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: DATABASE DESIGN AND IMPLEMENTATION

WEN TIAN

A MAJOR REPORT

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA

AUGUST 2001

Abstract

WEB BASED CINDI SYSTEM:
Database Design and Implementation

Wen TIAN

This report presents the database design and implementation of the CINDI System (Concordia INdexing and DIscovery) based on the World Wide Web. It is part of the work to develop a Web-based CINDI system. The information resource is described using a metadata called a Semantic Header, which is stored in MySQL database management system.

The Web based CINDI system is a 3-tier application that includes a Browser, an Apache Server and a MySQL Server. In the application, E-R (Entity-Relationship) modeling has been employed for the analysis and design of the MySQL Database Management System. The hierarchy searching of subjects was implemented. The Apache Web Server is used to communicate between the Browser and the Databases subsystems using the HTTP protocol. The interface was developed using PHP script language.

Acknowledgements

To begin with, I want to express my sincere gratitude to Dr. Bipin C. Desai, my supervisor, for his invaluable discussions and suggestions. Without his enthusiastic support and consistent guidance, this work would not have been possible.

I am sincerely grateful to Xiaomei Yang and Zhan Zhang who, as part of the CINDI project, suggested many ideas to me during our valuable discussions. Their amicable personalities also made the project more enjoyable.

Finally, I would like to dedicate this work to my wife, Liping Li and to my son, Daniel, for their continuous support and encouragement.

Contents

1 Introduction 1

1.1 Overview of the Virtual Library 1
1.1.1 The World-Wide Web (WWW) 1
1.1.2 Web Servers and Databases 2
1.2 CINDI System 3
1.2.1 Introduction 3
1.2.2 CINDI: The Project 4
1.3 Organization of the Report 6

2 Background 7

2.1 Information Retrieval 7
2.1.1 Meta-Data 7
2.1.2 Semantic Header 7
2.2 MYSQL Database 8
2.2.1 What is MySQL? 8
2.2.2 Why use MySQL? 9
2.2.3 The Main Features of MySQL 11
2.3 PHP 12
2.3.1 What is PHP? 12
2.3.2 What can PHP do? 13
2.4 Apache Server 14
2.5 ASHG 15

3 Architecture of the CINDI System 17

3.1 3-tier Architecture 17
3.1.1 Introduction 17
3.1.2 The Advantages of 3-tier Architecture 19
3.2 System Design 20

4 Semantic Header Database System Design and Implementation 24

4.1 Data Modeling and E-R Model 24
4.2 Logical Model 25
4.2.1 Identifying Entities and their Attributes 25
4.2.1.1 Resource Entity 26
4.2.1.2 Subject Entity 29
4.2.1.3 Resource_subject Entity 29
4.2.1.4 Resource_author Entity 29

4.2.1.5 Author Entity 30
4.2.1.6 Role Entity 30
4.2.1.7 Contributor Entity 31
4.2.1.8 Language Entity 31
4.2.1.9 Annotation Entity 32
4.2.1.10 Users Entity 32
4.2.1.11 Coverage Entity 32
4.2.1.12 System_req Entity 33
4.2.1.13 Identifier Entity 33
4.2.1.14 Classification Entity 34
4.2.2 Identifying Primary Keys 34
4.3 Physical Implementation 35
4.3.1 Implementation of the Semantic Header Database 35
4.3.2 Implementation of Intermediate Tables between ASHG Subsystem and Resource Registration Subsystem 39
4.4 Performance Considerations 42
4.4.1 Add Column Indices for Resource Searching 42
4.4.2 Optimizing Concurrent Select and Insert Operations 43
4.4.3 Using Persistent Connection 43
4.5 Implementation of Web Application with PHP 44
4.5.1 Connecting to MySQL Database 44
4.5.2 Executing SQL Query 45
4.4.2 Closing Persistent Connection 45

5 User Interactions with the CINDI System 46

5.1 Registration 46
5.1.1 Contributor Registration 46
5.1.2 User Registration 49
5.1.3 Login 49
5.1.4 Resources Registration 51
5.1.4.1 Manual Registration 51
5.1.4.2 Automatic Registration 55
5.2 Semantic Header Search 57
5.2.1 Search Query Structure 57
5.2.2 Performance Considerations for Search 59
5.2.3 User Interactions with CINDI Search Sub-System 60
5.3 Annotation 62

6 Conclusion and Future Work 65

6.1 Conclusion 65
6.2 Contribution of this Report 65
6.3 Future Work 66

Bibliography 67

List of Figures

Figure 1: PHP Script Example 13
Figure 2: 2-tier Architecture 18
Figure 3: 3-tier Architecture 18
Figure 4: The architecture of the CINDI system. 23
Figure 5: The E-R diagram of the Semantic Header Database 26
Figure 6: Semantic Header database implementation 39
Figure 7: Implementation of intermediate tables between the ASHG subsystem and the Resource Registration subsystem 42
Figure 8: Indices of the Semantic Header database 43
Figure 9: PHP scripts and MySQL example 44
Figure 10: Contributor registration 47
Figure 11: PHP/MySQL Code for contributorID existing checking 48
Figure 12: Inserting contributor's personal information in Semantic Header database 48
Figure 13: The user login interface 50
Figure 14: The processing for user login 51
Figure 15: Resource Manual Uploading 52
Figure 16: Generate pull down menu for language 54
Figure 17: The PHP/MySQL code to insert data in Semantic Header database 55
Figure 18: Part of PHP/MySQL code for automatic upload 57
Figure 19: The Current BNF Query Structure 59
Figure 20: Semantic Header Searching 61
Figure 21: HTML Query translated into SQL Query 61
Figure 22: Search Result 62
Figure 23: Annotation 63
Figure 24: The processing for annotation 64

List of Tables

Table 1: Comparison: Reading 2,000,000 rows by index 10
Table 2: Comparison: Inserting (350,768) rows 11
Table 3: Primary key for entities 35

Chapter 1

Introduction

1.1 Overview of the Virtual Library

A virtual library [BCD97] can be defined as the aggregation of remote access to the contents and services of libraries and other information resources, combining an on-site collection of current and heavily used materials in electronic form, with an electronic network which provides access to, and delivery from, external worldwide library and commercial information and knowledge sources. In essence, the user is provided the effect of a library through a synergy created by bringing technologically together the resources of many libraries and information services.

1.1.1 The World-Wide Web (WWW)

The Web makes it possible to access a virtual library anywhere on the Internet using a URL (Uniform Resource Locator). The URL is the address of a resource, or file, available on the Internet. A browser such as Netscape Navigator or Microsoft Internet Explorer (IE) interprets these resources or files and displays them on a screen. It also allows the user to navigate to other related documents by choosing links in the HTML (HyperText Markup Language) file. A related collection of such resources or files is called a Web site. The computer that holds these resources or files is called a Web Server. Currently most organizations maintain a Web site.

1.1.2 Web Servers and Databases

The Web is the cornerstone of E-commerce. Many organizations already offer E-commerce application through the Internet. Customers can place orders and pay bills 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 also provide an entry point to services based on information in a database. URL requests at a given site are processed by Web servers. According to the request, the server takes different actions. 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 browser. After the requester fills in the form, the data filled in the form by the user is returned to the Web server, and these information can be used as parameters to a program executed on the server. The results of the program can be used to create a customized HTML document that is returned to the browser.

The database connection can be made by invoking a program on the server. The program invoked by the Web server can generate a request to a database system. This capability allows users to make services that rely upon database access available over the Web. It is very common to access a database through the Web, especially in E-commerce applications.

1.2 CINDI System

1.2.1 Introduction

In order to make effective use of the wealth of information on the Internet, users need ways to locate pertinent information. Today, many search engines are available on the Internet, including AltaVista, Yahoo, Google, Northernlight, Lycos, Harvest, EINet, and Galaxy, etc. However, none of these systems is always successful in retrieving the documents sought [YZ01]. The reason 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. The existing search systems have several drawbacks:

1. They do not take into account the heterogeneous nature of the virtual libraries. The same document may be described in different ways in different virtual libraries.
2. They are not easy to use for new users. If new users do not know how to express their query criteria correctly, they may spend a long time getting their result.
3. Users are required to reference documents by words or phrases instead of by title, author, subject, date, etc.
4. The content of a document or file is not precisely described by many systems. Normally, before a user knows whether a specific file is pertinent, he often has to retrieve the pertinent source document / file and browse it.
5. To find relevant information is not easy. To address this problem, many systems use indexing systems, but most of these systems place a significant, unnecessary load on information servers and network links, because they use expensive object retrieval protocols to gather indexing information, and they do not coordinate information

gathering amongst each other. Each indexer gathers all of the information it needs, without trying to share overlapping information with other indexers [YZ01].

6. Most systems offer little support for customizing how they handle different information formats and index/search schemes.

1.2.2 CINDI: The Project

The CINDI project is a system proposed by Dr. Desai [BS94]. The objective of the project is to build a system that enables any resource contributor to catalog his own resource and any user to search for hypermedia documents using a typical search item 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 called the Semantic Header Database System. In addition the catalog for subjects will provide help when the user catalogs his index. The proposed system is characterized by the following:

1. A standardized metadata to describe each information resource – the Semantic Header [BCD95].
2. A Semantic Header Database that stores the Semantic Headers.
3. A Semantic Header registering system to register the Semantic Header into the Semantic Header Database System.
4. A search system that allows query entry for information discovery.
5. An annotation system.

6. A subject table that stores information about subjects classified using a standard cataloging scheme.
7. A set of graphical user interfaces to let a user register an index entry, make annotations to any index entry, and execute a search. The user can also browse the source resource using a Browser.

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

1. The Semantic Header allows the indexation of documents accessible by any protocol.
2. The Semantic Header includes annotations of reviewers and other users, thus offering the possibility of a more informed decision as to the pertinence of the source resource [YZ01].
3. The Semantic Header syntax offers a way to register standardized keywords chosen by the contributor of the resource. These make searching uniform. Since the contributor of the resource performs the registration of the Semantic Header in the database, it has the following advantages: low cost, accuracy, and efficiency [YZ01].
4. The Semantic Header is designed to describe each resource exactly. This abstract information provides a better idea of the resource than an excerpt prepared by systems such as Lycos, Google, etc.
5. The distributed nature of the Semantic Header database provides for scalability.

1.3 Organization of the Report

This report describes the architecture of the CINDI system, the design and implementation of the database system and the interface between the CINDI database system and the Web Browser.

Chapter 2 presents the background of the Web-based CINDI system. The first section gives an overview of information retrieval. The MySQL database is presented in section 2.2 and PHP is presented in section 2.3. Section 2.4 gives a brief view of the Apache server and section 2.5 presents the Semantic Header package ASHG (Automatic Semantic Header Generator).

Chapter 3 introduces the general idea of the CINDI system, including the 3-tier structure, the system design. Chapter 4 presents database system design, including logical model, physical implementation, performance considerations and the connection from PHP code. Section 4.1 gives an overview of data modeling; section 4.2 gives the design of the logical model. Section 4.3 gives the physical implementation of the database system. Section 4.4 gives the consideration of performance and section 4.5 presents the implementation of Web application with PHP.

The user interaction for registration, resource search and annotation are present in Chapter 5. Also the searching strategy is introduced in Chapter 5. Finally in Chapter 6 we draw our conclusions and provide suggestions for future work.

Chapter 2

Background

2.1 Information Retrieval

2.1.1 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 that 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 [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.

2.1.2 Semantic Header

The heart of any bibliography or indexing system is the record that is kept for each item being indexed. Standardization of a bibliographic entry better allows people to exchange information through a virtual library. The index structure for the indexing system was proposed by Desai and is called the Semantic Header [BCD95]. It was originally presented at the First International World Wide Web Conference in Geneva for WWW

resources. Since then, it has been extended to other resources accessible directly on the Internet [BCD95].

The Semantic Header may be considered as an application of the Standard Generalized Markup Language (SGML) [MAM01]. The objective of the Semantic Header is to include those elements that are most often used in the search for an information resource. Since the majority of searches begin with a title, the name of the author(s), subject and sub-subject [MAM01], the entry of these elements have been made mandatory in the Semantic Header. The abstract and annotations are, also, relevant in deciding whether or not the resource would be useful. These items are therefore also included.

2.2 MYSQL Database

2.2.1 What is MySQL?

MySQL, the most popular Open Source SQL database, is provided by MySQL AB [MSQL+]. MySQL AB is a commercial company that builds its business providing services around the MySQL database. The SQL part of MySQL stands for "Structured Query Language" - the most common standardized language used to access databases.

MySQL: A database management system.

A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, a database management system such as MySQL is necessary. Since computers are very efficient at handling large

amounts of data, database management plays a central role in computing, as stand-alone utilities, or as parts of other applications [PDMW99].

MySQL: A relational database management system

A relational database stores data in separate tables rather than putting all the data in one big storeroom. This adds speed and flexibility. The tables are linked by defined relations making it possible to combine data from several tables on request.

MySQL: Open source software

Open Source means that it is possible for anyone to use and modify the same code. MySQL is freely available on the Internet [1]. Anybody so inclined can study the source code and change it to fit their needs. MySQL uses the GPL (GNU General Public License), to define what may or may not be done with the software in different situations.

MySQL: A client/server system

MySQL is a client/server system that consists of a multi-threaded SQL server that supports different backbends, several different client programs and libraries, administrative tools, and several programming interfaces.

2.2.2 Why use MySQL?

MySQL is very fast, reliable, and easy to use. MySQL also has a very practical set of features developed in very close cooperation with users. MySQL was originally

---

[1] URL: http://www.mysql.com/downloads

developed to handle very large databases, much faster than existing solutions and has been successfully used in highly demanding production environments for several years [MSQL+]. Though under constant development, today MySQL offers a rich and very useful set of functions. The connectivity, speed, and security make MySQL highly suited for accessing databases on the Internet.

Table 1 and 2 shows the results of a performance comparison of MySQL to other database managers. Note that all experiments have been performed on the same NT 4.0 machine [MSQL+]. Both experiments clearly show that MySQL is much faster. For example, table 1 shows that informix_odbc takes about 260 times longer than mysql_odbc to read 2,000,000 rows. Table 2 shows oracle_odbc takes about 18 times longer than mysql_odbc to insert 350,768 rows.

Database	Seconds
mysql	367
mysql_odbc	464
db2_odbc	1206
informix_odbc	121126
ms-sql_odbc	1634
oracle_odbc	20800
solid_odbc	877
sybase_odbc	17614

Table 1: Comparison: Reading 2,000,000 rows by index

Database	Seconds
mysql	381
mysql_odbc	619
db2_odbc	3460
informix_odbc	2692
ms_sql_odbc	4012
oracle_odbc	11291
solid_odbc	1801
sybase_odbc	4802

Table 2: Comparison: Inserting (350,768) rows

2.2.3 The Main Features of MySQL

The following list describes some of the important characteristics of MySQL:

1. Fully multi-threaded using kernel threads. This means it can easily use multiple CPUs if available.
2. Multiple APIs support: C, C++, Eiffel, Java, Perl, PHP, Python and Tcl.
3. Multiple platforms support.
4. Many column types: signed/unsigned integers 1, 2, 3, 4, and 8 bytes long, FLOAT, DOUBLE, CHAR, VARCHAR, TEXT, BLOB, DATE, TIME, DATETIME, TIMESTAMP, YEAR, SET, and ENUM types [MSQL+].
5. Very fast join operations using an optimized one-sweep multi-join.
6. Full operator and function support in the SELECT and WHERE parts of queries.
7. SQL functions are implemented through a highly optimized class library and should be as fast as possible! Usually there is no memory allocation at all after query initialization.

8. A privilege and password system that is very flexible and secure, and allows host-based verification. Passwords are secure because all password traffic is encrypted when connected to a server.
9. Open Database Connectivity (ODBC) support.
10. Very fast B-tree disk tables with index compression.
11. Up to 32 indices per table are allowed. Each index may consist of 1 to 16 columns or parts of columns.
12. In-memory hash tables, which are used as temporary tables.
13. All columns have default values. INSERT can be used to insert a subset of a table's columns; those columns that are not explicitly given values are set to their default values.
14. No memory leaks. MySQL has been tested with Purify, a commercial memory leakage detector [MSQL+].
15. Aliases on tables and columns are allowed as in the SQL92 standard [MSQL+].

2.3 PHP

2.3.1 What is PHP?

PHP (Hypertext PreProcessor) is a server-side HTML-embedded scripting language. The PHP code is enclosed in special start tag (<? php) and end tag (?>) that allow user to jump into and out of PHP mode. For example, the following short code (Figure 1) will display "Welcome to CINDI System!" on the screen.

```
<html>
      <head>
            <title>Example</title>
      </head>
      <body>
            <?php
                  echo "Welcome to CINDI System!";
            ?>
      </body>
</html>
```

Figure 1: PHP Script Example

What distinguishes PHP from other scripting languages like client-side JavaScript is that the code is executed on the server. If there were a script similar to the above on the server, the client would receive the results of running that script, and the underlying code can't be determined by the users. The web server can even be configured to process all the HTML files with PHP, and then there's really no way that users can determine what the underlying code may be.

2.3.2 What can PHP do?

At the most basic level, PHP can do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies.

The strongest and most significant feature in PHP is its support for a wide range of databases. Writing a database-enabled web page is incredibly simple. The following databases are mainly supported: MySQL, Oracle, dBase, PostgreSQL, Sybase, IBM DB2, Informix etc [PHP+].

PHP also has support for talking to other services using protocols such as IMAP, SNMP, NNTP, POP3, HTTP and countless others.

2.4 Apache Server

Apache server means "A PAtCHy server". It is based on some existing code and a series of "patch files". The Apache server is a powerful, flexible, HTTP/1.1 compliant web server. It can be run on Windows NT/9x, Netware 5.x, OS/2, and most versions of Unix, as well as several other operating systems. It is an open source and it is freely available on the Internet [2].

According to the Netcraft survey, the Apache server has managed to capture over 58% of the server market and has been the most popular web server on the Internet since April 1996 [NTCR+]. Apache server has many advantages over other web servers, such as providing full source code and an unrestrictive license. It is also full of features. For example, it is compliant with HTTP/1.1 and extensible with third-party modules, and it provides its own APIs to allow module writing. Other useful features that have made it a popular web server include the capability to tailor specific responses to different errors, its support for virtual hosts, URL rewriting and aliasing, content negotiation and its support for configurable, reliable piped logs that allows users to generate logs in a format they want.

---

[2] URL: http://httpd.apache.org/dist/

2.5 ASHG

Automatic Semantic Header Generator (ASHG) is a program package that automatically generates Semantic Header for HTML, Latex, Text and RTF documents [SSD98]. It is part of the CINDI system and was developed specifically for the CINDI system. By using ASHG, it provides the document's contributor with
an initial set of subject classifications and a number of components of the Semantic Header for the document.

ASHG extracts some meta-information from a document and stores it in a Semantic Header. For instance, when a contributor uploads a new document to the CINDI system, fields such as document's title, abstract, keywords, subjects, dates, author's information are extracted. Using frequency occurrence and positional schemes, ASHG measures the significance of the words found in the previously mentioned list. The major steps used by ASHG are:

1. Recognizing Document Type: Currently, ASHG understands HTML, Latex, Text and RTF documents.
2. Applying ASHG's Extractor: A summarizer corresponding to the type of document applies the input document.
3. Classifying ASHG's Document: The document is assigned subject headings.
4. Output Semantic Header: The generated meta-information is stored in a Semantic Header.

5. Presenting Semantic Header: The generated Semantic Header is queried from database, sent to a HTML form and presented on the contributor's screen.
6. Validating Semantic Header: The contributor validates the Semantic Header generated by ASHG.

Chapter 3

Architecture of the CINDI System

3.1 3-tier Architecture

3.1.1 Introduction

In the traditional 2-tier model (figure 2), the client application contains the GUI (Graphic User Interface) code, business logic, security code, data access logic, etc. And the server side contains the database management system, some server side security code, etc. This type of architecture is suitable for deployment to a small group of users, but there are many problems in a large-scale application system:

1. Deployment and maintenance problem: Each time a change is made to a component or a database structure, every client has to be updated.
2. Scalability problem: All clients are hitting the same back-end data source and services, competing for resources.

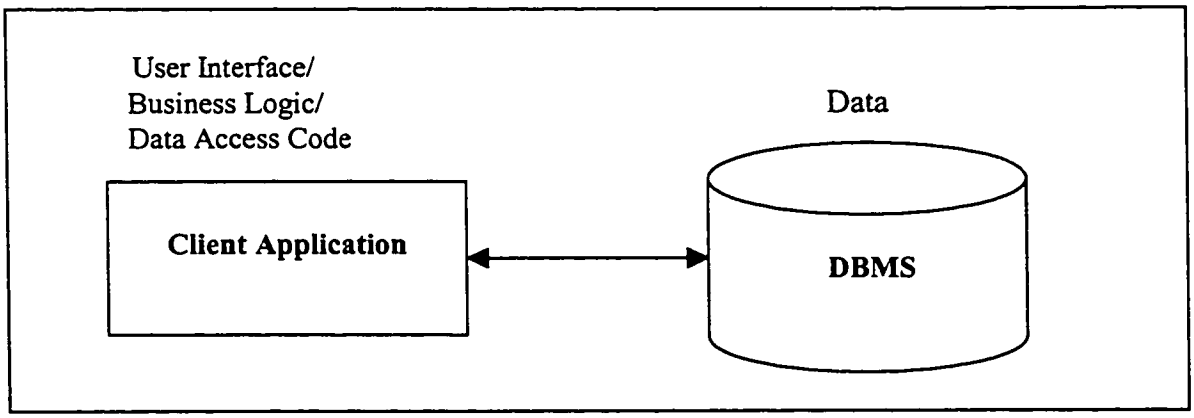

Figure 2: 2-tier Architecture

Compared with 2-tier architecture, 3- and n-tier architectures endeavor to overcome these problems. This goal is achieved primarily by moving the business logic from the client back to the server. Figure 3 shows a simplified form of 3-tier architecture.

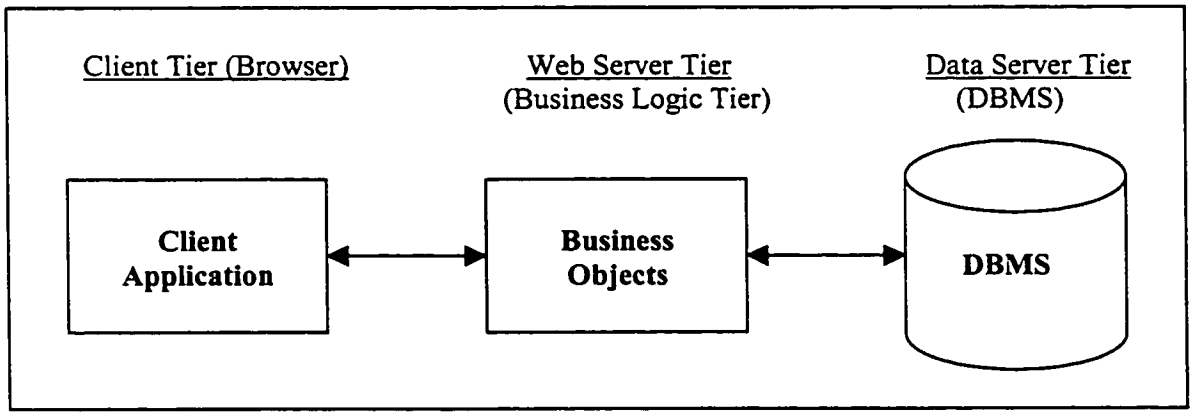

Figure 3: 3-tier Architecture

Client Tier (Browser)

The Client Tier is responsible for the presenting of data, the receiving of user events and the controlling of user interfaces. The actual business logic has been moved to Web Server Tier.

Web Server Tier (Business Logic Tier)

The Web Server Tier is new, i.e. it is not present in 2-tier architecture in this explicit form. Business-objects that implement the business rules "live" here, and are available to the client-tier. This level now forms the central key to solving 2-tier problems. This tier protects the data from direct access by the clients.

Data Server Tier

The Data Server Tier is responsible for data storage. Besides the widespread relational database systems, existing legacy systems databases are often reused here.

It is important to note that boundaries between tiers are logical. It is quite easily possible to run all three tiers on one and the same (physical) machine. The main importance is that the system is neatly structured, and that there is a well-planned definition of the software boundaries between the different tiers.

3.1.2 The Advantages of 3-tier Architecture

The advantages for 3-tier include:

1. Clear separation of user-interface-control and data presentation from application-logic.

2. Re-definition of the storage strategy would not influence the clients.
3. Business-objects and data storage should be brought as close together as possible, ideally they should be together physically on the same server.
4. In contrast to the 2-tier model, where only data is accessible to the public, business-objects can place applications-logic or "services" on the network.
5. As a rule, servers are "trusted" systems. Their authorization is simpler than that of thousands of "untrusted" client-PCs. Data protection and security is simpler to obtain.
6. Dynamic load balancing: if bottlenecks in terms of performance occur, the server process can be moved to other servers at runtime.
7. Change management: it's easy - and faster - to exchange a component on the server than to furnish numerous PCs with new program versions.

For all these reasons, the CINDI system has been made adaptable to 3-tier architecture, including Browser, Apache Web server and MySQL Database server.

3.2 System Design

System design is the high-level strategy for solving the target problem and building a solution. It includes decisions about the organization of the system into subsystems. It produces the system architecture.

The CINDI System is a web-based, 3-tier application. Users access the web site by entering the web address in a web browser and the web browser communicates with the web server through the HTTP protocol. The Web server runs PHP scripts and returns a

dynamic web page to the web browser based on the user requirements. If the request accesses data from the database, the web base middleware accesses the database. If not, the web base middleware responds with a corresponding web page to the web browser.

The CINDI system has six subsystems:

Resource Registration sub-system (RR): This subsystem is responsible for resource registration. The resource contributors use this subsystem to register their resources into the CINDI system.

Security Control sub-system (SC): This subsystem is used to register contributors' and users' information and to perform access authorization checking. Only registered contributors/users can use the CINDI system.

ASHG sub-system (ASHG): ASHG subsystem generates Semantic Header for HTML, Latex, Text and RTF documents. It provides the document's contributor an initial set of subject classifications and a number of components of the Semantic Header for the document.

Semantic Header Database sub-system (DB): All information including Semantic Headers, contributors/users personal information and user's annotation, etc. is stored in the Semantic Header database sub-system.

Search sub-system (SR): This subsystem is responsible for resource searching and for returning results to users.

Annotation sub-system (AT): Registered users can use this subsystem to read other users' annotations and/or make their own annotations.

As shown in figure 4, all new resource contributors as well as users must register their personal information using the Security Control sub-system (SC). After registering, each registrant will get a unique account and password by the e-mail provided during registration. Only after successfully registering in the CINDI system, contributors and users can use the system. For resource contributors, the Resource Registration sub-system (RR) is used to register resources into the CINDI system. Alternatively, the ASHG sub-system (ASHG) may be used to automatically generate a draft Semantic Header for HTML, Latex, Text and RTF documents and presents it to the resource contributor for validation. All the information that includes Semantic Headers, user information, resource contributor information, user's annotation, etc. are stored in Semantic Header database sub-system (DB). Registered users can use the Search sub-system (SR) to search information in the Semantic Header database, to get the resources and make annotations through the Annotation sub-system (AT).

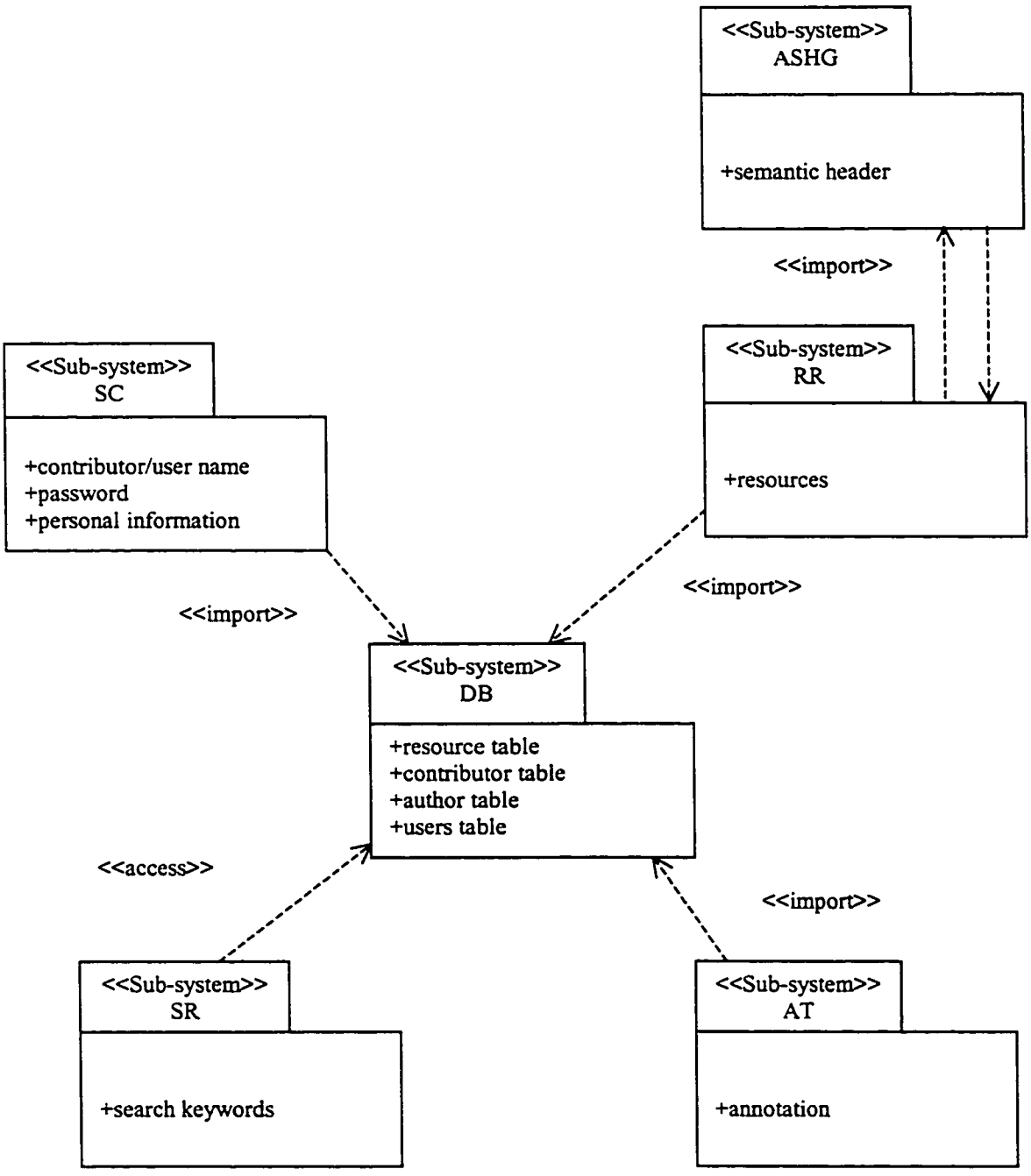

Figure 4: The architecture of the CINDI system.

Chapter 4

Semantic Header Database System Design and Implementation

4.1 Data Modeling and E-R Model

Data Modeling is the bridge between the real world and an information processing system, and helps to determine what data is required for a successful project. In database, a data model represents data structures and their characteristics, relations, constraints, and transformations. A well-abstracted data model will be economical and flexible to maintain and enhance.

The E-R (Entity Relationship) Model is a detailed, logical representation of the data modeling. An entity is a thing that can be distinctly identified. A relationship is an association between two or more entities. The E-R Model shows the entities within the database, the associations (or relationships) among the entities, and the attributes or properties of the entities and their relationships.

In the CINDI system, the E-R Model is used to present the data model of the Semantic Header.

4.2 Logical Model

It is the essence of the data model for identifying the correct entities and relationship between entities.

4.2.1 Identifying Entities and their Attributes

For cataloging and searching, a Meta-data description called a Semantic Header is used to describe an information resource [BS94]. As mentioned in section 2.1.2, the Semantic Header includes title, name of the authors, subjects, keywords, abstract and annotation, etc. Figure 5 shows E-R diagram of the Semantic Header Database. In the E-R diagram, each box is an entity and each entity is represented by a set of attributes. The entities and their attributes for the Semantic Header database are briefly described in the following sections.

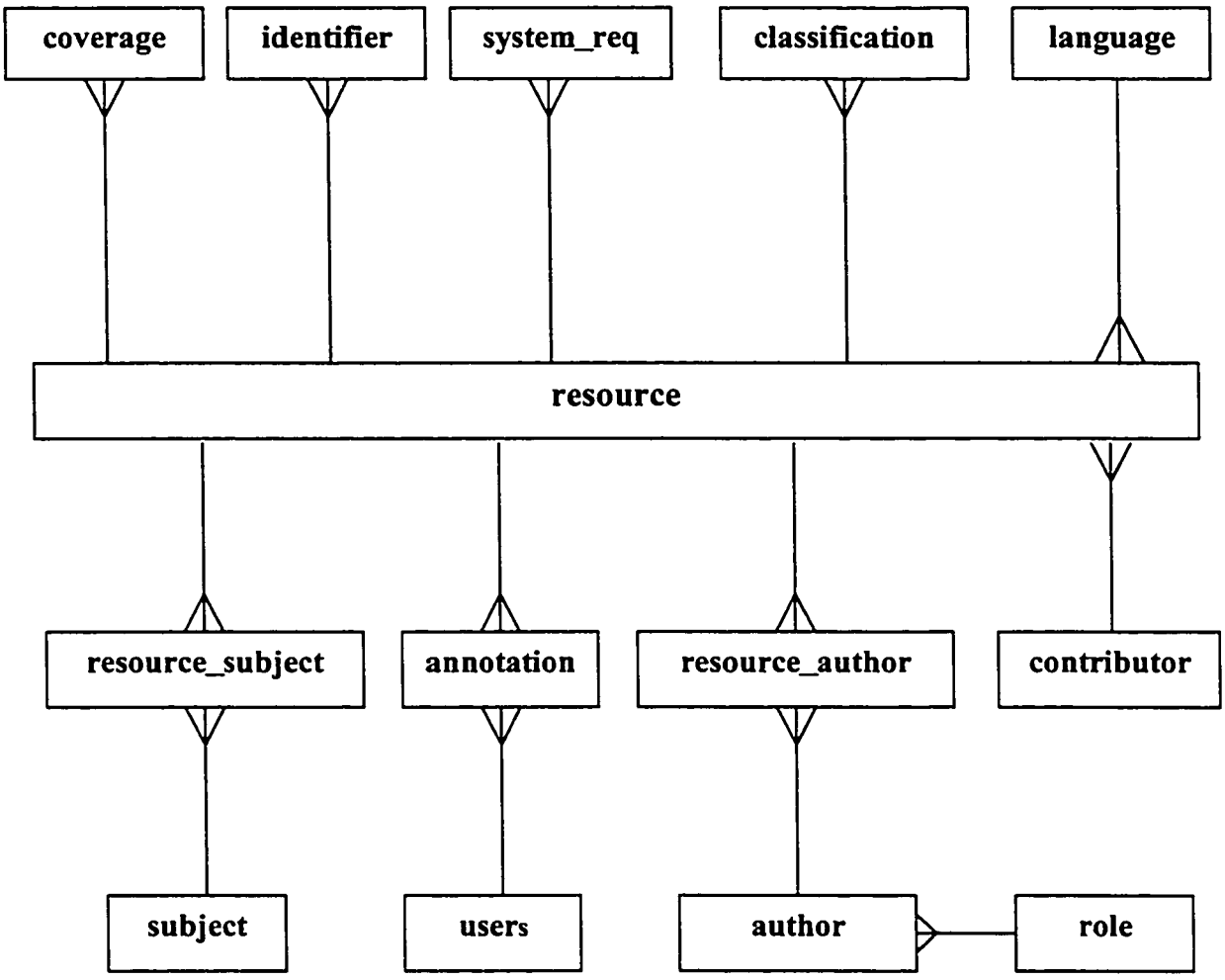

Figure 5: The E-R diagram of the Semantic Header Database

4.2.1.1 Resource Entity

The resource entity is the main entity for CINDI system. It includes the major Semantic Header information described by the following fields:

Title, Alt_title:

The title is a name given to the resource by its creator or a short description of the resource. It is a required field. The alt_title field is used to indicate an secondary title or a alternate short description of the resource.

Keyword:

This field is used for keeping keyword information for the resources. Each keyword is separated by a comma. It is used for keyword searching.

Language:

Identifies the language of the resource.

Create_date, Expiry_date, Upload_date, Update_date:

The date of creation, the date of expiration, the date of uploading and the date of updating for the resource. The create_date and the upload_date are required fields. The expiry_date is a optional field. The upload_date and the update_date are identified by a system-generated date.

Version:

The version number, if any, is given in this field.

Resource_format, Size:

The resource_format is used to describe the physical or electronic format of the resource, like ASCII, PDF, Postscript, GIF, etc. And the size field is used to give the corresponding value or size of the resource. Both of them are optional fields.

Annotation:

Text field for annotation when the resource is uploaded. Resource contributor or the author(s) makes this annotation.

Abstract:

Text field for abstract when the resource is uploaded. This is a required field. It is used for keyword searching.

Filename:

The file name of the resource. This is a required element. This is the link to the real resource.

Contributor_id:

The contributor ID for the resource. This is a required element.

4.2.1.2 Subject Entity

The details about each subject are indicated here. A subject of the resource is composed of three sublevels: subject, sub-subject, sub_sub-subject. Each subject includes a subject ID and a subject name.

4.2.1.3 Resource_subject Entity

One resource may belong to many different subjects. For instance, one resource may belong to Computer Science, Information storage and retrieval, Web_based online information services. It may also belong to Computer Science, Information storage and retrieval, Online information services. Meanwhile, one subject may belong to many resources. Therefore, there is a many—many relationship between the entity resource and the entity subject. In order to eliminate the many—many relationship, one intermediate entity should be added. This is the resource_subject entity that changes the many--many relationship between the resource entity and the subject entity to two one--many relationships: one—many relationship between resource entity and resource_subject entity, and one—many relationship between subject entity and resource_subject entity.

It includes the primary keys of the resource entity and the subject entity.

4.2.1.4 Resource_author Entity

One resource may have more than one author, and one author may register many resources in the CINDI system, so there is a many—many relationship between the entity resource and the entity author. In order to eliminate the many—many relationship, one

intermediate entity should be added here. This is the resource_author entity that changes the many--many relationship between the resource entity and the author entity to two one--many relationships: one—many relationship between resource entity and resource_author entity, and one—many relationship between author entity and resource_author entity.

It includes the primary keys of the resource entity and the author entity.

4.2.1.5 Author Entity

The details about the author(s) and/or other agent(s) responsible for the resource. It includes these fields: name, organization, address, phone number, e-mail address and role of the author or other agent, etc.

4.2.1.6 Role Entity

The details about the role of the author or other agent. Typical values for the role of the author or other agent could be author, co-author, designer, editor, programmer, creator, artist, publisher, etc. It includes the fields: role_id and the role name. Instead of keeping the particular role in the author entity, the role ID is stored in the author entity. In this way, it will be more flexible for role processing in the future, like query by role, sort by role or add/delete role, etc.

One author or other agent only has one role in the same resource, but one role may belong to many different authors or other agents. Therefore, there is one—many relationship between the role entity and the author entity.

4.2.1.7 Contributor Entity

The detailed information about the registered contributors. Contributors who want to upload resources must supply pertinent information, such as login account, last name, first name, organization name, detailed address information, e-mail, etc.

Obviously, there is one—many relationship between the contributor entity and the resource entity.

4.2.1.8 Language Entity

Detailed information about language is indicated here. Instead of keeping the particular language, like English, French, etc., in the resource entity, the language ID is stored in the resource entity. In this way, it will be more flexible for language processing in the future, like query by language, sort by language or add/delete language, etc.

Currently, if one resource has multiple language versions, there will be multiple Semantic Headers in the CINDI system. Therefore, there is one—many relationship between the language entity and the resource entity.

4.2.1.9 Annotation Entity

The annotations are made by users for a particular resource. Fields include: resource ID, user account and text field for annotation.

One resource may have many annotations made by many users. Therefore, there is one—many relationship between the resource entity and the annotation entity.

4.2.1.10 Users Entity

Users who registered for searching resources are indicated by user account, last name, first name, detailed address information, e-mail address etc.

Obviously, there is one—many relationship between the users entity and the resource entity.

4.2.1.11 Coverage Entity

It consists of a domain (target audience, coverage in a spatial and/or temporal sense, etc) and the corresponding value.

One resource may have more than one coverage information. For example, one resource may have both coverage in a spatial and also a temporal sense. But one coverage information can only be mapped to one particular resource. Therefore, there is a one—many relationship between the resource entity and the coverage entity.

4.2.1.12 System_req Entity

The system requirement such as hardware and software required to access, use, display or operate the resource is included in the Semantic Header. It consists of a domain of the system requirements (possible value are: hardware, software, network, protocol, etc.) and the corresponding value.

One resource may have more than one system requirements. For instance, one resource may have hardware and software requirements as well as network requirements. But one system requirement can only be mapped to one particular resource. Therefore, there is a one—many relationship between the resource entity and the system_req entity.

4.2.1.13 Identifier Entity

It includes two fields: the domain and the corresponding value. The possible domain values are: ISBN, URL, ISSN, etc.

One resource may have more than one identifier information. For example, one resource may be identified by a URL and a ISBN. But one identifier information can only be mapped to one particular resource. Therefore, there is a one—many relationship between the resource entity and the identifier entity.

4.2.1.14 Classification Entity

It includes two fields: the domain and the corresponding value. The possible domain values are: legal, security level, etc.

One resource may have more than one classification information. For example, one resource may have security restriction and copyright status. But one classification information can only be mapped to one particular resource. Therefore, there is a one—many relationship between resource entity and classification entity.

4.2.2 Identifying Primary Keys

A primary key is the combination of the values of one or more attributes that collectively and uniquely identify an entity.

For the resource entity, a number field resource_id is added to identify each resource and it is chosen as the primary key for the resource. In this way, it is convenient to implement. Each contributor has his own personal information. The contributor ID is used to uniquely identify each contributor. For the contributor's convenience, he can choose his own ID. In case of a duplicate, the contributor needs to choose another ID until he meets the unique requirement. The same process applies for the users entity. The user_id is the primary key. For the author entity, obviously, author's name can not be used as the primary key because two different authors may have the same name. So, a number field author_id is added to identify each author and it is chosen as the primary key of author. If two authors have the exact same information, like name, organization,

address, phone number and e-mail address, etc., they will be treated as one author and there is no need for new author_id. Otherwise, they will be treated as two different authors. For the subject entity, a number field is added as the primary key to uniquely identify each subject. Table 3 shows the primary key for entities.

Entity Name	Primary Key
resource	resource_id
contributor	contributor_id
author	author_id
users	user_id
resource_author	resource_id, author_id
role	role_id
annotation	resource_id, user_id
subject	subject_id,
resource_subject	resource_id, subject_id
language	language_id

Table 3: Primary key for entities

4.3 Physical Implementation

4.3.1 Implementation of the Semantic Header Database

In MySQL, entities and their attributes evolve into tables and columns. Figure 6 shows the MySQL implementation for the Semantic Header database. MySQL function AUTO_INCREMENT is used to get a sequence number as the primary key for resource_id, author_id and language_id, etc. As mentioned in section 4.2.2, each author

has a unique author_id. The author_ids that are relative to the resource are stored in table resource_author. Each author has one entry that includes resource_id and author_id. If there are three authors in one resource, then there are three tuples in this table. Through table resource_author, all authors' information of the resource can be obtained. For subjects in the resource, through table resource_subject, all subjects of the resource can be obtained. For the annotation table, there are three attributes: resource_id, user_id and annotation. Each annotation identified by the resource_id and the user_id. Therefore, all the annotations of the resource can be easily obtained. For the table identifier, table coverage, table system_req and table classification, they have one—many relationship with the table resource. By putting the resource_id in these tables, all these information about the resource can be obtained.

```
create table resource (
            resource_id       int(10) unsigned NOT NULL AUTO_INCREMENT,
            title             varchar(100) NOT NULL,
            alt_title         varchar(100),
            language_id       int(10) unsigned,
            keyword           varchar(200),
            created_date      date NOT NULL,
            upload_date       date NOT NULL,
            expiry_date       date,
            last_update       date NOT NULL,
            version           varchar(50),
            source            varchar(200),
            annotation        text,
            size              int(10) unsigned,
            resource_format   varchar(20),
            abstract          text NOT NULL,
            filename          varchar(100) NOT NULL,
            contributor_id    varchar(20)  NOT NULL,
            primary key(resource_id));
```

```
create table author (
            author_id      int(10) unsigned NOT NULL AUTO_INCREMENT,
            name           varchar(50)  NOT NULL,
            organization   varchar(100),
            address        varchar(50),
            apt_no         varchar(20),
            city           varchar(30),
            province       varchar(30),
            country        varchar(30),
            p_code         varchar(20),
            e_mail         varchar(50),
            phone          varchar(20),
            role           varchar(20)  NOT NULL,
            primary key(author_id));

create table role (
            role_id   int(10) unsigned NOT NULL AUTO_INCREMENT,
            role_name varchar(30),
            primary key(role_id));

create table subject (
            subject_id    int(10) unsigned NOT NULL,
            subject_name  varchar(80) NOT NULL,
            primary key(subject_id));

create table resource_subject (
            resource_id     int(10) NOT NUILL,
            subject         varchar(50) NOT NULL,
            sub_subject     varchar(50),
            sub_sub_subject varchar(50));

create table resource_author (
            resource_id       int(10) unsigned NOT NULL,
            author_id         int(10) unsigned NOT NULL,
            primary key(resource_id, author_id));

create table language (
            language_id    int(10) unsigned NOT NULL AUTO_INCREMENT,
```

```
            language_name  varchar(20),
            primary key(language_id));

create table users (
            user_id             varchar(20) NOT NULL,
            passwd              varchar(20) NOT NULL,
            fname               varchar(50) NOT NULL,
            lname               varchar(50) NOT NULL,
            organization        varchar(100),
            address             varchar(50),
            apt_no              varchar(20),
            city                varchar(30),
            province            varchar(30),
            country             varchar(30),
            p_code              varchar(20),
            e_mail              varchar(50),
            phone               varchar(20),
            primary key(user_id));

create table contributor (
            contributor_id    varchar(20)  NOT NULL,
            passwd            varchar(20)  NOT NULL,
            fname             varchar(50)  NOT NULL,
            lname             varchar(50)  NOT NULL,
            organization      varchar(100),
            department        varchar(100),
            address           varchar(50),
            apt_no            varchar(20),
            city              varchar(30),
            province          varchar(30),
            country           varchar(30),
            p_code            varchar(20),
            e_mail            varchar(50),
            phone             varchar(20),
            primary key(contributor_id));

create table annotation (
            resource_id int(10) unsigned NOT NULL,
```

```
            user_id     varchar(20) unsigned NOT NULL,
            annotation  text);

create table coverage (
            resource_id int(10) unsigned NOT NULL,
            coverage_domain   varchar(50),
            coverage_value    varchar(200));

create table system_req (
            resource_id       int(10) unsigned NOT NULL,
            system_req_domain varchar(50),
            system_req_value  varchar(200));

create table identifier (
            resource_id       int(10) unsigned NOT NULL,
            identifier_domain varchar(50),
            identifier_value  varchar(200));

create table classfication (
            resource_id int(10) unsigned NOT NULL,
            clasf_domain      varchar(50),
            clasf_value       varchar(200));
```

Figure 6: Semantic Header database implementation

4.3.2 Implementation of Intermediate Tables between ASHG Subsystem and Resource Registration Subsystem

In order to get a general interface with the ASHG (Automatic Semantic Header Generator) package, several tables are used to temporarily store Semantic Header information that are automatically generated by ASHG. After the contributor loaded a resource, the CINDI system will generate a unique ID for it. This ID is used to identify the resource in the tables. Once ASHG has stored the Semantic Header into the

intermediate tables, this ID is used to query the corresponding information. A C program is used to automatically clean these tables after the Semantic Header has shown on the contributor's screen for 30 minutes. The main information, such as title, keyword, subject, created_date and abstract, etc., is stored in the sh table. The author's information of the resource is stored in the sh_author table. Information about coverage, identifier, system requirements and classification is stored in the sh_covearage table, the sh_indentifier table, the sh_system_req table and the sh_classification table correspondingly. Figure 7 shows the MySQL implementation of intermediate tables between the ASHG subsystem and the Resource Registration subsystem. Actually, the structures of these tables are very similar to the tables: resource, author, coverage, system_req, identifier and classification. In the sh table, there are no update_date and last_update fields. Why are these tables needed? The reason is that, the Semantic Header information generated by the ASHG is only a draft. The contributor needs to check it and confirm it. This information can't be directly stored in the Semantic Header database, because this will give the database false information and render the Semantic Header database useless.

```
create table sh (
            sh_id          int(10) unsigned NOT NULL AUTO_INCREMENT,
            title              varchar(100) NOT NULL,
            alt_title          varchar(100),
            subject_1          varchar(50) NOT NULL,
            sub_subject1_1     varchar(50),
            sub_subject2_1     varchar(50),
            subject_2          varchar(50),
            sub_subject1_2     varchar(50),
            sub_subject2_2     varchar(50),
```

```
            subject_3          varchar(50),
            sub_subject1_3     varchar(50),
            sub_subject2_3     varchar(50),
            language_id        int(10) unsigned,
            keyword            varchar(200),
            created_date       date NOT NULL,
            expiry_date        date NOT NULL,
            version            varchar(50),
            source             varchar(200),
            annotation         text,
            size               int(10) unsigned,
            resource_format    varchar(20),
            abstract           text NOT NULL,
            contributor_id     varchar(20) NOT NULL,
            primary key(sh_id));

create table sh_author (
            sh_id      int(10) unsigned NOT NULL,
            name           varchar(50)  NOT NULL,
            organization   varchar(100),
            address        varchar(50),
            apt_no         varchar(20),
            city           varchar(30),
            province       varchar(30),
            country        varchar(30),
            p_code         varchar(20),
            e_mail         varchar(50),
            phone          varchar(20),
            role           varchar(20)  NOT NULL);

create table sh_coverage (
            sh_id              int(10) unsigned NOT NULL,
            coverage_domain    varchar(50),
            coverage_value     varchar(200));

create table sh_system_req (
            sh_id              int(10) unsigned NOT NULL,
            system_req_domain  varchar(50),
```

```
            system_req_value  varchar(200));

create table sh_identifier (
            sh_id             int(10) unsigned NOT NULL,
            identifier_domain varchar(50),
            identifier_value  varchar(200));

create table sh_classfication (
            sh_id             int(10) unsigned NOT NULL,
            clasf_domain      varchar(50),
            clasf_value       varchar(200));
```

Figure 7: Implementation of intermediate tables between the ASHG subsystem and the Resource Registration subsystem

4.4 Performance Considerations

In order to get good performance on resource registration and resource searching, the following features are added:

4.4.1 Add Column Indices for Resource Searching

Indices are used to quickly find rows with a specific value. Without an index MySQL has to start with the first record and then read through the whole table until it finds the relevant rows. The bigger the table, the more this costs. If the table has an index for the columns in the query, MySQL can quickly seek a position in the middle of the data file without having to look at all the data. For resource searching, the user may use author name, subject, etc. to find resources. It is better to have an index on these fields. As shown in figure 8, the name field in the author table is defined as an index. When the user uses author as his criteria to perform resource searching, there will be better performance

than without an index. For the same reason, an index on the resource_id and one on the subject field of the resource_subejct table are created. And indices on resource_id of identifier table, system_req table, classification table and coverage table are created.

```
Create index i_author_name on author(name(15));
create index i_reource_subject_rid on resource_subject(resource_id);
create index i_reource_subject_sid on resource_subject(subject);
create index i_identifier_id on identifier(resource_id);
create index i_system_req_id on system_req(resource_id);
create index i_classification_id on classification(resource_id);
create index i_coverage_id on coverage(coverage_id);
```

Figure 8: Indices of the Semantic Header database

4.4.2 Optimizing Concurrent Select and Insert Operations

In CINDI system, many users/contributors would concurrently search and insert data from/into the Semantic Header database. The INSERT DELAYED function is used to optimize these operations. By using this function, the row will be inserted when the table is not in use by any other thread, and insertions from many clients are bundled together and written in one block. This is much faster than by doing many separate inserts [MSQL+].

4.4.3 Using Persistent Connection

From PHP scripts, if there is a need to access data from MySQL database, a connection to MySQL database must be established. Instead of each connection for each database access, a persistent connection is used to connect to the database to avoid the connection overhead. When a persistent connection is requested, PHP checks if there's already an

identical persistent connection (that remained open from earlier). If the connection exists, PHP uses it. If the connection does not exist, PHP creates a new connection. An 'identical' connection is a connection that was opened to the same host, with the same username and the same password. This is more efficient than non-persistent connection. In this way, better performance is achieved.

4.5 Implementation of Web Application with PHP

4.5.1 Connecting to MySQL Database

In the CINDI system, the database name is cindi that includes all tables. The PHP scripts is used to develop the web-based CINDI system. From PHP scripts, there is a need to access database to store/retrieve data. The function `mysql_pconnect()` is used to open a link to the MySQL server. For example, the PHP scripts in figure 9 will get all the contributors' names.

```
<html>
<body>
<?php
      $db=mysql_pconnect("localhost","username","password");
      mysql_select_db("cindi",$db);
      $result = mysql_query("SELECT * FROM contributor",$db);
      printf("First Name:%s<br>\n", mysql_result($result,0,"fname"));
      printf("Last Name: %s<br>\n", mysql_result($result,0,"lname"));
      mysql_close($db);
?>
</body>
</html>
```

Figure 9: PHP scripts and MySQL example

In this example, the result of the connection is stored in the variable $db. Function `mysql_select_db()` then tells PHP that any queries are against the database that is named cindi.

4.5.2 Executing SQL Query

In figure 9, function `mysql_query()` is used to perform SQL querying. Using the database connection identifier, it sends a line of SQL to the MySQL server to be processed. The results that are returned are stored in the variable `$result`. Function `mysql_result()` is used to display the values of fields from the query. Using `$result`, starting from the first row, which is numbered 0, the value of the specified fields are displayed.

4.4.2 Closing Persistent Connection

The function `mysql_close($db)` is used to close links established by `mysql_pconnect()`.

Chapter 5

User Interactions with the CINDI System

5.1 Registration

The CINDI system supplies a registration interface for contributors and users. For resources registration, there are two choices: automatic generated Semantic Header and manual input Semantic Header. Currently, the Semantic Header for HTML, Latex, RTF and Text documents can be automatically generated, others need to use manual uploading.

5.1.1 Contributor Registration

The contributor registering graphical interface lets the resource contributors provide their personal information, and register as contributors of the CINDI system. The CINDI system will send an e-mail to the contributors to notify them of their contributor ID and password after they input the required information and choose a contributor ID. Figure 10 shows the user interface of the contributor registering.

Figure 10: Contributor registration

After the contributors have filled in the required information on their browser, this information is passed to Appache Server by HTML form. Using the database connection, in MySQL database, first the contributorID is checked. In case the contributorID is already in the database, then the database server returns a positive integer to the Web server, and the Web server returns an error message to the browser. Second, if all the required information is not null, then the contributor's personal information is inserted in the Semantic Header database.

Figure 11 shows the existing check for contributorID. Firstly, a connection to MySQL and to the cindi database must be established. Secondly, a query is performed from the contributor table. If there is any duplicate ID, then the return value will be greater than zero.

```
//connect to MySQL
$db=mysql_pconnect("localhost","username","password");
//select database
mysql_select_db("cindi",$db);

if ($submit) {
//check existing for the contributor ID
      $result0 = mysql_query("select count(*) exist from contributor
                  where contributor_id='$contributor_id'");
                  if (mysql_result($result0,0,"exist") >0) {
            print "<tr><td><font size=5>ERROR: Contributor ID
                  already existed. Please choose another one.</font>
                  </td></tr>";
      }
}
```

Figure 11: PHP/MySQL Code for contributorID existing checking

Figure 12 shows how to insert the contributor's personal information in the Semantic Header database. The insert statement is used to insert data into the contributor table.

```
// insert data into contributor table
$result = mysql_query("insert into contributor values
('$contributor_id','$passwd','$fname','$lname','$organization',
'$address','$apt_no','$city','$province','$country','$p_code',
'$e_mail','$department')");
```

Figure 12: Inserting contributor's personal information in Semantic Header database

5.1.2 User Registration

The user registering graphical interface lets a user provide his personal information, and register as a user of the CINDI system. The CINDI system will send an e-mail to user to notify him of his user name and password after he has entered the required information and chosen a user name. The user interface is very similar to the contributor registration interface (Figure 10). Besides the different tables, the underlying process is similar to the contributor registration process (Figure 11 and 12).

5.1.3 Login

Registered contributors and users can use the graphical user interface (figure 13 shows the interface for user login) to log into the CINDI system. After contributors/users enter in their ID and password, this information is passed to Apache Server by HTML form. Using the database connection, in MySQL database, a valid checking is performed and a number is returned to the Web server. Based on the value of the return number, the Web server returns a error message to the browser if the ID does not exist or there is a password error, otherwise it redirects the contributors/users to corresponding page.

Figure 14 shows the processing for user login. After the user clicks the submit button, a query to check the user_id and the password is performed on the table users in MySQL database. If the user_id exists and the password is correct, then the search page is shown. Otherwise, an error message is given.

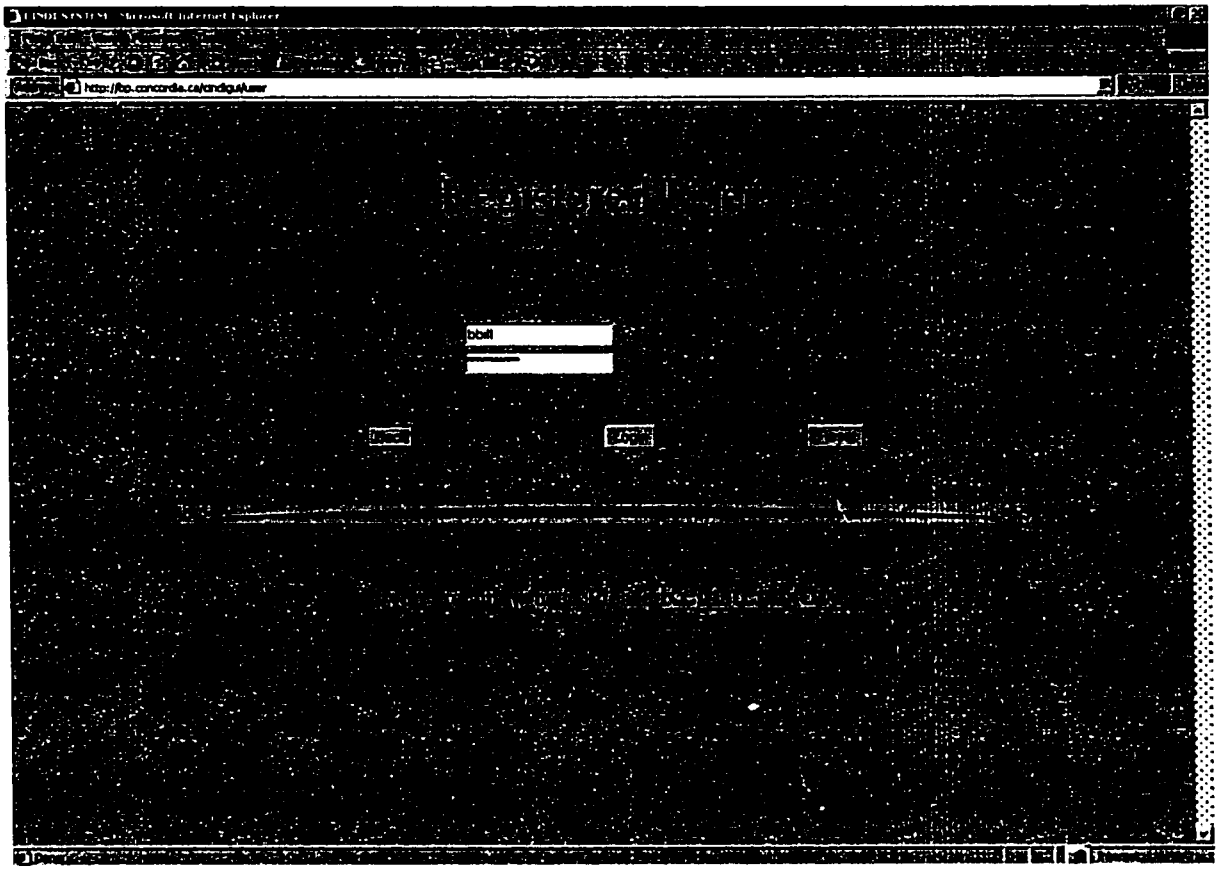

Figure 13: The user login interface

```
if ($submit) {

// valid checking
$result = mysql_query("select count(*) exist from users where
            user_id='$userid' and passwd='$passwd'");

//if it is valid, redirect to search function page
if (mysql_result($result,0,"exist") >0) {
            include("search.php");
} else {

      //if it is invalid, give error message

      print "<table width=80% align=center>
            <tr><td colspan=2>   </td></tr>
```

```
            <tr><td colspan=2>   </td></tr>
            <tr><td width=50%><font size=6>Invalid Login. Please
                  check your user name and password. And";

      print "<a href='javascript:history.go(-1)'> Try again</a>";

      print "or <a href=\"mailto:cindi@bp.concordia.ca\">
            Contact us</a></td>
            <td width=70%> </td></font></tr></table>";
      }
}
```

Figure 14: The processing for user login

5.1.4 Resources Registration

As mentioned in section 5.1, the CINDI system supports two kinds of Semantic Header registration: Manual Registration and Automatic Registration.

5.1.4.1 Manual Registration

After a registered contributor logs in to the CINDI system, there is a choice of manual uploading or automatic uploading. If the contributor clicks on the manual upload link, he will use the manual upload method to register Semantic Header. Figure 15 shows the graphical interface. The required fields include: the title, the author(s), the subject, the abstract and the filename (see the asterisk). The required fields checking is performed by the PHP code.

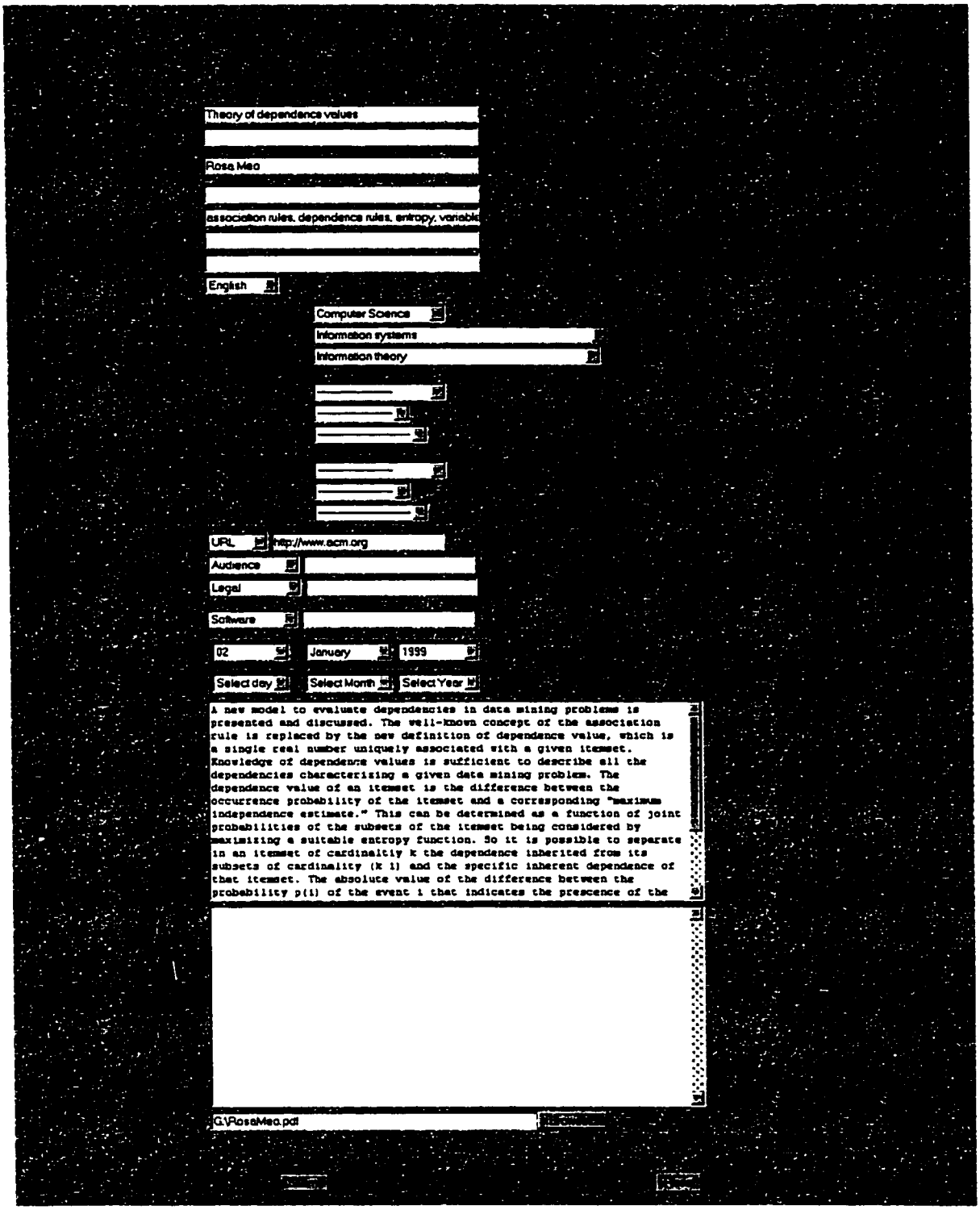

Figure 15: Resource Manual Uploading

After the contributor logs into the system and chooses the manual upload link, the browser sends a request to the Web server. The Web server executes the PHP scripts and connects to the Semantic Header database to get the data, such as language, subject, identifier, coverage, classification, system requirement, etc (figure 16 shows an example for this processing). The Web server sends back this information to the browser. After the contributor fills in all required information, the form data is sent to the Web server and the Web server connects to database and inserts data in the corresponding tables. In order to implement the relationship between the resource table and the relative tables, information such as title, alternate title, keywords, created date, abstract, filename, size and format of the resource, etc. is inserted in the table resource first. By using AUTO_ INCREMENT for the resource ID, a new resource ID for the resource is thereby generated. Then, function LAST_INSERT_ID() is used to get the resource_id for using in other tables, such as resource_subject, identifier, coverage, system_req, classification and resource_subject.

For author's information, first a check is performed based on author name, organization, address, phone number and e-mail address, etc. If the author already exists in the author table, then no record is inserted into the author table and the corresponding author ID and the resource ID is inserted into the resource_author table. Otherwise, the author's information is inserted in the author table and the new author ID and the resource ID is inserted in the table resource_author. Figure 17 shows the detailed PHP/MySQL code for inserting Semantic Header in the database.

```
echo "<select align=left name='language'>";

      // connect to MySQL
      $db=mysql_pconnect("localhost","username","password");

      // select database
      mysql_select_db("cindi",$db);


      // query data from language table
      $result_subject=mysql_query("select * from language",$db);

      while ($myrow=mysql_fetch_row($result_subject)) {
        print("<OPTION value=$myrow[0]");
        if ($cplanguage==$myrow[0]) {
              print(" selected");
        }
        printf(">%s",$myrow[1]);
        echo "</OPTION>";
      }
      echo "</select></td></tr>";
```

Figure 16: Generate pull down menu for language

```
//insert record into table resource
$result = mysql_query("insert delayed into resource values
      (null,'$title', '$alt_title', '$language_id', '$keyword',
      '$created_date', '$upload_date', '$expiry_date', '$last_update',
      '$version', '$source', '$annotation', $size, '$resource_format',
      '$abstract', '$filename', '$userid')");

// insert record into table identifier
if ($identifier_type != "0" )
$result1 = mysql_query("insert into identifier values
            (LAST_INSERT_ID(),'$identifier_type','$identifier')");


// insert record into table converage
if ($coverage_type != "0" )
$result2 = mysql_query("insert into coverage values
            (LAST_INSERT_ID(),'$coverage_type','$coverage')");

// insert record into table system_req
if ($sysrequirement_type != "0" )
$result3 = mysql_query("insert into system_req values
      (LAST_INSERT_ID(),'$sysrequirement_type','$sysrequirement')");

// insert record into table classification
if ($classification_type != "0" )
$result4 = mysql_query("insert into classification values
      (LAST_INSERT_ID(),'$classification_type','$classification')");

// insert record into table resource_subject
```

```
if ($subject1 != "0" )
$result5 = mysql_query("insert into resource_subject values
      (LAST_INSERT_ID(),'$subject1','$sub_subject1',
            '$sub_sub_subject1')");


if ($subject2 != "0" )
$result6 = mysql_query("insert into resource_subject values
      (LAST_INSERT_ID(),'$subject2','$sub_subject2',
            '$sub_sub_subject2')");

if ($subject3 != "0" )
$result7 = mysql_query("insert into resource_subject values
      (LAST_INSERT_ID(),'$subject3','$sub_subject3',
            '$sub_sub_subject3')");

//check existing for author
$result=mysql_query("select count(*) exist from author where
name='$authorarray[$loop]' and organization='$organization' and
address='$address' and phone='$phone' and e-mail='$e-mail' ");

      if (mysql_result($result,0,"exist") >0) {
      // for existing author, get the author ID and insert
      // record   into table resource_author. No new record
       insert into table   author.
            $result=mysql_query("select author_id from author where
                  name='$authorarray[$loop]' and
                  organization='$organization' and address='$address'
                  and phone='$phone' and e-mail='$e-mail' ");
             $myrow=mysql_fetch_row($result);
              mysql_query("insert into resource_author values
                  ('$resource_id', '$myrow[0]') ");
       } else {
            // For new author, insert record into table author and
            // record into table resource_author.
                  $result1 = mysql_query("insert into author values
                        (NULL,'$authorarray[$loop]','$organization',
                        '$address','$phone','$e-mail')");
                  mysql_query("insert into resource_author values
                        ('$resource_id', LAST_INSERT_ID()");
                 }
       }
```

Figure 17: The PHP/MySQL code to insert data in Semantic Header database

5.1.4.2 Automatic Registration

The ASHG subsystem of the CINDI system can generate a draft Semantic Header for HTML, LATEX, TEXT and RTF documents. After a registered contributor logs into the CINDI system and chooses automatic upload, the CINDI system will invoke the ASHG

subsystem to generate the semantic header, and store this information in the intermediate tables (see section 4.3.2). After the Semantic Header is abstracted, the ASHG subsystem will show a window asking the contributor to continue further processing, and subsequently shows the Semantic Header that has been generated to let the contributor verify and possibly modify this information.

The underlying process is very similar to manual uploading. However, the Semantic Header information is obtained from table sh, sh_author, sh_identifier, sh_coverage, sh_classification and sh_system_req according to the unique file ID at first. Then, items in pull down menus is highlighted such as language, subject, identifier, etc. Figure 18 shows part of the PHP/MySQL code for getting data from intermediate tables.

```
//query data from table sh
$result = mysql_query("select * from sh where sh_id=$sh_id",$db);
$myrow=mysql_fetch_row($result);

// display title and alt_title
echo"<tr><td width=11% align=left valign='top'>

    <form  name=form ENCTYPE=multipart/form-data method='post'
      action='auto_submit.php?ID=$myrow[0]'>";

      echo "Title:*<td width=79%><input type='text' name='title'
      value='$myrow[1]' size=50 maxlength=100></tr>";

      echo"<tr><font size=6><td width=11% align=left>
            Alt_title:</td></font><font size=5><td width=79%><input
            type='Text' name='alt_title' value='$myrow[2]' size=50
            maxlength=100></td>
            </tr>";

//query data from table author_table, separated by comma
$result_author = mysql_query("select *  from sh_author where
                  sh_id=$sh_id",$db);
while ($myrow_author=mysql_fetch_row($result_author)){
   $author_array.=$myrow_author[1] . ","
}

// display author
echo" <tr><font size=6><td width=11% align=left >Author/ Other
      Agents:*</td></font><font size=5><td width=79%><input
```

```
type='Text' name='author'  value='$author_array' size=50
maxlength=100>(use comma to seperate)</td>
</tr>";
```

Figure 18: Part of PHP/MySQL code for automatic upload

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. According to the request, the CINDI system performs Semantic Header query. Once the query is processed, the result is sent back to the client.

5.2.1 Search Query Structure

The content of the search query contains the title and/or author and/or keyword and/or subject and/or period of created date. By filling in the HTML form and then submitting it to the server, what a user does, in essence, is create and transmit a string to the Web server, and passes it to the Semantic Header database as an SQL query string to perform a query. The set of all possible strings that the user can create and submit currently, is described by the grammar written in BNF [GM86] as presented in figure 19. The grammar determines the set of strings that can have optional parts connected by the logical AND, OR. The optional parts are: title, author, keywords, subject, and period of created date. Their exact orders and combinations are given again by following the BNF rules of the grammar. 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. In MySQL database, AND has the higher priority than

OR. The user can search for Semantic Headers for a given title, which could be an exact or a substring title. For given authors that are separated by commas, it can be combined by AND or OR. To perform a search, the separated authors are treated as substrings name. Also keywords can be separated by comma, combined by AND or OR, and separated keyword(s) are treated as substring in search. The user also can choose a subject at a general level, and/or corresponding sub_subject level and/or corresponding sub_sub_subject level to perform a search. Lastly, the user can perform a search by giving the period of the created date of the resource.

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

<operand>          ::= <title> | <subject> | <author_unit> |
                       <keyword_unit> | <date>

<op>               ::= <AND> | <OR>

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

<exacttitle>       ::= <string>

<substringtitle>   ::= <string>

<subject>          ::= <general> | <general> <AND> <sub_subject> |
                       <general> <AND> <sub_subject>
                       <AND> <sub_sub_subject>

<general>          ::= <string>

<sub_subject>      ::= <string>

<sub_sub_subject>  ::= <string>

<author_unit>      ::= <author> | <AND | OR> <author_unit>

<author>           ::= <string>

<keyword_unit>     ::= <keyword> | <AND | OR> <keyword_unit>

<keyword>          ::= <string>

<date>             ::= <from_date> | <to_date> |
                       <from_date> AND <to_date>

<from_date>        ::= <day> - <month> - <year>
```

```
<to_date>        ::= <day> - <month> - <year>
<string>         ::= <character> | <character> <string>
<character>      ::= a|A|b|B|c|C| … |x|X|y|Y|z|Z|0|1|2| … |7|8|9
<day>            ::= 1|2|3|4| … |28|29|30|31
<month>          ::= 01|02|03| … |10|11|12
<year>           ::= 1990|1991|1992| … |2008|2009|2010
```

Figure 19: The Current BNF Query Structure

5.2.2 Performance Considerations for Search

In order to get a better performance for Semantic Header search, besides the performance considerations mentioned in section 4.4, there are the following additional considerations for the search.

Query relative tables only

To perform a Semantic Header search, first the resource ID that uniquely identifies the resource is needed. To get the resource ID, it must be queried from the resource table and/or the relative tables depending on the query criteria. For instance, if the user only gives query criteria for title, keyword and date, then a query is needed only from the resource table. On the other hand, if the user gives all criteria, including subject, author, a join query is needed from these tables: resource, subject, subject_resource, author, and resource_author. For different cases, query SQL statements are generated based on different table(s). In this way, a better performance is achieved.

Using case-insensitive search in string matching

By default, MySQL searches are case-insensitive. This default is used when a resource search is performed, because to get case-sensitive matching there is a need to perform additional search that is more costly, such as INDEX or STRCMP.

Eliminating merge join

Merge join normally has a bad performance. By using primary keys and column indices, merge joins are eliminated in the Semantic Header search.

Arrange the order of the join tables

There are five tables (resource, subject, resource_subject, author, resource_author) joined together to perform a resource search in the worse case scenario. In this case, MySQL performs four separated joins: a join of two tables to generate a set of records, then a join between that set of records and the third table, and the fourth table and the last table. If the large tables are joined in the first join, the size of the tables will impact on each successive join and will negatively impact on the overall performance of the search. For this point, the join order is arranged as author, resource_author, subject, resource_subject and resource.

5.2.3 User Interactions with CINDI Search Sub-System

Figure 20 shows the graphical user interface of the Semantic Header search sub-system. Users can enter search criteria, such as title, author(s), keyword(s), subject(s) and period

of created date. In this particular example, the user is searching for Semantic Headers using "Minimizing" as exact title and "Naveen Garg" as author name and "approximation" as keywords and the created date in "Jan. 1, 1998" to "Jan. 1, 2000". The corresponding SQL query is given in figure 21.

Figure 20: Semantic Header Searching

```
select DISTINCT c.resource_id
from author a, resource_author b , resource c,
where b.resource_id=c.resource_id and a.author_id=b.author_id
      and c.title like "Minimizing" and a.name like "Naveen Garg" and
      (abstract like "approximation" or keyword like "approximation")
      and created_date>"01-01-1998" and created-date<"01-01-2000" LIMIT
      $start, $rows_per_page;
```

Figure 21: HTML Query translated into SQL Query

In order to separate the result in a page order, the LIMIT function is used in the SELECT statement. The current setting is 3 resources per page. After the resource ID is obtained, then the details of resource, such as title, author(s), subject, abstract, and keyword, etc. are queried. Figure 22 gives the result of the previous search example.

Figure 22: Search Result

5.3 Annotation

The CINDI system allows users to view annotations made by other users and allows them make their own annotations. From the search result, there is an annotation link. When the

user clicks the link, an annotation window is presented to the user. Figure 23 shows the graphical interface for annotation.

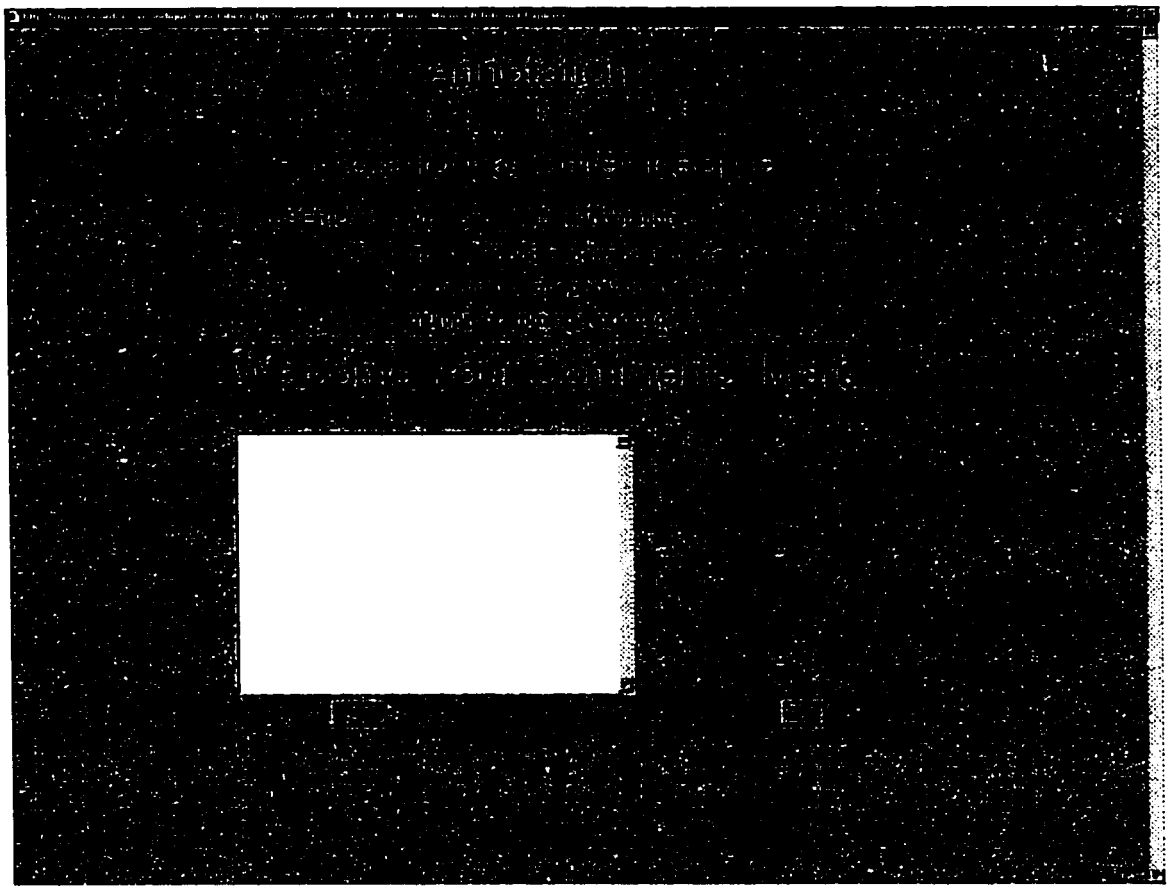

Figure 23: Annotation

First, the corresponding annotation made by other users is presented. Then, the annotation that made by the current user is inserted in annotation table. Figure 24 shows this processing.

```
// query annotations  relative to the resource
$result_subject=mysql_query("select annotation from annotation
      where resource_id=$resource_id",$db);

// output annotations
      echo"<tr><font size=6><td width=11% align=left
            valign='top'>Annotation of other
            people:</td></font><font size=5><td width=79%><textarea
            name='annotation' cols='70' rows='15' wrap='on'
            maxlength=1000> $annotation </textarea></td>
      </tr>";

// insert annotation to database
$result = mysql_query("insert into annotation
            values('$resource_id',null,'$annotation')");
```

Figure 24: The processing for annotation

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The web-based CINDI system meets the challenges of finding hypermedia document on the Internet by defining a Meta-Data structure, which allows automatic and manual extracting of Meta-Data from resources. It provides a group of graphical user interfaces to interact in the registering and discovery process.

6.2 Contribution of this Report

The contributions made by this major report to the CINDI system are listed below:

1. The design and implementation of the database subsystem for Semantic Headers. This is the fundamental contribution made by this major report to the CINDI system.
2. The design and implementation of subject hierarchy. This is one of the major issues in the design and implementation the CINDI system. This problem is solved by using primary key structure, PHP/Java Scripts.
3. The design and implementation of the searching strategy. By optimizing the query, a better search performance is achieved.

6.3 Future Work

We believe that the current prototype of the CINDI system satisfies the needs of Internet users for effective retrieval of electronic information resources. However, in the near future, some new functions could be built and some existing functions could be improved.

1. A major extension to the system would be to build a distributed system accessible in different locations.
2. Because the current MySQL version doesn't support Foreign key, application codes are used to control the data constraint. If the new version of MySQL supports Foreign Key, it can be use to improve the data constraint.
3. The subject hierarchy function could be extended. Currently, the CINDI system supports 3 level subject hierarchies that were implemented by a primary key structure and PHP/Java Scripts. In the future, if MySQL supports the hierarchy query (like Oracle CONNECT BY), it can be use to support dynamic multiple subject levels.
4. We should extend the search function. The current CINDI system supports search criteria such as title, keywords, author, subject and period of created date. In the future, we can extend it to support more search criteria, such as language, identifier, classification and system requirements, etc.
5. For the convenience of both contributors and users, an automated ID/password lookup facility could be built. If the contributors/users have already registered in the CINDI system and have forgotten their login ID/password, then they can use this function to get their login ID/Password by the e-mail provided during registration.

Bibliography

[APCH+] Apache Online Document, http://www.apache.org

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

[BCD+] Bipin C. Desai, The Semantic Header and Indexing and Searching on the Internet, http://www.cs.concordia.ca/~faculty/bcdesai/cindi-system-1.0.html

[BS94] Bipin C. Desai, Shinghal Rajjan, A System for Seamless Search of Distributed Information Sources, May 1994. http://www.cs.concordia.ca/~faculty/bcdesai

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

[BCD96] Bipin C. Desai and Shinghal Rajjan, Resource Discovery: Modeling, cataloguing and searching. *In Proceedings of the Seventh International Conference and Workshop on Database and Expert System Applications (DEXA'96),* pages 70-75. IEEE Press, Zurich, Switzerland, 1996.

[BCD97] Bipin C. Desai, Supporting Discovery in Virtual Libraries, Journal of the American Society of Information Science (JASIS), 48-3, pp. 190-204, 1997.

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

[JE99] Jeri Edwards, *3-Tier Server/Client at Work,* John Wiley & Sons, 1999.

[MAM01] Mohamed Amokrane Mechouet, Web Based CINDI System, Department of Computer Science, Concordia University, 2001

[MSQL+] MySQL Documents, http://www.mysql.com/documentation/

[NTCR+] The Netcraft Web Server Survey, http://www.netcraft.com/survey/

[PDMW99] Paul DuBois and Michael Widenius, *MySQL,* Addison Wesley, 2000.

[PHP+] PHP Documents, http://www.php.net/docs.php

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

[SSD98] Sami Samir Haddad, Automatic Semantic Header Generator, Department of Computer Science, Concordia University, 1998.

[YZ01] Youquan Zhou, CINDI: The Virtual Library Graphical User Interface, Department of Computer Science, Concordia University, 2001.