation.

The Network: Net8

To link the individual databases of a distributed database system, a network is necessary. All Oracle databases in a distributed database system use Oracle's networking software, Net8, to facilitate inter-database communication across a network. Net8 (as shown in figure 4)

connects clients and servers that operate on different computers of a network. It also allows database servers to communicate across networks to support remote and distributed transactions in a distributed database. The connectivity that is necessary to transmit Structured Query language (SQL) requests and receive data for applications that use the system, is made transparent through Net8. Net8 takes SQL statements from a client and packages them for transmission to an Oracle server over a supported industry-standard communication protocol. Net8 also takes replies from a server and packages them for transmission back to the appropriate client. Net8 performs all processing independent of an underlying network operating system.

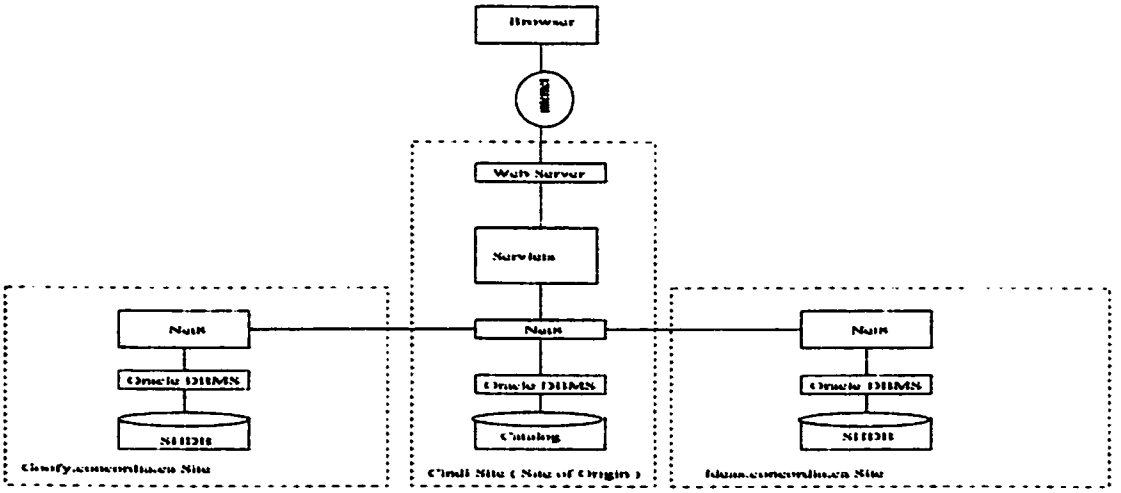

Figure 4: Architecture of the CINDI System

2.2.4 Oracle's Distributed Database Concepts

Schemas and Schema Objects

A schema is a collection of database objects that are available to a user. Schema objects are the logical structures that directly refer to the database's data. Schema objects include such structures as tables, views, sequences, stored procedures, synonyms, indexes, clusters and database links.

Tables

A table is the basic unit of data storage in an Oracle database. The tables of a database hold all of the user-accessible data.

Synonyms

A synonym is an alias for a table, view, sequence, or program unit. A synonym is not actually a schema object itself, but rather a direct reference to a schema object. Synonyms are used to mask the real name and owner of a schema object and provide public access to a schema object. They also provide location transparency for tables, views, or program units of a remote database and simplify the SQL statements for database users.

Database Links

Oracle uses *database links* to facilitate connections between the individual databases of a distributed system. A database link defines a *path* to a remote database by uniquely identifying and specifying the location of a remote database.

Schema Object Name Resolution

To resolve application references to schema objects (a process called name resolution) Oracle forms object names using a hierarchical approach. For example, within the single CINDI database in *goofy* site, Oracle guarantees that a schema "mechouet" has a unique name and that within a schema, the "Header" table object has a unique name too. As a result, a schema object's name "mechouet.header" is always unique within the database. Furthermore, Oracle can easily resolve application references to an object's local name. In a distributed database,

a schema object such as a table is accessible to all applications in the system. Oracle simply extends the hierarchical naming model with global database names to effectively create global object names and resolve references to the schema objects in a distributed database system.

Triggers

Oracle allows you to define procedures that are implicitly executed when an INSERT, UPDATE, or DELETE statement is issued against the associated table. These procedures are called database triggers. Triggers (one or more) are implicitly fired (executed) by Oracle when a triggering INSERT, UPDATE, or DELETE statement is issued, no matter which user is connected or which application is being used. Triggers are stored in the database separate from their associated tables. Triggers can be used, for example, to restrict Data Manipulation Language (DML) operations against a table to those issued during regular business hours. A trigger could also restrict DML operations to occur only at certain times during weekdays. Other uses for triggers are to automatically generate derived column values and prevent invalid transactions. Triggers also enforce complex security authorizations, referential integrity across nodes in a distributed database, and complex business rules. They provide transparent event logging, sophisticated auditing, and maintain synchronous table replicates, gathering statistics on table access.

2.2.5 World Wide Web

The Web makes it possible to access a file anywhere on the Internet using a URL. http://cindi.cs.concordia.ca/index.html is an example of URL that identifies a file called index.html, stored in the default home directory on cindi.cs.concordia.ca machine. This file is a formatted document using HTML and contains several links to other files (identified through their URLs). A browsing tool such as Netscape Navigator interprets the formatting commands to display the document in an attractive manner and allows the user to navigate to other related documents by choosing links. A collection of such documents on a given machine is called a Web site, and most organizations today maintain a Web site.

Audio, video, and even computer programs (written in Java, a highly portable language, and typically performing complex animations) can be included in HTML documents that could be automatically generated using visual editors. When a user retrieves such a document using

a suitable browser, images in the document are displayed, audio and video clips are played, and embedded programs are executed at the user's machine. The result is a rich multimedia presentation. The increased accessing of these HTML files using Internet browsers has fuelled the growth of the Web.

2.2.6 Web Servers and Databases

The Web is widely expected to be the cornerstone of electronic commerce. Many organizations already offer products through their Web sites, and customers can place orders by visiting a Web site. For such applications, a URL must identify more than just a file, however rich the contents of the file; a URL must provide an entry point to services based on information in a database. To understand how such an entry point can be supported, and the connection between the web and database systems, it is necessary to consider how Web sites are administered. As shown in figure 5, a Web server is a program that waits for URL requests at a given site. If the requested URL is a file name, the server returns a copy of the file. If the requested URL identifies a program to be executed at the server's site, the Web server creates a process to execute the program, and communicates with this process using the CGI (Common Gateway Interface) protocol. If an HTML document is a form, the form is returned to the requester (a user running an Internet browser). After the requester fills in the form, the form is returned to the Web server, and the information filled in by the user can be used as parameters to a program executing at the server. The results of the program can be used to create a customized HTML document that is returned to the requester.

The ability to use a URL to invoke a program at the server leads us to the database connection. The program invoked by the Web server can generate a request to a database system. This capability allows us to easily place a database on a network, and make services that rely upon database access available over the Web. Applications that access a database through the Web are likely to become very common. The CGI protocol, which creates one process per request, is inefficient and cannot deal with a large number of requests. Alternative protocols, in which the program invoked by a request is executed within the Web server process, have been proposed by Microsoft ISAPI (Internet Server API) and by Netscape NSAPI (Netscape Server API). Indeed, the TPC-C benchmark has been executed, with good results, by sending requests from 1500 PC clients to a Web server, and through it to an SQL database server [RGR98].

2.2.7 Web Interfaces to Databases

Interfacing databases to the World Wide Web is important for two reasons. First, with the growth of electronic commerce on the Web, databases used for transaction processing must be linked with the Web. The HTML forms interface is convenient for transaction processing. The user can fill in details on an order form, and can click a submit button to send a message to the server to process the information that has been filled in. A program corresponding to the order form is executed, in turn executing a transaction on a database at the server site. The results of the transaction can be formatted into HTML and displayed to the user.

The second reason for connecting a database to the Web is that fixed HTML sources for display to users have limitations. One limitation is that the use of fixed Web documents does not allow the display to be tailored to the user data. Also, when the data is updated, the Web documents become outdated if they are not updated simultaneously. The problem becomes more acute if multiple Web documents replicate important data and all must be updated.

We can fix these problems by generating Web documents dynamically. When a document is requested, a program can be executed at the server site, which runs queries on the database and generates a document based on the query results. The Web document displayed can be tailored to the user based on user information stored in the database. Data in Web documents can also be defined by queries on a database, so that whenever relevant data in the database are updated, the Web documents will be updated, too.

2.3 Modeling Web Applications with UML

There is a subtle distinction between a web application and a web site. A web application contains a web site where user input (navigation through the site and data entry) affects the state of the business (beyond, of course, access logs and hit counters). In essence, a web application, as presented in figure 5, uses a web site as the front end to a more typical application. The architecture for a web site is rather straightforward and contains three principal components: a web server, a network connection, and one or more client browsers. The web server distributes pages of formatted information to clients that request it. The

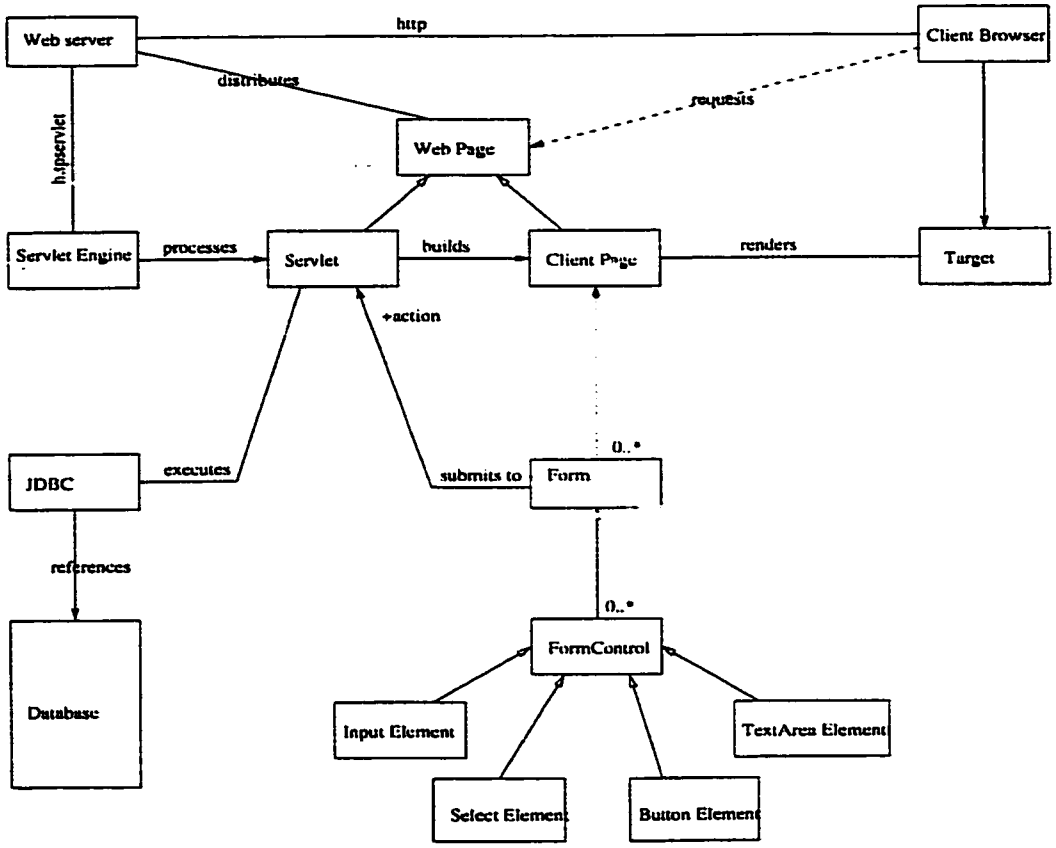

Figure 5: UML Model of Web Application Architecture

request is made over a network connection and uses the HTTP protocol. Some web sites require clients to login, and some allow anonymous access.

2.3.1 Java Servlet Applications

A Servlet is a program that runs within a Web server and performs actions in response to HTTP requests. As shown in figure 5, in the HTTP based request-response paradigm, a client user agent (a web browser or any such application that can make HTTP requests and receive HTTP responses) establishes a connection with a web server and sends a request to the server. If the web server has a mapping to a Servlet for the specified URL in the request, the web server delegates the request to the specified Servlet. The Servlet, in turn, processes the request and generates an HTTP response. Servlets allow us to write Java programs that drives the back end of our Web site. To run a Servlet, our Web server must have a *Servlet engine.*

2.3.1.1 HTTP Request

The interface HttpServletRequest is the first abstraction provided by the Servlet API. This interface encapsulates HTTP request from a user agent. When the Servlet engine receives a request, an object of this type is constructed and passed on to a Servlet. This object provides methods for accessing parameter names and values of the request, other attributes of the request, and an input stream containing the body of the request.

2.3.1.2 HTTP Response

The HttpServletResponse interface of the Servlet API provides an encapsulation of the HTTP response generated by a Servlet. This interface defines an object created by the Servlet engine that lets a Servlet generate content in response to a user agent's request. This object provides methods to set type and length of the content, character encoding, response status including errors, and an output stream into which binary response data may be written. Alternatively, this object also provides a print writer object for writing formatted text responses.

2.3.1.3 Application Logic and Content Generation

The HttpServlet interface specifies methods for implementing the application logic and generating content in response to an HTTP request. These methods handle the GET and POST requests of HTTP. The GET method is designed for getting information (a document, a chart, or the results from a database query), while the POST method is designed for posting information (a credit card number, some new chart data, or information that is to be stored in a database) [HC98]. The GET method, although it is designed for reading information, can include as part of the request some of its own information that better describes what to get (such as an x,y scale for a dynamically created chart. This information is passed as a sequence of characters appended to the request URL in what is called a query string. The POST method uses a dufferent technique to send information to the server because in some cases it may need to send megabytes of information. A POST request passes all its data, of unlimited length, directly over the socket connection as part of its HTTP request body. The exchange is invisible to the client.

2.3.2 HTML Pages

By far the most fundamental component of a web application is the page. Browsers request pages (or conceptual pages) from servers. Web servers distribute pages of information to browsers. In essence, the organization of a web page makes up the user interface for the application. In web applications, the browser acts as a generalized user interface container with specific user interfaces being defined by each page.

2.3.3 HTML Forms

Any serious web application accepts more than navigation input from its users. Web applications, as shown in figure 5, often elicit textual, selectable and Boolean information. The most common mechanism for collecting this type of user input is with HTML forms. An HTML form is a collection of input fields that are rendered in a Web page. The basic input elements are: a text-box, text-area, check-box, radio button group and a selection list.

Name or ID identifies all the input elements on a form. Each form is associated with an action page. This action page represents the name (and location) of the page that is to receive and process the information contained in the completed form. The action page is almost always a dynamic page, containing server side scripts (or compiled code). When a user completes a form, the user submits the form back to the server with a page request for the action page. The web server finds the page and interprets (or executes) the page code. The code in the page has the ability to access any information in the form submitted with the request, and is the major mechanism for obtaining user input in a web application.

2.3.4 JDBC Servlet Applications

A CGI database script is an external program run by the Web Server to access the database and create output in the form of an HTML document, which is then presented on a web client. This is a proven architecture but has limitations. These limitations can create significant problems when trying to develop enterprise-based web applications. There is a new architecture that not only solves these problems, but also gives code portability and the ability to allow the server-side application to interface with a wide range of relational databases. That architecture is using Java DataBase Connectivity (JDBC) with Java Servlets to replace CGI. JDBC (Java version of Open DataBase Connectivity (ODBC)) is a Java based API that is now a standard part of Java and is included in the Java Development Kit. Servlets are the server side Java counterpart to applets without a graphical user interface. They are protocol and platform-independent components, which extend Java-enabled web servers (e.g., Apache, Netscape, and IIS). Servlets maintain a connected state, can use a standard database API (JDBC) and have a significant performance increase because they have no heavy process startup and initialization for each client request like CGI.

2.3.4.1 Maintaining State and Sessions with Servlets

A typical CGI architecture uses cookies (a cookie is a bit of information sent by a Web server to a browser that can later be read back from that browser) on either the client or server (or both) to maintain some sense of state or session (a session is a series of requests from the same client that occur during a given time period). This cookie technique, however, does not solve the problem of keeping the connection "alive" between the CGI application and

the database. We are still required to re-establish or maintain a connection during a client session. This is required because CGI applications are forked processes, which are spawned for each client request, then die. Servlets, on the other hand, can maintain state and session identity because they are persistent and all client requests are processed until the servlet is shut down by the web server, or explicitly through a destroy method. A servlet technique used here to maintain state/session is through creating a threaded session class or object (there is a similar feature also available in Sun's Java Web Server 1.1). Each client request is stored and maintained in the servlet. When a client first makes a request, the client is assigned a new session object and a unique session ID. These values are stored in a servlet hash table. When the client issues another request, the session ID is passed and the session object information is retrieved to re-establish session state.

2.3.4.2 Servlet Performance Advantages over CGI

The most important performance feature of Servlets is that they do not require the creation of a new process for each request. Servlets support threads, so there can be one Servlet invocation to support multiple clients. In the web server environment, Servlets can run in parallel within the same process as the server, an arrangement which provides significant performance advantages over CGI. This is because Servlets only require lightweight thread context switches. CGI requires heavier weight process startup and initialization code on each request. Servlets contain an initialization method that allows for "expensive" processes such as database connections to be performed only once at Servlet start-up time. This means that all of the client requests to a Servlet can share global resources and take advantage of caching.

2.3.4.3 JDBC Standard

JDBC's main advantage is that we can develop an application (in Java) that can inter-operate with multiple data-sources. JDBC has the same advantages and disadvantages as ODBC and can be easily incorporated into a Servlet. Use of JDBC means that pure Java programs do not have to call native methods to access relational data. This means that it is possible for an organization, which has a large variety of hardware and a centralized relational database management system, to move very easily to a client-server implementation. The

same program can run on all the various client platforms against data on a central server. Virtually all commercial relational databases support an SQL interface. JDBC supports an underlying SQL implementation. Clearly not all DBMS products are the same. Nor do they all implement the same dialect of SQL.

JDBC uses a two level architecture to hide differences in the underlying DBMS platforms from the programmer. The *JDBC API* and *JDBC Driver API* represent the *two* level architecture. Using JDBC application (or applet) makes use of classes in the JDBC API. The JDBC classes basically consist of a set of interfaces (that reside in the java.sql package). The JDBC API interfaces are useless on their own. In order to make use of them, we need a JDBC driver. A JDBC driver is a set of classes that implement all of the methods present in the JDBC API interfaces. Since each driver exposes only those methods defined in the JDBC API interfaces, every driver looks exactly the same from the Java programmer's standpoint. The JDBC is heavily based on the ANSI SQL-92 standard, which specifies that a JDBC Driver should be SQL-92 entry-level compliant to be considered a one-hundred percent JDBC-compliant driver. The JDBC API communicates with a JDBC driver, whereas the JDBC API is database independent (and identical across all hardware/software platforms), and the driver is not. The driver communicates directly with a relational database product.

2.3.4.4 ODBC's Part in the JDBC

The JDBC and ODBC share a common parent. Both are based on the same X/OPEN call level interface for SQL. Though there are JDBC drivers emerging for many databases, we can write database-aware Java programs using existing ODBC drivers [PK97]. In fact, Javasoft and Intersolv have written a JDBC driver, the JDBC-ODBC bridge, as shown in Figure 6, that allows developers to use existing ODBC drivers in Java Programs.

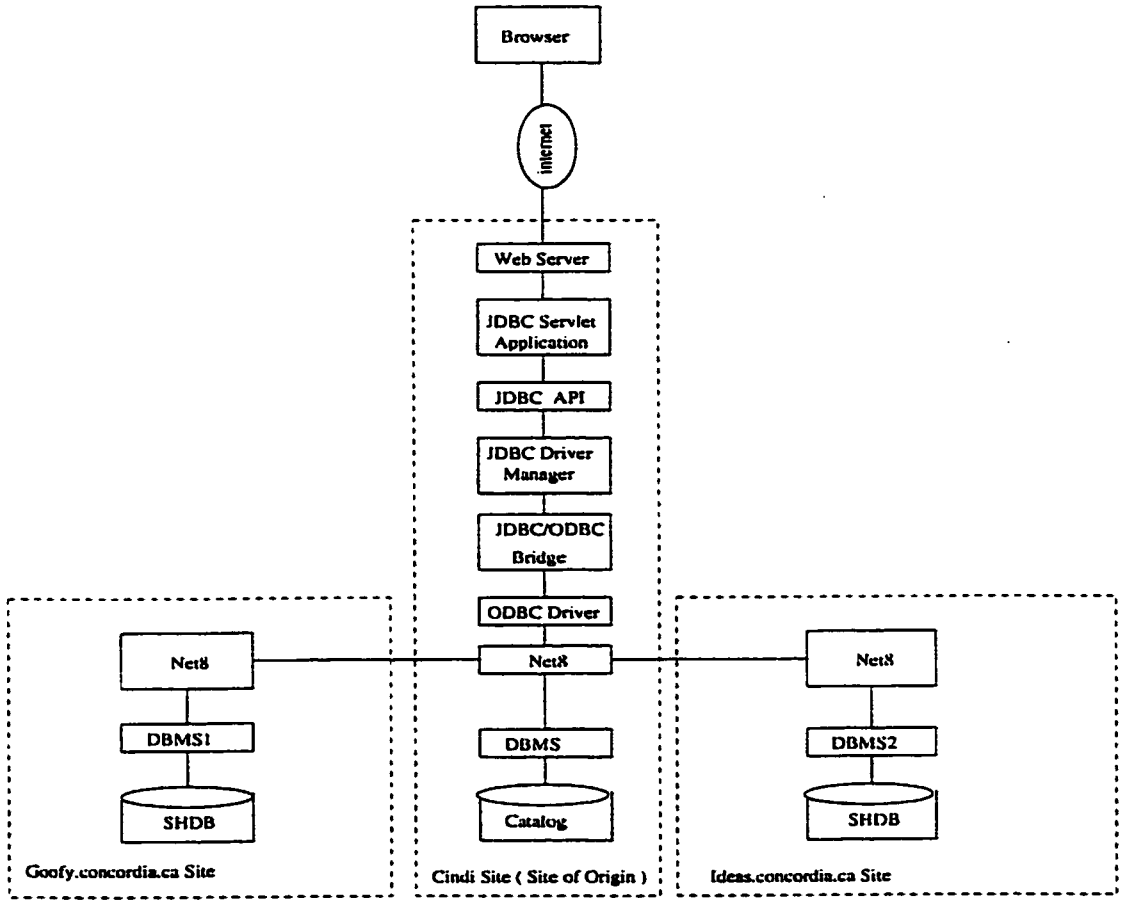

Figure 6: Detailed Architecture of the CINDI System

Chapter 3

Database Design of the CINDI System

3.1 UML Design

With the advent of intricately complex systems, a clear and concise way of representing them visually has become increasingly important. The Unified Modelling language (UML) was developed by Grady Booch, Jim Rumbaugh and Ivar Jacobson in response to that need [PJ99]. UML has been chosen to model the database of the CINDI system despite its derivation from a tradition of object programmers and not data modellers. The reason for this choice is that UML is a richer language for describing relationships between entities, and is therefore able to model all of the relationships that Entity Relationships Diagrams (ERDs) do. In addition, UML is able to describe relationships that ERDs cannot.

Some relationships have different meanings from other relationships. For example, the relationship between a Semantic Header and its Semantic Header Detail, which is described in section 3.2.3.7, is called "Close Association". UML provides two kinds of close association: aggregation and composition, which are new relationships for ER modellers.

3.2 Semantic Header Specification

The identification of the correct classes and the relationships between them is the essence of the data model, which is shown in figure 7. In order to create the CINDI database, classes must be found, identified and described using the following method:

3.2.1 Identifying Classes and their Attributes

For cataloging and searching, a Meta-Data description called a Semantic Header is used to describe an information resource [BS94]. 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], the entry of these elements is mandatory in the Semantic Header. The abstract and annotation as well, are relevant in deciding whether or not the resource would be useful; these items are also included. The elements of the Semantic Header are briefly described below:

3.2.1.1 Semantic Header Class

Title, Alt_title:

The first field of the Semantic Header is the title (title for non-document like resources may require some creativity. For example, the title of a satellite image could be generated from the name of the satellite, its location, date, time, etc.) of the resource. This title 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.

Language, Character Set:

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

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

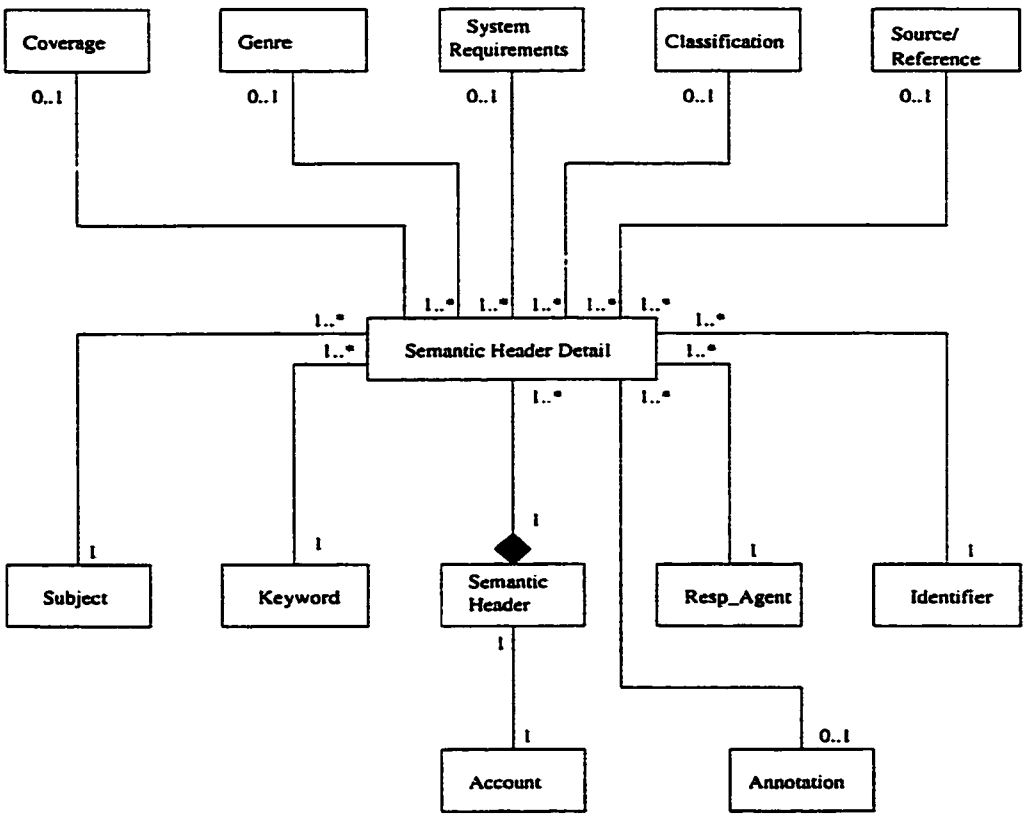

Figure 7: UML Diagram of the Semantic Header Database

Version:

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

Abstract:

The abstract is an optional element given in the next field. The abstract is provided by the author of the resource.

3.2.1.2 Subject Class

The subject and sub-subjects of the resource are indicated in the next field which is a repeating group (a multi-part field with one or more occurrences of items in the group). A subject of the resource is composed of three sub-levels: general, sub-level1 and sub-level2. All resources must have at least one occurrence for this field.

3.2.1.3 Resp_Agent Class

The details about the author(s) and/or other agent(s) responsible for the resource is given in the next repeating group (for resources such as satellite image, the agent may be the agency controlling the satellite or the satellite itself). The sub-fields are: role of the agent (typical values for role of the agent could be author, co-author, designer, editor, programmer, creator, artist, corporate entity, publisher, etc), name, organization, address, phone number, fax number and e-mail address.

3.2.1.4 Keyword Class

The list of keywords is included in this field. At least one keyword is required for this field.

3.2.1.5 Identifier Class

The next element is a repeating group for recording the identifiers of the resource. Each occurrence of this group consists of two sub-fields: one for the domain and the other for the corresponding value. 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) [BLTC], or URN [SOLL] etc., and the value contains the corresponding coded identifier. Since a resource in an 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 URN name may be used instead of an URL location if the item is likely to move or is accessible from multiple locations (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). The identifier(s) can be used to locate the resource.

3.2.1.6 Classification Class

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.

3.2.1.7 Coverage Class

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.

3.2.1.8 System Requirements Class

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.

3.2.1.9 Genre Class

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.

3.2.1.10 Source/Reference Class

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

3.2.1.11 Annotation Class

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. The annotations are optional elements. The sub-fields are name, organization, address, phone number, fax number, e-mail address and text (the annotation made by the author and/or independent users of the resource).

3.2.1.12 Account Class

The last two items in the Semantic Header are the User ID and the Password. Any update to the changeable part of the Semantic Header requires these fields to be filled in. Each User

ID is assigned one, and only one, password.

Physical implementation of Classes

As introduced in section 2.3.4.3, JDBC is used to access relational databases. In Oracle's relational architecture, classes and their attributes evolve into tables and columns. Sequence numbers are used to generate unique numbers for numeric columns of the CINDI database tables. Figure 8 shows the oracle implementation of our model.

```
create sequence count_header increnment by 1;
create sequence count_subject increnment by 1;
create sequence count_resp_agent increnment by 1;
create sequence count_keyword increnment by 1;
create sequence count_ident increnment by 1;
create sequence count_clasf increnment by 1;
create sequence count_cover increnment by 1;
create sequence count_sys_req increnment by 1;
create sequence count_genre increnment by 1;
create sequence count_source increnment by 1;
create sequence count_annotate increnment by 1;
create sequence count_account increnment by 1;
create table header (
        header_ID               number(10)      NOT NULL,
        title                   varchar2(200)   NOT NULL,
        alt_title               varchar2(200),
        character               varchar2(200),
        language                varchar2(200)   NOT NULL,
        date_created            date            NOT NULL,
        date_expiry             date,
        date_updated            date,
        version                 varchar2(10)    NOT NULL,
        abstract                varchar2(2000),
        account_ID              number(10)      NOT NULL );
create table subject (
        subject_ID              number(10)      NOT NULL,
        general                 varchar2(200)   NOT NULL,
        level1                  varchar2(200)   NOT NULL,
```

```
        level2                  varchar2(200)   NOT NULL );
create table resp_agent (
        agent_ID                number(10)      NOT NULL,
        role_name               varchar2(200)   NOT NULL,
        name                    varchar2(200)   NOT NULL,
        organization            varchar2(200),
        address                 varchar2(200),
        phone                   varchar2(200),
        fax                     varchar2(200),
        email                   varchar2(200) );
create table keyword (
        keyword_ID              number(10)      NOT NULL,
        value                   varchar2(200)   NOT NULL );
create table identifier (
        ident_ID                number(10)      NOT NULL,
        domain                  varchar2(200)   NOT NULL,
        value                   varchar2(200)   NOT NULL );
create table classification (
        clasf_ID                number(10)      NOT NULL,
        domain                  varchar2(200)   NOT NULL,
        value                   varchar2(200) );
create table coverage (
        cover_ID                number(10)      NOT NULL,
        domain                  varchar2(200)   NOT NULL,
        value                   varchar2(200) );
create table sys_req (
        req_ID                  number(10)     NOT NULL,
        component               varchar2(200)  NOT NULL,
        value                   VARCHAR2(200) );
create table genre (
        genre_ID                number(10)     NOT NULL,
        form                    varchar2(200),
        size_bytes              varchar2(200) );
create table source (
        source_ID               number(10)     NOT NULL,
        relationship            varchar2(200),
        domain                  varchar2(200) );
```

```
create table annotation (
        annotate_ID             number(10)      NOT NULL,
        name                    varchar2(200),
        organization            varchar2(200),
        address                 varchar2(200),
        phone                   varchar2(200),
        email                   varchar2(200),
        text                    varchar2(2000) );
create table account (
        account_ID              number(10)      NOT NULL,
        user_ID                 varchar2(200)   NOT NULL,
        password                varchar2(200)   NOT NULL
        header_ID               number(10)      NOT NULL );
create table header_DTL (
        header_ID               number(10)      NOT NULL,
        subject_ID              number(10)      NOT NULL,
        keyword_ID              number(10)      NOT NULL,
        agent_ID                number(10)      NOT NULL,
        ident_ID                number(10)      NOT NULL,
        clasf_ID                number(10)      NULL,
        cover_ID                number(10)      NULL,
        genre_ID                number(10)      NULL,
        req_ID                  number(10)      NULL,
        source_ID               number(10)      NULL,
        annotate_ID             number(10)      NULL,
        CONSTRAINT      header_DTL_ID_pk
        PRIMARY KEY(header_ID, subject_ID, ident_ID, keyword_ID, agent_ID)
) ORGANIZATION INDEX;
```

Figure 8: Semantic Header Database implementation

3.2.2 Identifying Primary Keys

A primary key is the combination of the values of one or more attributes that collectively and uniquely identify an object in a class. Each component of a primary key must never be null. Because of physical implementation issues, primary keys must never change.

Physical Implementation of Primary Keys

Figure 9 shows the oracle implementation of Primary Keys.

```
ALTER TABLE header
ADD CONSTRAINT header_ID_pk
PRIMARY KEY(header_ID);
ALTER TABLE subject
ADD CONSTRAINT subject_ID_pk
PRIMARY KEY(subject_ID);
ALTER TABLE resp_agent
ADD CONSTRAINT agent_ID_pk
PRIMARY KEY(agent_ID);
ALTER TABLE keyword
ADD CONSTRAINT keyword_ID_pk
PRIMARY KEY(keyword_ID);
ALTER TABLE identifier
ADD CONSTRAINT ident_ID_pk
PRIMARY KEY(ident_ID);
ALTER TABLE classification
ADD CONSTRAINT clasf_ID_pk
PRIMARY KEY(clasf_ID);
ALTER TABLE coverage
ADD CONSTRAINT cover_ID_pk
PRIMARY KEY(cover_ID);
```

```
ALTER TABLE coverage
ADD CONSTRAINT cover_ID_pk
PRIMARY KEY(cover_ID);
ALTER TABLE sys_req
ADD CONSTRAINT req_ID_pk
PRIMARY KEY(req_ID);
ALTER TABLE genre
ADD CONSTRAINT genre_ID_pk
PRIMARY KEY(genre_ID);
ALTER TABLE source
ADD CONSTRAINT source_ID_pk
PRIMARY KEY(source_ID);
ALTER TABLE annotation
ADD CONSTRAINT annotate_ID_pk
PRIMARY KEY(annotate_ID);
ALTER TABLE account
ADD CONSTRAINT account_ID_pk
PRIMARY KEY(account_ID);
```

Figure 9: Primary Keys implementation

3.2.3 Identifying Relationships

Just as with ER diagramming, in object modelling it is crucial to identify and construct appropriate interactions or relationships between classes. The following are the types of relationships discovered between classes:

3.2.3.1 Association Relationship

Association is one of the most common relationship types encountered. Using an association relationship between two classes does not indicate how the classes interact, but merely represents the fact that these two classes have something to do with each other. This does not say anything about the type of relationship, but simply that one exists.

Implementation of associations

An association relationship is the means by which Oracle enforces data integrity between a child and a parent table. These relationships are physically implemented through the use of foreign key constraints as shown in Figure 10. Foreign Keys of child tables make reference to the primary key, or unique identifier of the parent table.

```
ALTER TABLE header_DTL
ADD CONSTRAINT header_fk
FOREIGN KEY (header_ID) REFERENCES header(header_ID);
ALTER TABLE header_DTL
ADD CONSTRAINT subject_fk
FOREIGN KEY (subject_ID) REFERENCES subject(subject_ID);
ALTER TABLE header_DTL
ADD CONSTRAINT agent_fk
FOREIGN KEY (agent_ID) REFERENCES resp_agent(agent_ID);
ALTER TABLE header_DTL
ADD CONSTRAINT keyword_fk
FOREIGN KEY (keyword_ID) REFERENCES keyword(keyword_ID);
ALTER TABLE header_DTL
ADD CONSTRAINT ident_fk
FOREIGN KEY (ident_ID) REFERENCES identifier(ident_ID);
ALTER TABLE header_DTL
ADD CONSTRAINT clasf_fk
FOREIGN KEY (clasf_ID) REFERENCES classification(clasf_ID);
ALTER TABLE header_DTL
ADD CONSTRAINT cover_fk
FOREIGN KEY (cover_ID) REFERENCES coverage(cover_ID);
ALTER TABLE header_DTL
ADD CONSTRAINT requir_fk
FOREIGN KEY (req_ID) REFERENCES sys_req(req_ID);
```

```
ALTER TABLE header_DTL
ADD CONSTRAINT genre_fk
FOREIGN KEY (genre_ID) REFERENCES genre(genre_ID);
ALTER TABLE header_DTL
ADD CONSTRAINT source_fk
FOREIGN KEY (source_ID) REFERENCES source(source_ID);
ALTER TABLE header_DTL
ADD CONSTRAINT annotate_fk
FOREIGN KEY (annotate_ID) REFERENCES annotation(annotate_ID);
```

Figure 10: Foreign Keys implementation

Cardinality

Cardinality refers to creating association sets that relate one or more objects in the first class to one or more objects in the second class. In UML, a number placed on each side of the relationship represents the cardinality associated with objects in that class. Allowable values are single integers (for example, "1"), a list of integers (for example, "0,1,2"), a range of integers (for example, "0 .. 2"), or a combination (for example, "2,4,6,10 .. 20"). However, such complex cardinality rules are extremely rare. To represent that any number of objects are allowed in the association, the symbol "*" is used. "*" refers to any nonnegative integer. "1" refers to any positive integer.

Optional and Mandatory Relationships

One of the common confusions with UML diagramming is whether to place a "1" or a "0..1" on a given side of a relationship. The "1" indicates that every object must be involved in the relationship where "0..1" indicates that every object may be involved in the relationship. There is a similar confusion about when to use "*" (meaning 0 to N) and a "1..*". "*" alone means that not necessarily every object is involved in the relationship. "1..*" means that the relationship is mandatory.

As shown in figure 7, a Semantic Header must always have at least one Semantic Header Detail, so the cardinality on the Semantic Header Detail side would be "1..*". Since each Semantic Header Detail must be associated with a Semantic Header, a 1 is placed next to the Semantic Header, because every Semantic Header Detail is associated with exactly one Semantic Header.

3.2.3.2 One-to-One Relationship, mandatory on both sides

Each Semantic Header must have an account which contains a UserID and Password as attributes. The purpose of this design is to protect certain header data from unwanted access.

Relational Implementation

This relationship (as shown in figure 11) is implemented by creating two tables, HEADER

and ACCOUNT, and are related to each other through the use of two foreign keys, one in each table referring to the other. To make the two tables entirely dependent upon one another, a circular reference has been created between them.

```
ALTER TABLE header
ADD CONSTRAINT account_ID_fk
FOREIGN KEY (account_ID) REFERENCES account(account_ID);
ALTER TABLE account
ADD CONSTRAINT header_ID_fk
FOREIGN KEY (header_ID) REFERENCES header(header_ID);
```

Figure 11: One-to-One Relationship, mandatory on both sides implementation

3.2.3.3 One-to-Many Relationship, optional on both sides

Each Semantic Header could have any number of Coverage, Genre, System Requirements,Classification, Source/Reference or Annotation.

Relational Implementation

The key to implementing an optional One-to-Many relationship is to declare the columns in the dependent table that form the primary key to be NULL (as shown in figures 12 and 8). This can be observed via the COVER_ID column in the HEADER_DTL table. The COVER_ID column of the HEADER_DTL table makes references to the COVERAGE table via the COVER_ID foreign key.

```
ALTER TABLE header_DTL
ADD CONSTRAINT cover_fk
FOREIGN KEY (cover_ID) REFERENCES coverage(cover_ID);
```

Figure 12: One-to-Many Relationship, optional on both sides implementation

3.2.3.4 One-to-Many Relationship, mandatory on both sides

Each Semantic Header must have at least one Subject, one Keyword, one Resp_Agent and one Identifier.

Relational Implementation

The key to implementing an optional One-to-Many relationship is to declare the columns in the dependent table that form the primary key to be NOT NULL (as shown in figure 13 and 8). This can be observed via the SUBJECT_ID column in the HEADER_DTL table. The SUBJECT_ID column of the HEADER_DTL table makes references to the SUBJECT table via the SUBJECT_ID foreign key.

```
ALTER TABLE header_DTL
ADD CONSTRAINT subject_fk
FOREIGN KEY (subject_ID) REFERENCES subject(subject_ID);
```

Figure 13: One-to-Many Relationship, mandatory on both sides implementation

3.2.3.5 N-ary Relationship

A Semantic Header Detail is a result of the relationship among Subject, Keyword, Semantic Header, Resp_agent, and Identifier.

Relational Implementation

N-ary relationship (as shown in figure 8) is implemented as a series of NOT NULL foreign keys (subject_ID, keyword_ID, header_ID, agent_ID and ident_ID) to an intersection table HEADER_DTL.

3.2.3.6 Or Relationship

A Semantic Header Detail may be a result of the relationship among Coverage, Genre, System Requirement, Classification, Source/Reference, Annotation, or neither.

Relational Implementation

Or relationship, as shown in figure 8, is implemented as a series of NULL foreign keys (cover_ID, clasf_ID, genre_ID, req_ID, source_ID and annotate_ID) to an intersection table HEADER_DTL.

3.2.3.7 Composition Relationship

A Semantic Header object is said to be a composition of Semantic Header Detail object since each Semantic Header Detail object is a part of a Semantic Header object. When considering the composition relationship, there are rules that are enforced. Semantic Header Detail objects may not exist unless they are part of a specific Semantic Header object. Similarly, Semantic Header Detail objects may not exist independently; they must always belong to one and only one Semantic Header object and they have no independent meaning apart from that Semantic Header object. Finally, Semantic Header Detail object may not be a composition child of more than one object at a time; it is from one specific Semantic Header object.

Relational Implementation

Oracle supports a form of table organization called Index Organized Tables (IOTs). In IOTs, the table data is organized as an index on the primary key of the table. That is, in this table, rows are physically clustered (and ordered) on the primary key. This can be quite useful in modelling composition (parent-child relationships). The Header Detail table is organized as an Index Organized Table with primary key as (header_ID, subject_ID, ident_ID, keyword_ID, agent_ID). This guarantees that all the lines of a Header (for a given header_ID) will be physically clustered as shown in Figure 8.

Chapter 4

Architecture of the CINDI System

The CINDI system

The CINDI (Concordia INdexing and DIscovery) System is a system proposed by Desai et al [BS94]. The objective of the system is to enable any resource provider to catalog his/her own resource and any user to search for hyper-media documents using typical search items such as Author, Title, Subject, etc. The system will offer a bibliographical database that provides information about documents available on the Internet. A standardized index scheme will be used to ensure homogeneity of the syntax and semantics of such an index. These index entries are stored in a distributed Semantic Header Databases System. The system is based on a set of nodes connected to the Internet, each node containing a Semantic Header database.

4.1 Distributed Databases Management

4.1.1 Managing Global Database Names

In a distributed database system, each ORACLE database must have a unique global database name so those objects within the distributed database can be uniquely identified.

A global database comprises two parts:

1. A name component (used for local administrative operations such as startup, recovery operations and shutdown).

2. A network domain component (used to indicate the database's location within a network structure).

In other words, $GlobalDatabaseName = InstanceName + HostName$. Each database in a distributed database is distinct from all other databases in the system and has its own global database name. Oracle forms a database's global database name by prefixing the database's network domain with the individual database's name. To manage the CINDI distributed databases on the two different network domains, GOOFY.CONCORDIA.CA and IDEAS.CONCORDIA.CA, the following SQL statements were created:

```
On GOOFY.CONCORDIA.CA  Site:
============================
SQL> ALTER DATABASE RENAME GLOBAL_NAME TO cind.goofy.concordia.ca;

On IDEAS.CONCORDIA.CA  Site:
============================
SQL> ALTER DATABASE RENAME GLOBAL_NAME TO cind.ideas.concordia.ca;
```

Figure 14: Global Names creation

After running these two SQL statements, the following two databases' global names will be created:

- CIND.GOOFY.CONCORDIA.CA
- CIND.IDEAS.CONCORDIA.CA

4.1.2 Transparency in a Distributed Database System

The goal of transparency is to make a distributed database system appear as though it is a single oracle database. Consequently, the system does not burden developers and users with complexities that would otherwise make distributed database application development challenging and detract from user productivity. The following sections explain more about transparency in a distributed database system.

4.1.2.1 Location Transparency

Oracle server provides the means to make data objects such as tables in remote databases appear to an application developer or user of that data object as if they are in the local database. This is called *location transparency.* An Oracle distributed database system has features that allow application developers and administrators to hide the physical location of database objects from applications and users. Location transparency exists when a user can universally refer to a database object such as a table, regardless of the node to which an application connects.

Location transparency has several benefits, including the simplicity of access to remote data, because database users do not need to know the physical location of database objects and administrators can move database objects with no impact on end-users or existing database applications.

4.1.2.2 Enforcing Location Transparency

In the CINDI system, synonyms are used to establish location transparency for the tables and supporting objects in an application schema. The following statements create a synonym in CIND.GOOFY.CONCORDIA.CA database for the *Header* table that resides in CIND.IDEAS.CONCORDIA.CA remote database:

```
CREATE PUBLIC SYNONYM header2
FOR cindidba.header@cind.ideas.concordia.ca
```

Figure 15: Synonym creation

Now, rather than access the remote header table with a query such as:

```
SELECT title, alt_title
FROM   cindidba.header@cind.ideas.concordia.ca h
WHERE  h.header_ID = 10;
```

Figure 16: Remote Query

An application can issue a much simpler query that does not have to account for the location of the remote header table.

```
SELECT title, alt_title
FROM   header2 h
WHERE  h.header_ID = 10;
```

Figure 17: Simple Remote Query

4.1.3 Managing the Database Links

Database links are essentially transparent to the users of an Oracle distributed databases system, because the name of a database link is the same as the global name of the database to which the link points. A database link is created at the GOOFY.CONCORDIA.CA site to allow the dynamic replication of the same Database at the IDEAS.CONCORDIA.CA site, since in the CINDI system, the Semantic Header database in the GOOFY.CONCORDIA.CA will be replicated at the IDEAS.CONCORDIA.CA site. The following SQL statement creates a database link in the CIND.GOOFY.CONCORDIA.CA database that describes a path to the remote CIND.IDEAS.CONCORDIA.CA database.

```
On GOOFY.CONCORDIA.CA  Site:
============================
SQL> CREATE PUBLIC DATABASE LINK  cind.ideas.concordia.ca
     CONNECT TO cindidba  IDENTIFIED BY ********
     using 'ideaslocation';

Where : ideaslocation is a Service Name created on GOOFY.CONCORDIA.CA site
        using Oracle Net8 EASY CONFIG and described in figure 19
```

Figure 18: Database Link creation

```
# D:\ORANT\NET80\ADMIN\TNSNAMES.ORA
# Configuration File:D:\orant\net80\admin\tnsnames.ora
# Generated by Oracle Net8 Assistant

IDEASLOCATION.WORLD =
  ( DESCRIPTION =
    ( ADDRESS = ( PROTOCOL = TCP )
        ( HOST = ideas.concordia.ca )
        ( PORT = 1521 )
    )
    ( CONNECT_DATA = ( SID = CIND ) )
  )
```

Figure 19: Service Name creation

After creating a database link, applications connected to the CIND.GOOFY.CONCORDIA.CA database can access data in the remote CIND.IDEAS.CONCORDIA.CA database.

4.1.4 Triggers to replicate Tables

Triggers can supplement the standard capabilities of Oracle to provide a highly customized database management system. Triggers can be used to enforce dynamic data replication as shown in Figure 20. When an INSERT statement is issued against the Header table in the CIND.GOOFY.CONCORDIA.CA database, the copy_header trigger is fired to replicate the

Header table into the remote CIND.IDEAS.CONCORDIA.CA database. Similar triggers have been created to replicate other tables.

```
CREATE TRIGGER copy_header
AFTER INSERT ON header
FOR EACH ROW
BEGIN
      INSERT INTO header2
      VALUES ( :new.header_id, :new.title, :new.alt_title, :new.character,
               :new.language, :new.date_created, :new.date_expiry,
               :new.date_updated, :new.version, :new.abstract);

END;
```

Figure 20: Database replication using triggers

4.2 Distributed Databases Administration

4.2.1 Catalog Management in Distributed Databases

A Catalog of distributed databases stores all the information which is useful to the system for accessing data correctly and efficiently, and for verifying that users have the appropriate access rights to them. Catalogs are used for translating applications (Data referenced by applications at different sites are mapped to physical data), optimizing applications (Data allocation, access methods available at each site are required for producing an access plan) and executing applications (Catalog information is used to verify that access plans are valid and that the users have the appropriate access rights).

4.2.2 Catalog Structure

In order to send queries into a specific oracle DBMS, a catalog described as follows is needed:

- Fragmentation description - The description of the replica (Oracle Database Name).
- Allocation description - It gives the mapping between replicas and physical images.

Since the CINDI distributed databases are controlled by Java applications, a JDBC URL that identifies each individual database, is needed.

The JDBC URL structure is defined as follows [PK97]:

```
jdbc:<sub-protocol>:<sub-name>
```

```
Where: jdbc is the standard base,
       sub-protocol is the particular data source type, and
       sub-name is an additional specification that can be used by the sub-protocol.
```

In addition, the Apache Web Server in the CINDI system is located in the Windows NT machine; therefore **JDBC-ODBC** bridge is used with the following URL format:

```
jdbc:odbc:datasourcename, username, password
```

Catalogs can be allocated in distributed databases in many different ways. In the CINDI system, the interface to the distributed databases is accessed via the World Wide Web and controlled by applications. Therefore a centralized catalog is chosen. In a centralized catalog, the complete catalog is stored at one site; the site where the Web Server resides. In the CINDI.CS.CONCORDIA.CA site which hosts thew Apache Web server, the catalog is created as shown in figure 21. In this particular catalog, the Primary CINDI Oracle Database with its jdbc:odbc:goofycind location and the Replicated CINDI Oracle Database with its jdbc:odbc:ideascind location were created. To add new databases nodes to the CINDI system, we need to append the access informations to the catalog rather than add these informations into Java applications.

```
CREATE TABLE cindi_catalog (
        name            varchar2(200),
        url             varchar2(200),
        userid          varchar2(200),
        password        varchar2(200)
);

INSERT INTO cindi_catalog
VALUES('CIND','jdbc:odbc:goofycind','cindidba','*********');

INSERT INTO cindi_catalog
VALUES('CIND','jdbc:odbc:ideascind','cindidba','*********');

COMMIT;

Where: goofycind is the Primary CIND Database located in the goofy.concordia.ca site and
       ideascind is the Replicated CIND Database located in the ideas.concordia.ca site.

       goofycind and ideascind are ODBC System Data Source Name defined in the
       cindi.cs.concordia.ca site using the Microsoft ODBC Data Source Administrator.
```

Figure 21: Catalog structure implementation

4.3 Implementing Web Applications with Java

4.3.1 Getting a Connection

The first step in using a JDBC driver to get a database connection involves loading the specific driver class into the application's Java Virtual Machine (JVM). This makes the driver available later, when we need it for opening the connection. An easy way to load the driver class is to use the Class.forName() method:

```
Class.forName(''sun.jdbc.odbc.JdbcOdbcDriver'');
```

When the driver is loaded into memory, it registers itself with the java.sql.DriverManager class as an available database driver. The next step is to ask the driverManager class to open a connection to a given database where the database is specified by a specially formatted

URL. The method used to open the connection is DriverManager.getConnection(). It returns a class that implements the java.sql.Connection interface:

```
Connection con = DriverManager.getConnection(''jdbc:odbc:database'',
                                             ''user'', ''password'');
```

A JDBC URL identifies an individual database in a driver-specific manner.

4.3.2 Connecting to the Distributed Databases

The cindi_catalog table which resides in the CINDI.CS.CONCORDIA.CA (Central Site) is used to get the physical locations of the CINDI distributed databases. In the CINDI registration sub-system, the connection to the goofy site is shown in figure 22. For the CINDI search sub-system, as indicated in figure 23, if the goofy site is down, the connection will be directed automatically to the ideas site.

```
try
{

  // try to connect to goofy site

  con = DriverManager.getConnection(''jdbc:odbc:goofycind'', ''cindidba'',
                                                             ''*******'');

}

catch
{

  // a Problem occured while connecting goofy site

}
```

Figure 22: Connecting to the Distributed Databases in the Registering Sub-system

```
try
{
  // try to connect to goofy site
  con = DriverManager.getConnection(''jdbc:odbc:goofycind'', ''cindidba'',
                                                              ''********'');
}
catch
{
  // Problem with connection to goofy, try to connect to ideas site
  con = DriverManager.getConnection(''jdbc:odbc:ideascind'', ''cindidba'',
                                                              ''********'');
}
```

Figure 23: Connecting to the Distributed Databases in Search Sub-system

4.3.3 Executing SQL Queries

To use a database, we need to have some way to execute queries. The simplest way to prepare a query is to use the java.sql.Statement class; the same statement object can be used for multiple unrelated queries. Statement objects are never directly instantiated; instead, a program calls the createStatement() method of Connection to obtain a new Statement object:

Statement stmt = con.createStatement();

A query that returns data can be executed using the executeQuery() method of Statement. This method executes the statement and returns a java.sql.resultset that encapsulates the retrieved data:

ResultSet rs = stmt.executeQuery("SELECT * FROM header");

To register a Semantic Header, the data needs to be entered into the database and a Resultset is not expected, therefore the executeUpdate(; method is used as follows:

stmt.executeUpdate("INSERT INTO header VALUES('title', ... ");

Chapter 5

Dynamic Behaviour of the CINDI System

The dynamic model describes those aspects of a system concerned with the sequencing of operations over time. In other words, the model 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, regardless of what the operations do, what they operate on, or how they are implemented. In the following sections, the dynamic model is described into scenarios and event traces.

Scenarios and Event Traces

A scenario is a sequence of events that occur 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 sequence of events and the objects exchanging events can both be shown in pictorial form called an Event Trace Diagram. In the diagram, 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.

To begin, a typical scenario that gives a general overview of the dynamic behaviour of the whole system is presented.

The CINDI system is divided into two sub-systems :

- Semantic Header registration
- Semantic Header search

5.1 Semantic Header Registration

The user in the client site makes a request by filling in some of the fields in the Internet browser. The client connects to the server through the Internet, and sends the user's request in an HTML form. At the *cindi* server site, the server receives this HTML form and obtains the location of the *goofy* site from the local server database. At this point, based on the information retrieved from the HTML form and the location of the goofy site, the server calls appropriate database functions to process the registration query. The server then checks if the Semantic Header already exists using the SHN (Semantic Header Name) which is the combination of title, name of the first author, first subject, creation date and version. In the event that the Semantic Header already exists in the databases, the server cancels the registration process and sends a message to the client site indicating that the Semantic Header already exists. Otherwise the registration is processed and the triggers are fired to replicate the same content of the database into the *ideas* site. Finally, the server sends the confirmation through the Internet to the client site. In the client site, the user receives a message saying that both the registration and the replication were successful.

5.1.1 Scenarios and Event Traces for CINDI Registration Sub-System

To begin registration, as shown in figure 24 and detailed in Appendix B, the user enters a set of elements of the Semantic Header, such as title, author and subjects, by completing the HTML form which assists the user to choose subject hierarchies. Choosing a subject

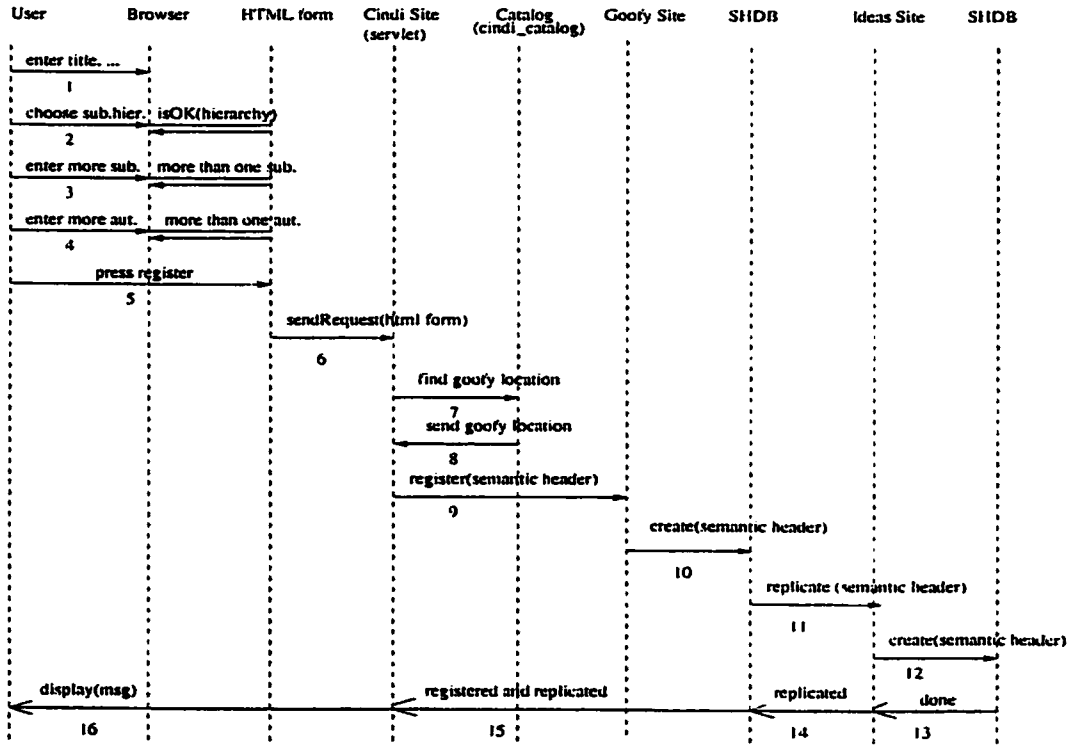

Figure 24: Event Trace Diagram for CINDI Registration Sub-System

can be done by pressing on the corresponding icon. There are three levels in one subject hierarchy: general, level1 and level2. (The selection process of each level is emphasized in section 5.1.2). Upon achieving level2, the user can enter more than one subject; the user can also enter more than one author. Once the HTML form is completed and submitted to the *cindi.cs.concordia.ca* (central server) site, the servlet at the *cindi.cs.concordia.ca* site receives the Semantic Header information and retrieves the location of the *goofy.concordia.ca* Oracle database server (from the cindi_catalog table). The servlet then sends the registration request to the *goofy.concordia.ca* Oracle database server, where upon the *goofy.concordia.ca* Oracle database server creates a new Semantic Header and fires triggers to replicate the same Semantic Header into the *ideas.concordia.ca* Oracle database server. At this point, the *ideas.concordia.ca* creates a new Semantic Header. Once this is created, the servlet at the *cindi.cs.concordia.ca* central server site sends a message to the user saying that both the registration and the replication of the Semantic Header were successful.

5.1.2 User Interactions with the CINDI Registration Sub-System

The Semantic Header registration provides a graphical user interface, as shown in figure 23-27, to facilitate the provider (author/creator) of a resource registering the bibliographic information about the resource. The interface allows the provider to enter this information and provides control by means of selection of the controlled terms. To initialize a registration of a new Semantic Header, a user completes the HTML form ensuring that certain entries which are indicated by an asterisk, such as title and subject fields, are mandatory. For the subject entry, the user must choose three levels of one subject hierarchy by pressing on the corresponding icon. Subject entry fields are not editable, which means that they cannot be manually entered. The system, however, offers selection items. For the general level, Computer Science and Electrical Engineering are the two options provided. The user must select a higher level in the subject hierarchy before selecting a lower level. The user must fill out the general level before looking up level1, and similarly choose level1 before looking up level2. When pressing the general icon, the matching items for general level will be displayed, and similarly by selecting level1 and level2, the matching items for each level will be displayed. Once level2 is chosen, the user can enter more than one subject by pressing on the ACCEPT/NEXT button.

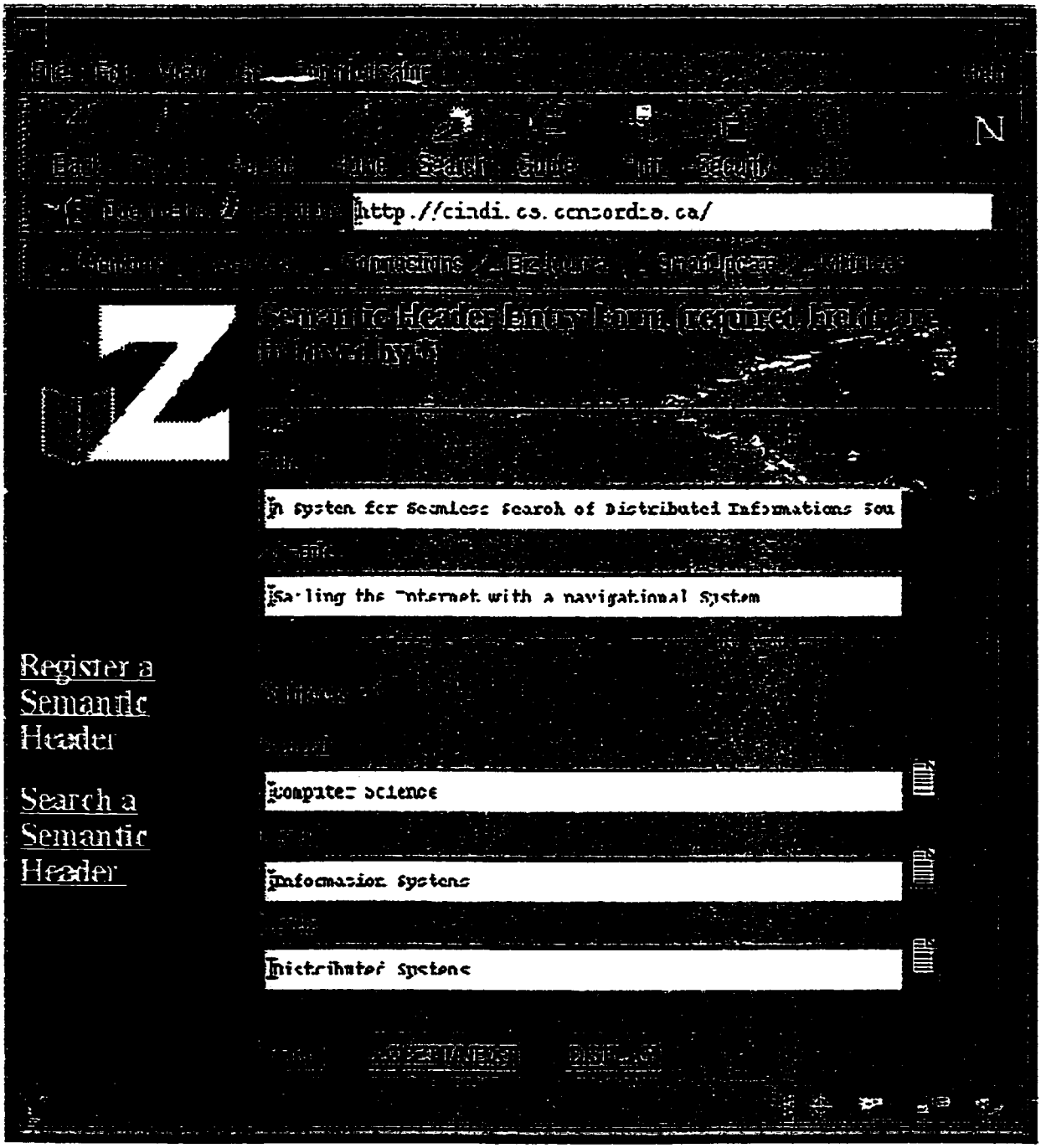

Figure 23: Part One of the Semantic Header Web GUI Registration

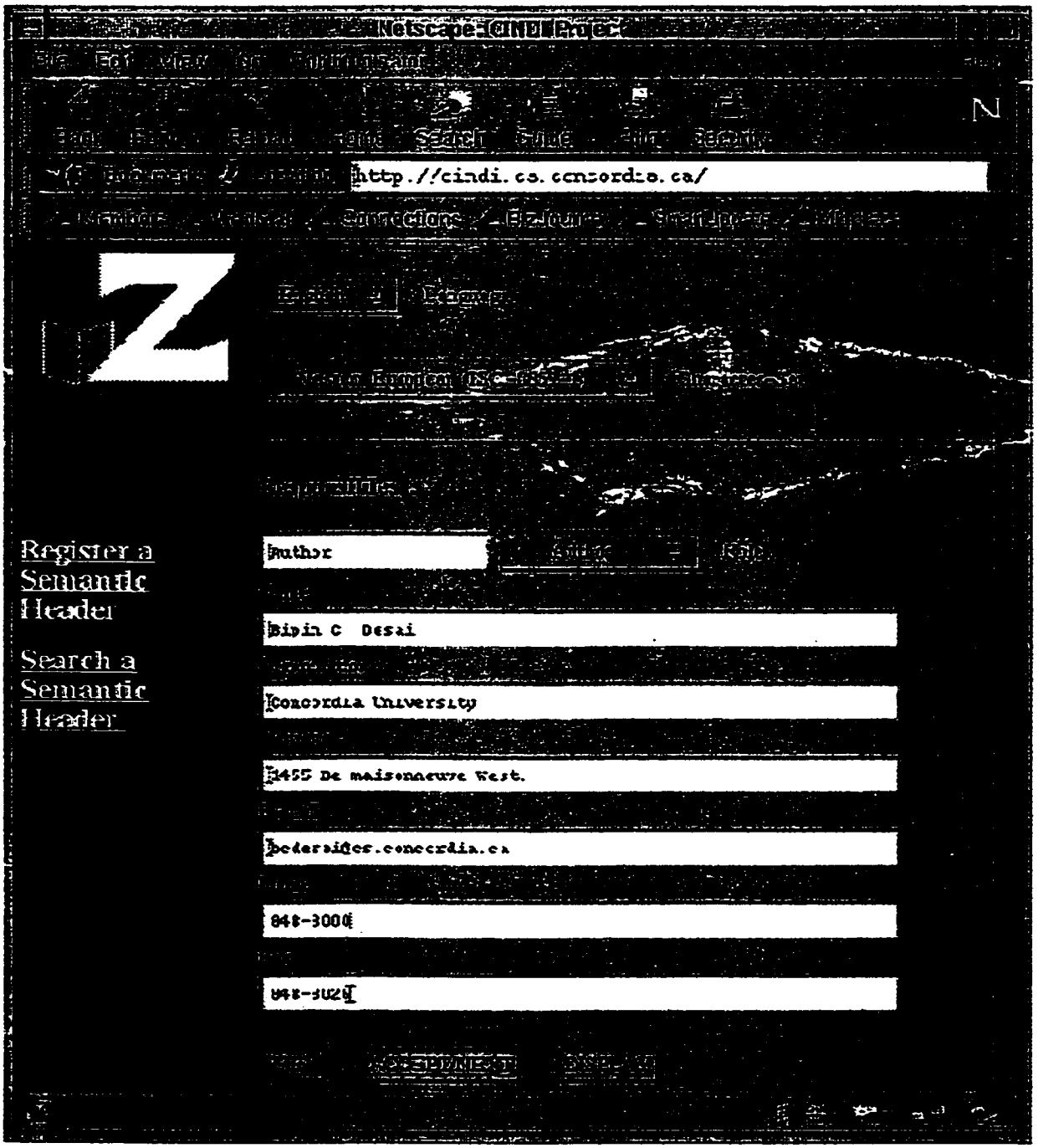

Figure 24: Part Two of the Semantic Header Web GUI Registration

For author entry as well, the user has, as indicated in figure 26, the option to enter more than one author by pressing on the ACCEPT/NEXT button. Also included on the HTML form, as shown in figure 27, are the Keywords field, where more than one keyword can be entered using commas to indicate the different keywords. In the remaining fields, as seen in figure 28, there are either selection items provided by the user interface or manual entries typed in by the user.

Figure 29 displays the final entry in the HTML form where the User ID and the associate encoded password must be provided before registering the Semantic Header. To finalize registration, that is after all entries have been selected or entered manually, the user must click on the REGISTER button. If the entries are accepted and validated by the user interface, the Semantic Header will be registered at the *goofy.concordia.ca* site and replicated at the *ideas.concordia.ca* site. The user will then be notified in message form, as indicated in figure 30. If the Semantic Header already exists in the databases, the registration process is cancelled and the user is notified by the following message "The Semantic Header already exists ... ".

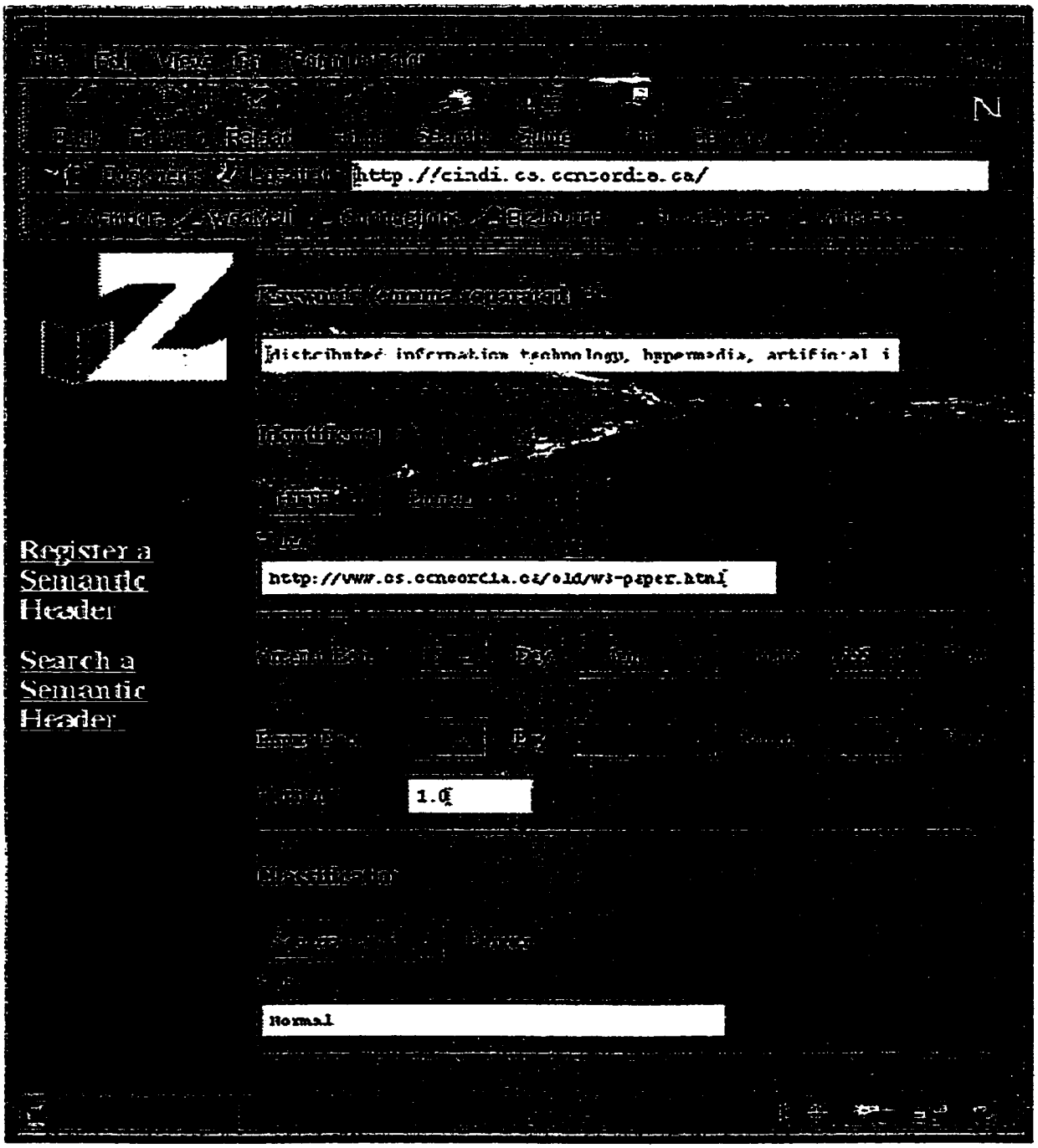

Figure 25: Part Three of the Semantic Header Web GUI Registration

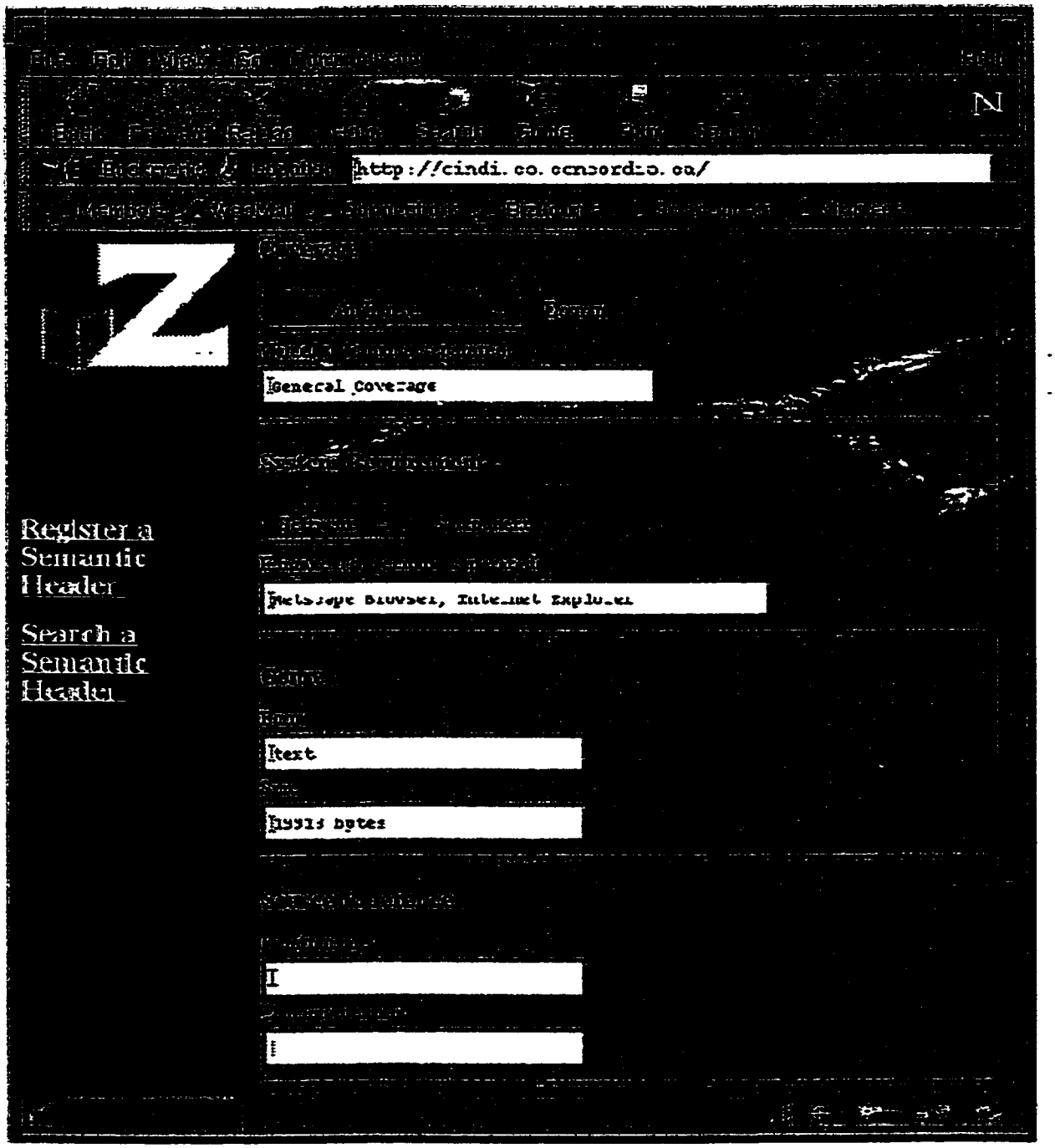

Figure 26: Part Four of the Semantic Header Web GUI Registration

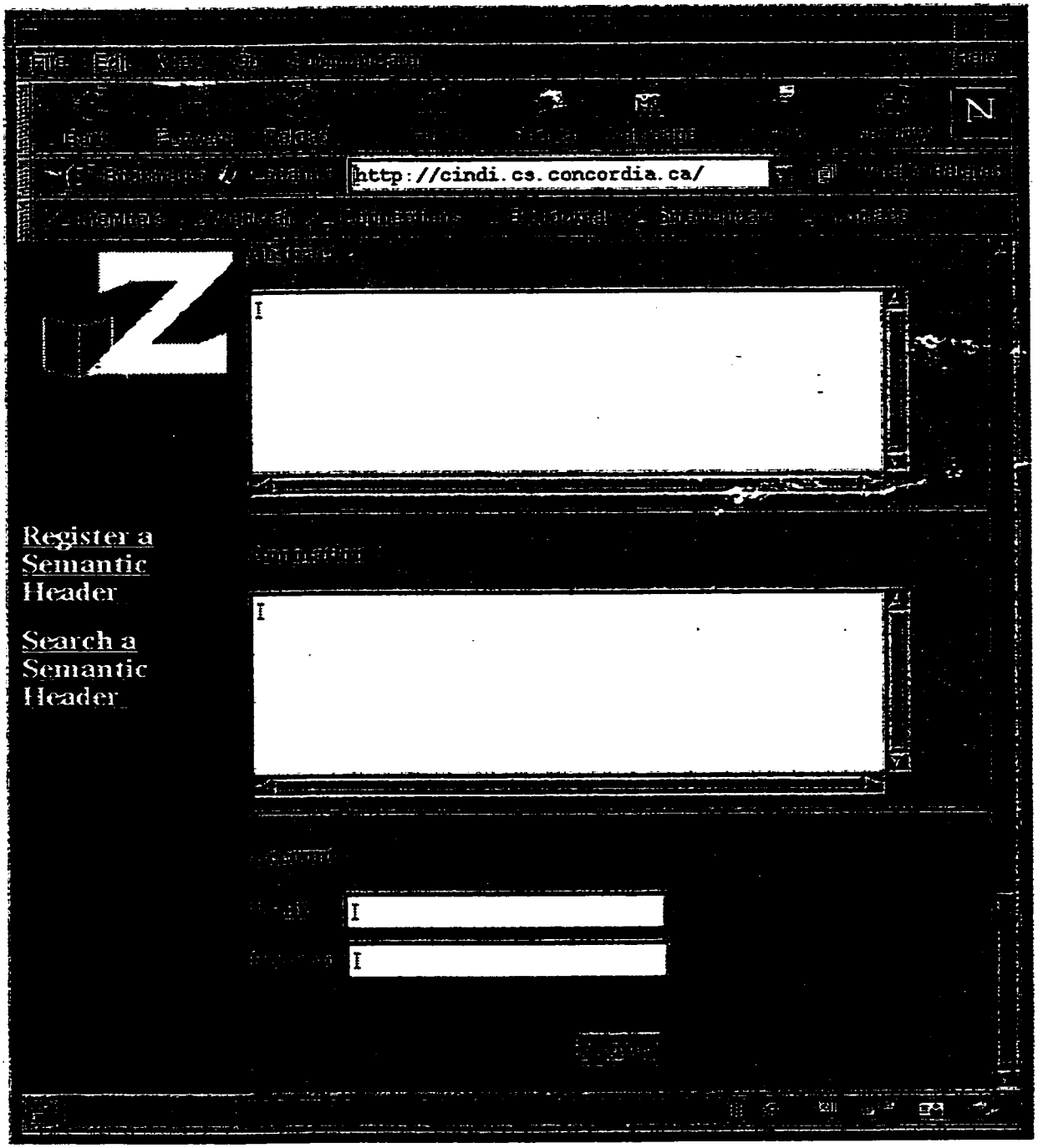

Figure 27: Part Five of the Semantic Header Web GUI Registration

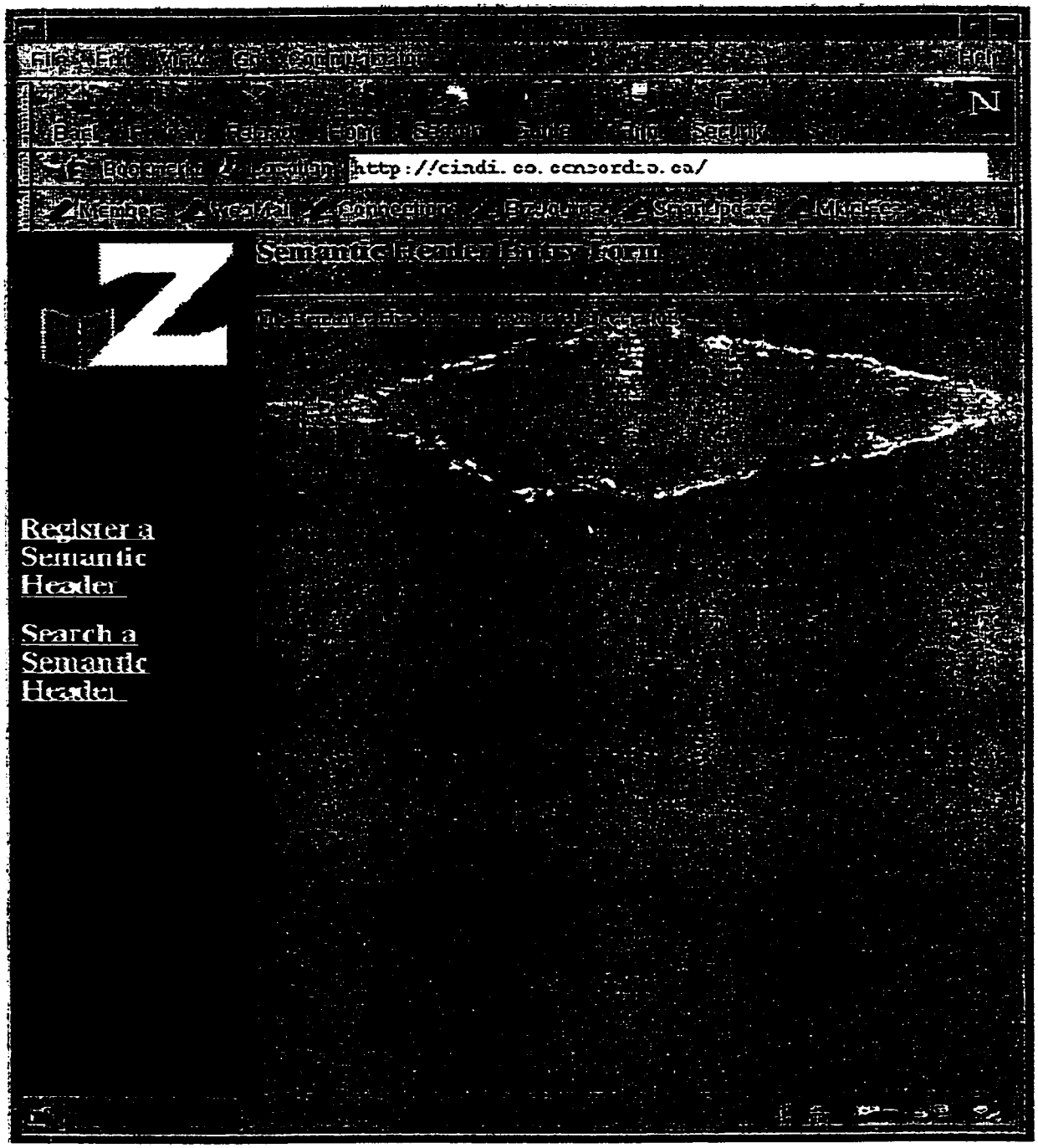

Figure 28: Result of the Semantic Header Web GUI Registration

5.2 Semantic Header Search

The user in the client site makes a request by filling in some of the fields in the Internet browser. The client connects, through the Internet, to the server and sends the user's request in an HTML form. At the *cindi* server site, the server assigns a unique session ID to this particular user and sets its timeout to ten minutes, receives this HTML form and retrieves the location of the *goofy* and the *ideas* sites from the local server database. At this point, based on the information retrieved from the HTML form and the state of the *goofy* site, the server calls appropriate database functions to process the search query against the *goofy* site if it is not down. Otherwise it processes the query against the *ideas* site. Once the query is processed, the result is sent back to the *cindi* site. Finally, the *cindi* server site, using the unique session ID, forwards the results through the Internet to the specific client site. In case of concurrent access to the CINDI system; each user in a particular site, receives a set of Semantic Headers, corresponding to its session ID, as results.

5.2.1 Search Query Structure

The content of the search query contains the title and/or subject and/or author and/or keyword. By filling in the main HTML form and then submitting it to the server, what a user does, in essence, is create and transmit a string to the server, which in turn will be translated into an SQL query. The set of all possible strings that the user can create and submit, is succintly described by the grammar written in BNF [GM86] as presented in figure 31. In simple words, the grammar determines the set of strings that can have optional parts connected by the logical AND, OR. The optional parts are: title, subject, author and keyword. Their exact orders and combinations are given again by following the BNF rules of the grammar. BNF is a notation for specifying the syntax of a language, but not its semantics. We assume the semantics of the first rule regarding the operator precedence is the same as the default followed by SQL processors for the boolean expressions in the Where Clause. AND has a higher priority than OR. The user can search for Semantic Headers for a given title, which could be exact or a substring title. A search can also be conducted for a given subject at the general level, general and level1 or general, level1 and level2. For a given author which could be an exact or substring name, the user can search for Semantic Headers. The same applies to the keywords.

```
<search>          ::=  <operand> [ <op> <operand> [ <op> <operand>
                       [ <op> <operand> [ <op> <date> ] ] ] ]

<operand>         ::=  <title> | <subject> | <author> | <keyword> | <date>

<title>           ::=  <exacttitle> | <substringtitle>

<subject>         ::=  <general> | <general> AND <level1> |
                       <general> AND <level1> AND <level2>

<author>          ::=  <exactname> | <substringname>

<date>            ::=  <afterdate> | <beforedate> | <afterdate> AND <beforedate>

<afterdate>       ::=  <day> AND <month> AND <year>

<beforedate>      ::=  <day> AND <month> AND <year>

<op>              ::=  AND | OR

<keyword>         ::= <string>
<exacttitle>      ::= <string>
<substringtitle>  ::= <string>
<general>         ::= <string>
<level1>          ::= <string>
<level2>          ::= <string>
<exactname>       ::= <string>
<substringname>   ::= <string>

<string>          ::= <character> | <character> <string>
<character>       ::= a|b|c| ... |x|y|z|A|B|C| ... |X|Y|Z|0|1|2| ... 7|8|9

<day>             ::= 1|2|3| ... |29|30|31
<month>           ::= 'January' | 'February' | 'Mars' | 'April' |
                      'May' | 'June' | 'July' | 'August' | 'September' |
                      'October' | 'November' | 'December'
<year>            ::= 1999 | 2000 | 2001 | 2002
```

Figure 31: BNF Query Structure

5.2.2 Scenarios and Event Traces for the CINDI Search Sub-System

There are two scenarios in the Search Sub-System. In the first scenario, if the *goofy* site is not down, as presented in figure 32 and detailed in Appendix B, the user can then enter a query via the HTML form that is a set of Semantic Header elements such as title, subject, author, and keyword. The user has the option to search by using the following key terms: title, subject, author, or keyword. The user can also search by entering a combination of two or more key terms. The HTML form helps the user to choose subject hierarchies. This can be accomplished by pressing on the corresponding Icon. There are three levels in the hierarchy: general, level1 and level2. At the *cindi* central server site, the servlet uses the parser to parse and translate the information received from the HTML into SQL query, then obtains the location of the *goofy* and *ideas* database servers from the central catalog (cindi_catalog table). At this point, the servlet sends the request to the *goofy* server database, where the translated SQL query will be executed and the results retrieved. The servlet at the *cindi* central server site, formats the results into HTML and sends it back to the client browser where it will be displayed.

In the second scenario, if the *goofy* site is down, as shown in figure 33 and detailed in Appendix B, the connection to the database will be redirected automatically to the *ideas* site, after the user has entered a query via the HTML form that is a set of Semantic Header elements such as title, subject, author, and keyword. The user has the option to search by the following key terms: title, subject, author, or keyword. The user can also search by the combination of two or more key terms. The HTML form helps the user to choose subject hierarchies. This choice can be made by pressing on the corresponding icon. There are three levels in the hierarchy: general, level1 and level2. The servlet at the *cindi* central server site, receives the request from the HTML form and uses the parser to translate it into an SQL query. At this point, the *cindi* server obtains the location of the *goofy* database site and the location of the ideas site from the central catalog (cindi_catalog) and sends the request to the *ideas* server database where it will be executed. The servlet at the *cindi* central server site, formats the results into HTML and sends it back to the client browser. Finally, the client browser displays the results to the user.

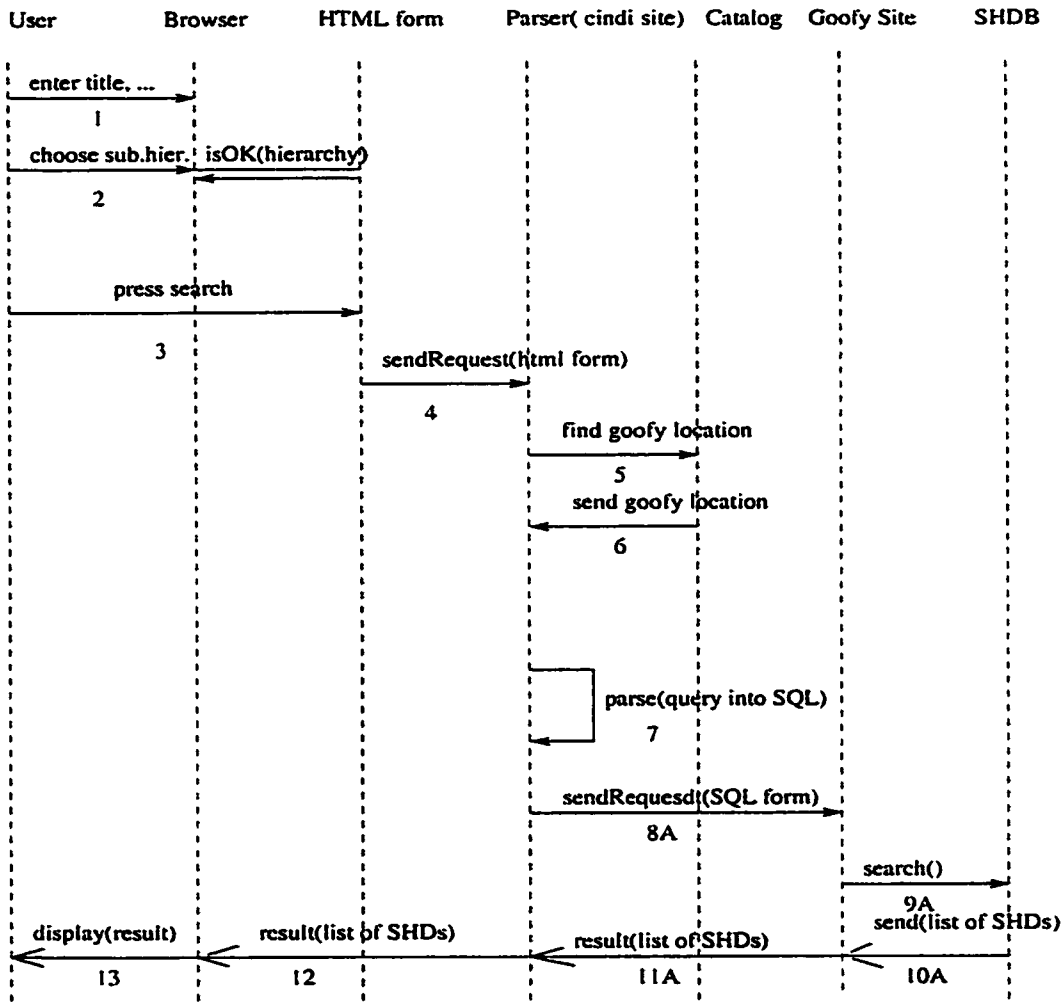

Figure 32: Event Trace for 'Semantic Header Search' (goofy site is not down)

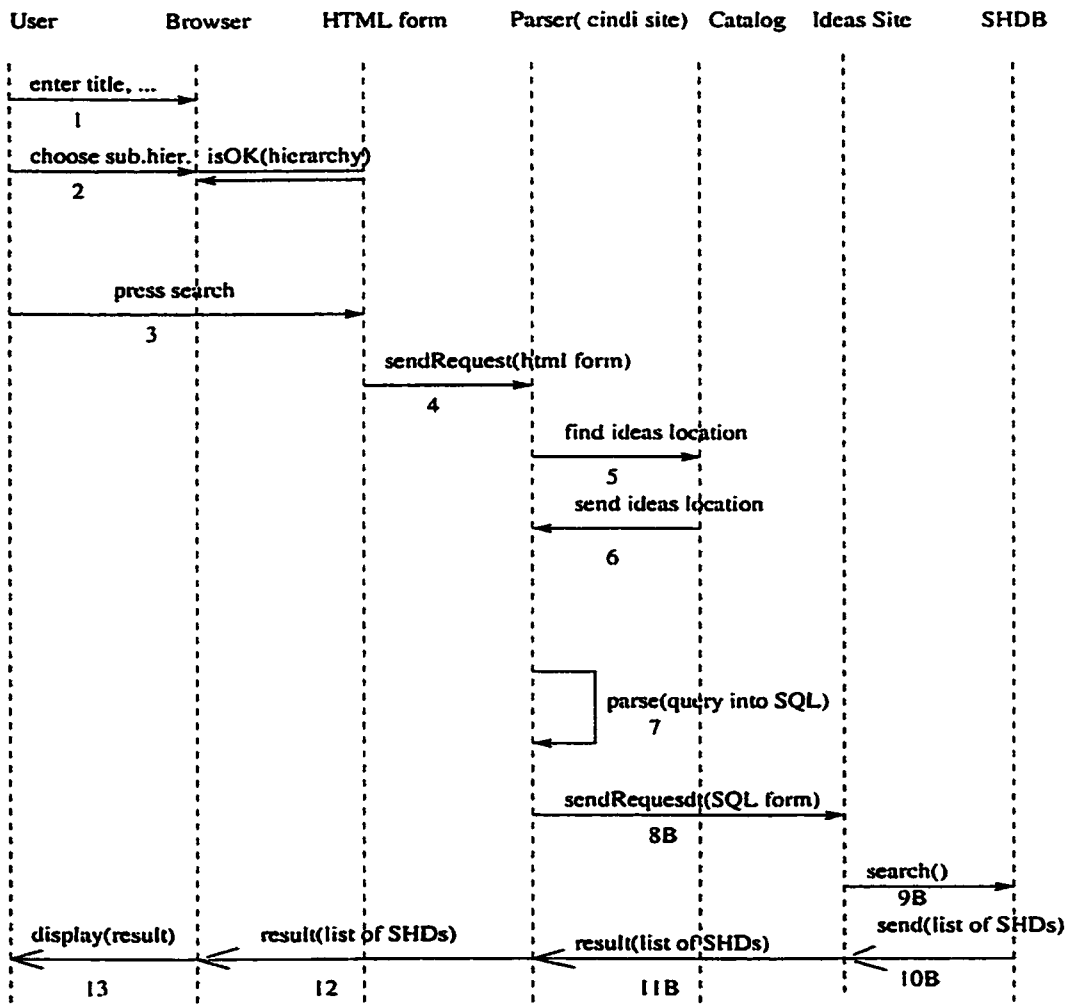

Figure 33: Event Trace for 'Semantic Header Search' (goofy site is down)

5.2.3 User Interactions with the CINDI Search Sub-System

The Semantic Header search sub-system provides a graphical user interface as seen in figure 33, to facilitate searches for Semantic Headers. In this particular example, the user is searching for Semantic Headers using "search" as substring title. To use the CINDI system, a user must enter "search" in the title field and click on the substring check-box. Then, as indicated in figure 34, the user must click on the SEARCH button to send the HTML form to the *cindi.cs.concordia.ca* server site where the servlet receives this information and uses the parser to translate it into an SQL query. The corresponding SQL query is given in figure 32.

```
SELECT DISTINCT header_dtl.header_ID, header_dtl.ident_ID
FROM   header_dtl.header_ID = header.header_ID AND
       header.title LIKE '%search%';
```

Figure 32: HTML Query translated into SQL Query

The servlet retrieves the locations of the *goofy* and *ideas* sites from the cindi_catalog table. Once the locations are retrieved, the servlet tries to connect to the *goofy* site; if it is down, it automatically redirects the connection to the *ideas* site. Using the JDBC, the servlet sends the query to the Oracle server database in SQL form, and then gets the results back. Once the results are retrieved, the servlet will format them into HTML, as shown in figure 35. By clicking on the Anchor provided in figure 35, the complete Semantic Header will be viewed. By clicking on the Anchor provided in figure 36, the actual resource will be viewed as demonstrated in figure 37.

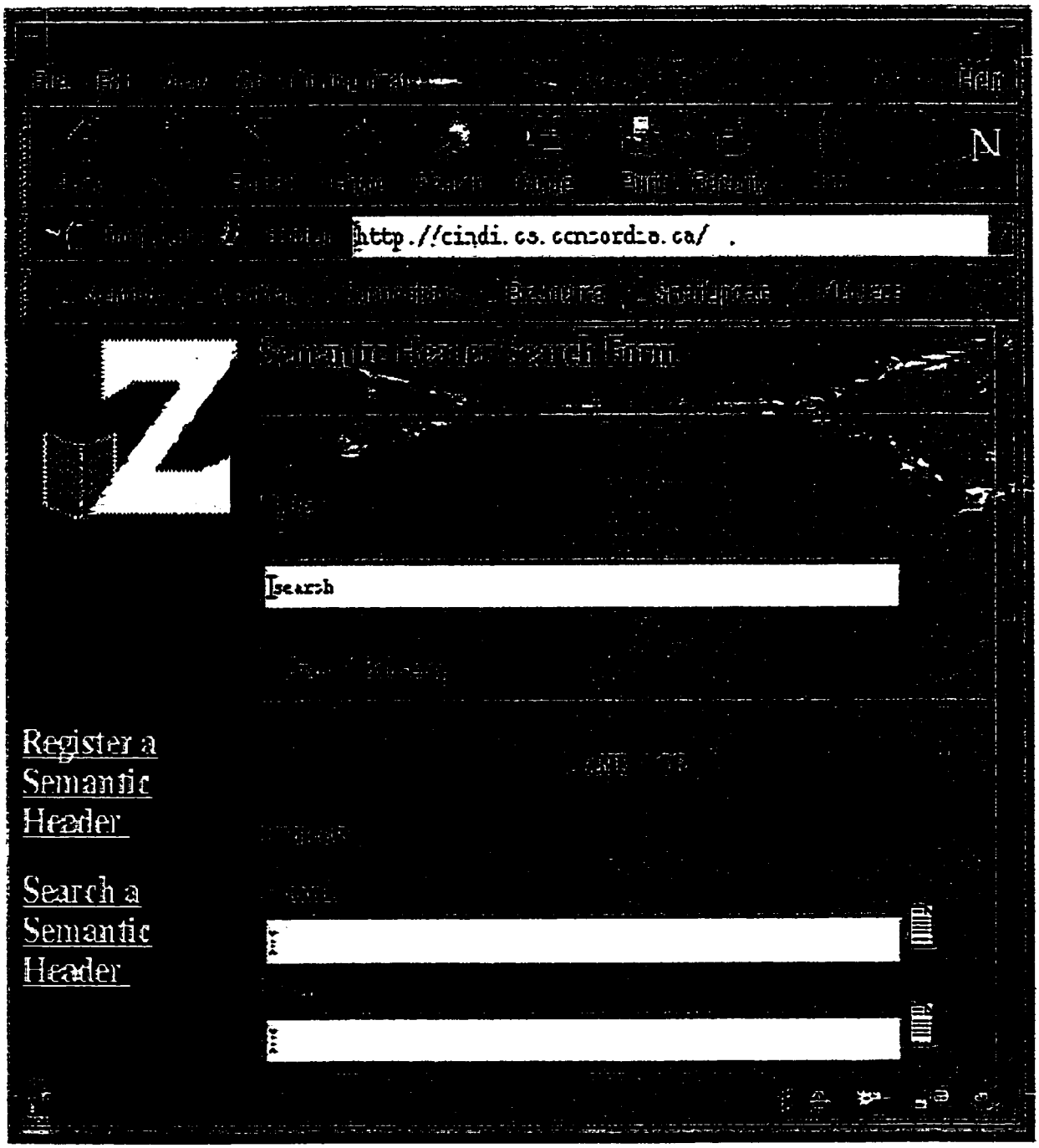

Figure 33: Part One of the Semantic Header Web GUI Search

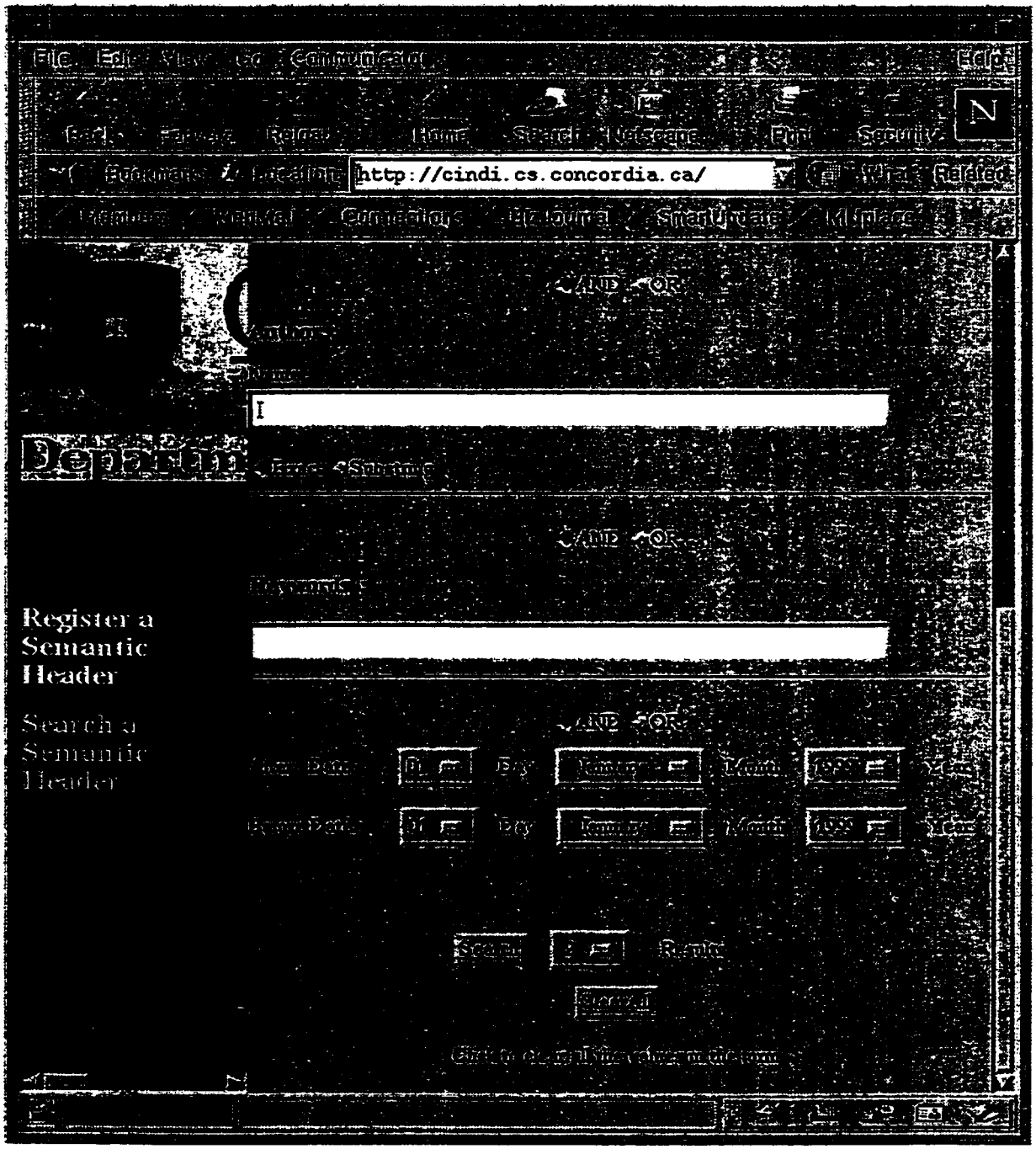

Figure 36: Part Two of the Semantic Header Web GUI Search

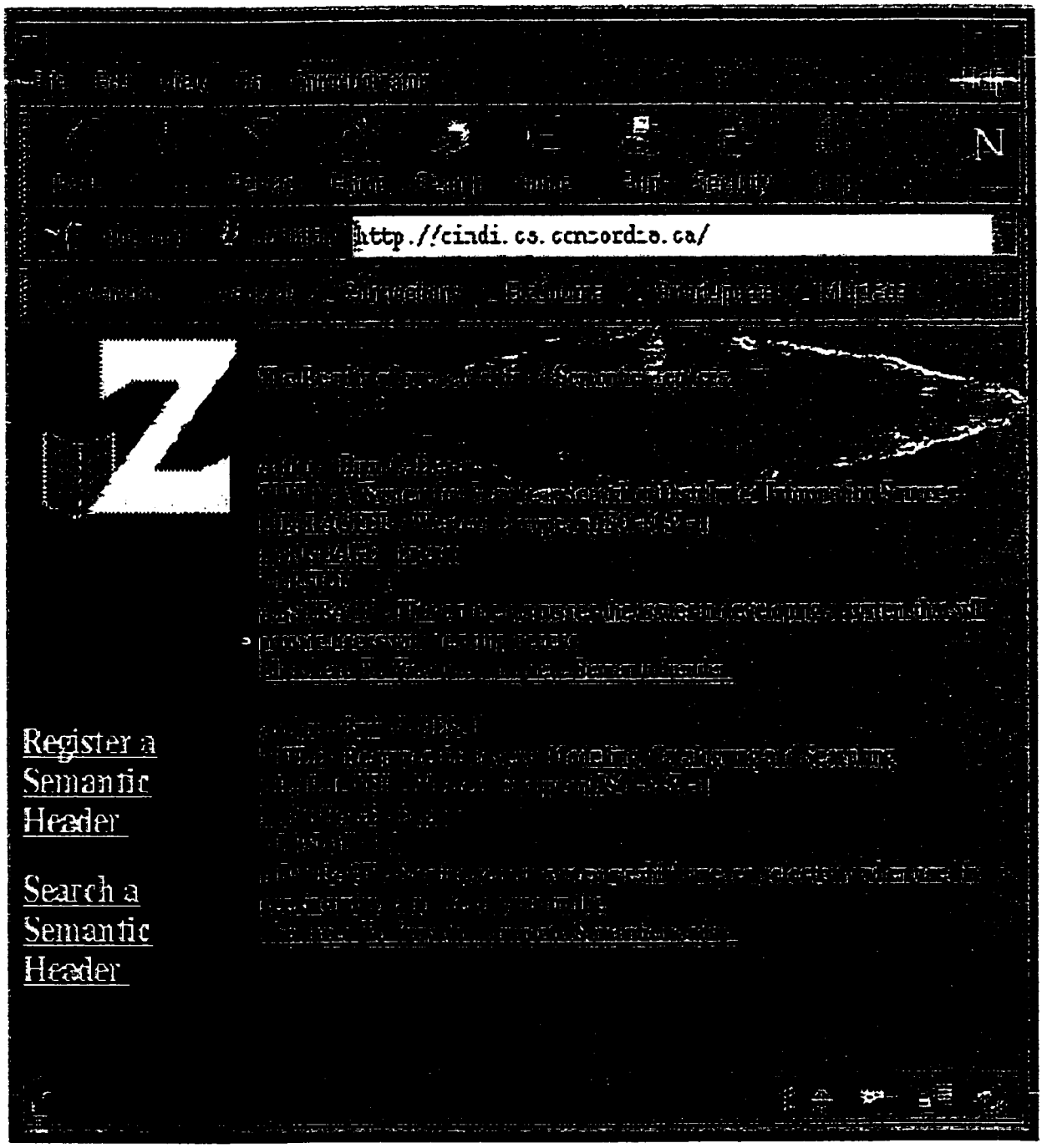

Figure 35: Part One of the Result of the Semantic Header Web GUI Search

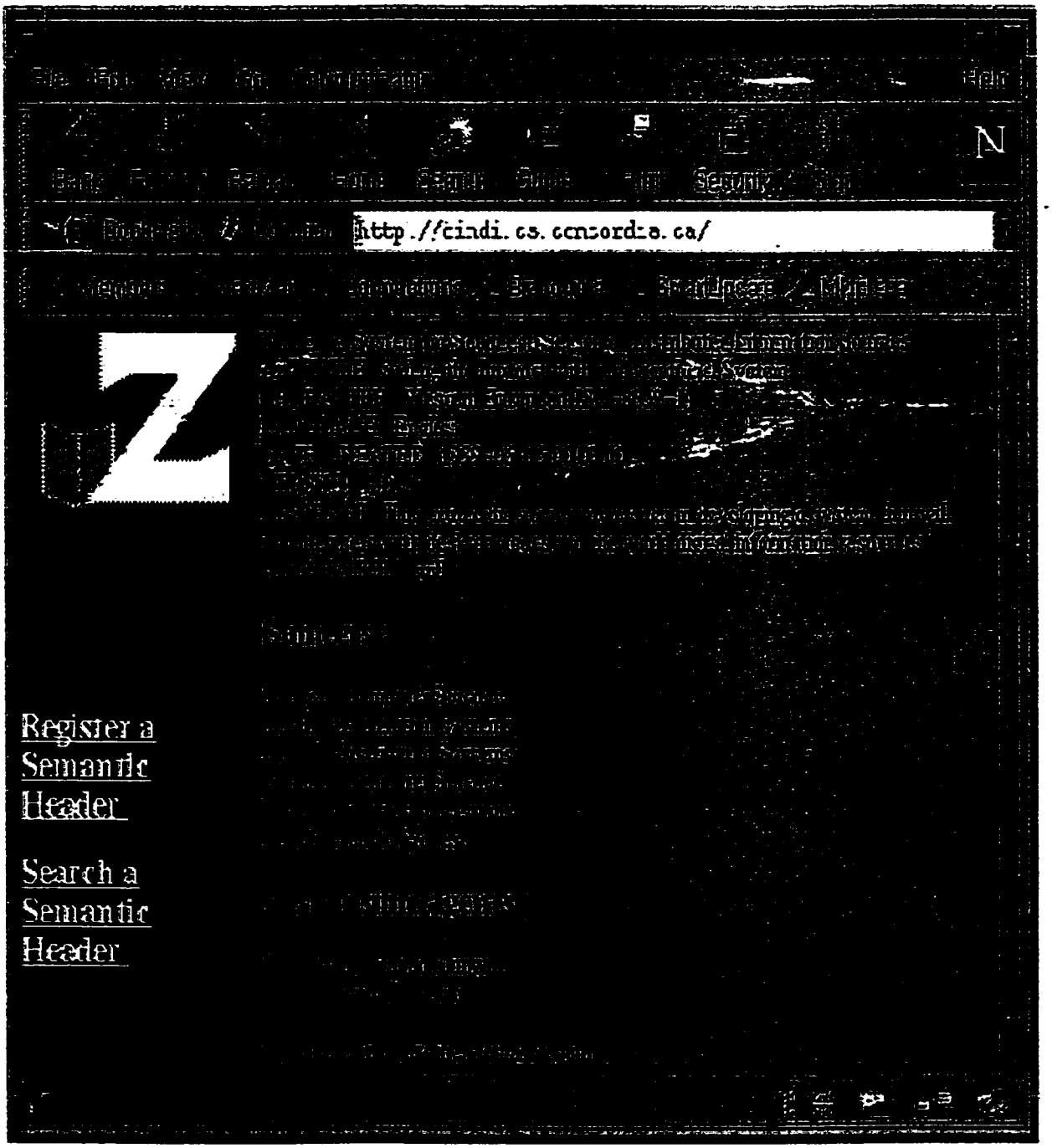

Figure 36: Part Two of the Result of the Semantic Header Web GUI Search

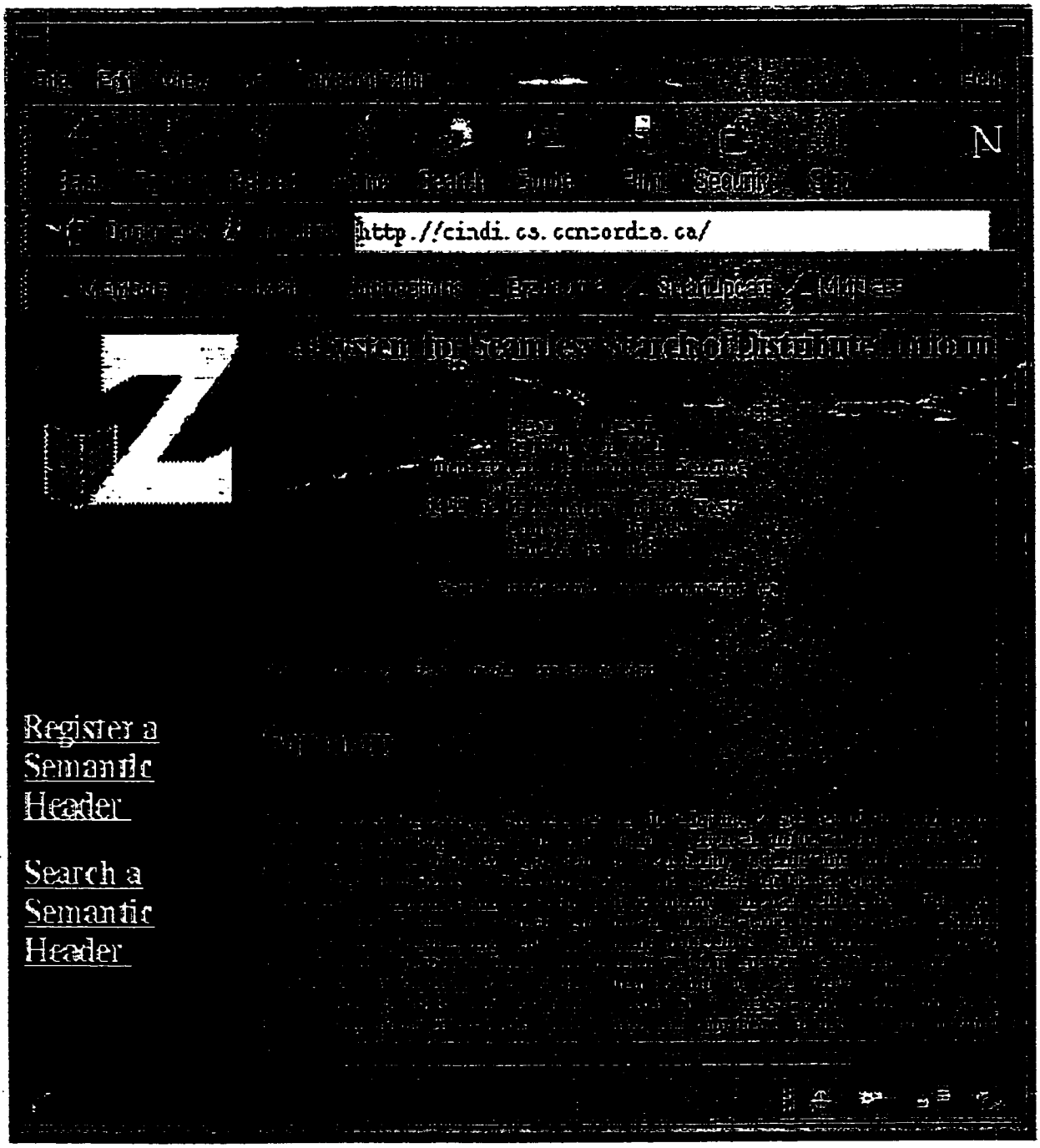

Figure 37: Part Three of the Result of the Semantic Header Web GUI Search

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The proposed Semantic Header based system meets the challenges of the coming information age by defining a Meta-Data structure, which allows automatic and semi-automatic (human assisted) extracting of Meta-Data from resources. A distributed indexing system was built with an intuitive graphical user interface to interact in the registering and discovery process. The distributed and replicated nature of the Semantic Header databases provides reliability and scalability.

The size of the CINDI database is reduced from the whole document storage into its Semantic Header storage, which allows our CINDI system to provide the correct results with minimum time processing.

6.2 Contribution of this Thesis

The design and implementation of the distributed databases subsystem for indexing and retrieval of Semantic Headers is one of the fundamental contributions made by this thesis to the CINDI project. The design and implementation of Web applications using HTML and Java-Script in the Client Side and Java Servlets in the Server Side, with Oracle JDBC as a

back ends database, also contributes to the CINDI project. Using the Web applications, the design and implementation of the the indexing/registering and search sub-systems, which incldude the registering and search query User Interfaces, were the key factors contributing to the interaction with the distributed databases through the Internet.

6.3 Future Work

The current prototype of the CINDI system satisfies the needs of Internet users for effective retrieval of electronic information resources. In the near future, however, other functionalities could be built. A major extension to the system would be to build a distributed system based on the subject areas. In the current system, all the existing Semantic Headers and their related subjects are stored in the same site. Presently, in the main site, there are two subject areas, Computer Science and Electrical Engineering subjects. Further extension would require that the Computer Science subject and all its related Semantic Headers, be stored in one site, and the Electrical Engineering subject and its related Semantic Headers, be stored in a different site. The databases on different subjects will be maintained at different nodes of the Internet.

References

[ABS00] Serge Abiteboul, Peter Buneman, dan Suciu, *Data on the Web : From Relations to Semistructered Data and XML, 2000.*

[BCD90] Bipin C. Desai, *An Introduction to Database Systems. West St Paul, 1990.*

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

[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., *World Wide Web: 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.

[BS94] Bipin C. Desai, Shinghal Rajjan, *A System for seamless search of distributed information resources, May 1994.* http://www.cs.concordia.ca/w3-paper.html.

[BS96] Bipin C. Desai, Shinghal Rajjan, *Resource Discovery: Modeling , cataloguing and searching. In Proceeding of the Seventh International conference and workshop on Database and Expert Systems Applications (DEXA'96), pages 70-75. IEEE Press, Zurich, Switzerland, 1996.*

[BCD95] Bipin C. Desai, *Report of the Metadat Workshop, Dublin. March 1995.* http://www.cs.concordia.ca/faculty/bcdesai/metadata/metadat-workshop-report.html.

[FD95] Fung R. and Del Favero B. *Applying Bayesian Networks to Information Retrieval, Communication of the ACM, Vol38, No. 3, pp. 42-57, March 1995.*

[GM86] Gehani Narrain, McGettrick Andrew *Software Specification techniques. Addison-Wesley, 1986.*

[HC98] Jason Hunter, William Crawford *Java Servlet programming. O'REILLY, 1998.*

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

[KOS96] Koster, M., *The Web Robots Pages, 1996.* http://info.webcrawler.com/mak/projects/robots/robots.html.

[LJ96] Lewis D., Jones K. *Natural Language processing for information retrieval, Communications of the ACM, Vol 39, pp. 92-101, January 1996.*

[PJ99] Paul Dorsey, Joseph R. Hudicka *Design using UML Object Modeling. Osborne/McGraw-Hill, 1999.*

[PK97] Pratik Patel, Karl Moss *Java Database Programming with JDBC, 1997. Coriolis Group Books.*

[RGR98] Raghu Ramakrishnan *Database Management Systems, 1998. McGraw Hill.*

[SOLL] Sollin K., Masinter L. *Functional requirements for Uniform Resource Name, RFC1737.* ftp://ds.internic.net/rfc/rfc1737.txt.

Glossary

CGI: (Common Gateway Interface) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state: a text file that doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information.

DBMS: (DataBase Management System) A DBMS is a computerised record-keeping system that stores, maintains and provides access to information.

ERD: (Entity-Relationship Diagram) is a snapshot of data structures.

FTP: (File Transport Protocol)is a software that allows users to tranfer files on the network.

HTML: (HyperText Markup Language) This is the format of files published on the World Wide Web.

Internet: The Internet is a worldwide communications network originally developed by the U.S. department of defense as a distributed system with no single point of failure. Long the province of scientists and academics, the development of easy-to-use software for accessing the net has generated an explosion in commercial use.

markup: Markup is anything added to the content of the document that describes the text.

tag: A tag is a marker embedded in a document that indicates the purpose or function of the element. Each element has a beginning tag and an end tag.

World Wide Web: Often referred to as WWW or the Web, this usually refers to information available on the Internet.

Web Browser: A web browser is a software used to access the information available on the Internet. The two most well-known web browsers are Netscape Navigator and Microsoft Internet Explorer, which are used by the vast majority. Other browsers are available as well.

Appendix

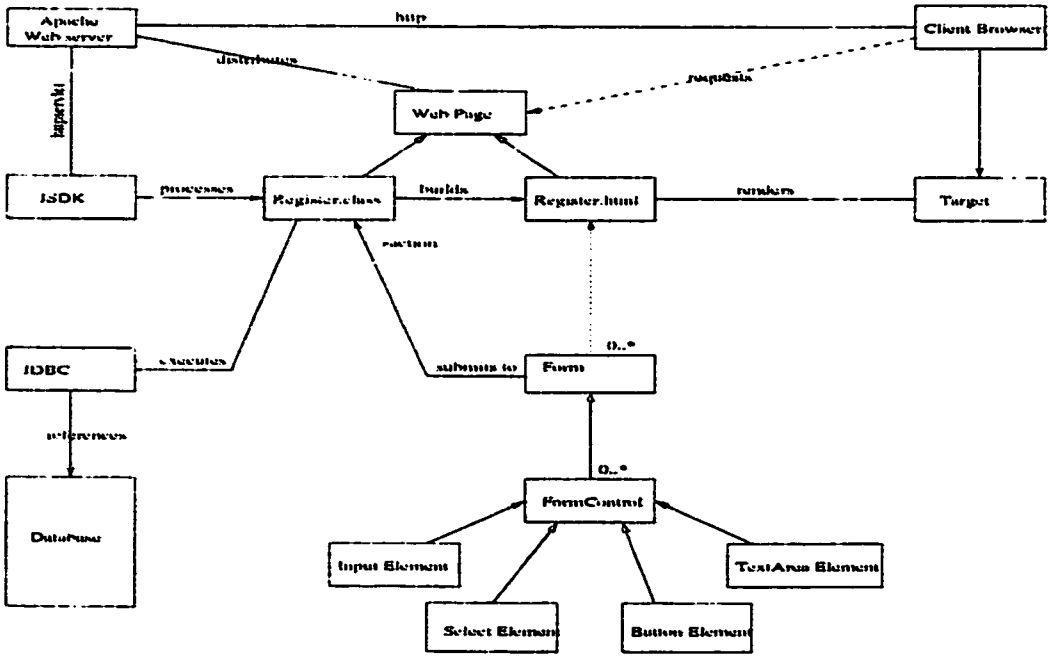

Figure 40: UML Model of the Registration Sub-system

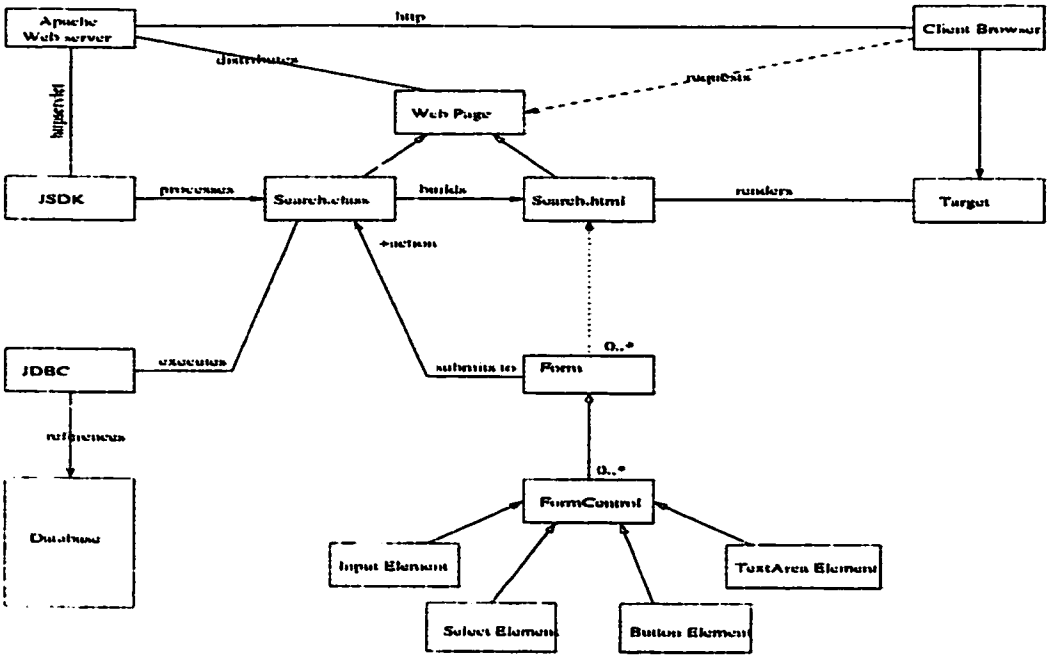

Figure 41: UML Model of the Search Sub-system

Appendix B

Detailed Messages for Registration Subsystem

Client Side

message 1 : The user enter title, Alt_title, Language, Character_Set, Keywords, Identifier, Creation_Date, Expirty_Date, Version, Classification, coverage, System Requirements, Abstract, Annotation

message 2 : The user choose subjects

message 3 : The user enters one or more subjects : add_subjects()

message 4 : The user enters one or more authors : add_authors()

message 5 : The user press the REGISTER KEY

message 6 : validating the entries, then sending them to the cindi.cs server site: validate_form()

message 16 : User gets a notification that the registration and replication is succefull

Server Side

message 7 : find the goofy database server location

message 8 : get the goofy database server location from cindi_catalog

message 9 : connect the goofy database server and sends the registration SQL query request : getConnection()

message 15 : insertion and duplication was succefull : executeUpdate()

Database Side

message 10 : goofy database server inserts the Semantic Header into the database

message 11 : goofy database server replicate the Semantic header into the ideas database

message 12 : ideas database server inserts the semantic header into the database

message 13 : insertion into ideas was succefull

message 14 : Replication was succefull

Detailed Messages for Search Subsystem

Client Side

message 1 : The user enters title AND/OR author name AND/OR Keywords AND/OR Date

message 2 : The user choose a subject

message 3 : The user press the SEARCH Key

message 4 : validating the entries, then sending them to the cindi.cs server site : validate_form()

message 13 : the browser formats the results and displays them to the user

Server Side

message 5 : find the goofy and ideas database server locations

message 6 : get the goofy and the ideas database server locations from cindi_catalog

message 7 : the cindi.cs translate the Search query from HTML form into SQL form : parse()

message 8A : If the goofy site is not down, connect the goofy database server and sends the search SQL query request : getConnection()

message 8B : If the goofy site is down, connect the ideas database server and sends the registration SQL query request : getConnection()

message 11A: search servlet gets the results from the goofy database server

message 11B: search servlet gets the results from the ideas database server

message 12 : search servlet forwards the results to the Client Browser

Database Side

message 9A : goofy database server searches for the Semantic Headers : executeQuery()

message 9B : ideas database server searches for the Semantic Headers : executeQuery()

message 10A: goofy database server gets the results

message 10B: goofy database server gets the results