INFORMATION TO USERS

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

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

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

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

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

UMI®

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

Université d'Ottawa · University of Ottawa

The Design and Implementation of the User Interface and the Web Server Extension to a Relational Database Application

BY

YAOPING WANG

A thesis submitted to the
School of the Graduate Studies and Research
In partial fulfillment of the requirements for the degree of

Master of Applied Science
In
Electrical and Computer Engineering
Ottawa-Carleton Institute of Electrical Engineering
School of Information Technology & Engineering (SITE)
Faculty of Engineering
University of Ottawa
Ottawa Ontario

December 1999

To *Amy*

Abstract

The Web provides new opportunities for gathering information from multiple, distributed, heterogeneous information sources. However, this distributed environment poses difficult technical problems for the information-seeking client, including finding the information sources relevant to an interest, formulating questions in the terms that the sources understand, interpreting the retrieved information, and assembling the information retrieved from several sources into a coherent answer. We have developed a relational database application to tackle these problems.

The Global University Database (GUD) project, funded by Nortel, has been developed by the GUD group at the Multimedia and Mobile Agent Research Lab (MMARL), University of Ottawa, Canada. The project was initiated around September 1997. The idea is to retrieve the *University*-relevant information through the Internet, and to populate a local database with the data extracted. The database may be updated to reflect the changes that could have occurred on the university Web sites from which the data have been extracted. The users will then access and query the database as a single source of information.

The main contributions of this research are the design and implementation of the user interface and Web server extension of this relational database system. The development of the system architecture is based on the four-tier system architecture. We believe our achievements in this project are:

- We have designed and implemented a user interface to a relational database

application. The user interface design process is fully examined and the implementation and testing procedure is well performed.

- We have designed and implemented a Web server extension to the application. The Data Processing Agent (DPA), which is used to handle the requests from the user, is a platform-independent, database-driven application server extension.

The thesis consists of several chapters: introduction; overview of the GUD project; design of the GUD user interface; design of the GUD Web server extension; implementation; conclusion and future work.

Acknowledgements

I wish to express sincere gratitude to my supervisor, Dr. Ahmed Karmouch, whose interest has made this experience possible. His support and patience are highly appreciated.

Special thanks go to Mr. Vu Pham for his ideas, insight, and inspiration during the development of this project. I also want to thanks for other GUD team members for their excellent work and cooperation.

Mr. Thong Pham, our system administrator, has spent lots of time to help me manage my NT station. I will definitely miss this extremely smart, knowledgeable young guy.

I am also grateful to all my other friends and colleagues at the Multimedia and Mobile Agent Research Lab. Thank you all for creating a very positive atmosphere that makes it easier to withstand the difficulties that sometimes arise.

This research was supported by the Nortel Networks. I would like to thank Dr. Ahmad Zaid, Advanced Research manager, Nortel, for the logistical support and the resources made available, without which this project would not have been possible.

Thanks also go to Eng Tan, my colleague at the Object People, who stayed with me several weekends to code the JSP page.

Finally, I would like to thank Jing Liu for her dedicated love, patience and support of my academic pursuits.

Table of Contents

List of Figures and tables

Acronyms

AM	Administration Manager
API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
DC	Database Connector
DLL	Dynamic Link Library
DPA	Data Processing Agent
GUD	Global University Database
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
IB	Information Broker
JDBC	Java Database Connectivity
ODBC	Open Database Connectivity
PM	Populating Mediator
RMI	Remote Method Invocation
RPC	Remote Procedure Control
UM	Updating Mediator
WMA	Web Mining Agent

Chapter 1

Introduction

1.1 Motivations

The Web is rapidly becoming the foundation of an information economy. The much-heralded "information technology age" has delivered a stunning variety of on-line information resources: nation-wide Yellow Pages, job listings, stock market quotations, retail product catalogues, and much more. With widespread adoption of open standards such as TCP/IP and HTTP, and extensive distribution of inexpensive or even free software such as Internet Explorer, those resources are ever more widely available.

The inevitable result of this growth is that current technologies for accessing information on the Web are inadequate. The exponential growth of information within the Web has created an overabundance of information and a poverty of human attention, with users citing the inability to navigate and find relevant information on the Web as one of the biggest problems facing the Web today. [Montebello98].

As a matter of fact, the growth brings with a number of related challenges for information searching. First, non-technical users should be able to benefit from the information available on the Web without being overwhelmed by technical detail. Second, users should be freed from mundane and repetitive browsing tasks. Third, and

most critical, information from the Web should be available in the format and combination that best fit the user task (e.g. upon the user's preference), regardless of the pages on which the information was originally found.

There has been much work on software systems to help reduce this information overload. We have seen systems that reduce the amount of information coming to a user by *filtering*, and we have seen *indexers* that provide convenient ways to search the vast information spaces to find a specific item. However, it is evident nowadays that finding some specific information is still time-consuming and tedious.

1.2 Objectives

The primary goal of research presented here is to put forth new techniques that can be used to help efficiently filter in the desired specific information and manage users' attention processes when dealing with large, unstructured, heterogeneous information environments.

Our first main objective is the information extraction. Roughly speaking, by the phrase "information extraction" we refer to the procedure of identifying and organizing relevant fragments in a document while discarding extraneous text. In the literature, such specialized procedures are commonly called wrappers [Childovskii97, Roth97]. Of course, this emphasis of extraction ignores many important issues, such to discover resources in the first place (e.g. by software agent) [Bowman94]; how to query these sources [Doorenbos97]; and which services beyond extraction wrappers should provide [Papakonstantinou95, Roth97].

Once a resource's information is extracted in this manner, it can be

manipulated in many different ways. Eventually, it has to be stored somewhere, like local database, and to be presented to the user via Web- based browsers or standalone Graphic User Interface (GUI).

It comes up with our second objective: information access and presentation. Constantine [Constantine95] points out that the reality is that a good user interface allows people who understand the problem domain to work with the application without having to read the manuals or receive training. When we already have the data available, we need to provide an efficient way for the user to access the data source.

In the Multimedia and Mobile Agent Research (MMARL) Laboratory at the University of Ottawa, our research group has designed and implemented a Globe University Database (GUD) project to offer a solution to the information overloading. The idea is to retrieve the *University*-relevant information through the Internet, and to populate a local database with the data extracted. The database may be updated to reflect the changes that could have occurred on Web sites from which the data have been extracted. The users will then access and query the database as a single source of information. To construct a desired user interface for the GUD project to be easily used and a robust web server extension for the request to be processed form the core of this thesis.

1.3 Contributions

The GUD project is a comprehensive application development platform created to harness the benefit of the Internet for core business applications. As discussed earlier, the

primary motivation of this thesis involves efficient data access, processing, and presentation. The main contribution of this research is the design and implementation of the user interface and Web server extension of a university database system. The development of the system architecture is based on the four-tier system architecture. We believe our achievements in this project are:

- We have designed and implemented a user interface to a university database application. The user interface design process is fully examined and the implementation and testing procedure is well performed.
- We have designed and implemented a Web server extension to the application. The Data Processing Agent (DPA) is a cross-platform, database-driven, application server extension. The cutting-edge technologies, like Java Servlet API and JDBC, are successfully deployed to implement the components inside the DPA, namely Information Broker (ID), Data Mediator (DM), among others.
- We have used the newest Java Server Page (JSP), working together with the servlets, to separate the tasks of handling the presentation logic with the application logic. The usage of JSP not only simplifies the development and maintenance procedure, but also emphases the reusable component (using JavaBean) and enhances the overall system performance.

Our GUD team presented our system at the OCRI'99 Ottawa technology showcase, and drew lots of attentions. One paper arose from this research, for the IEEE Canadian Conference on Electrical and Computer Engineering 1999, Edmonton. [Wang99]

1.4 Organization

The remainder of this thesis is organized as follows:

Chapter 2: Overview of the Global University Database (GUD) Project

In this chapter, the GUD project is first introduced. The GUD overall four-tier system architecture is reviewed. The four tiers are the front-end user, the Data Processing Agent (DPA), the Web Mining Agent (WMA), and the back-end database.

Chapter 3: User Interface Design on the GUD Project

In this chapter, the user interface design process is first overviewed. And then the concept of the user interface design is introduced, followed by the discussion of the user interface prototyping. The user interface process of the GUD project is fully examined, while the GUD data model, business function model, and user interface prototype are illustrated. The issues of the GUD project user interface design, namely usage profile diversity, browser diversity, and form validation, are also addressed.

Chapter 4: Issues of Design of the Web Server Extension to the GUD Project

The Web Server extension consists of software components to perform the Web-based server-side data computing. In this chapter, the design issues of GUD Web server extension -- Data Processing Agent (DPA) are discussed. The DPA framework is presented first. The working mechanism of the DPA is then elaborated. As part of DPA, the Data

Mediator (DM) interfaces with the WMAs and performs the database table creating, database populating and updating. The process flowchart is then examined.

Chapter 5: Project Implementation

In this chapter, the implementation details of the user interface and DPA in the GUD project is reported. The implementation language and tools are briefly introduced. The three packages in the GUD project, namely *applet*, *broker*, and *mediator*, are illustrated respectively. The user interface and query processing samples are also presented.

Chapter 6: Conclusion

The summary of the overall writing is enlightened in this chapter. The suggestions for the future research are also listed.

References

Appendix A: The GUD Package and Class Hierarchy

Appendix B: The GUD JSP Sample Code

Appendix C: The GUD JavaBean Sample Code

Appendix D: Publications

Chapter 2

Overview of the Global University Database (GUD) Project

2.1 Introduction

As discussed in the chapter 1, our research group has developed a relational database application – Global University Database (GUD) project -- to alleviate the problems of information overloading. In this chapter, We will review the overall GUD framework, while the four-tier system architecture is introduced. The four tiers are composed of the front-end client, the Data Processing Agent (DPA --- the GUD Web server extension), the Web Mining Agent (WMA), and the back-end GUD database.

Before we get into the GUD four-tier paradigm, We will review the evolution of the Client/Server computing, and make strong arguments why we deploy the multi-tier architecture and what benefits we can get from it.

2.2 The Benefits of the Multi-tier System Architecture

The term client-server architecture is a general description of a networked system where clients program initiates contact with a separate server program (possibly on a different machine) for a specific function or purpose. The client exists in the position of the requester for the service provided by the server. The first generation system is a two-tiered architecture where a client presents a GUI interface to the user, and uses the user's

data entry and actions to perform requests of a database server running on a different machine. In this case, application logic is typically coupled to the client. [Jenkins97]

A new generation of client-server implementation takes a step further and adds a middle tier to achieve a 'three-tier' architecture. Generally, client-server can be implemented in an 'N-tier' architecture where application logic is partitioned. This leads to faster network communications, greater reliability, and better overall performance. [Edwards98]

2.2.1 Two-Tier Architectures

The most common implementation of two-tier is to place application logic in the client, making it a 'fat' client - 'thin' server architecture (Figure 2.1). The database then simply reports the results of queries. This is typically implemented via dynamic SQL using a call level interface (CLI) such as Microsoft's Open Database Connectivity (ODBC). In this scenario remote database transport protocols are used to carry the transaction.

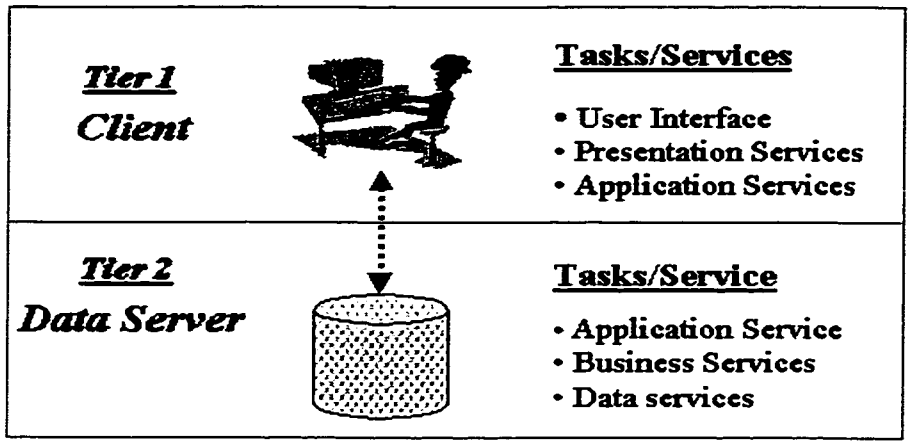

Figure 2.1 The Two-Tier System Architecture

The drawback in this first-generation design is that the GUI is tightly bound to the application logic on the client side. As a result, as client applications become more complex, the application logic expands creating overloaded clients-also known as the "fat client" syndrome. Both the network transaction size and query transaction speed are slowed by this interaction. These architectures are not intended for mission critical applications.

2.2.2 Three-Tier Architectures

A significant reduction in network footprint and enhancement of performance is possible in the alternative 'three-tier' client-server architecture. A middle tier may be inserted in between a 'thin' client and 'thin' server, thus achieving a three-tier structure. The client interacts with the middle tier via a standard protocol such as DLL, API, or RPC.[Stallings97]. The middle-tier interacts with the server via standard database protocols (Figure 2.2).

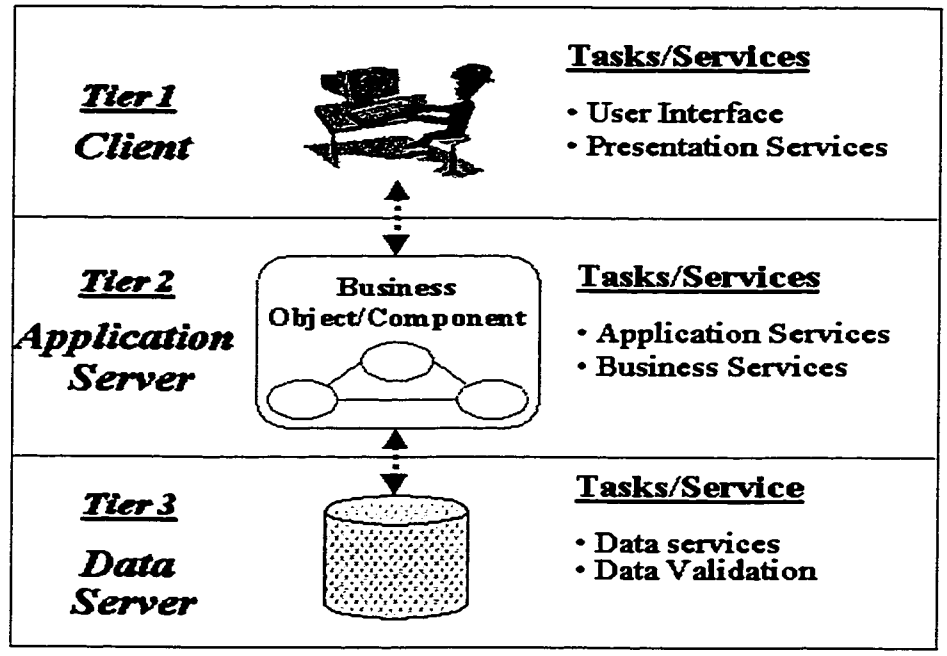

Figure 2.2 The Three-Tier System Architecture

The middle-tier contains most of the application logic, translating client calls into database queries and other actions, and translating data from the database into client data in return. If the middle tier is located on the same host as the database, it can be tightly bound to the database via an embedded application interface. This yields a very highly controlled and high performance interaction, thus avoiding the costly processing and network overhead. Furthermore, the middle tier can be distributed to a third host to gain processing power capability.

Through standard tiered interfaces, services are made available to the application. A single application can employ many different services that may reside on dissimilar platforms or are developed and maintained with different tools. This approach allows a developer to leverage investments in existing systems while creating new application that

can utilize existing resources.

2.2.3 N-Tier Architectures

The three-tier architecture can be extended to N-tiers when the middle tiers provide connections to various types of services, integrating and coupling them to the client, and to each other. An N-tiered system can also be created by partitioning the application logic among various hosts. Encapsulation of distributed functionality in such a manner provides significant advantages such as reusability, and thus reliability (Figure 2.3).

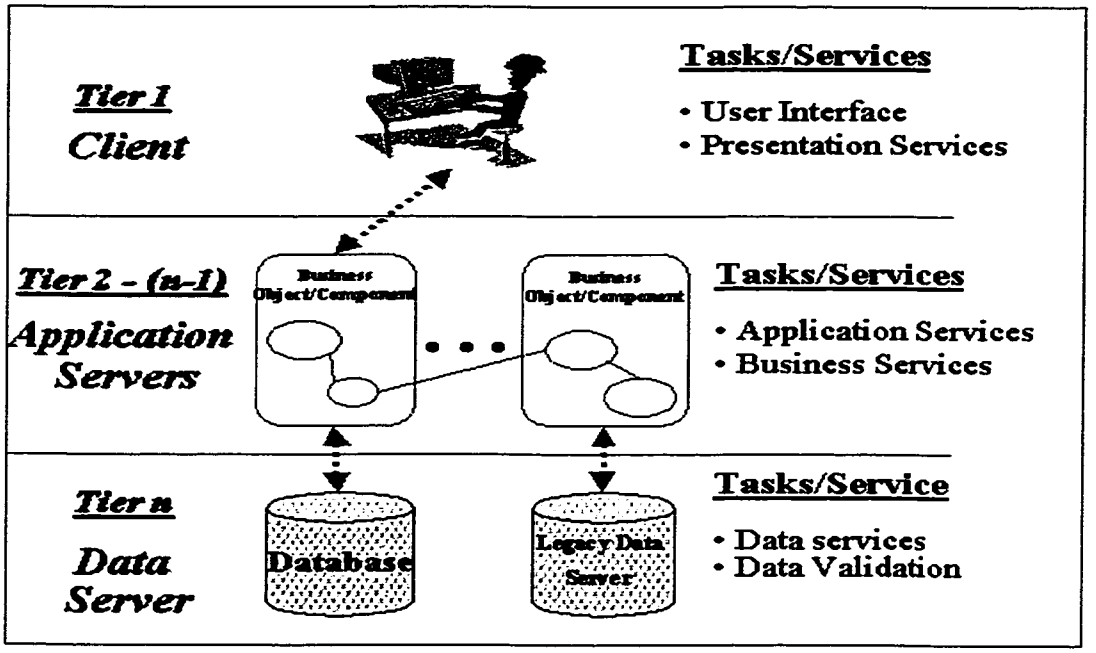

Figure 2.3 The N-Tier System Architecture

Several considerations make an N-tiered architecture desirable, including scalability, performance, client variability, network efficiency, versioning of business logic, distribution of services, and security. [Edwards98]

Scalability

N-tier architecture increases application scalability and performance by enabling a great number of concurrent client connections. In a two-tier model, each client maintains a connection to the database. In an N-tier model, clients only connect to the application server. The application server can multiplex requests from clients through a pool of pre-allocated database connections. This reduces load on the database server, while load on the application server tier can be balanced across multiple application servers. This redistribution increases the processing capacity of the application and improves overall performance and scalability.

Multiple Clients

The N-tier model leads to trimmer clients because most of the logic runs in the middle tiers. Clients therefore require less memory. This insures that a broad range of client platforms can run the application. The server-centric approach allows for multiple clients to be written if necessary. A full-featured client for the desktop can be simplified and provided in a slimmer version for devices such as cellular phones or Personal Digital Assistants (PDAs).

De-coupled Business Logic

With an N-tiered architecture, the business logic can be modified at the application server tier without impacting the client. It is therefore possible to adapt business logic to meet changing business needs with much less work than if it is embedded in the client tier.

Because the business logic resides in the application server tier, it is easy to deploy multiple ``flavors" of clients without having to duplicate the business logic for each. Thus, the business logic can be maintained in one place, eliminating the potential for inconsistencies. The need for "flavors" increases as the definition of users evolves and different clients are required for a variety of users and situations.

Manageability

This model facilitates deployment of applications without any knowledge of the client. The only requirement on the client is a browser. From a management perspective, the application distribution and management problem space is greatly reduced since only servers need to be managed. No installation or configuration of applications is necessary at the client.

Reusable Services

One final advantage of this architecture is that by viewing applications as collections of services, the services that make up an application can be leveraged and made available to other applications that require them for their own purposes. One example of service reuse is that applications can share an authentication mechanism (service) rather than deploy and maintain their own. This cuts down on the administrative and development work required for each application.

2.3 Introduction of the GUD 4-Tier System Architecture

The Global University Database (GUD) project, developed by Multimedia &

Mobile Agent Research Lab, University of Ottawa, is an information retrieving, updating, processing, and presenting system to which the end users like professors, researchers and students can query the university-related information. Basically, the GUD stores information on up-to-date researching, teaching and other activities happened at the universities across the North America and Europe.

Our group has proposed the 4-tier GUD architecture, illustrated as figure 2.4.

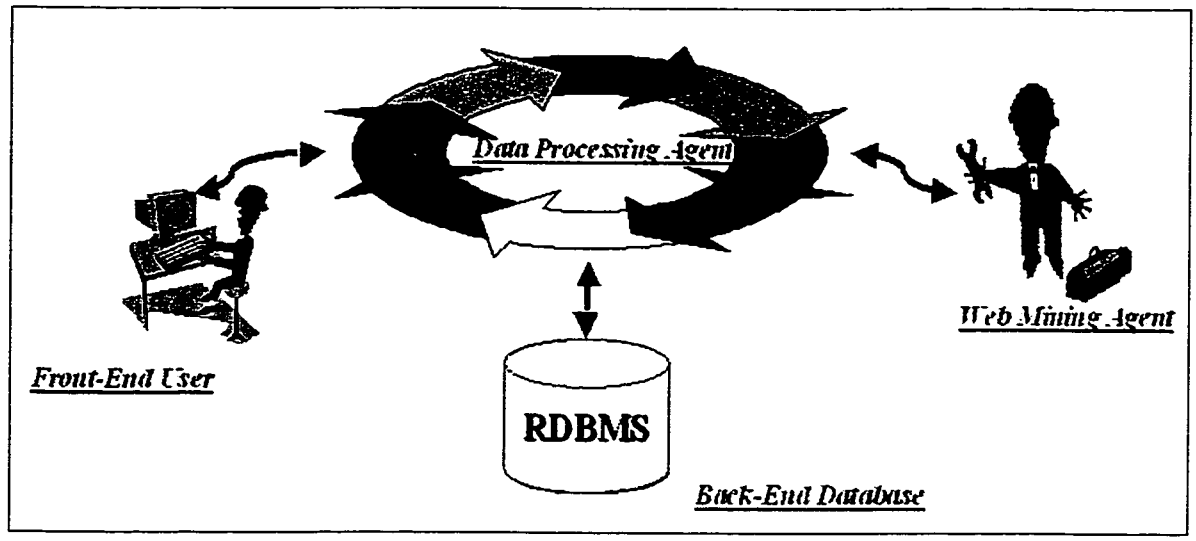

Figure 2.4 The Four-Tier GUD System Architecture

Tier 1 of the architecture is the front-end client. It is responsible for basic rudimentary rules like form validation that can be performed on the client, data presentation and display.

Tier 2 is middle-ware software, which stores all the business rules for the application. It works as the Data Processing Agent (DPA) as a whole. The DPA plays the roles in providing transaction services (as in processing the request/response

paradigm), metering services (as in acting as a transaction monitor to limit the number of simultaneous requests to a given server), administration services (as in tracking the user session activities).

Tier 3 is the Web Mining Agent (WMA), another middle-ware agent. The WMA retrieves and manipulates the university-related data from the Internet, and populates or updates them into the database through the DPA.

Tier 4 is the back-end database server. It stores the university-related data extracted from the university web sites by the WMAs.

In this four-tier scenario, the amount of logic in the client is a minimum, i.e. just a browser like Internet Explorer, which means avoiding installations of any vendor specific software at the client side. The system is robust and efficient. Distributing tiers on different physical computer nodes and different processes improves performance and increases coordination and shared information in the whole system. Moreover, the isolation of application logic into separate components promotes code reusability and provides better manageability and greater scalability. For example, the WMA can be easily replaced with other software components for new applications, and the complete GUD database engine can be changed without affecting any other part of the system.

2.4 Summary

In this chapter, we briefly overview the GUD project. The GUD project is an information management system to provide a solution to the information overloading. The idea is to retrieve the university-relevant information through the Internet, and to

populate a local database with the data extracted. The database may be updated to reflect the changes that could have occurred on Web sites from which the data have been extracted. The users will then access and query the database as a single source of information.

A 'Multi-tier', often referred to as 'three-tier' or 'N-tier,' architecture provides greater application scalability, lower maintenance, and increased reuse of components. In our GUD project, we have deployed four-tier system architecture. We have briefly reviewed the overall GUD architecture in this chapter. In chapter 3, we will address the design issues of the GUD user interface (i.e. tier 1). We will discuss the development of the GUD Web server extension --- DPA (i.e. tier 2) in chapter 4.

Chapter 3

User interface Design on the GUD Project

3.1 Introduction

The only part of the system that the ordinary user is usually in direct contact with is the human-computer interface, A usable information system has to support the user's work effectively, be efficient and satisfactory to the user. What users want is for developers to build applications that meet their needs and that are easy to use. Constantine [Constantine95] points out that the reality is that a good user interface allows people who understand the problem domain to work with the application without having to read the manuals or receive training.

In this chapter, the user interface design process is first overviewed. And then the concept of the user interface design is introduced, followed by the discussion of the user interface prototyping. The user interface process of the GUD project is fully examined, while the GUD data model, business function model, and user interface prototype are illustrated. The issues of the GUD project user interface design, namely usage profile diversity, browser diversity, and form validation, are addressed before the summary

concludes the chapter.

3.2 An Overview of the User Interface Design process

The objective of any user interface developer should be to design and implement quality user interfaces, Unfortunately it's not always that easy to define what is meant by a "quality user interface." It's like describing a quality furniture arrangement. Some work better than others, but there is never one perfect solution. For the purposes of this thesis, a "quality user interface" shall be defined as any user interface that is intuitive, easy to use, and allows the users to maximize their efficiency and effectiveness when using it. [Shneiderman98]

The best way to ensure quality user interface design is to use an orderly and well-defined design process that is specifically geared to producing quality results. This applies to the design of the application that the user interface supports as well as the design of the user interface itself. The great user interface will not be well received if the application itself is poorly designed. There are many resources devoted to the topic of application design methodologies (notably James Martin's "Rapid Application Development" [Martin91]). In this thesis we will discuss some overriding principles that are general to most methodologies, and will focus in particular on aspects that relate specifically to user interface design.

All application design methodologies break the development process down into discrete phases. Below are some typical phases.

Table 3.1 The User Interface Development Phases

Phase	Description
Requirements	Determine the requirements for the application
Conceptual Design	Model the underlying business that the application will support
Logical Design	Design in general terms how the application will operate
Physical Design	Design in specific terms how the application will be constructed
Construction	Construct the application
Usability Testing	Test the usability of the user interface

The Requirements Phase

Before the design process begins in earnest, there's a lot of work that needs to be done. If the application is to be accepted by those who have a stake in it (stakeholders), it is imperative that they be involved from the very beginning. They should be sought out and polled as to what they consider their requirements for the application to be. The more broad a representative group of stakeholders is involved, the more broad the acceptance will be when it is delivered.

Below are some steps that will help lead to a successful requirements phase.

- Assemble the design team

        - Identify all stakeholder groups
        - Select representatives to participate on the design team (there will be a significant time commitment involved)
    - Gather Requirements:
        - Interview as many stakeholders as possible to determine:
            - What the underlying business problems are that this application should address
            - What benefits the application should provide
            - What the critical success factors are
    - Define the scope of the project
        - Review the requirements that have been gathered
        - Make decisions about what will be included and what will not
        - If the scope of the project gets too big, consider breaking it down into stages.

It really can't be overstated how important the Requirements Phase is to the success of any project. Any user interface, no matter how well designed, won't be well received if its users feel disenfranchised from the design process. By establishing the scope of the project early, the expectations of the stakeholders will also be established early.

The Conceptual Design Phase

This phase is concerned with modeling the underlying business that the

application will support. User interface considerations are not addressed at this time. The objective is to model the business irrespective of any implementation issues.

One approach divides the conceptual design into three parts. The *Data Model* identifies data entities and defines the relationships between them. The *Business Function Model* defines the scope of the business with respect to the application, and breaks it down into component business functions. The *Communications Model*, also referred to as a Data Flow Diagram, maps the interactions between the component business functions and the data entities.

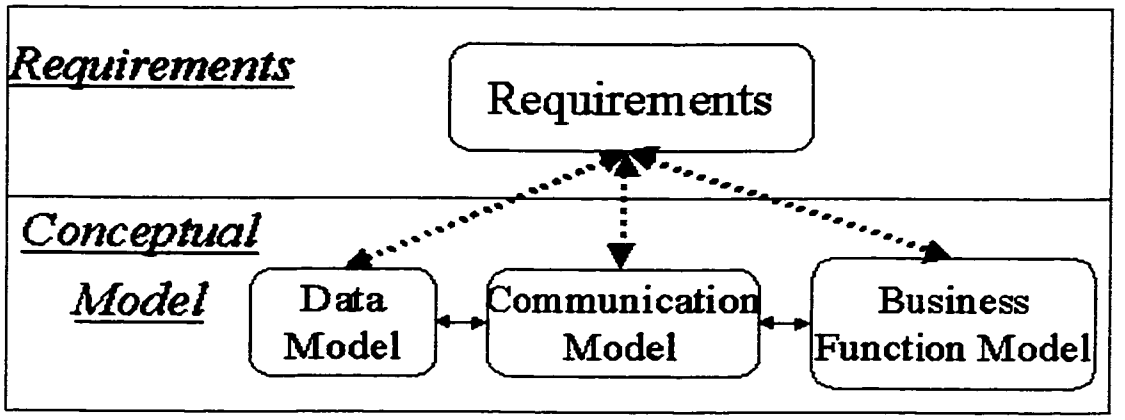

Figure 3.1 The Concept Model in the User Interface Design

Logical Design

The Logical Design phase gets into more specifics about how the application will support the business that was modeled in the Conceptual Design phase. The *prototyping* of user interfaces should kick off the Logical Design phase (note: the user interface prototyping will be discussed at next section). By determining what events will occur on the client (e.g. clicking a button or selecting something from a scrolling list), the logical

processes that support them can be designed. These events and processes are then broken down into "system functions."

Determining the specific technology (i.e. hardware and software) in which the application will be developed is somewhat of a chicken-and-egg process. The earlier the technology is determined, the more decisions can be made about what the user interface will and won't be able to do. But by the same token, the requirements about what the user interface should be able to do can drive the decision of what technology ultimately gets selected. The bottom line is that the sooner decisions are made regarding the technology, the more successful the logical design will ultimately be.

Along these lines, it is important to determine the minimum hardware configuration the application will cater to. For example, a particular user interface might look very good on a large color monitor, but be very difficult to use on a small black and white monitor. The decision must be made as to whether the user interface will cater to the small black and white monitor, or if it's okay to design around the large color monitor. Both solutions are valid, depending on the circumstances, but it must be a conscious decision.

Physical Design

The Physical Design phase is concerned with determining how the logical design will be implemented on specific physical platforms. The technology in which the application will be developed must be conclusively determined before the Physical Design Phase can begin.

This phase will not directly impact user interface design, unless it is revealed

that it is not possible to physically implement certain aspects of the logical design.

Construction

It is in the construction phase that the application is actually programmed. The most important thing to consider at this point is that just because the application is being constructed does not mean that the design process has concluded. When the users get something in their hands that actually functions, they will inevitably change their minds as to how they want things to work.

For this reason it's important to get functional interfaces in the users' hands as early as possible. If they request a change that winds up being somewhat fundamental, less re-designing will be required the earlier the change is identified.

Usability Testing

Usability Testing is a technique that can validate the user interface design and reveal areas in which it requires refinement. The basic concept is to simply observe users as they operate the interface. They should be instructed to verbalize their thought process as they go along. For example, they should be saying things like, "I want to find an invoice for a customer. I see a button that says, 'Invoices,' but I don't know if that will display one or create a new one..." By understanding what the testers are thinking it will be possible to ascertain where they are having problems understanding and using the user interface.

It should be noted that this is often more an exercise in testing how easy a user interface is to learn than how easy it is to use. If the learnability of the user interface is

the primary goal of the design, uninitiated users should be selected, they should be given minimal instruction or guidance, and the observers should look for areas in which the testers are having trouble figuring the user interface out. If the ease of use of the interface is the primary goal, novice users should be selected and the observers should look for areas in which the testers are having trouble generally using the interface or remembering how certain things work.

Either way it is very important that this technique not be used as a substitute for fundamental interface design. It is a mistake to de-emphasize the design process with the rationalization that shortcomings will be revealed during usability testing. That would be like ignoring the recipe for a meal because it is possible to add salt and pepper later. Usability testing is a process of validation and refinement. It is not part of the design process itself.

The more testers that participate in this exercise, the better the results will be. If one or two users have trouble in one particular area, that might not necessarily indicate a problem. But if a majority of testers run into the same problem or similar patterns emerge, it can be apparent that certain parts of the user interface require attention.

3.3 The Concept of the User Interface Design

A basic user interface usually includes things like menus, forms, windows, the keyboard, the mouse, the "beeps" and other sounds the computer makes, and in general, all the information channels that allow the user and the computer to communicate. The user interface can be treated as a software component which accepts information from the user and which presents information to the user. There are some concepts related to

the user interface design as follows. [Shneiderman98]

Usability vs. Learnability

Many people consider the primary criterion for a good user interface to be the degree to which it is easy to learn. This is indeed a laudable quality of any user interface, but it is not necessarily the most important.

The goal of the user interface should be determined in the design process. Consider the example of a visitor information system located on a kiosk. In this case it makes perfect sense that the primary goal for the interface designers should be ease of operation for the first-time user. The more the interface walks the user through the system step by step, the more successful the interface would be.

In contrast, consider a data entry system used daily by an office of heads-down operators. Here the primary goal should be that the operators could input as much information as possible as efficiently as possible. Once the users have learned how to use the interface, anything intended to make first-time use easier will only get in the way.

User interface design is not a "one size fits all" process. Every system has its own considerations and accompanying design goals. The Requirements Phase is designed to elicit from the design team the kind of information that should make these goals clear.

Metaphors

Metaphors help users define what the interface can do and what information is contained in a program by relating it to a known function or process taken from an area or discipline familiar to the users. Program metaphors such as books, book shelves, space

exploration, or buildings with different rooms help users to organize the program's content and the access to the content contained in the program. [Jones89]

The use of metaphor can be helpful when it fits well into a situation, but it is not a panacea and is not guaranteed to add value. The use of icons as metaphors for functions is a good example. It can be a gamble if someone will understand the connection between an icon and the function.

While metaphors aren't always as useful as other solutions, it is important to note that in the right situation they can be a vital part of a quality user interface. The folder is a particularly useful and successful metaphor. Its purpose is immediately apparent, and by placing one folder inside another the user creates a naturally intuitive hierarchy. The counterpart in the character user interface is the directory/subdirectory construct. This has no clear correspondence to anything in the physical world, and many non-technical people have difficulty grasping the concept.

The bottom line is that if a metaphor works naturally, use it by all means. But at the first hint that the metaphor is not clearly understood or has to be contorted in order to accommodate reality, it should be strongly considered as to whether it will really help or not.

Intuitiveness

It is generally perceived that the most fundamental quality of any good user interface should be that it is intuitive. The problem is that "intuitive" means different things to different people. To some an intuitive user interface is one that users can figure out for themselves. There are some instances where this is helpful, but

generally the didactic elements geared for the first-time user will hamper the effectiveness of intermediate or advanced users.

A much better definition of an intuitive user interface is one that is easy to learn. This does not mean that no instruction is required, but that it is minimal and that the users can "pick it up" quickly and easily. First-time users might not intuit how to operate a scroll bar, but once it is explained they generally find it to be an intuitive idiom.

Designing intuitive user interfaces is far more an art than a science. It draws more upon skills of psychology and cognitive reasoning than computer engineering or even graphic design. The process of Usability Testing, however, can assess the intuitiveness of a user interface in an objective manner. Designing an intuitive user interface is like playing a good game of tennis. Instructors can tell you how to do it, but it can only be achieved through hard work and practice with a lot of wins and losses on the way.

Consistency

Consistency between applications is always good, but within an application it is essential. The standard GUI design elements go a long way to bring a level of consistency to every panel, but "look and feel" issues must be considered as well. The use of labels and icons must always be consistent. The same label or icon should always mean the same thing, and conversely the same thing should always be represented by the same label or icon. [Shneiderman98]

Simplicity

The complexity of computers and the information systems they support often

causes us to overlook Occam's Razor, the principle that the most graceful solution to any problem is the one which is the most simple.

A good gauge of simplicity is often the number of panels that must be displayed and the number of mouse clicks or keystrokes that are required to accomplish a particular task. All of these should be minimized. The fewer thing users have to see and do in order to get their work done, the happier and more effective they will be.

A pitfall that should be avoided is "featuritis," providing an over-abundance of features that do not add value to the user interface. New tools that are available to developers allow all kinds of things to be done that weren't possible before, but it is important not to add features just because it's possible to do so. The indiscriminate inclusion of features can confuse the users and lead to "window pollution." Features should not be included on a user interface unless there is a compelling need for them and they add significant value to the application.

Prevention

A fundamental tenet of graphic user interfaces is that it is preferable to prevent users from performing an inappropriate task in the first place rather than allowing the task to be performed and presenting a message afterwards saying that it couldn't be done. This is accomplished by disabling, or "graying out" certain elements under certain conditions.

One of the advantages of graphic user interfaces is that with all the options plainly laid out for users, they are free to explore and discover things for themselves. But this requires that there always be a way out if they find themselves somewhere they realize the shouldn't be, and that special care is taken to make it particularly difficult to

"shoot themselves in the foot." A good tip to keep users from inadvertently causing damage is to avoid the use of the Okay button in critical situations. It is much better to have button labels that clearly indicate the action that will be taken.

Locus of Control

The user should always feel in direct control of the computer interface, and should never feel that the computer has "automatically" taken actions that could arbitrarily change the user's preferences, destroy data, or force the user to waste time. Well designed interfaces are also forgiving of user's mistakes, and are stable enough to recover "gracefully" if the user makes mistakes, supplies inappropriate data, or attempts to take an action that might result in irreversible loss of data.

Aesthetics

Finally, it is important that a user interface be aesthetically pleasing. It is possible for a user interface to be intuitive, easy to use, and efficient and still not be terribly nice to look at. While aesthetics do not directly impact the effectiveness of a user interface, users will be happier and therefore more productive if they are presented with an attractive user interface.

3.4 The User Interface Prototyping

The conventional means today is the only way to build good user interface is by iterative refinement [Buxton80]: build an initial version of the user interface, and then test it with users and revise it as many times as you have money and time for it. [Gould85] Nevertheless, building and repeatedly revising the actual software system is

difficult and expensive.

Prototyping is an important technique to reduce the cost and risk involved in developing complex software system [Rudd94]. It essentially involves building a small-scale version of complex software in order to acquire critical knowledge required to build the system.

Rapid prototyping is increasingly recognized as a valuable approach [Shneiderman98], particularly with tools such as the User Interface Management System, which allows one to build a framework for the prototype that then becomes the basis for the final product.

In presenting the case for the need for prototypes, Wasserman and Shewmake [Wasserman90] cite several reasons:

- It enables the user to evaluate the interface in practice and to suggest changes to the interface.
- It enables the developer to evaluate user performance with the interface and to modify it so as to minimize user errors and improve user satisfaction.
- It facilitates experimentation with a number of alternative interfaces and modification of interfaces.
- It gives the user a more immediate sense of the proposed system and thereby encourages users to think more carefully about the needed and desirable characteristics of the system.

- It reduces the likelihood of project failure.

Other benefits include: reducing the total development product costs, improving the product's functional specification in quality and completeness and providing a common reference point for all members of the design team, users and marketing.

Prototyping is an iterative analysis technique in which users are actively involved in the mocking-up of screens and reports. The purpose of a prototype is to show people the possible design(s) for the user interface of an application. As we see in Figure 3.2, there are four steps to the prototyping process:

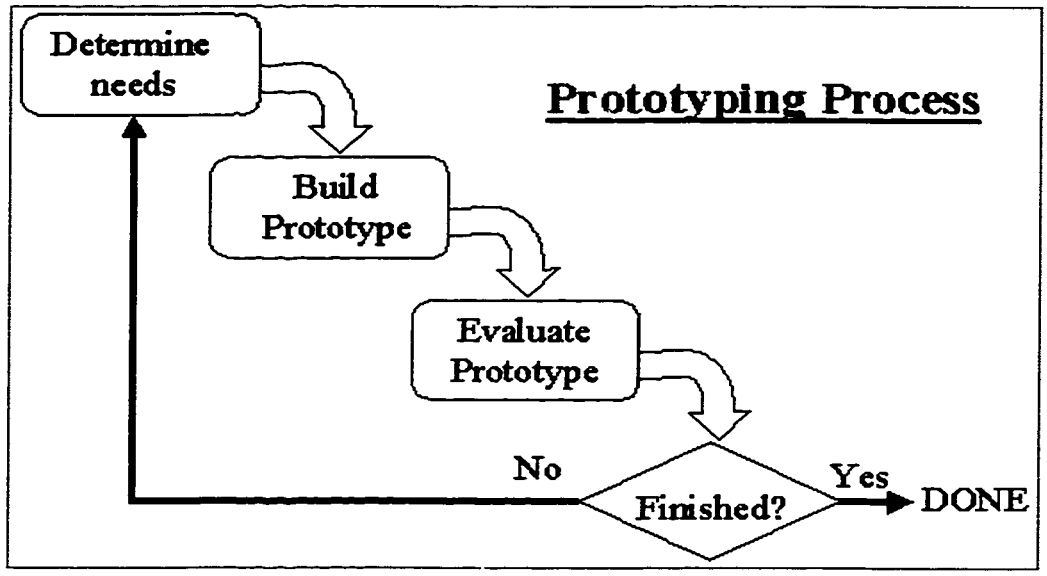

Figure 3.2 User Interface Prototyping Process

Determine the needs of the users. The requirements of the users drive the development of the prototype as they define the business objects that our system must support. We can gather these requirements in interviews, in CRC (class responsibility collaborator) modeling sessions, in use-case modeling sessions, and in class diagramming

sessions [Ambler98].

1. Build the prototype. Using a prototyping tool or high-level language we develop the screens and reports needed by the users. The best advice during this stage of the process is to not invest a lot of time in making the code "good" because chances are high that we may just scrap our coding efforts anyway after evaluating the prototype.
2. Evaluate the prototype. After a version of the prototype is built it needs to be evaluated. The main goal is that we need to verify that the prototype meets the needs of our users. There are three basic issues needed to be addressed during evaluation: What's good about the prototype, what's bad about the prototype, and what's missing from the prototype. After evaluating the prototype we'll find that we may need to scrap parts, modify parts, and even add brand-new parts.
3. Determine if we're finished yet. We want to stop the prototyping process when we find the evaluation process is no longer generating any new requirements, or is generating a small number of not-so-important requirements.

3.5 The GUD User Interface Design Process

One part of our task in the GUD project was the design and implementation of the user interface. The user interface design process was performed following the conventional methodologies, discussed at the previous section.

Requirements

After examining the sponsor's demands and discussing with the GUD team members, we list the requirements the GUD user interface should provide:

- As an information delivery system, the user should be easily to access to the information source. The user interface should provide intuitive menus and links, straightforward query pages, help and tips for the novice users
- As a Web-based application, the browser issue needs to be addressed. We will defer the discussion of the browser issue to the next section. Nonetheless, the user should have no problem to use the system via currently available browsers in the consistent way.
- To ease the maintenance and management, there should be little or even none user site (client) software installation. All the user needs is a freely available popular browser.
- The initial stage of the user interface development should be focused on the general functionality. The sophisticated components could be built upon the success of the earlier versions.
- The security issue might be addressed. It could be implemented at the system level, as well as the application level. The registration page should be provided to create the new user profile.

Concept Design

The GUD project presents the university-related information to the users. We define the data model, communication model, and business function model in the concept design phase:

- Data Model:

The GUD data model defines the relationships between the GUD university-related entities.

As showed in the figure 3.3, the entity *University* contains the general university information, like the university name, location, and the Web URL. Each university has faculties (*Faculty*), departments (*Department*), and faculty members (Staff). Each staff has his/her research topics (*Research Interest*), affiliates to research groups (*Research Group*), and involved in research projects (*Research Project*). Each university possesses patents (Patent), and receives research grants and funds from agent (*Agent*)

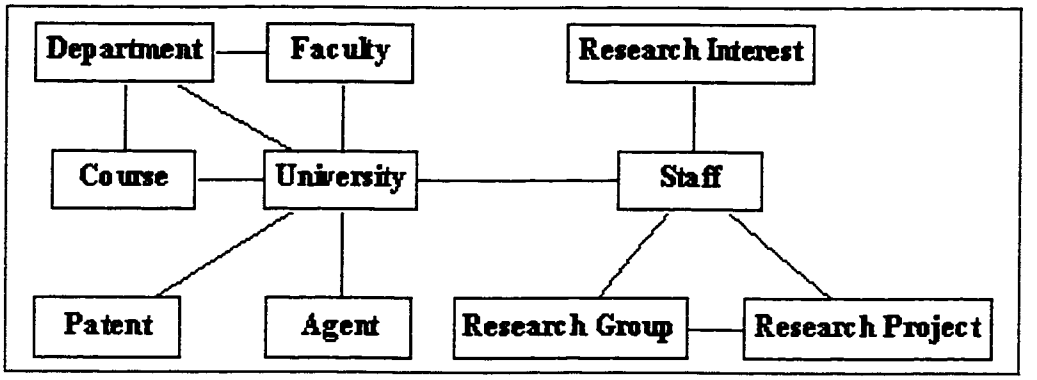

Figure 3.3 The GUD User Interface Data Model

- Business Function Model:

The GUD business function model defines the scope of the application, i.e. university information system, and itemized the component business function. The GUD user interface business functions are decomposed to *Registration* (provide the registration service), *Online Help* (provide the help service), *Query* (the user starts the search here), and *Feedback* (the user gives the feedback). The *Query* is further broken

down to University (university general query), Research (research-related query), People (Reseracher-related query) and General (the user makeup query).

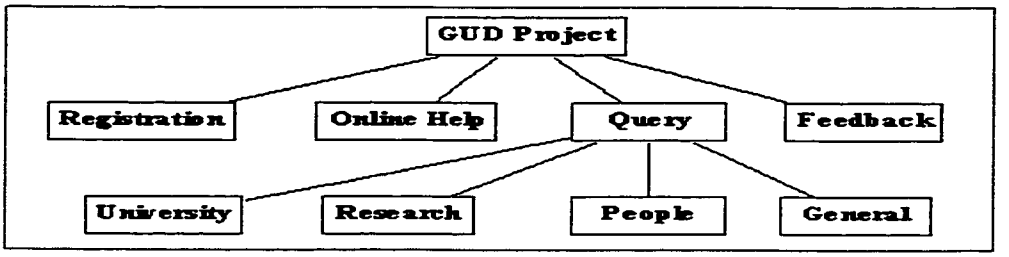

Figure 3.4 The GUD User Interface Business Function Model

- Communication Model:

The GUD communication model maps the interaction between the GUD data model and business function model. In this Web-based application, the Web server extension acts as a middleware to link the business function model with the data model. The interaction between the business function model and the Web server extension could be conducted through standard http protocol, the Java RMI, and COBRA, while the Java JDBC, or Microsoft ODBC, among others, bridges the middleware with the data model.

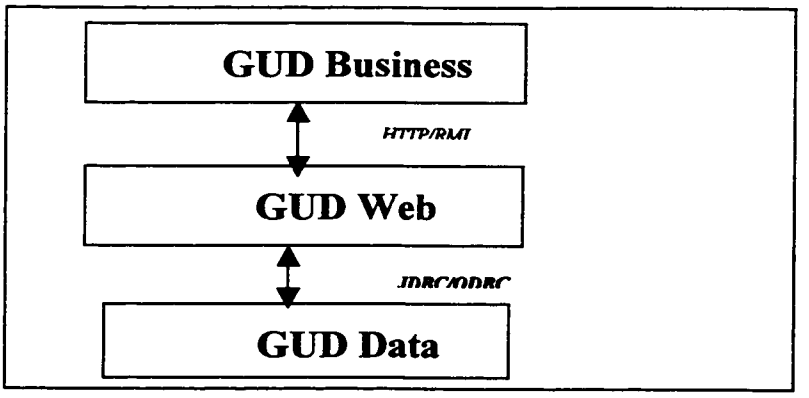

Figure 3.5 The GUD User Interface Communication Model

Logical Design

In this design phase, we specify the hardware and software on which the GUD project will be built. Java is employed to build extensible and platform-independent business functions running on the GUD Web server (Java Web Server, or MS IIS) on the Windows NT workstation. The Web-based user interface is developed using standard HTML embedded scripts and Applets. The GUD working browser are Internet Explorer 4+ or Netscape Navigator 4+, the discussion of the browser diversity issue will be addressed at the next section.

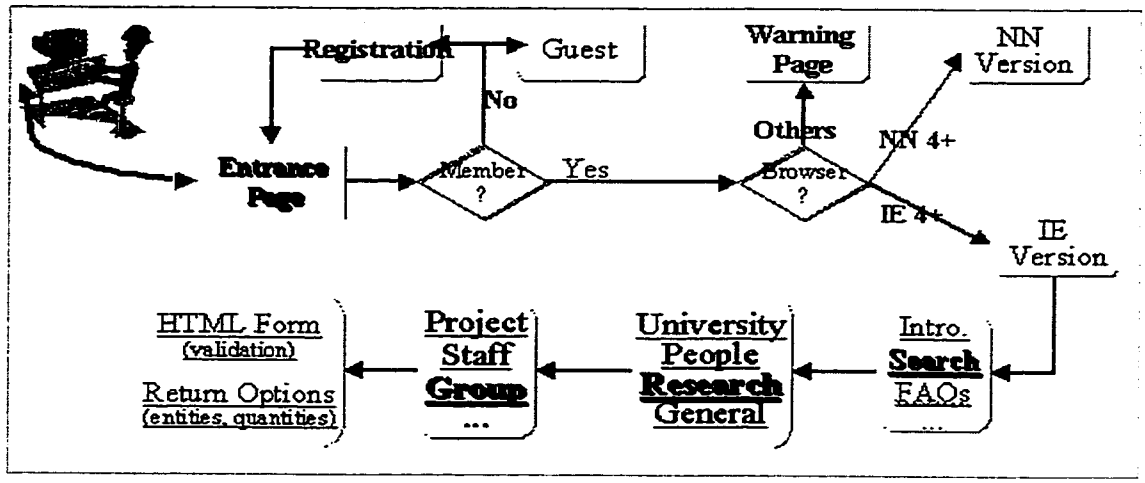

Figure 3.6 The GUD User Interface Prototype

The GUD user interface should provide user-friendly, easily navigated, lightweight and intuitive interface to the user. The first time user could quickly pick up the system, while the frequently user could use short cut, macro and other means to speed up the search.

For the access control, a user profile will be used to store the user information. A registration page is provided for user registration, while a guest page is helpful for the

new user to try the system before registering as an authorized user.

Since the GUD system working browsers are Internet Explorer (IE) 4+ and Netscape Navigator (NN) 4+, there is a need to develop two sub-sections corresponding to two browser types. To assist the new users, a warning page should be provided to alarm the user who is using the different browser (version) other than IE 4+ or NN 4+. The testing program is being used to distinguish the browser version and automatically link to the correction subsection pages.

Before the user enters the system, the testing script embedded in the "entrance page" automatically tests the user's browser type and browser version and decides how to process. The user is not aware of the background testing. If IE 4+ is used, the browser will automatically request pages (e.g. *IE front page)* that fit IE 4 features; if Navigator 4+ is used, the browser will request pages (e.g. *NN front page)* that work best with Navigator 4+. Otherwise, the *warning page* is displayed to alert the user to update his browser.

Figure 3.5 illustrates the IE 4+ case. After opening *IE front page*, the user interacts with the system through menus, hyperlinks, forms, buttons and checkboxes. The *menu frame* contains the hyperlinks to the FAQ's page, registration page, search page, etc., which are projected to the *main frame*. The *main frame* is the place to display the pages, make the query and display the results. Client-side form validation and the advanced return options are also processed here.

The *construction* and *usability testing* phases of the GUD project will be discussed in the next chapter.

3.6 Issues on the GUD User Interface Design

In this section, we will address some issues arisen from the GUD system user interface design, namely the usage profile diversity, browser diversity and form validation.

3.6.1 Usage Profile Diversity

"Know the user" was the first principle in Hansen's [Hansen71] classic list of user-engineering principle. The successful user interface design always starts from the considerations of the possible users. There is no exception in our GUD Database System. Since our system is mainly on the University-related information, we expect that the users come from academic units, including professors, researchers and students. We further divided our potential users by three groups: the novice or first-time user, the intermediate user, and the expert user.

A once-in-a-while user with a minimum of learning will want to be able to get at least a few straightforward things done. In fact, even an expert user in one domain will be a novice in others. Users will be clerical workers, information specialists, executives, engineers, and others. Attention to novice-oriented and to tutorial help features is required.

Users also want and deserve the reward of increased proficiency and capability from improvements in their skills, their knowledge, their conceptual orientation to the problem domain and to their workshop's system of tools, methods, conventions, etc. "Advanced vocabularies," short concise control notation and conventions in every special

domain will be important and unavoidable.

The novice user has little knowledge on the task or interface concept. In contrast, the first time user might know exactly what his task is but has shallow knowledge on the interface. In face this type of user, we need providing instructions, dialog boxes, FAQs and online help. To reduce his/her anxiety and frustration, step-by-step online tutorial and constructive error message are effective. Restricting vocabulary to a small number of familiar, consistently used concept term is essential to begin developing the user's knowledge.

Most of the users casually use the system with certain knowledge. They have stable task concepts and broad knowledge of the interface concepts, but have difficulty in memorizing the structure of the menus or the location of features. Consistent sequences of actions, meaningful messages, and guides to frequent patterns of usage will assist this type of users to perform their tasks properly. Well-organized reference manual usually is welcome by these casual users.

Expert users are completely familiar with their task and interface concepts and seek to get their jobs done quickly. They ask for rapid response time, brief and non-distracting feedback, and the capacity to carry out actions with just a few strikes or clicks. To these users, we need to provide shortcut, macro, abbreviation and other accelerators.

3.6.2 Browser Diversity

The user accesses the system by remitting the URL to our system server and loading the server-sent documents (HTML file) in his/her browser. There are two most

popular browsers, Microsoft's Internet Explorer and Netscape's Netscape Navigator. There are also different visions associated with them, including IE 2, IE 3, IE 4 and Navigator 2, Navigator Gold 3, Navigator 4. There are even further minor versions like Gold 3.01 and Navigator 4.04. Besides Internet Explorer and Netscape navigator, there are also some browsers such as HotJava and so on. With the same HTML document, different browser displays different "look and feel". Things get worse when the different browser is running on the different operating system, like Windows, Unix, and Mac.

In our GUD user interface design, we only deal with Internet Explorer 4 and Netscape Navigator 4 versions. There are three reasons to do so: first, both browsers are most used, well evolved, freely downloadable, and comparatively stable; second, both support most of the emerging technologies. The latest major versions, IE 4 (including IE 5) and Navigator 4, therefore, are chosen for deploying the newest techniques, like Java, frames, and JavaScript; third, it's relatively easier to maintain only two deviations. However, we leave room for the later development for the other browsers' users.

3.6.3 Form Validation

HTML forms are a widely used method of gathering data on the Web. Creating and posting a form is the easy part; the hard part is sorting through the data after they come in and making sense of them. As often being happened, blank and incorrect-formatted-filled forms can clog up the database, increase the network traffic and take up space on the server.

Form validation can help alleviate this problem. Using either client-side or server-side validation, it's possible to detect form errors and alert the user before they reach

the database. Client-side validation process with scripting language like JavaScript is quick and doesn't disturb the server, but it is susceptible to browser incompatibilities. Server-side validation works well on all browsers at the expense of the slower process and more network traffic.

In the GUD system, the client-side validation is employed. The validation process is fast and efficient, and there are no compatibility problems since we work only on IE 4 and Navigator 4, and scripting functions well on both versions. However, we also want to perform server-side validation. It can be used either to catch the 'bad data' passed through the client-side validation script, or to directly function the validation if we need to deal with other browser versions later (e.g. IE 3).

3.7 Summary

User Interface design is important for several reasons. First of all, the more intuitive the user interface, the easier it is to use, and the easier it is to use the cheaper it is. The better the user interface, the easier it is to train people to use it, reducing the training costs. The better the user interface, the less help people will need to use it, reducing the support costs. The point to be made is that the user interface of an application will often make or break it. Although the functionality that an application provides to users is important, the way in which it provides that functionality is just as important. An application that is difficult to use won't be used.

In this chapter, we have addressed the design issues of the GUD user interface. The design process is theoretically examined, and the design concepts are carefully

introduced. The GUD user interface prototype is presented, followed by the discussion of the design issues. In the next chapter, we will investigate the design scheme of the GUD Web server extension --- Data Processing Agent (DPA).

Chapter 4

Web Server Extension Design on the GUD Project

4.1 Introduction

A Web server is the software that processes the Web communications. It uses standard HTTP protocol to translate requests from a browser into the useful information. The Web server is the focal point for an enterprise-class middle tier, because it provides much of the infrastructure and functionality required to support distributed applications. As a class of software, the Web servers can provide a range of services needed by many applications, including centralized management, load balancing, and transaction management. A Web Server extension is an application module to provide middleware services for the specific application needs.

In this chapter, we will discuss the design issues of GUD Web server extension -- Data processing Agent (DPA). The DPA framework is presented first. The working mechanism of the DPA is then elaborated. As part of DPA, the Data Mediator (DM) interfaces with the WMAs and performs the database table creating, data populating and updating. The process flowchart therefore is examined. The chapter ends with the summary.

4.2 The Data Processing Agent (DPA) Framework

In our research group, we have developed a GUD Web Server Extension -- Data Processing Agent (DPA). The overall DPA framework is shown in Figure 4.1. The extension consists of several components that are performing their respective own functionality. The integration of those components is a benefit of our research agenda. By combining components in a single module, that have been used individually, we gain new insight and discover new research directions for the components. The most important components follow in rough order of their invocation in the DPA.

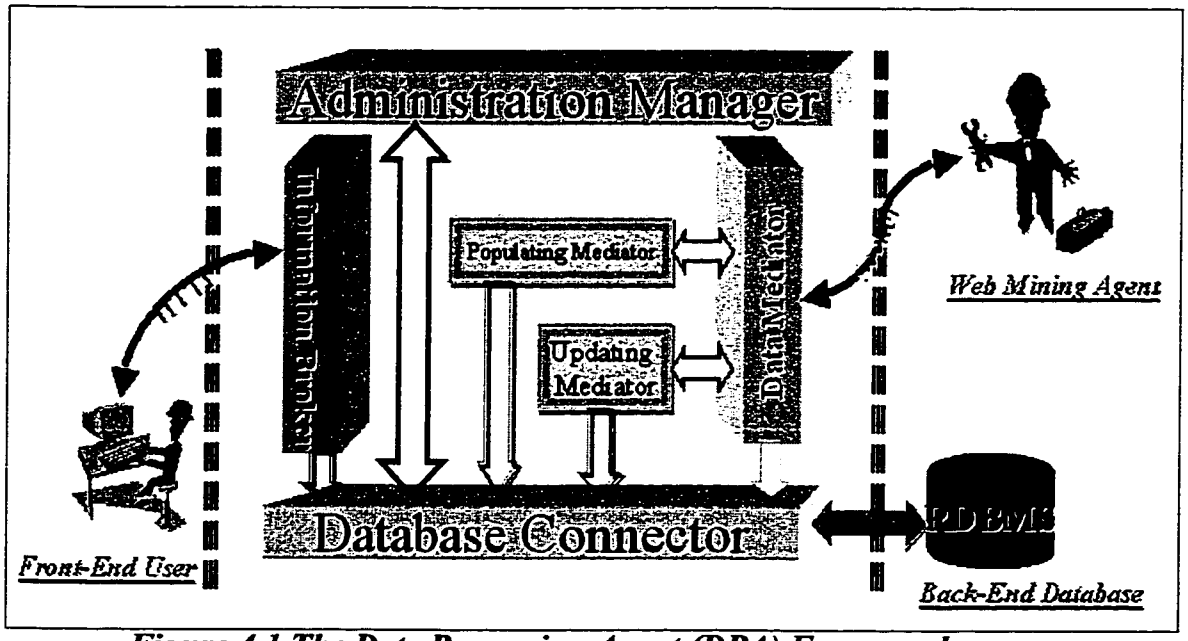

Figure 4.1 The Data Processing Agent (DPA) Framework

- IB: *Information Broker* works intermediately between the front-end user and the GUD database. IB should receive and parse the user request, connect to the database, and send the formatted results back to the user. IB may also process the server-side form validation.

- DM: *Data Mediator* is communicated with the Web Mining Agents (WMAs). DM keeps listening to the data transmission request from the WMAs, therefore it can be treated as a dumb server running all the time, while the WMAs reap the data from the Internet and relay them to DM. The DM manipulates the data and passes them over populating mediator (PM) or updating mediator (UM) to populate/update the database.

- PM: *Populating Mediator* is an object being used to insert the new record into the database

- UM: *Updating Mediator* is an object being used to update the existed record at the database.

- DC: *Database Connector* bridges IB and DM with the database. The DC could be used to access to multi-vendor database and isolate the low-level details from the application components. Even in the GUD system, we mainly deal with text data, which the relational database is deployed, the multimedia type data like image, audio and video could be added and the DC should provide a general interface to the comprehensive database servers.

- AM: *Administration Manager* monitors the performance of the system, adjusts the system parameters, and manages the user profile and system security.

The DPA is designed to be a robust and maintainable media for communicating among the GUD database, the WMAs, and GUD user interface. It has the following properties:

1. ***The DPA is platform independent.*** It is written in Java. It operates on any platform that is capable of running 100% Java compatible VM.

2. ***The DPA is database independent.*** It communicates with the data layer via a database-independent JDBC API and utilizes a standard subset of SQL that is supported by all database vendors.

3. ***The DPA is efficient.*** It intelligently shares database connections between multiple client applications, which increases database capacity.

4. ***The DPA is scalable.*** One instance of the DPA can work with many databases at the same time. Several instances of the DPA can operate on the same database simultaneously.

4.3 The Services in DPA

The DPA consists of several separate server processes (services). Each process performs an independent logical task, which facilitates the implementation of modifications on a code that is easily maintained and supported. The following is a list of these processes:

Query Service:

One component of the DPA - Information Broker (IB) - implements the query service. It works like a broker between the front-end user and the back-end database. The IB can handle multi-user access requests by spawning one process accompanying to each user, instead of each request, which significantly improves the performance.

Access Control Service:

The DPA authorizes users' logons and logoffs, as well as controls users' access to different "functional areas" of the application. When a user logs in, the DPA queries the user table and gets the user's access rights. The access rights are keys to various interface screens. The user's ACL (Access Control List) object is created that encapsulates the user's information including the access rights. Each user interface screen has a "lock" that contains information of what kind of user may access it and what visual controls should be enabled or disabled or made invisible, etc. This approach allows interception of any unauthorized access in the very beginning.

Administration Service:

The DPA provides the administration service that is a separate application implementing the following functions:

- Monitors database connections
- Monitors users' database statements
- Logs off users and frees resources.

Data Manipulation Service:

The Data Mediator interfaces with the Web Mining Agents, receives the piped university-related data, manipulates the data, and connects to the database to process the data populating.

The figure 4.2 below shows the working mechanism of the DPA. On the one side, the Data Mediator (DM) interfaces with the Web Mining Agents (WMAs), receives and

manipulates the data. The data then are relayed to the PM (Populating Mediator) or UM (Updating Mediator) to process the data populating or updating. The whole process can be done anytime (like evening or weekend)

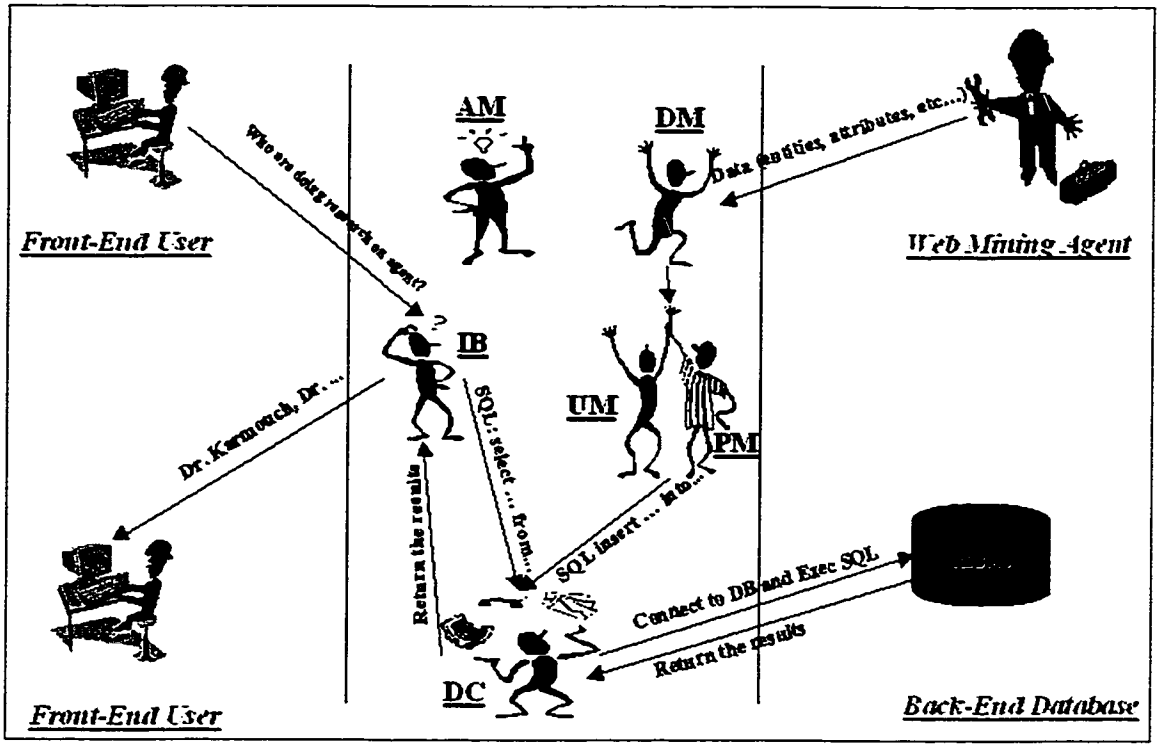

Figure 4.2 The DPA in Action

On the other side, the request from the user is handled by the Information Broker (IB). The IB parses the request, generates the SQL command, and then connects to the database through the DC (Database Connector). The results are dynamically formatted to html form and sent back to the user.

4.4 The Working Flowchart of the Data Mediator (DM)

The Data Mediator (DM) is used to interface with the WMA, manipulate the received data, and process the database population. In this section, we will illustrate the

working flowchart of the DM.

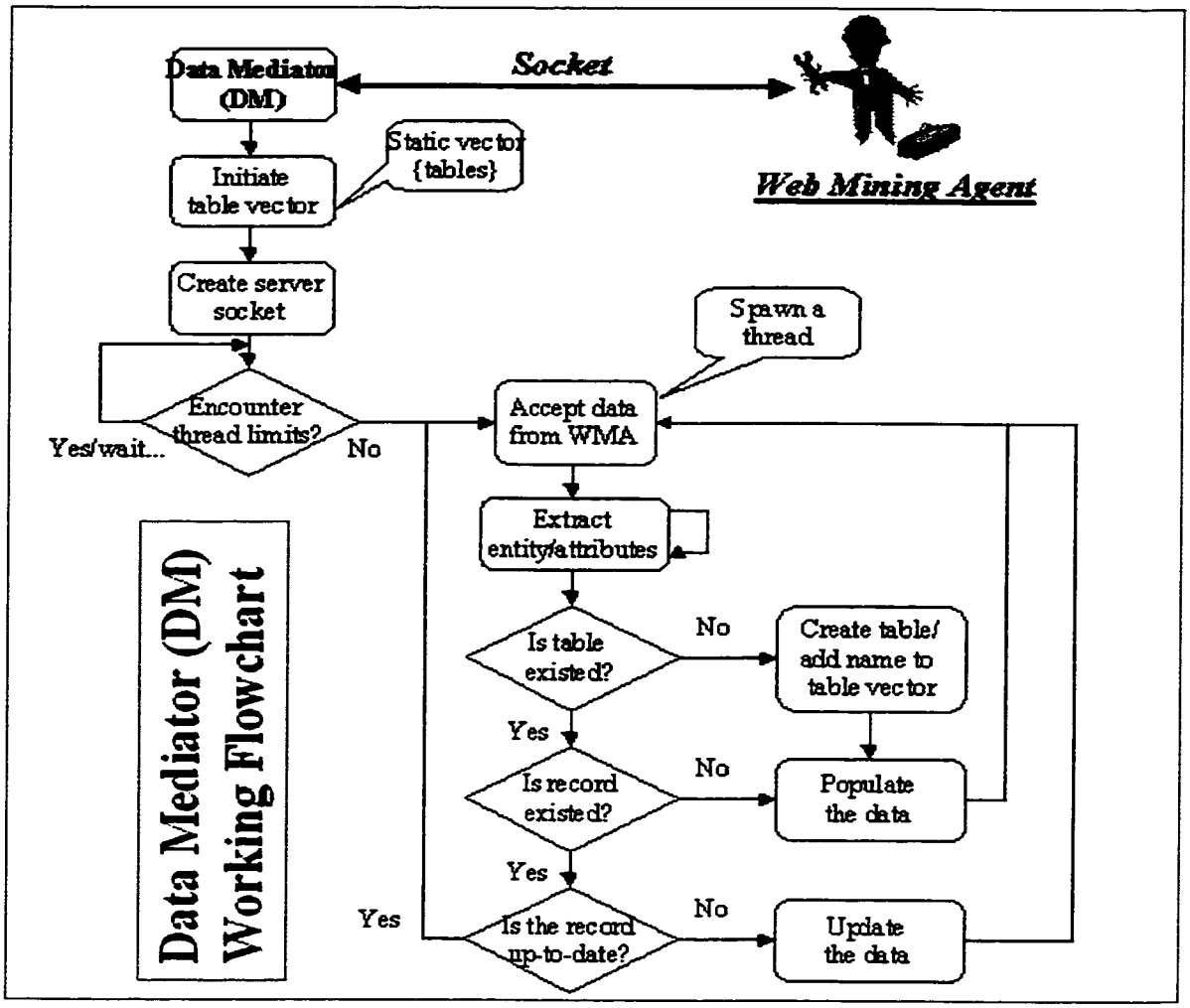

Figure 4.3 The flowchart of the DM

As shown in figure 4.3, the DM is running as a daemon server, and listening to the request from the WMAs. Each connection request from a WMA is counted, and the request will be rejected if the limit is encountered. The database metadata (like tables) are retrieved and static table vector is created to store the table list whenever the DM is initiated.

The reaped data from the WMA are channeled to the DM through the dedicated socket. The DM extracts the piped data, and generates the SQL command. If the data entity (table name) does not exist in the database, the table creating is processed first before the data populating/updating. And then, the DM checks whether the data record already exists in the database, and performs the database populating (insert) if not. If the record exists, the DM further checks whether the record is up-to-date. The updating is processing if the answer is no, and going back to the next data record without doing anything if the answer is yes.

4.5 Summary

In this chapter, we have analyzed the DPA framework, its features, and the services functioned inside. The GUD's DPA is a cross-platform, database-driven, application server extension. With DPA, the core application resides on a Web server, and clients interact with the application using a standard Web browser. DPA builds the web pages dynamically, by constructing the page from information stored in one or more databases.

In the next chapter, we will report the implementation details of the GUD project.

Chapter 5

Implementation

5.1 Introduction

In this chapter, we will go through the implementation of the GUD project. The implementation details include two parts: the user interface, and the Web server extension -- DPA. We first introduce the programming languages and tools being used in the GUD project, namely html, JavaScript, Java, among others. And then we will describe three packages in the GUD project and the classes inside each of them. The GUD project samples are presented. The snapshots are provided to show the layout of the GUD user interface, the query process and the result pages. The summary concludes the chapter.

5.2 Implementation languages and tools

In the GUD project, we used html, Java (applets), and JavaScript to implement the user interface, while we employed Java Servlets, JDBC API to construct DPA. The Java Web Server was initially used to be the system Web server, and it was later replaced by MS IIS and Live Software's Jrun [Live99]. MS SQL Server 6.5 was the RDBMS to store the GUD data. We also used Unified Modeling Language (UML) and Rational Rose

[Rational99] tool to visually model the DPA static structure, i.e. the system package and class diagram. Java Server Page (JSP) is introduced later to handle the presentation logic.

5.2.1 The World Wide Web, HTML and Java

The World Wide Web is a vast collection of information that is spread across hundreds of thousands of computers around the world. Almost every item of information on the World Wide Web can be accessed directly.

The World Wide Web used three new technologies:

- HTML (HyperText Markup Language) used to write Web pages.
- HTTP (HyperText Transfer Protocol) to transmit those pages.
- Web browsers to receive the data, interpret it, and display the results.

Using HTML, almost anyone with a text editor and access to an Internet site can build visually interesting pages that organize and present information in a way seldom seen in other online venues. In fact, Web sites are composed of *pages* because the information on them looks more like magazine pages than traditional computer screens. [HTML99]

HTML isn't the only way to present information on the Web, but it's the glue that holds everything together. In addition to being a markup language for displaying text, images, and multimedia, HTML provides instructions to Web browsers in order to control how documents are viewed and how they relate to each other.

Java is perhaps one of the most talked about advances in Internet technology since

the World Wide Web. Perhaps the most visible examples of Java technology-based software today are on the Internet. They're nimble, interactive programs called "applets." Applets work inside Web browsers on computers and other devices. And there are other kinds of Java technology-based software: programs written in the Java programming language can run directly on a computer (without requiring a browser), or on Web servers (servlets), on large mainframe computers, or other devices.

In our GUD project, Java technology is extensively exploited. We have used Java applets in the user interface and the servlets, JSP and JDBC in the DPA. The servlets and JDBC will be introduced at the following sections. We have used JavaServer Page to handle the presentation logic which was previous handled by the servlets.

5.2.2 JavaScript and the Form Validation

Now it's time to introduce JavaScript, which we used extensively in our user interface implementation. JavaScript is one of a new breed of Web languages called *scripting languages*. These are simple languages that can be used to add extra features to an otherwise dull and dreary Web page. Scripting languages like JavaScript make it easy for nonprogrammers to improve a Web page.

JavaScript was originally developed by Netscape for its browser, Netscape Navigator. It includes a convenient syntax, flexible variable types, and easy access to the browser's features. It can run on the browser without being compiled, and the source code can be placed directly into a Web page. [Netscape99]

One of the most important uses of JavaScript is to validate form input to server-

based programs such as server-side applications or CGI programs. Client-side JavaScript statements embedded in an HTML page can respond to user events such as mouse-clicks, form input, and page navigation. For example, you can write a JavaScript function to verify that users enter valid information into a form requesting a telephone number or zip code. Without any network transmission, the HTML page with embedded JavaScript can check the entered data and alert the user with a dialog box if the input is invalid. This is useful because:

- It reduces load on the server. "Bad data" are already filtered out when input is passed to the server-based program.
- It reduces delays in case of user error. Validation otherwise has to be performed on the server, so data must travel from client to server, be processed, and then returned to client for valid input.
- It simplifies the server-based program.

As we know, Java is powerful enough to add animation, sound, and other features to an applet--but it's very cumbersome to directly interact with an HTML page. JavaScript isn't big or powerful enough to match Java's programming power, but it is uniquely suited to work directly with the elements that comprise an HTML document. By combining the best features of Java and JavaScript, we can significantly improve the system performance and offer a new level of interactivity.

5.2.3 CGI, Java Servlets and User Interactivity

The web server has to handle the query and access the databases. These

processing functions can be supported through many interfaces. The Common Gateway Interfaces (CGI) and the Servlet API, among them, are intended to be vendor-independent. CGI is simple and widely supported. However, it is not very efficient because of the common technique of spawning a new process on the server side for each CGI request. The overhead can be significant for popular sites. Vendors have developed proprietary alternative such as ISAPI (Microsoft) and NSAPI (Netscape) to address the performance issue. Javasoft proposed servlets as a standard alternative to CGI.

5.2.3.1 Overview of CGI

The Common Gateway Interface, or CGI, permits interactivity between a client and a host operating system through the World Wide Web via the HyperText Transfer Protocol (HTTP). It's a standard for external gateway programs to interface with information servers, such as HTTP or Web servers. A plain HTML document that the Web server delivers is static, which means it doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information. [NCSA99]

The CGI scripts are programs which handle information requests and return the appropriate document or generate a document on the fly. The CGI can be used for a variety of purposes, the most common being the handling of FORM requests for HTTP. An HTTP Server is often used as a gateway to a legacy information system; for example, an existing body of documents or an existing database application. The CGI is a convention between HTTP servers about how to integrate such gateway scripts and programs.

5.2.3.2 Introduction of Java Servlet API:

Java servlets are protocol- and platform- independent server-side components that can be dynamically loaded like applets. The web server deals with the connection process: accepting and establishing a request from the client. Once a connection is accepted, it is processed by servlets. A servlet, when used in the context of the web server, can be regarded as analogous to the CGI script. [Java99]

A CGI script is usually run in response to a web request, for instance, processing a form submitted by a user using the web site. A servlet, can be used for the same task, but has the advantage of being much faster and efficient. Servlets offer better performance than most legacy CGI application because servlets are only loaded once and used over and over by the server. In addition, servlets using JDBC to connect to any database servers are truly object-oriented and portable across all servers and operating systems (unlike NSAPI and ISAPI).

5.2.3.3 The Features of Java Servlets

By using existing mature server technology, such as Web servers, in combination with servlets, sophisticated applications can be written to hide the complexities and platform-specific behaviors of the particular server implementation. Servlets save the developer from worrying about the inner workings of the server. Form data, server headers, cookies, and the rest are handled by the servlet API's underlying classes. One of the biggest differences between today's server-side applications and servlets is performance. A single JVM runs on the server and the servlet is loaded once when it is

called. It is not loaded again until the servlet changes, and a modified servlet can be reloaded without restarting the server or application.

The servlet stays resident in memory so it is fast. Static or persistent information can be shared across multiple invocations of the servlet, allowing shared information between multiple users. Servlets are also modular; each servlet can perform a specific task and can be tied together. In short, servlets can talk to each other.

Generally speaking, Java Servlet API brings the power of Java to the servers like:

- **CGI replacement.** Servlets offer extensive APIs to respond in the best way to an HTTP request POST-ed from an HTML form. Through these APIs, servlets provide far more functionalities than traditional CGI, as well as compelling performance advantages over both the CGI approach and the Fast-CGI approach. One of the biggest performance features of servlets is that they do not require creation of new process for each incoming request.

- **Multi-user interoperability.** Since servlets can handle multiple requests concurrently, the requests can be synchronized with each other to support multi-user applications such as on-line conferencing.

- **Mobile collaboration.** Developers could define a community of active agents, which share work among each other. The code for each agent would be loaded as a servlet, and the agents would pass data to each other. Using the Serialization API, they could even pass Java classes to one another, but also Servlet files. Servlets could then literally crawl over the network, moving from one place to

another, helped by other servlets.

- **Load balancing.** Servlets running on an overloaded host could forward some of their requests to other servers. This technique can also be used to partition a single logical service between several servers.

5.2.4 JDBC and database connectivity

Traditional web access to the database was done with the use of CGI script accessing the database and creating output in the form of HTML document presented to the browser. The requests and responses were transmitted using HTTP protocol, and there was no session notion and it was not able to preserve the database transaction logic. Servlets address these problems. A servlet class allows a user over the network to query and update a database through JDBC.

5.2.4.1 What is JDBC

JavaSoft created the JDBC (Java Database Connectivity) specification to meet the urgent need for a standard database management system (DBMS) application programming interface (API) for Java. JDBC provides database access via Java that's independent of both the platform and the database host system the application runs on. The specification enables you to write Java code that establishes a connection to a Structured Query Language (SQL)-capable data source, sends SQL statements to the data source, and returns status messages and data records resulting from the SQL execution. JDBC also offers advanced functionality such as automatic conversion of different

database data types to Java data types, the streaming of large data records, cursors, and multiple result data sets. [Java99]

JDBC mandates no specific requirements on the underlying DBMS. Rather than dictating what sort of DBMS an application must have to support JDBC, the JDBC specification places all its requirements on the JDBC implementation. A JDBC implementation is mandated to support at least ASNI SQL-92 Entry Level. Since most common RDBMS and OORDBMS support SQL-92, JDBC can access most of the popular database systems.

5.2.4.2 JDBC Layers

JDBC consists of two main layers: the JDBC API supports application-to-JDBC Manager communications; the JDBC Driver API supports JDBC Manager-to-Driver implementation communications. The Manager handles communications with multiple drivers of different types from direct-interface implementations in Java to network drivers and ODBC-based drivers.

In terms of Java classes, the JDBC API consists of:

- java.sql.Environment - allows the creation of new database connections;
- java.sql.Connection - connection-specific data structures;
- java.sql.Statement - container class for embedded SQL statements;
- java.sql.ResultSet - access control to results of a statement.

The JDBC Driver API is contained in java.sql.Driver. Each driver must provide

class implementations of the following virtual classes: java.sql.Connection, java.sql.Statement, java.sql.PreparedStatement, java.sql.CallableStatement, AND java.sql.ResultSet. These virtual classes describe the API but are specific to how each database functions. The JDBC-ODBC bridge performs translations of JDBC calls to that which can be understood by ODBC clients at a C language level. This bridge is quite small due to the similarities of the specifications. The bridge is needed to isolate native C language calls to a controlled area while maintaining compatibility with non-JDBC databases. In our GUD project, the type 1 JDBC-ODBC Bridge was used to connect to the Microsoft SQL Server.

5.2.5 Java Server page (JSP) and JavaBeans:

The typical Web application involves generating large amounts of user interface contents such as HTML. As most of the contents are static, the others are dynamic, depending on the results of computations. The basic servlet mechanism discussed earlier specifies that servlets (Information Brokers) not only implement business logic but are also responsible for generating the user interface content that is sent to the client. However, this practice does not adequately address separation of skills since it couples two elements of the application, the user interface design and the program logic design that need to evolve somewhat independently. In addition, the presence of many print statements required to generate the HTML makes it harder to understand and develop the actual computation. We introduce Java Server Pages (JSPs) into our GUD project at the later stage of the system development.

5.2.5.1 JSP -- Separating the Presentation Logic with the Business Logic:

JavaServer Pages (JSP) are HTML pages extended with a number of mechanisms to allow dynamic content to be added to the page as it is sent to the client. All processing of JSP HTML extensions is done on the server and each of the extensions is either removed or replaced before the page is sent to the client. No additional client support is required.

JSP provides a way to combine the worlds of HTML and Java servlet programming. JSP are text files that look very much like HTML pages. The HTML is enhanced with new tags that specify the programming of a servlet to control the generation of dynamic content.

If the user has requested information that's in static pages that reside on the http server, the response will be an HTML version of the stored page. For dynamic responses, a call will go from the http server to an application server, where JSP and servlets are managed. The application server can be configured to pre-load servlets so they can quickly respond even to the first user's request.

JSP files only need to be compiled and loaded once. This avoids the CGI-bin problem of spawning a new process for each HTTP request, or the runtime parsing required by server-side includes. If a newer version of the JSP becomes available later, the application server will compile the newer one and load its version of the servlet.

5.2.5.2 JavaBean – Reusable Component

One of the most powerful features of JSP technology is the ability to access

reusable components, such as JavaBeans, from within a JSP file.

A JavaBean is a reusable software component. JavaBeans are Java objects that use a standard "method" naming conventions. This gives them predictable behaviour and makes tool-based manipulation easy. When used with servlets and JSPs in an application server, JavaBeans represent the business logic for an application. They provide:

1. *Properties* which can be set to configure their behaviour, and encapsulate the generated dynamic content.
2. *Methods* exposing the services of the business logic.

When a user sends a request by way of the browser, it may be for static or dynamic information. If dynamic, the request will be handled by a servlet. The servlet locates one or more JavaBeans needed to perform the request. It configures the beans using request parameters and invokes the required business logic methods. The servlet then stores a reference to the result bean so it is available to a JSP. The JSP uses the dynamic content available in the result bean to deliver a response back to the browser.

The JSP accesses the result bean object via a tag.

```
<jsp:useBean id="QueryResult" scope="session"
class="com.gud.servlets.ResultsetBean" />
```

The tag specifies how to find or create a bean. If an existing bean is to be accessed, it is either retrieved from the session or the request context. If a bean needs to be created, it is instantiated from a serialized bean, or a new bean is constructed.

5.2.5.3 JSP Application Model Used in GUD Project:

JSP pages may be included in a number of different application

architectures or models, and may be used in combination with different protocols, components and formats.

In our GUD project, we have deployed a much flexible model using JSP. In this configuration, the Web-based client make a request directly to the Information Broker (Servlets), which actually parses the request into a valid SQL statement, send the SQL to the GUD database server to be executed, wraps the results into a result bean and invokes the JSP page. The JSP page accesses the dynamic content from the bean and sends the formatted results (as HTML) to the browser. (Figure 5.1)

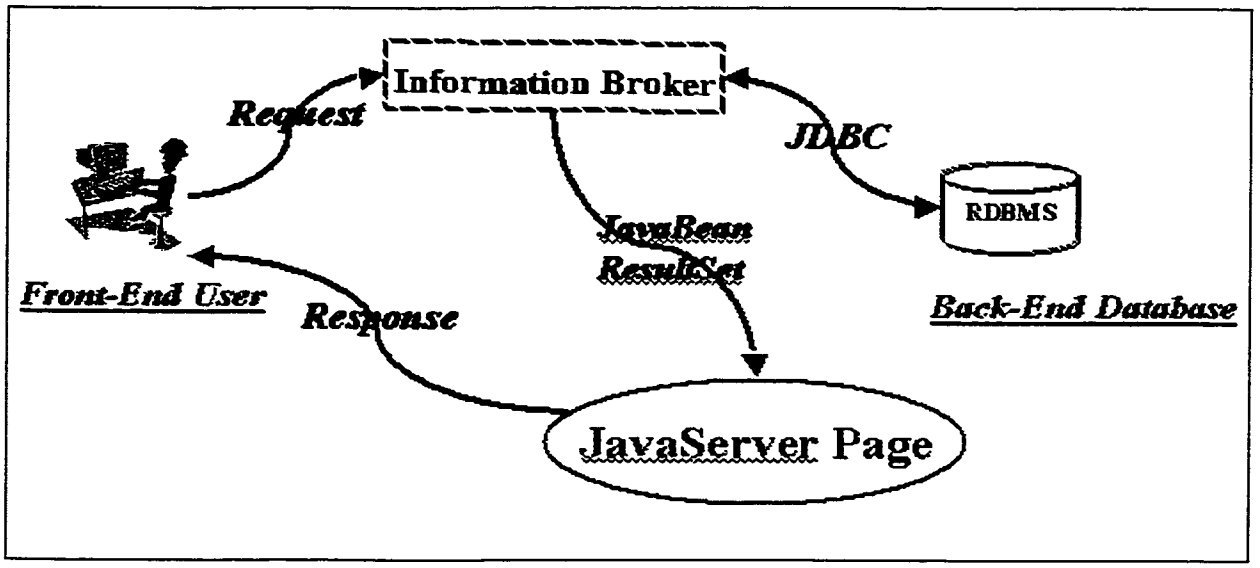

Figure 5.1 JSP Application Model used in GUD Project

This approach creates more reusable components that can be shared between applications. As we have successfully implemented the IB components, we can reuse those servlets and Beans and efficiently develop JSP pages to handle the presentation logic task.

More important, there is only one connection to the database for multiple Web-based clients. After the connection is established, all the following requests (read) from different sources are being processed by the same connection. It could significantly boosts the system performance if the JDBC driver could handle the concurrent reading well.

5.3 Classes Descriptions

In this section, we will introduce the Java classes that are implemented to perform the functions of Information Broker, Data Mediator, and other components in the GUD project. There are three packages: UI package contains Java applets (banners, menus, etc); IB package contains Java servlets to process the query request; DM package contains Java application to interface with the WMAs and perform the database populating and updating.

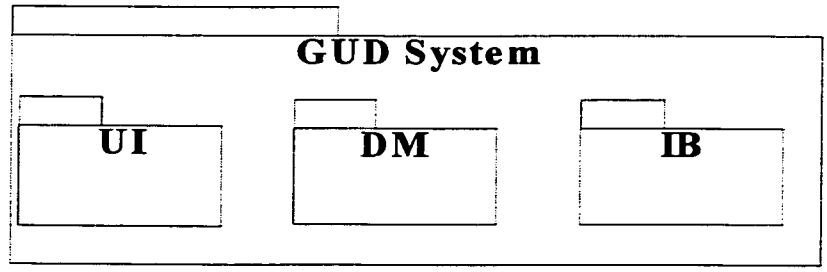

Figure 5.2 The GUD project package diagram

5.3.1 The Information Broker (broker) Package

There are several classes in the broker package working together to perform the

user request handling. (see figure 5.3)

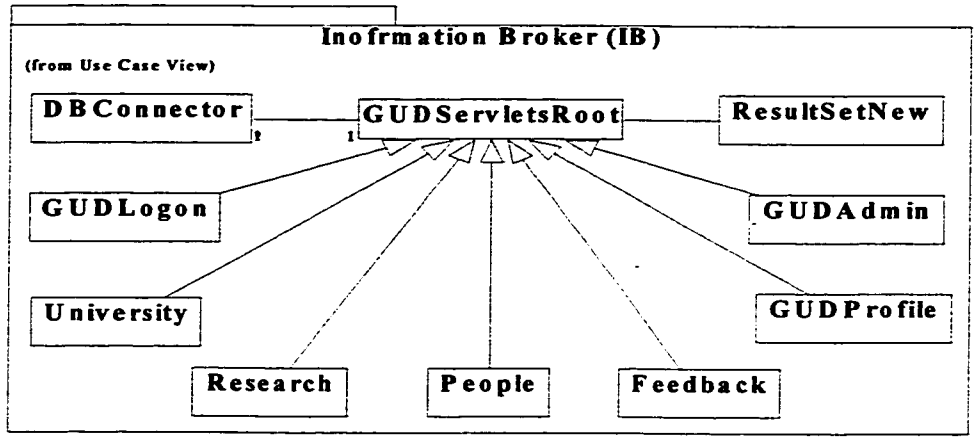

Figure 5.3 The GUD Information Broker Package

GUDServletsRoot:

GUDServletsRoot.class is the super class of all GUD servlets. It provides the common interface to generate html header, footer, and results body. The getFormData() method extracts the user request input and creates a hashtable to hold the name/value pairs. The service() method is the main method to process the request (through two interfaces HTTPServletRequest and HTTPServletResponse) inherited from the HTTPServlet class and will be overridden by its subclass.

GUDLogon:

GUDLogon.class is a derived servlet (from GUDServletsRoot) and is used to process the logon procedure.

GUDProfile:

GUDProfile.class is a derived servlets to create the user profile (registration

procedure).

GUDAdmin:

GUDAdmin.class is a derived servlet to manage the user access control.

DBConnector:

DBConnector.class is used to setup the connection with the database and close the connection after the process is complete. In the GUD project architecture, it acts as the

Database Connector component inside the DPA. We use MS SQL Server6.5 database server and therefore JDBC-ODBC Bridge is used. It is a singleton class, so there is only one instance of the class for the connection purpose. All users are sharing with the same connection for query. The connect() method initiates the connection to the database when the Web server starts or when the first query comes in, while close() closes the connection after the web server shuts down. GetConnectionHandle() returns a Connection object handler after the connection is setup.

ResultSetNew:

ResultSetNew.class is a higher level abstraction of the JDBC ResultSet object, We can easily wrap any ResultSet object in this class by simply creating one on the spot using the new operator.

University:

University.class is an information broker to handle the query request related to university, execute the query, and format the result sent back to the user.

Research

Research.class is an information broker to handle the query request related to research, execute the query, and format the result sent back to the user.

People:

People.class is an information broker to handle the query request related to university faculty member, execute the query, and format the result sent back to the user.

5.3.2 The Data Mediator (mediator) Package

There are ten classes in the mediator package to function the data parse, manipulating and database populating/updating.

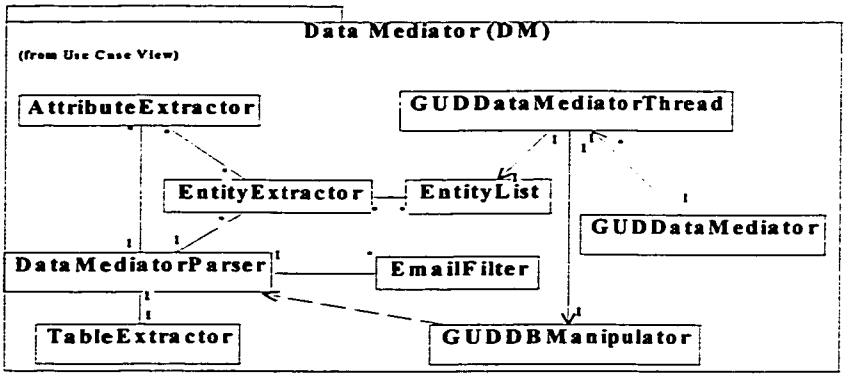

Figure 5.4 The GUD Data Mediator Broker Package

GUDDataMediator:

GUDDataMediator.class is a daemon server that keeps listening to the connection requests from the clients (i.e. Web Mining Agents). It will spawn a thread to handle every

WMA until the maxim thread number (10 in this case) is reached.

GUDDataMediatorThread

GUDDataMediatorThread.class a thread to handle the data piped from the WMAs. When the connection is setup, the data will be piped from a WMA to the DPA through the dedicated socket. The constructor includes three parameters: socket object is tunnel between a WMA and the DPA, TableExtractor object stores the database table information (metadata), and a boolean to turn on/off an applet (GUDThreadApplet) to display the data transmission process.

GUDDBManipulator

GUDDBManipulator.class is used to execute the formatted SQL command and implement the database populating/updating. The start() method creates a *DataMediatorParser* object and initiates the data manipulation (entities and attributes extraction, table creation, SQL command generation). The executeSQL() executes the sql command inside the GUD database and commits the transaction. This method is synchronized for the sake of the concurrency control.

DataMediatorParser

DataMediatorParser.class is used to parse the data (entities, attributes) from the WMAs.

The parseEntity() calls the *EntityExtractor* object and extracts the entities into an entity array, while the parseAttribute() calls the *AttributeExtractor* object and extracts the attributes of a given entity into an attribute array. The processTableCreation() creates a

table upon a entity name if the table does not exist in the GUD database. The genePopuplateOrUpdateSQL() generates the SQL command string to perform the database inserting or updating. If the checkExist() returns false, the record does not exist, and insert command will be executed. If it return true, the record exists, the updating is to be performed.

TableExtractor

TableExtractor.class extracts and stores the static structure of a database (table list).

EntityExtractor

EntityExtractor.class is used to manipulate information of an entity. This class was initially coded by Hicham Ouahid, and was later modified by myself. In the class, The getEntityName() returns the entity name. The addAttribute(AttributeExtractor) add an AttributeExtractor object (which contains the attribute information) into the entity vector. The getAttributeNumber() returns the number of attribute in an entity.

AttributeExtractor

AttributeExtractor.class is used to extract information of the attributes of the entity (e.g. attribute name, attribute value, and attribute type). This class was initially coded by Hicham Ouahid, and was later modified by myself. In the class, the attribute name could be university name, staff email, and so on. The attribute value likes "university of Ottawa", "Karmouch@site.uottawa.ca", etc. There are three attribute types: Primary key, foreign key, and regular.

EntityList

EntityList.class is used to formulate an entity vector. This class was coded by Hicham Ouahid, and used in the DM.

EmailFilter

EmailFilter.class is a utility tool to filter out the wrong formatted email data.

5.3.3 The User Interface (applet) Package

The UI package contains several applet to function as banners, menus, etc. and being used in the front-end user interface to ease the browsing, navigating and querying.

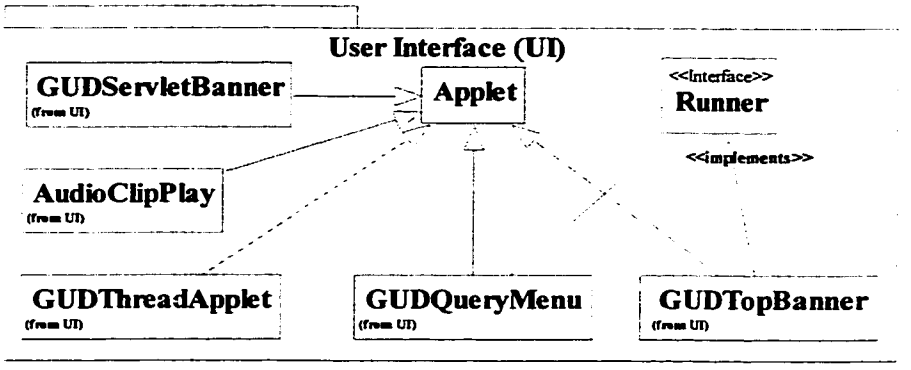

Figure 5.5 The GUD User Interface Package

GUDServletBanner

GUDServletBanner.class draws a text string (in our application, the string is "the results from the GUD servlets") on the screen by "fanning out" from a central point and fades in text color as it moves.

AudioClipPlay

AudioClipPlay.class is an applet playing background music when the user

browsing the system. The music could be stopped, resumed, and looped.

GUDTopBanner

GUDTopBanner.class is an applet fading on/off at the GUD front page. The applet is loaded only once into the top frame of the GUD frameset. All other interactions are processed in other frames (menu frame or main frame). It significantly reduces the load time, at the same time, makes the pages animated and attractive.

GUDQueryMenu

GUDQueryMenu.class is an applet being displayed as a pup-down menu in the GUD search page for easy navigation

5.4 Implementation Samples

When accessing the GUD welcome site, the user is asked to logon. If he/she is not the authenticated user, the user is asked to fill out the registration form before use the system. (The user can also access the system using a *guest* account, but can not make the query) After the user successfully logins the system, the JavaScript program embedded in the HTML file automatically test the user's system – Browser, JavaScript version and operating system – and decide how to process.

5.4.1 The Welcome Page

The welcome page is the entrance of the GUD project. The new user needs to register before use the system, and all the users can logon as a "guest" to surf the system. Besides the system level access control, we developed the application access control

plan. By default, all the new registered users have the lowest access privilege. The user can only modify his/her profile, like changes of email account, but not the access privilege. The system admin however can modify (grant or revoke) the user access privilege.

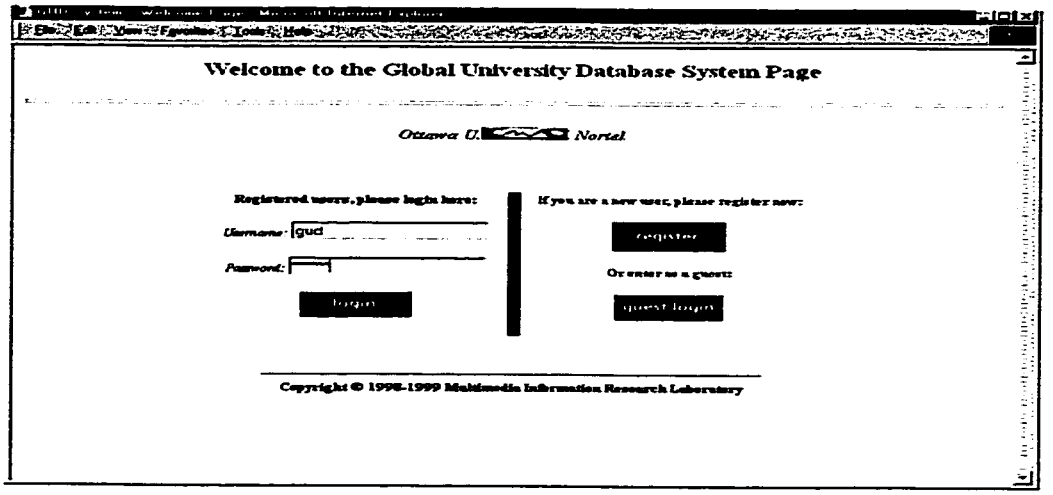

Figure 5.6 The GUD Welcome Page

5.4.2 The registration page

The first time user needs to register before fully use the system. The user is asked to create his/her own profile by choosing username/password pair and inputting his/her full name and email address. The form validation is functioned by the underground JavaScript program. The username is unique as primary key. The user access control is by default the lowest level and cannot be modified by the end user, but by the system administrator.

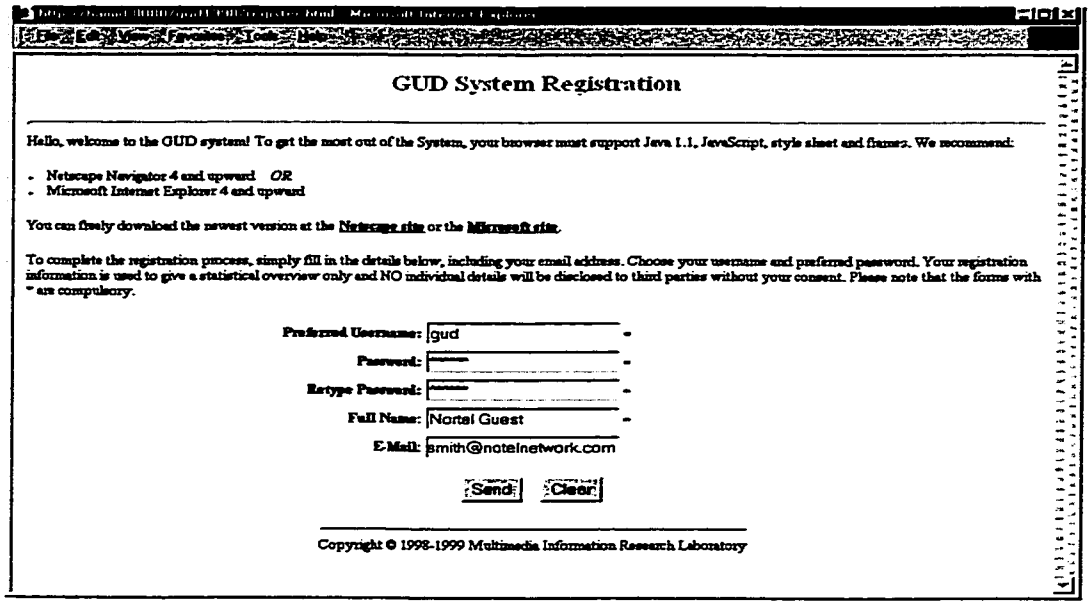

Figure 5.7 The GUD Registration Page

5.4.3 The User Interface Layout

The layout of the GUD project front page consists of three parts: banner, menu, and main. The banner frame shows the logo and link of the project group and funded company; the menu frame lists the selections that the user can use with the system.

Basically, all the clickable links are mapped to the main frame. The banner and menu frames are initially loaded only once and used over and over until the user exit the system. We use the frameset's feature to reduce the download time. Also, we deploy the Image map and CSS in HTML 4 to improve the performance

The main component in the menu frame is of course "search", from which the user makes the query and gets the results back. According to our GUD database schema, we provide a series of requested forms that the user can type in or select on. These forms

consist of entities like university name from University Table, E-mail address from Staff table, etc.

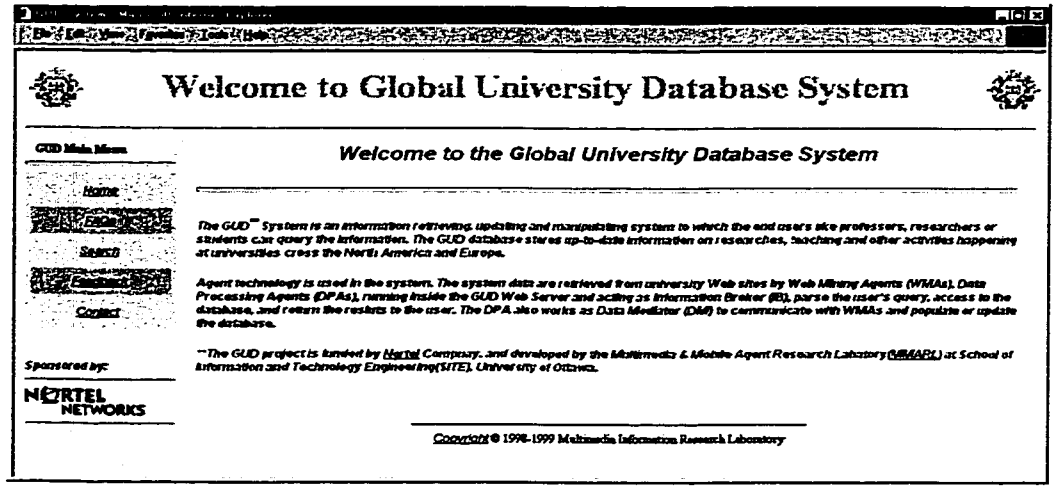

Figure 5.8 The GUD User Interface Layout

Here we should be careful since we face different users, as mentioned earlier. The Novice or the first time user may need easy-to-follow, simple search command and corresponding online help. By contrast, the expert user does appreciate quick search return, handy shortcuts. He or she may want to customize the display, format the data according to his/her preference, and set the data return preference (e.g. top 10 or all found results). Therefore, we have to bear in mind these factors. Even for the time beings, we focus on the key functionality. After this stage, we will add more choices to fit different users' flavors.

5.4.4 The FAQ, feedback, and search tip pages:

The user interface provides some "helper" pages for the user to learn and use the GUD system. The faq page lists the frequent asked questions about the GUD. (Figure

5.9)

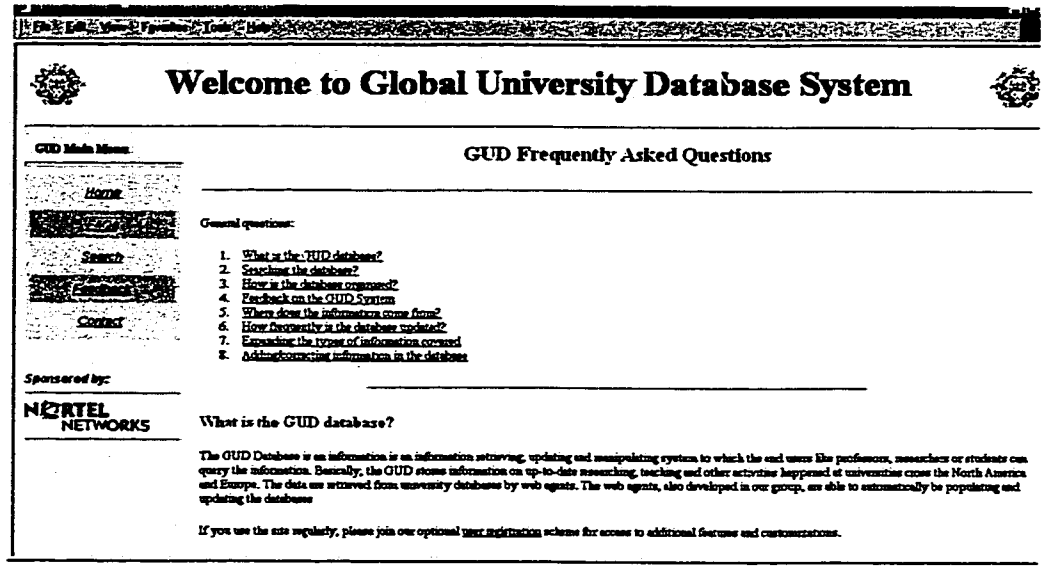

Figure 5.9 The GUD FAQ Page

The feedback page takes the comments and suggestions from the user (figure 5.10).

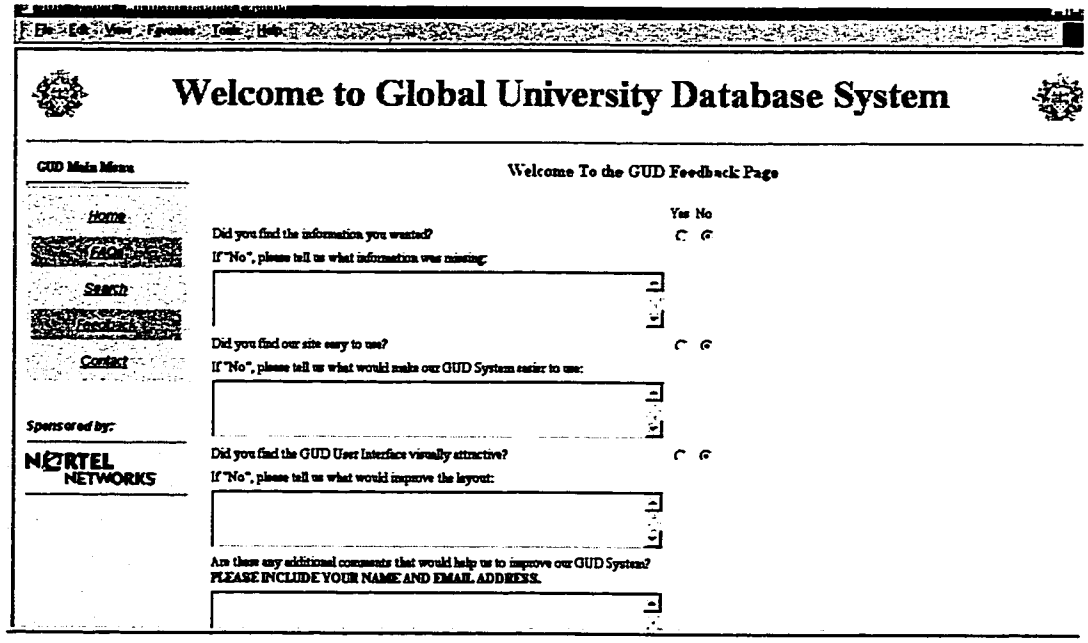

Figure 5.10 The GUD Feedback Page

The search tip page give away some tricks when making the query.

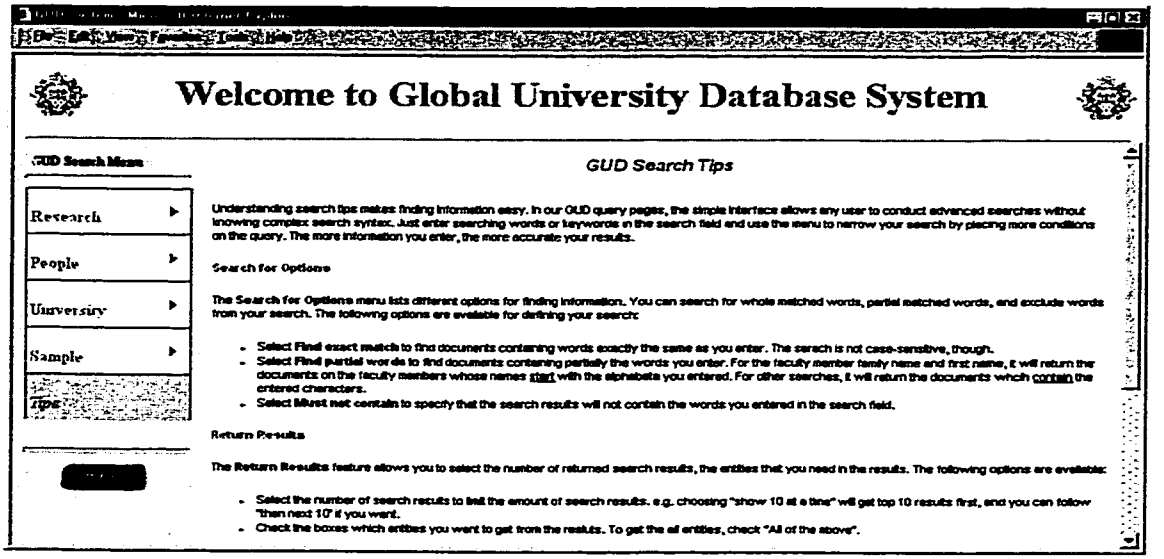

Figure 5.11 The GUD Search Tips Page

5.4.5 The GUD Query Sample pages:

The query is handled by the Information Broker (IB). IB parses the request string, connects the database via Database Connector (DC). The results set is passed to the JSP pages through a JavaBean. The JSP page dynamically generates the result into an html file and sends the file back to the user.

The snapshots below demonstrate several query processes on the GUD database. In each query, the user navigates to the intended search page, inputs the query selection criteria, and selects the return options.

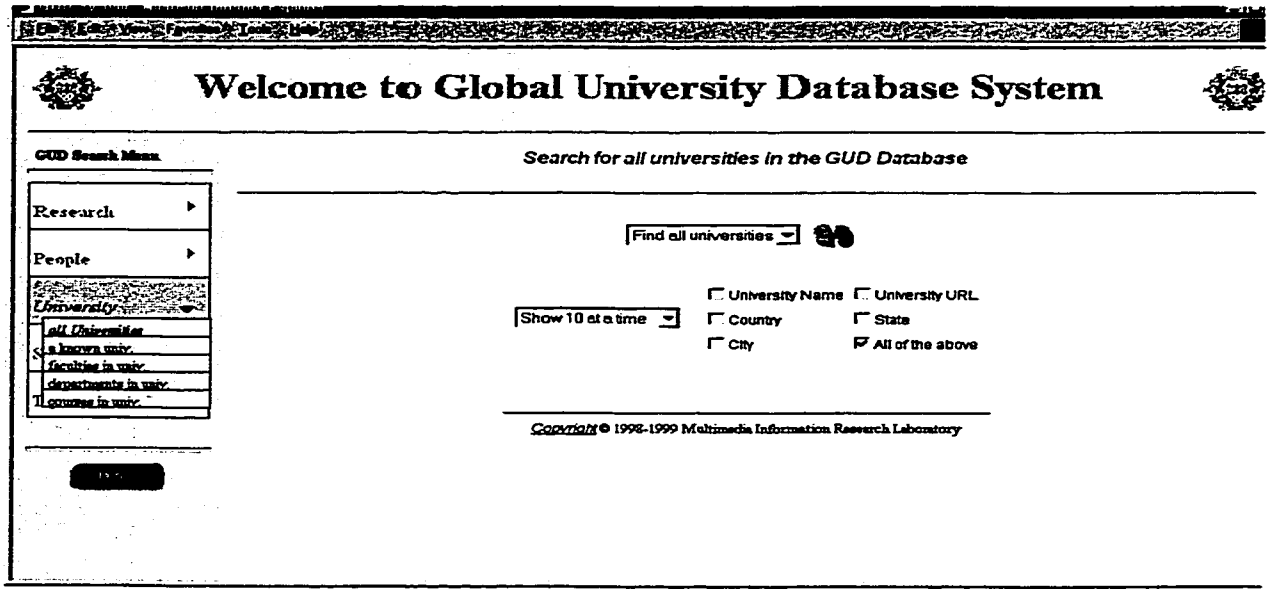

Figure 5.12 Search Sample 1A: Search All Universities

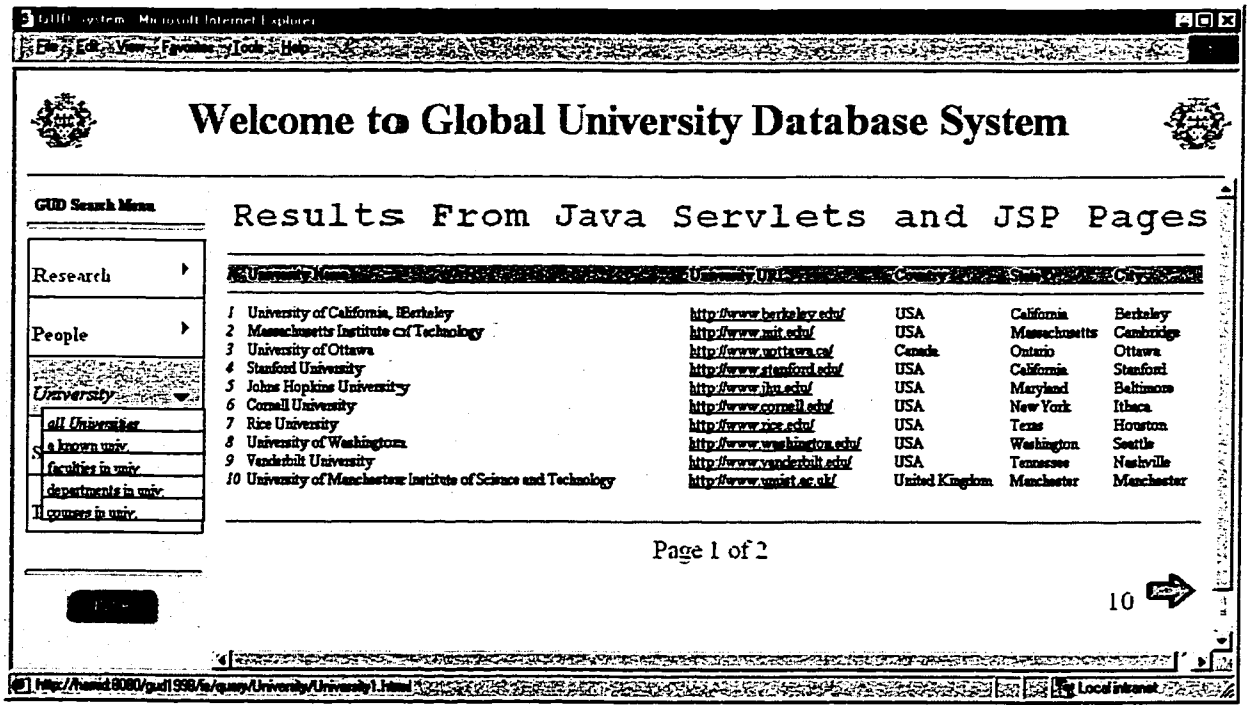

Figure 5.13 Search Sample 1B: All Universities Results

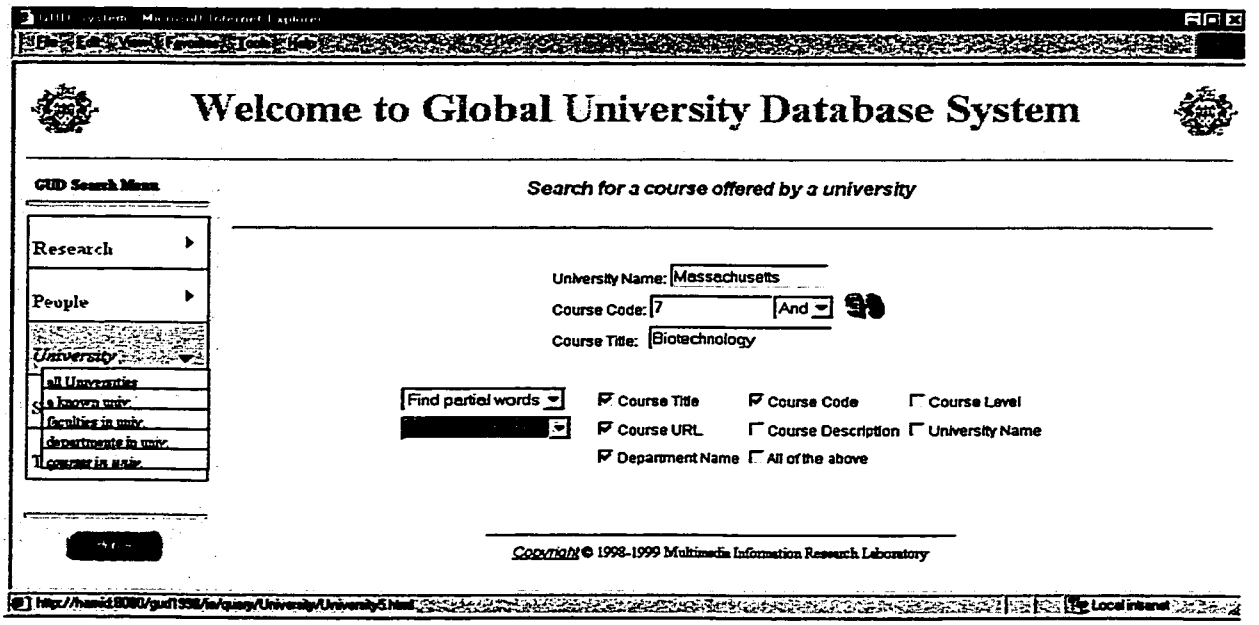

Figure 5.14 Search Sample 2A: Courses Offered by a Given University

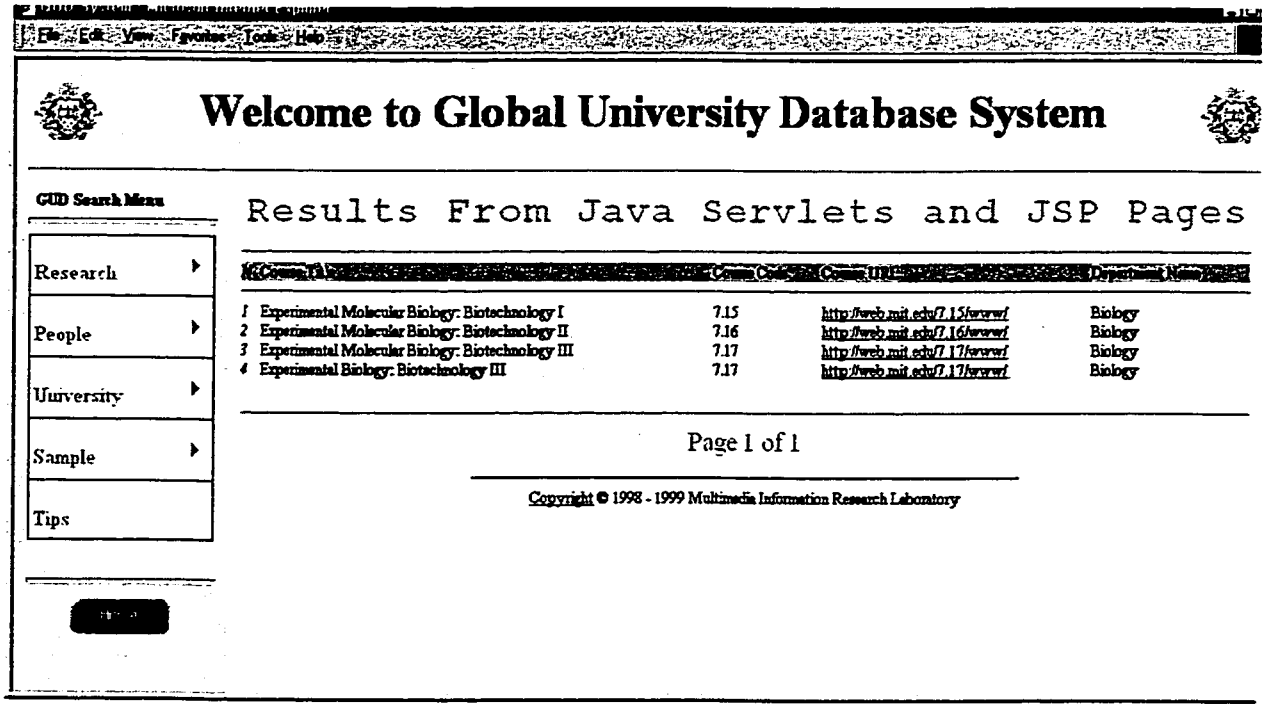

Figure 5.15 Search Sample 2B: Courses Query Results

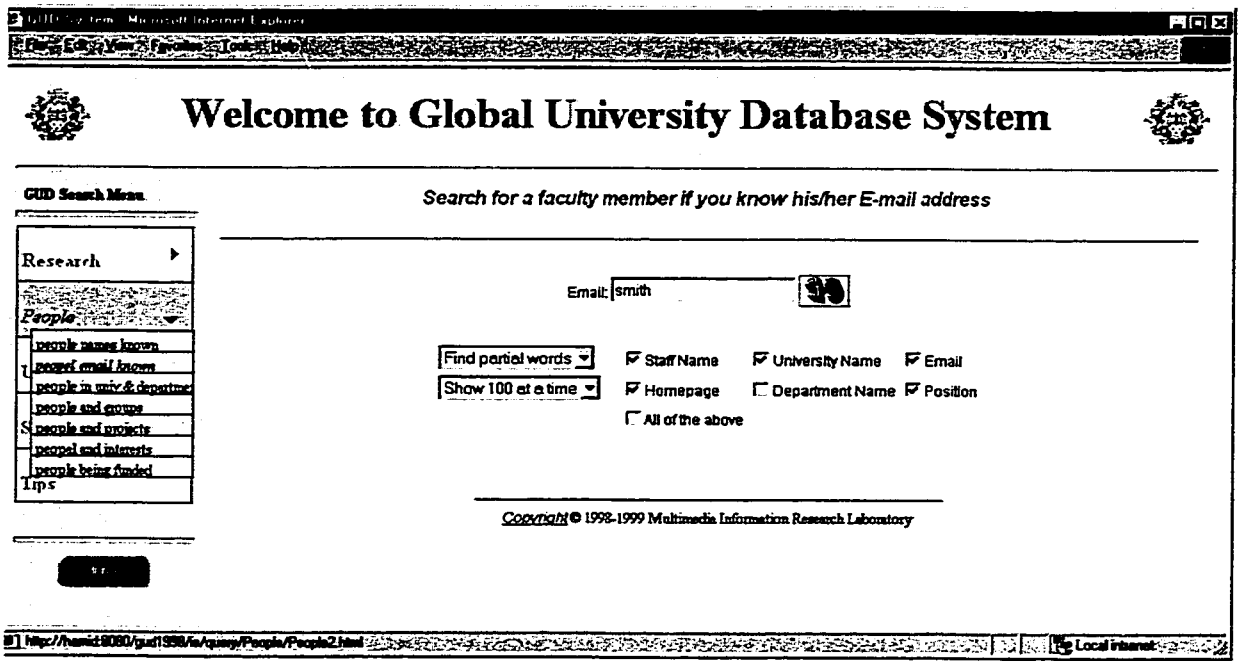

Figure 5.16 Search Sample 3A: Known Email Search for Researcher-Related Info

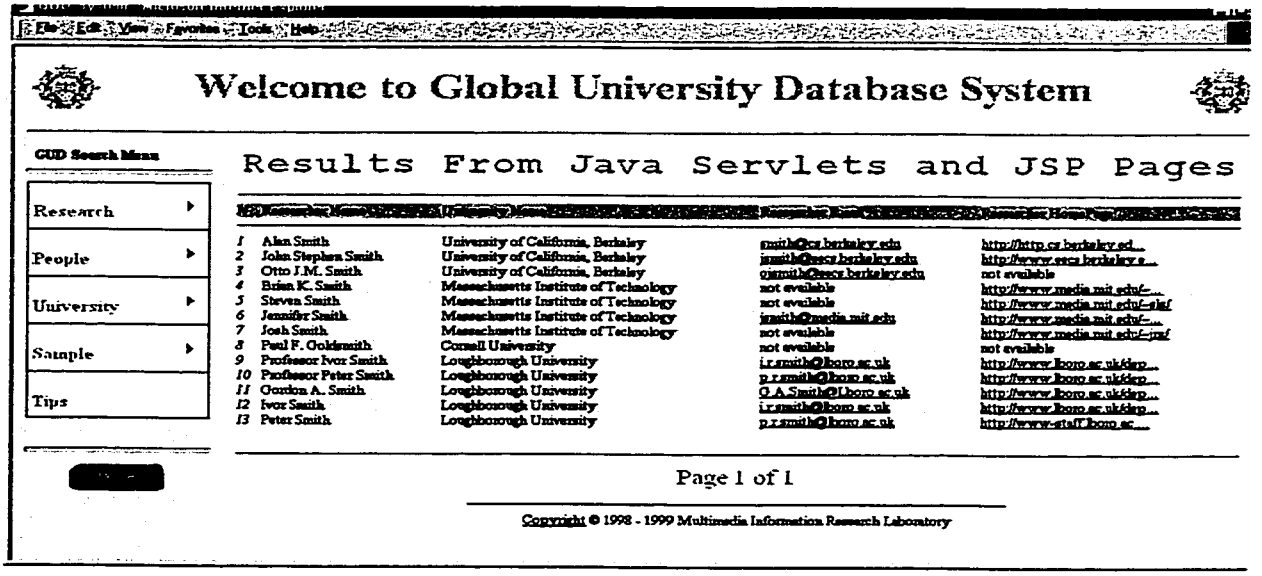

Figure 5.17 Search Sample 3B: Email Query Results

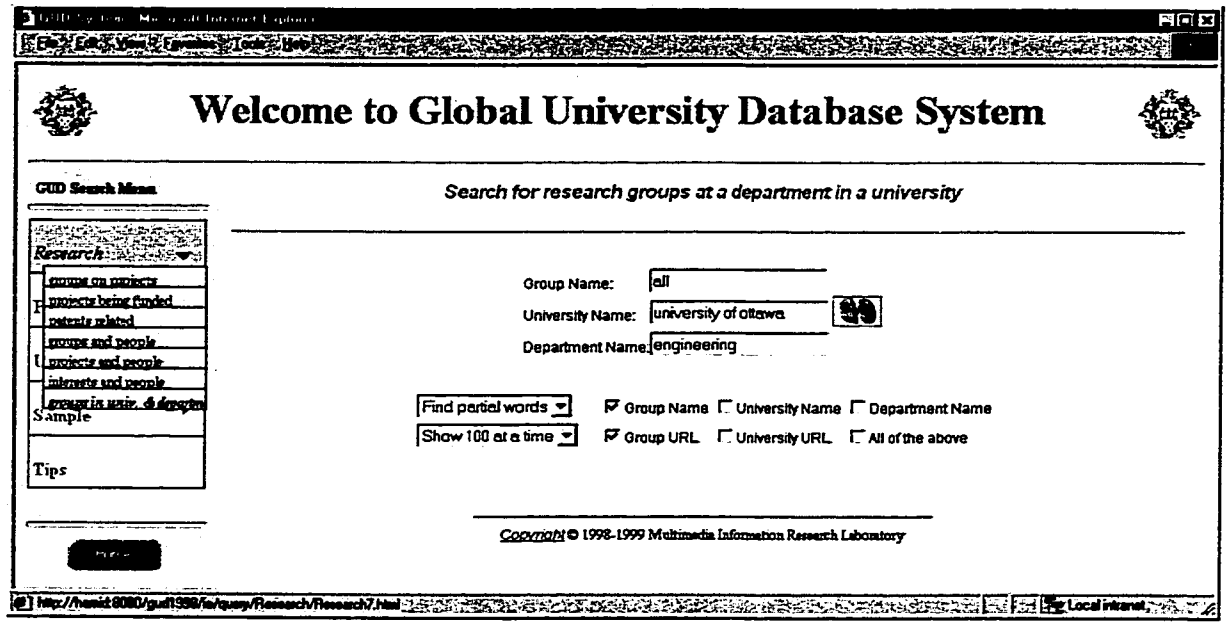

Figure 5.18 Search Sample 4A: Research Group at a Given University Query

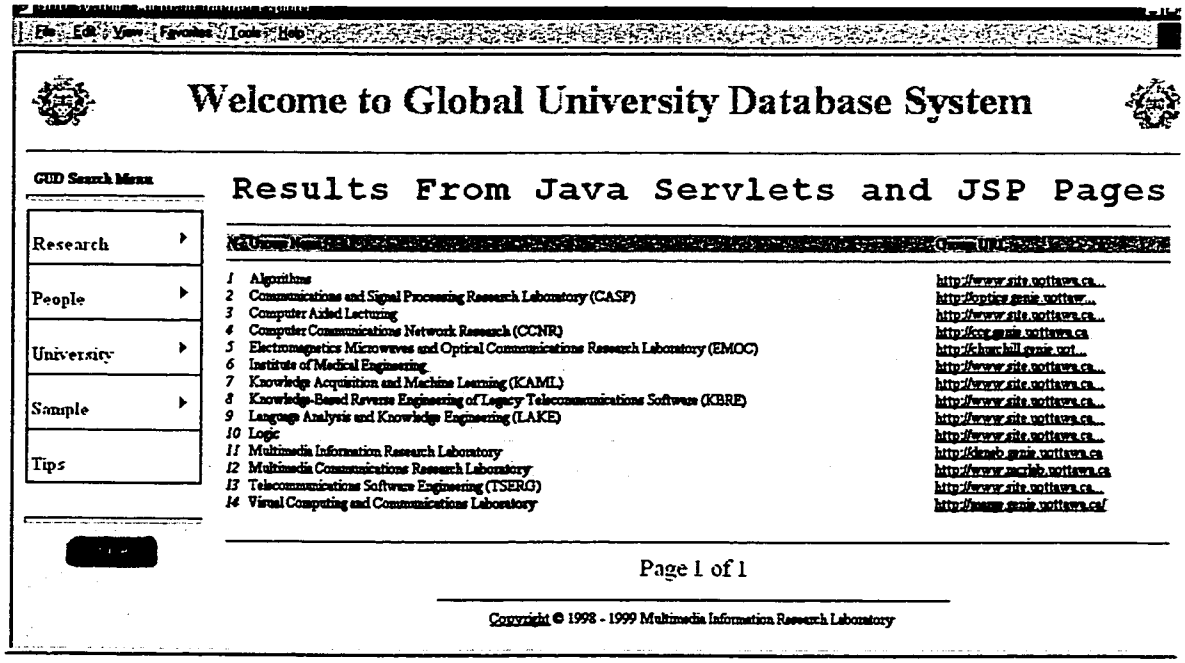

Figure 5.19 Search Sample 4B: Groups at a Given University Query Results

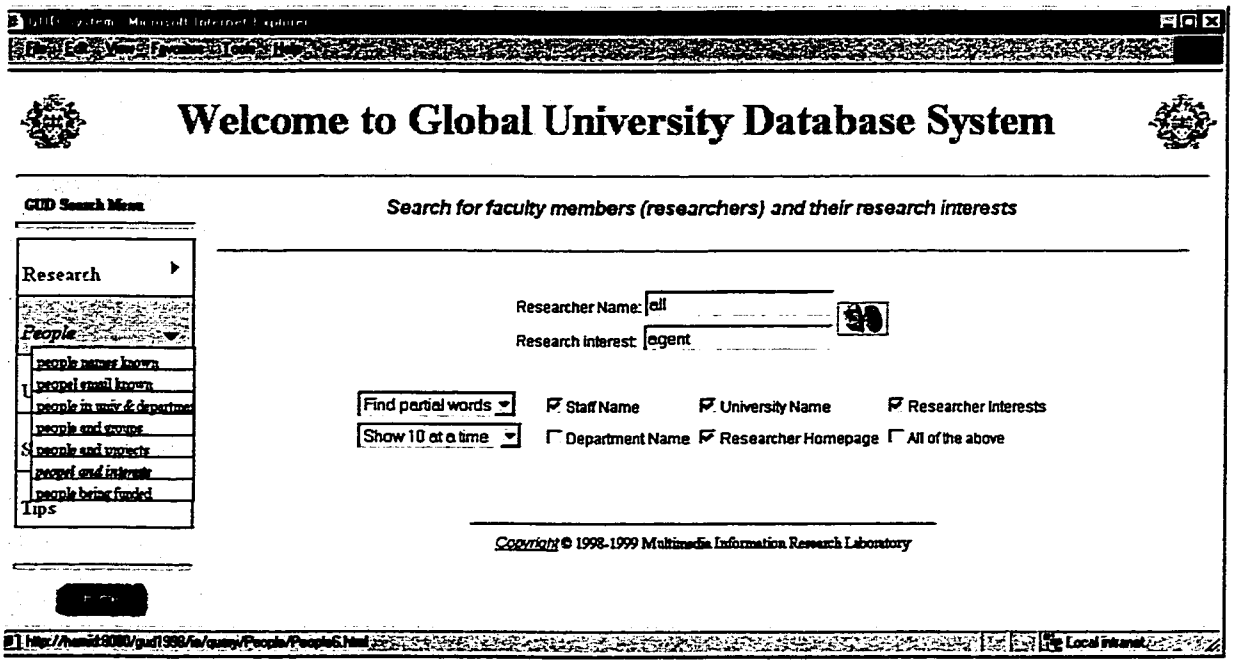

Figure 5.20 Search Sample 5A: Researcher's Interest Query

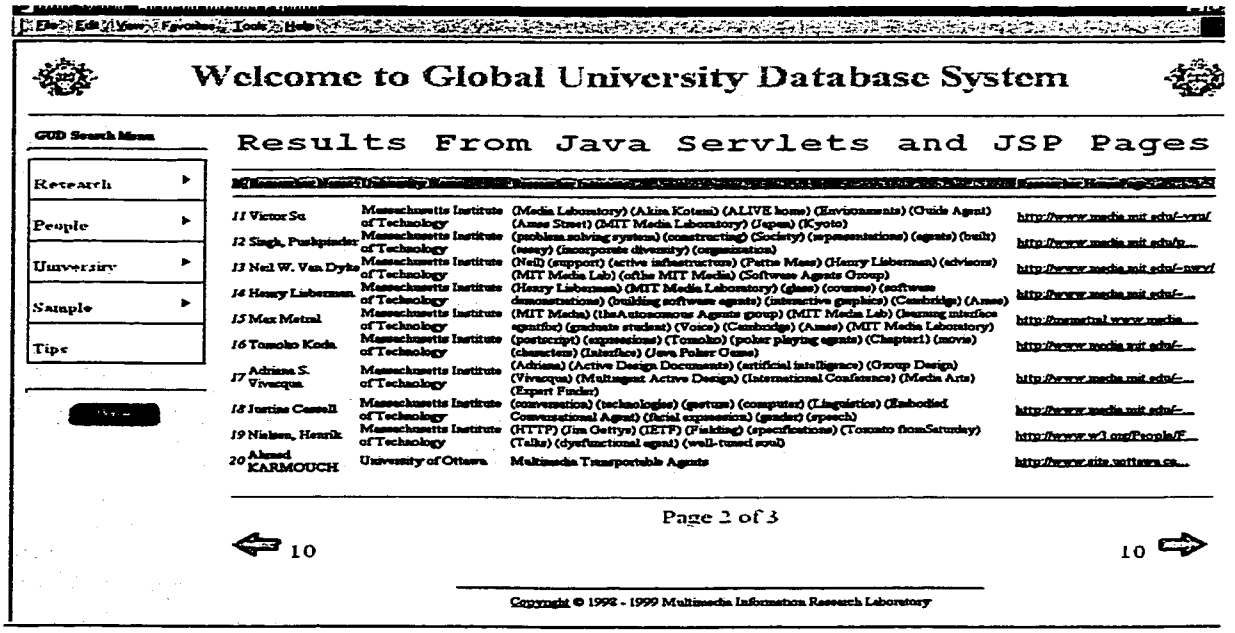

Figure 5.21 Search Sample 5B: Interest Query Results

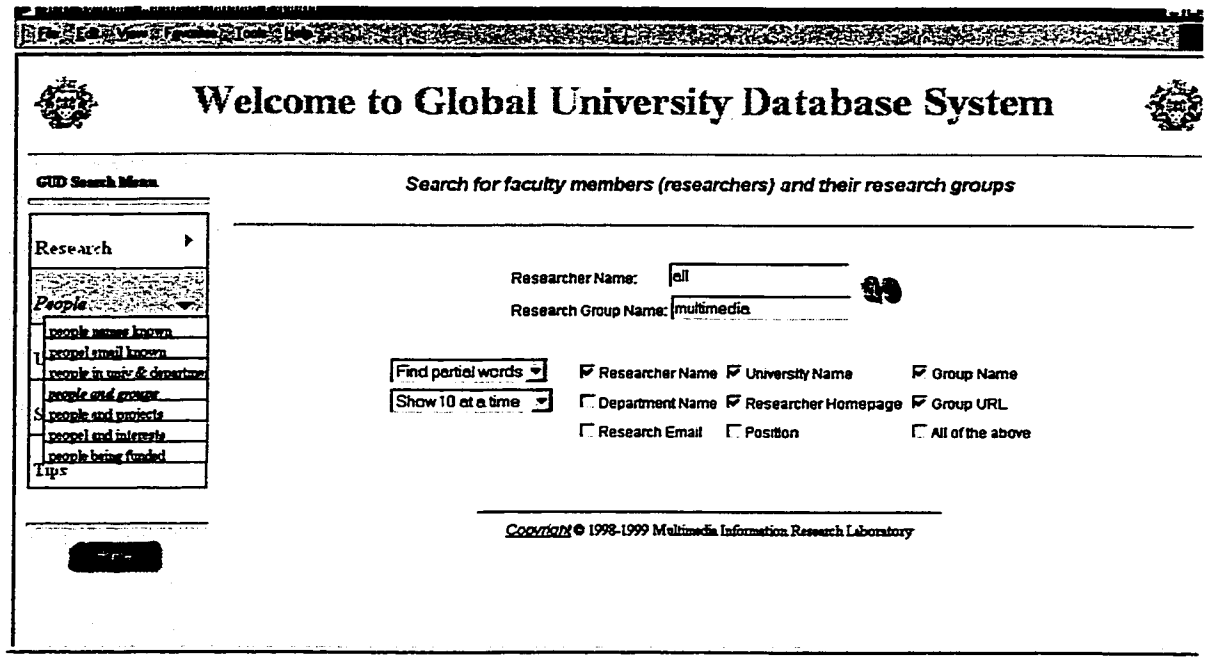

Figure 5.22 Search Sample 6A: Research Group and Researcher Query

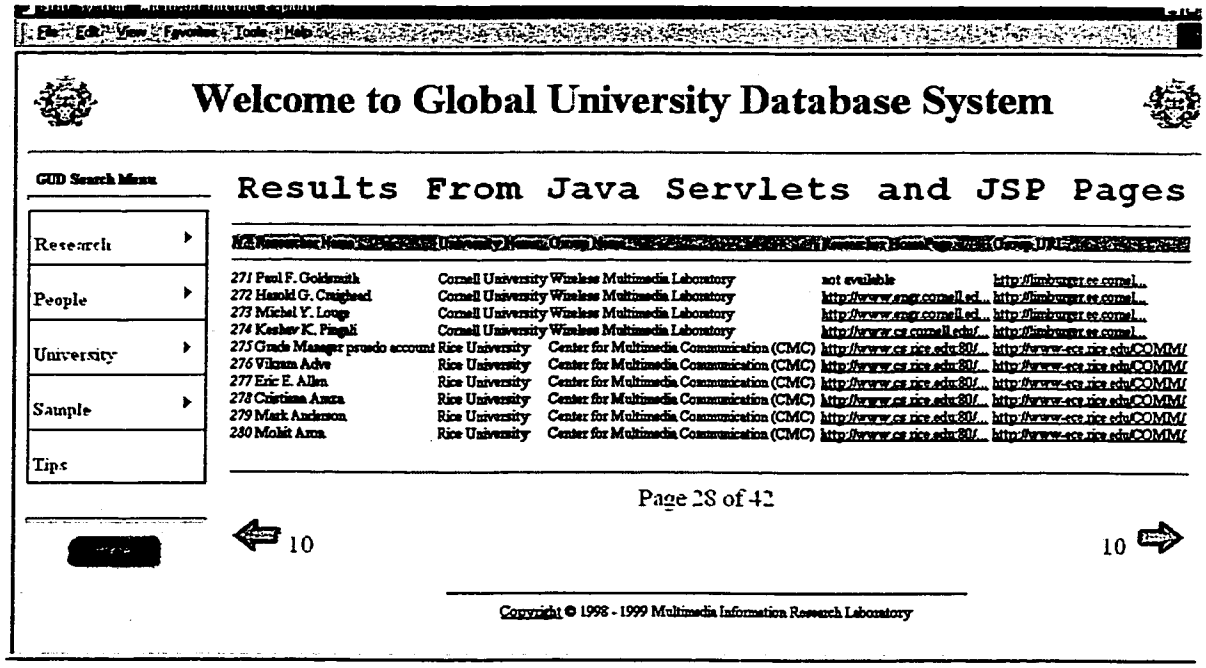

Figure 5.23 Search Sample 6B: Group and Researcher Query Results

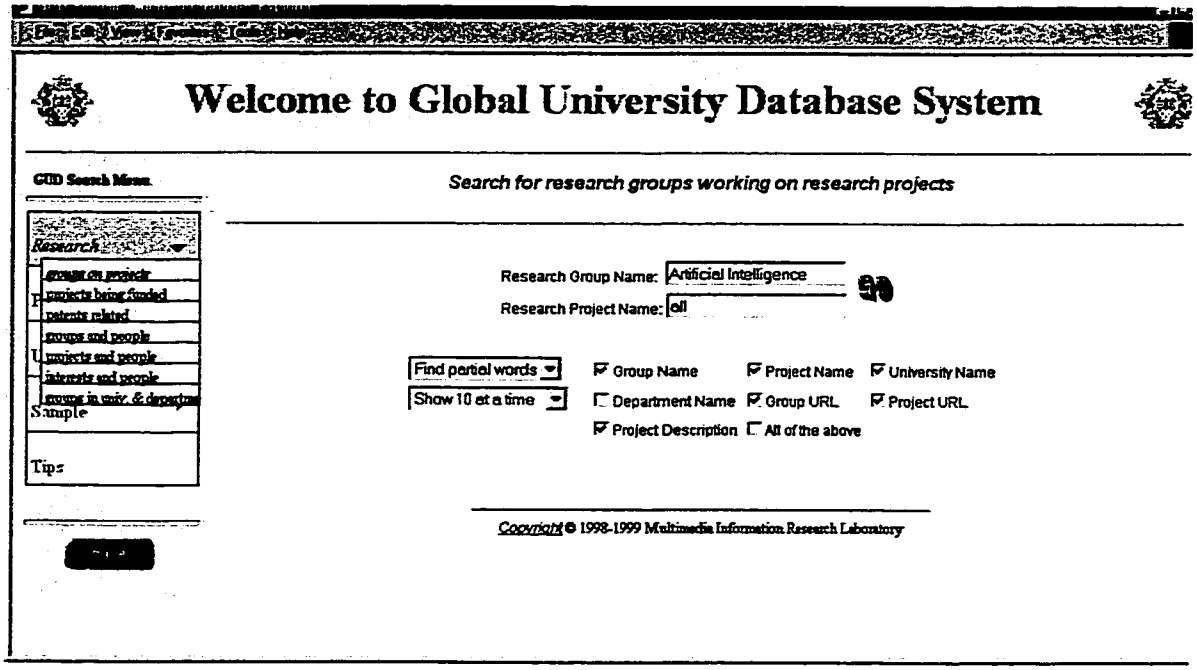

Figure 5.24 Search Sample 7A: Research Project in Group

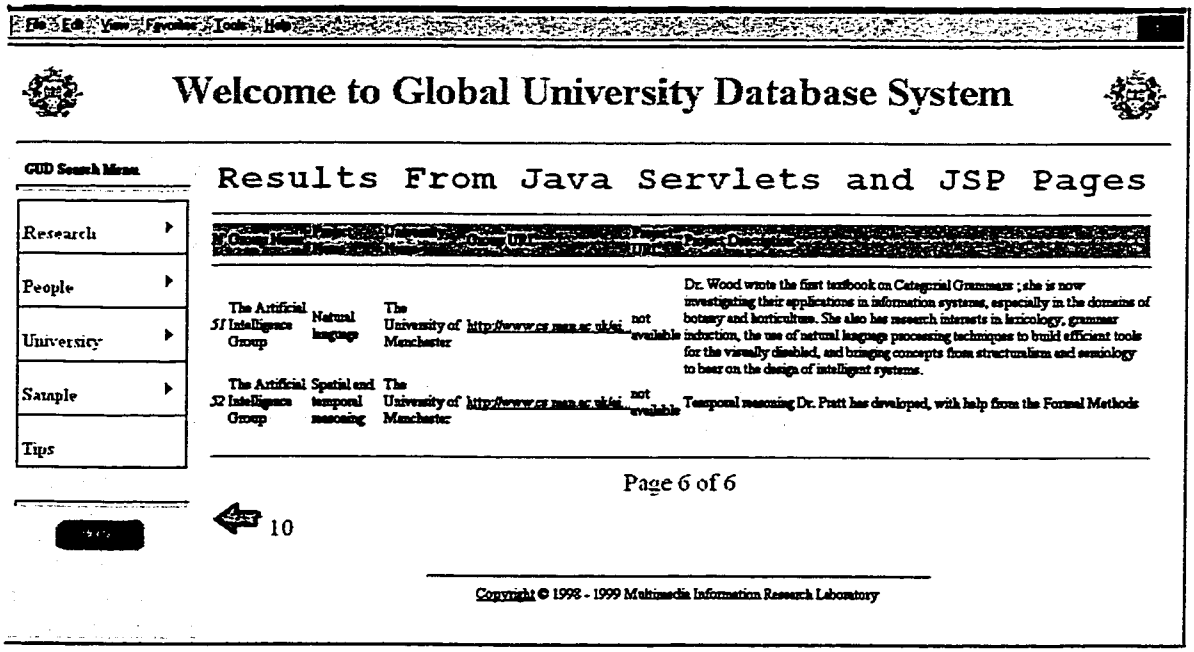

Figure 5.25 Search Sample 7B: Project in Group Query Results

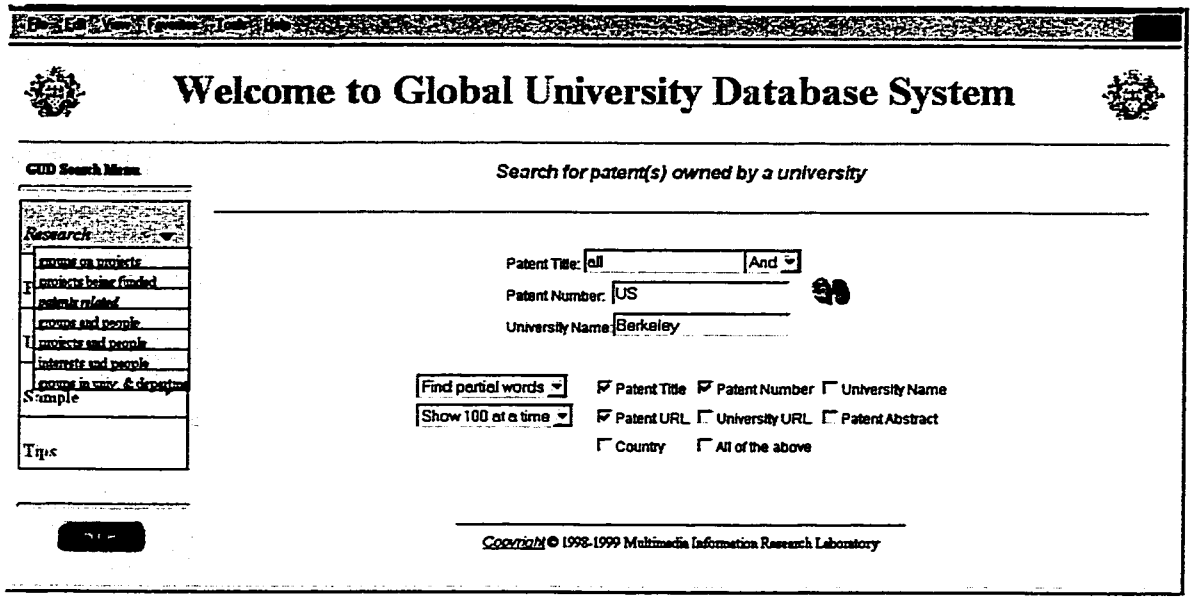

Figure 5.26 Search Sample 8A: Patents Query

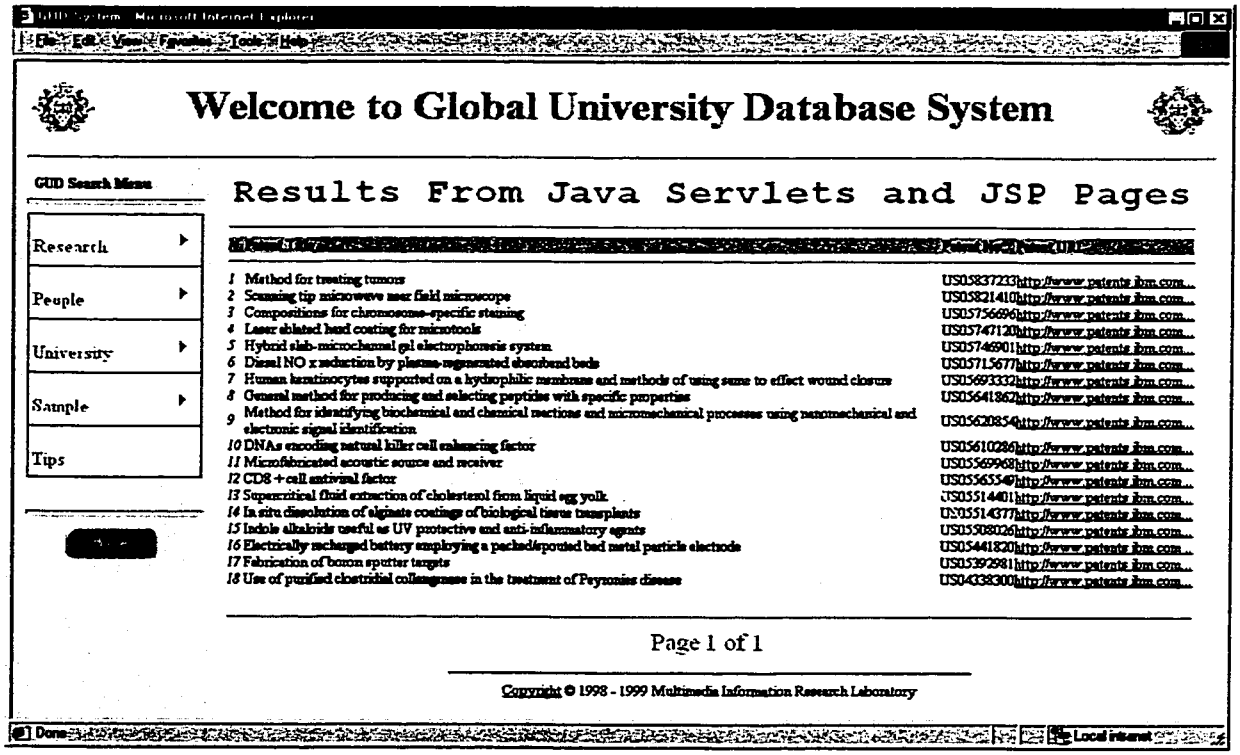

Figure 5.27 Search Sample 8B: Patents Query Results

5.5 Summary

In this chapter, we have reported the implementation details of our research. Our task includes the implementation of GUD user interface and Web server extension. In the user interface part, we use standard html with the addition of JavaScript, which is used to perform the version checking and form validation. In the Web server extension (DPA) part, Java Servlet is deployed, instead of CGI, to handle the request from the end user and return the formatted result dynamically. JDBC is used to connect the database. At the later development stage, we add the JSP pages to handle the presentation logic of the project.

Our GUD group integrated with everyone's work successfully, did the testing, and released the first version of the GUD project in January 1999. The system contained data of 13 universities from Canada, USA, and Europe. There was academic information on faculties and departments, courses, faculty members and their interests, research projects, research groups, and patents, etc. As the data were stored in the local database, the search process is much simple and efficient.

Chapter 6

Conclusion

The GUD project incorporates thin client architecture for reducing the complexity of business critical systems and ensuring low-cost deployment, less configuration, less maintenance, and a robust application environment. The GUD user interface provides a friendly, efficient browsing and querying environment for the end user. The GUD Data Processing Agent (DPA) allows for application logic and data access to be aggregated in a central application server. The application server becomes a one-to-many service provider of business objects and business processes. Therefore, business logic can be reused among multiple applications, and application logic need.

The GUD project allows the user to create scalable Web Server applications with DPA, which enables the user to use his/her existing client/server knowledge to create, manage, and deliver data over the Web. The applications are running from the GUD Web Server when each time a user accesses to the system. This means that the application takes up no disk space on the client machine. Because the DPA is platform-neutral, it can operate and fully support Internet strategies from different vendors, thereby reducing the risk of platform selection.

The GUD project is a comprehensive application development platform created to harness the benefit of the Internet and Intranets for core business applications. In our project, upon the request of the sponsor, the system provides university-related

information. Nevertheless, the GUD framework can be applied to any Internet-based application, like financial transaction, online shopping, and library searching, among others.

Our GUD team successfully integrated the individual work and released the first version in January 1999. The first version had 13 sample universities from Canada, USA, and Europe. The project was also presented at the CITO'99 workshop show in Ottawa, April 1999, and drawn lots of attentions from the participants. However, the application is by no means complete. There are a number of enhancements that can be made to it, which are proposed below.

As we implemented in the GUD project, the user interface pages were pre-defined. Therefore, the SQL query was static and predefined as well. It fits the application well, as we have already defined the database schema, and the way to pre-define the pages can efficiently guide the user (especially the novice user) through the query procedure. However, to make the system scalable and robust, the intelligent user interface (i.e. the pages generated automatically upon the data structure) should be built and tested. In fact, we have done some research on database creating (by the DPA) which makes the system database schema independent. The future work should step further to develop the intelligent user interface.

In our GUD system, the DPA interfaces with the WMA (implemented by Java and XML) and extracts the piped data before create the database and populate/update the database. Some information (e.g. the data hierarchical relation) is lost during the extraction process. Instead of manipulating the database upon the extraction of the piped

data, the better way is to make use of the XML's DTD files, which contains the complete application data structure. The same argument could be applied to the creation of the intelligent user interface. The DPA could be a generic Web server extension. The Information Broker (IB) inside dynamically generate the intelligent interface according to the specific DTD files, while the Data Mediator (DM), on the other hand, creates the database based on the same DTD files. In this scenario, any application could be easily developed by simply defining the application data model and DTD. There is little change needed to be made in the generic DPA, which make the system development simple, robust, and cost-efficient.

In our GUD project, all the data, retrieved by the WMA from the university Web sites, are text-based. One direction of the future work could be on the multimedia application area. Even it's possible to simply retrieve the hyperlink of the multimedia files (image, audio, video, etc), and let the user accesses and loads the multimedia files directly from the source sites, it could be useful, for the sake of reliability and efficiency, to retrieve the multimedia files into the local database. There are some topics on how to retrieve the multimedia files, how to store them, how to transform the files to the end user, etc. It's definitely an interested research domain and worthy of being exploited.

References

[Ambler98] Ambler, S.W., "Building Object Applications That Work: Your Step-By-Step Handbook for Developing Robust Systems with Object Technology", New York: SIGS Books, 1998

[Bowman94] Bowman, M., Danzig, P., Manber, U., and Schwartz, F. "Scalable Internet discovery: research problems and approaches", C. ACM, 37(8), page 98-107, 1994

[Buxton80] Buxton, W. and Sniderman, R. "Iteration in the Design of the Human-Computer Interface", In Proc. Pf the 13th Annual Meeting of the Human Factors Assoc. of Canada, 72-80, 1980

[Casterfranchi95] Casterfranchi, C., "Guarantees for autonomy in cognitive agent architecture", In Woolridge, M. and Jennings, N. R., ed., *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, page 56-70. Springer-Verlag: Heidelberg, Germany, 1995

[Childovskii97] Childovskii, B., Borghoff, U., and Chevalier, P. "Towards Sophisticated Wrapping of Web-based Information Repositories", In Proc. Conf. Computer-Assisted Information Retrival, p123-135, 1997

[Constantine95] Constantine, L. L., "Constantine on Peopleware", Englewood Cliffs, NJ: Yourdon Press, 1995

[Cooper95] Cooper, Alan, "About Face --The Essentials of User Interface Design", IDB Books, 1995

[Doorenbos97] Doorenbos, R., Etzioni, O., and Weld, D. "A saclable comparison-shopping agent for the World-Wide Web", In Proc. Autonomous Agents, p39-48, 1997

[Edwards98] Edwards, Jeri, "3-Tier Client/Server", Wiley Publication, 1998

[Friedman97] Friedman, M. and Weld, D. "Efficiently executing information gathering plans", In Proc. 15th Int. Joint Conf. AI, p. 785-791, 1997

[Gould85] Gould, J. and Lewis, C. "Designing for Usability - Key Principle and What Designer Think", Communications of the ACM 28(3):300-311, March, 1985

[HTML99] W3C, "HyperText Markup Language Home Page", www.w3.org/MarkUp/

[IBM93] IBM, "Systems Application Architecture – Common User Access Guide to User Interface Design", IBM Corporation, 1993

[Java99] Sun, "The Source for Java Technology", www.javasoft.com

[Jenkins97] Jenkins, Neil, "Client/Server Unleashed", Sams Publication, 1997

[Jones89] Jones, M. K., "Human-computer interaction: A design guide", Englewood Cliffs, NJ: Educational Technology Publications, 1989

[Krulwich97] Krulwich, Bruce, "Automating the Internet: Agents as User Surrogates" IEEE Internet Computering, Vol. 1, No. 4, 7-8 1997

[Laurel91] Laurel, B. (Ed.), "The art of human-computer interface design", Menlo Park, CA: Addison Wesley, 1991

[Live99] Live Software, "JRun and Servlets", www.livesoftware.com

[Martin91] Martin, James, "Rapid Application Development", MacMillan Publishing Company, 1991

[Mayhew92] Mayhew, D.J., "Principles and Guidelines in Software User Interface Design." Englewood Cliffs NJ: Prentice Hall, 1992

[McConnell96] McConnell, S., "Rapid Development: Taming Wild Software Schedules", Redmond, WA: Microsoft Press, 1996

[Microsoft95] Microsoft, "The Windows Interface Guidelines for Software Design", Redmond, WA: Microsoft Press, 1995

[Montebello98] Montebello, M. "Information Overload - An IR problem?", Proceedings of the String Processing and Information Retrieval: A South American Symposium, 1998

[NCSA99] NCSA, "The Common Gateway Interface RFC Project", NCSA, 1999

[Netscape99] Netscape, "JavaScript Developer Center", http://developer.netscape.com/tech/javascript/index.html

[Papakonstantinou95] Papakonstantinou, Y., Garcia-Monlina, H., and Widom, J. "Object exchange across heterogeneous information sources" In Proc. 11th Int. Conf. Data Engineering, p251-260, 1995

[Rational99] Rational, "Unified Modeling Language (UML)", www.rational.com

[Roth97] Roth, M. and Schwartz, P. "Don't scrap it, wrap it! A wrapper architecture for legacy data sources", In Proc. 22nd VLDB Conf., p266-275, 1997

[Rudd94] Rudd, J. and Isensee, S. "Twenty-two Tips for a Happier, Healthier Prototype", ACM Interaction 1(1):35-41, January, 1994

[Shneiderman98] Shneiderman, Ben, "Designing the User Interface", 3rd ed., Addison Wesley, 1998.

[Sridharan97] Sridharan, Prashant, "Advanced Java Networking", Prentice Hall, PTR, 1997.

[Stallings97] Stallings, William, "Data Communications", 5th Edition, Prentice Hall, 1997.

[Wang99] Wang, Yaoping, Pham, Vu and Karmouch, Ahmed, "The issues of design a Global University Database System", In proceeding of CCECE99, Edmonton, 1999

[Wasserman90] Wasserman, A.I., and Shewmake, D.T., "The role of prototypes in the user software engineering (USE) methodology", In: J. Preece & L. Keller (Eds.), Human Computer Interaction. Hemel Hempstead: Prentice Hall in association with the Open University, 385-401, 1990

[Weinschenk95] Weinschenk, S. and Yeo, S., "Guidelines for Enterprise-wide

GUI design", John Wiley & Sons, Inc., 1995

[XML99] W3C, " Extensible Markup Language ", www.w3.org/xml

Appendix A.

GUD Class Hierarchy

- class java.lang.Object
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container
      - class java.awt.Panel
        - class java.applet.Applet
          - class com.gud.applets.GUDQueryMenu
          - class com.gud.applets.GUDServletsBanner (implements java.lang.Runnable)
          - class com.gud.applets.GUDTopBanner (implements java.lang.Runnable)
  - class com.gud.servlets.DBConnector
  - class com.gud.persistence.Department
  - class com.gud.servlets.EmailFilter
  - class com.gud.persistence.Faculty
  - class javax.servlet.GenericServlet (implements java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig)
    - class javax.servlet.http.HttpServlet (implements java.io.Serializable)
      - class com.gud.servlets.GUDServletsRoot
        - class com.gud.servlets.Copyright
        - class com.gud.servlets.GUDLogon
        - class com.gud.servlets.GUDProfile
        - class com.gud.servlets.People
        - class com.gud.servlets.Research
        - class com.gud.servlets.University
  - class com.gud.persistence.Group
  - class com.gud.persistence.Interest
  - class com.gud.persistence.Project
  - class com.gud.servlets.ResultsetBean (implements java.io.Serializable)
  - class com.gud.persistence.Staff
  - class com.gud.persistence.University

Appendix B.

JavaBean Sample code

```
package com.gud.servlets;

/*
* @(#)ResultsetBean.java    1.0 GA
*Copyright (c) 1998-1999 Multimedia Information Research Lab, All Rights Reserved.
* This software is the confidential and proprietary information of Multimedia Information Research
* Lab (MIRL), University Of Ottawa ("Confidential Information"). You shall not disclose such
* Confidential Information and shall use it only in accordance with the terms of the license
* agreement you entered into with MIRL.
*/
import java.util.*;
import java.sql.*;

/**
 * This Bean object is being used to hold the query results passed from the servlet
 * It is used by JSP to generate the dynamic content of html file
 *
 * @version 1.0 GA
 * @author Yaoping Wang
 */

public class ResultsetBean implements java.io.Serializable {
  //return numbers chosen from the user, and the start index in a returned page
  private int pageRowSize, startIndex;

 //flags to show weather it's the first page, the last page, and the first JSP call
  private boolean isFirstPage,isLastRow, isFromServlet;
  //hold the column name vector
```

```
  private Vector columnNames;

 //hold whole results, a vector of vector
  private Vector resultRows;

/**
 * ResultsetBean constructor to take the vector of column names, result rows,
 */
public ResultsetBean(Vector columnNames, Vector resultRows) {
  this.columnNames = columnNames;
  this.resultRows = resultRows;
}

/**
 * ResultsetBean constructor to take the vector of column names, result rows,
 * and page row size required by the user.
 */
public ResultsetBean(Vector columnNames, Vector resultRows, int pageRowSize) {
  this.columnNames = columnNames;
  this.resultRows = resultRows;
  this.pageRowSize = pageRowSize;
  isFirstPage = true;
  isLastRow = false;
  startIndex = 1;
  isFromServlet = true;
}
```

Appendix C

JSP page sample code

(DisplayResults.jsp)

```
<-- Retrieve query results from the JavaBean and process the html generation -->
<jsp:useBean id="QueryResult" scope="session" class="com.gud.servlets.ResultsetBean" />

<html><body>

<-- global varibles -->
<%! int totalPage=0, currentPage=1; %>

<-- local variables -->
<% Vector resultRows = QueryResult.getResultRows();
  Vector columnNames = QueryResult.getColumnNames();
  int rowSize = resultRows.size();
  int startIndex=1, pageRowSize=10;
  QueryResult.renameDisplayedColumnName();

  <-- the first call from the servlet -->
  if(!QueryResult.getIsFromServlet()) {
        try{
              startIndex = new Integer(request.getParameter("startIndex")).intValue();
              pageRowSize = new Integer(request.getParameter("pageRowSize")).intValue();
              if(pageRowSize!=1) {
                    currentPage = startIndex/pageRowSize+1;
              } else {
                    currentPage = startIndex/pageRowSize;
              }
        } catch(NumberFormatException e) {}
  <-- the following calls from the cache -->
  } else {
        startIndex = QueryResult.getStartIndex();
```

```
        pageRowSize = QueryResult.getPageRowSize();
        QueryResult.setIsFromServlet(false);
        currentPage=1;
        totalPage = rowSize/pageRowSize + ((rowSize%pageRowSize==0)?0:1);
    }

    Vector displayRows = QueryResult.getResultsByRange(startIndex, pageRowSize); %>

    <%-- if the result set is empty, return another JSP page saying "nothing found" --%>
    <% if (rowSize == 0) { %>
        <jsp:forward page="/jsp/noFound.jsp"/>
    <% } else { %>
<center><body bgcolor='#f0f0f0'><applet code='com.gud.applets.GUDServletsBanner.class'
codebase='../gud1998/applet' height=39 width=810></applet><hr size=2><center>
        <table width="100%" cellspacing cellpadding> <tr>
        <%-- at least one row of data found, display the field names in the table --%>
        <td valign = middle bgcolor='#b0b0b0'><em>N</em></td>
        <% for (int i = 0; i < columnNames.size(); i++) { %>
          <td valign = middle bgcolor='#b0b0b0'> <%= columnNames.elementAt(i) %></td>
        <% } %> </tr><tr>
        <% for (int i = 0; i <= columnNames.size(); i++) { %>
          <td><hr></td>
        <% } %> </tr>

        <%-- display rows of data into a table --%>
        <% for (int j = 0; j < displayRows.size(); j++) { %>  <tr>
          <td><em><%= startIndex+j %></em></td>
          <% for (int i = 0; i < columnNames.size(); i++) { %>
                <% String datum = (String)((Vector)displayRows.elementAt(j)).elementAt(i);
                  String fieldName = (String) columnNames.elementAt(i); %>
                <%-- datum either starts with No or empty, display "not available" --%>
                <% if (datum.startsWith("No")||datum.equals("")) { %>
                        <td valign = middle
                        <% if (j%2 == 0) %> bgcolor='#eeeeee'>
                        <% else { %> bgcolor='#dddddd'> <% } %>
                  <%= "not available" %></td>

                <%-- display hyperlink, if the length too long, truncate to size 30 --%>
        <% } else if (fieldName.endsWith("URL") || fieldName.endsWith("HomePage")
        ||fieldName.endsWith("Publications")) { %>
```

```
                    <% String datumSub = datum;
                    if (datum.length() > 30 ) {
                            datumSub = datum.substring(0,26) + "...";
                } %>
                    <td valign = middle
                            <% if (j%2 == 0) %> bgcolor='#eeeeee'>
                            <% else { %> bgcolor='#dddddd'><% } %>
                <a href='<%= datum %>'> <%= datumSub %></a></td>

                <%-- display email link, if the length too long, truncate to size 30 --%>
                <% } else if (fieldName.endsWith("Email")) { %>
                  <-- filter out wrongly-formatted email data -->
                  <% String datumSub = QueryResult.getFormatedEmail(datum);
                  if (datum.length() > 30 ) {
                            datumSub = datum.substring(0,26) + "...";
                  } %>
                 <td valign = middle
                  <% if (j%2 == 0) %> bgcolor='#eeeeee'>
                  <% else { %> bgcolor='#dddddd'> <% } %>
                            <% if (datumSub == null || datum.startsWith("#")){ %>
                                    <%= "not available" %></td>
                            <% } else {%>
                                    <a href='mailto:<%= datum %> '> <%= datumSub %></a></td>
                            <% } %>
                <% } else { %>
                            <td valign = middle
                            <% if (j%2 == 0) %> bgcolor='#eeeeee'>
                            <% else { %> bgcolor='#dddddd'> <% } %>
                <%= datum %> </td>
                <% } %>
        <% } %>
    </tr>
  <% } %>
  </tr></table><p><hr size=2>

<-- display the information at each page like "Page 4 of 10" -->
<table width="100%"><tr><td align=center><font size="+3" color="red">Page <%=currentPage%> of
<%=totalPage%></td></tr></table>
```

```
<-- display the "previous" arrow if the desplayed page is not the first page -->
<table width="100%"> <tr> <td align='left'>
<% if (startIndex > 1) { %>
        <form action="/jsp/GUDDisplay.jsp" method="post">
        <input type="hidden" name="startIndex" value=<%= startIndex - pageRowSize %>></input>
        <input type="hidden" name="pageRowSize" value=<%= pageRowSize %>></input>
        <input type="image" src="../gud1998/image/previous.gif"  width="40" height="30" >
         <font size="+3" color="green"><%= pageRowSize %></font>
        </form>  </td> <% } %>

<-- display the "next" arrow if the desplayed page is not the last page -->
<td align='right'>
<% if (!QueryResult.getIsLastRow()) { %>
        <form action="/jsp/GUDDisplay.jsp" method="post">
        <font size="+3" color="green"><%= pageRowSize %></font>
        <input type="hidden" name="startIndex" value=<%= startIndex + pageRowSize %>></input>
        <input type="hidden" name="pageRowSize" value=<%= pageRowSize %>></input>
        <input type="image" src="../gud1998/image/next.gif"  width="40" height="30" >
        </form>  </td> <% } %> </tr></table>
<% } %>

<-- display the footnote -- copyright information -->
</table><hr width='54%' color='#808000'><center>
<a href='Copyright'>Copyright</a> © 1999 Multimedia Information Research Laboratory
</body> </html>
```

Appendix D

Publications

Wang, Yaoping, Pham, Vu and Karmouch, Ahmed, *"The issues of design a Global University Database System"*, In proceeding of CCECE99, Edmonton, 1999