Developing a Courseware Database for The AudioGraph

A Thesis presented in partial fulfilment of
the requirements for the degree of

Master
in
Computer Science

At Massey University, Palmerston North,
New Zealand.

Jun Pan

2000

Dedication

To memory of my auntie, Jin Yuan Pan

To my eldest sister, Feng Lan Pan

To my lovely daughter, Shu Ke

Acknowledgments

I would like to take this opportunity to thank the people who have helped to make my thesis a reality.

First, I would like to thank my supervisor, Chris Jesshope, for providing valuable guidance and suggestions along the way. I would also like to thank Elizabeth Kemp, who was very careful of checking my thesis and giving suggestions.

Second, I would like to give my thanks to Zhenzi Zhang, my colleague and my best friend, for her ability to endure my demands for improvement. Without her continued help, my thesis wouldn't be handed in on time. Thanks to my friend, Margaret Rollinson for checking my English grammar and for making this a smooth and understanding process. Thanks to my colleagues, Simon Zhang, Robin Luo and Yongqiu Liu. They gave me valuable feedback for my coding test.

Finally I give my thanks to my lovely daughter, Shu for her support and understanding during thesis writing. Giving up mum time has not been easy for her. Shu, keep working on your computer and someday you can go beyond your mum.

Contents

Abstract

The goal of this project is to investigate and prototype a database driven server for the editing and delivery of multimedia courseware. This project required the analysis, design, and construction of a client/server based, distributed educational system. The components of the project are a relational database server with a particular database schema that can be downloaded or distributed with an existing project and the AudioGraph. The AudioGraph is an application using a multi-media tool to publish university lectures, tutorials or training material on the Web. The front-end interface is a Java application that lets the lecturers or students interact with the database. This system can be used to keep track of various stages of courseware development and web publishing. The overall aim was a flexible and adaptive system with the current lecture development and environment maintained. The system may be distributed on Windows NT, Unix and Macintosh platforms and so is portable and extendible and is platform-independent. The background and technology employed in the project is introduced. Each stage of the project process is explained in terms of the development lifecycle of the system. A limitation imposed by multi-platform compatibility is discussed and the achievement is presented by screenshots. Through the report, the structure of the file, run time environment, inter-process communication, user interface, and server access are explained.

List of Figures

Definitions

AudioGraph	A multimedia authoring tool for Web publishing, designed by researchers at Massey University and distributed by NZEdSoft.
API	Application Programming Interface: the group of system tools that allows an application to access the system.
AWT	Abstract Window Toolkit: the Java API that enables programmers to develop Java applications with GUI components.
CASE tool	Computer Aided Software Engineering tool: this refers to any computer-based tool for software planning, development, and evolution.
CORBA	Common Object Request Broker Architecture: the OMG (see OMG definition) platform-independent technique for programs running on different machines to communicate with each other.
COM	Component Object Model: Microsoft's interface specification for hooks into Java for inter-program communication.
DBA	Database Administrator is responsible for the physical realization of the database system.
DBMS	Database Management System: this enables users to create, define and maintain the database and provides access controls to this database.
DCOM	Distributed Component Object Model is a set of Microsoft concepts and program interfaces in which a client program object can request services from server program objects on other computers in a network.
DCE	Distributed Computing Environment: an industry-standard software technology for setting up and managing computing and data exchange in a system of distributed computers.
DDL	Data Definition Language: a language can be used to define the database.
DML	Data Manipulation Language: this enables users to insert, update, delete and retrieve data from the database.
ANSI	The American National Standards Institute, which is the primary organization for fostering the development of technology standards in the United States.
ER	Entity-Relationship is a methodology to model a relational database.
FTP	File Transfer Protocol: this is basically a tool used for transferring and handling files across computers anywhere on the Internet.
GUI	Graphical User Interface to a computer.
HCI	Human-Computer Interaction is the study of how people interact with computers and to what extent computers are or are not developed for successful interaction with human beings.
IDE	The Integrated Development Environment is a collaborative strategic infrastructure embodying data standards that support business processes across a number of geographically dispersed and heterogeneous organizations.

JDBC Java Database Connectivity is an application program interface specification for connecting programs written in Java to the data in a popular database.

JDK Java Development Kit: a development environment for writing applets and applications that conform to the Java platform.

JFC Java Foundation Classes: this extends the original AWT and is portable and compatible with all AWT-based applications.

JIT Just-in-Time compiler, which improves the performance of Java run time and is used to implement a Java Virtual Machine.

JNMI Java Native Method Interface

JRE Java Runtime Environment.

JVM Java Virtual Machine: this provides an application with a guaranteed run time environment.

MIME Multi-Purpose Internet Mail Extensions are an extension of the original Internet e-mail protocol that let people use the protocol to exchange different kinds of data files on the Internet.

MRJ Macintosh Runtime for Java refers to the Java Virtual Machine and associated class libraries for the Macintosh operating system.

mSQL Mini SQL is a lightweight database engine. It offers a subset of SQL as its query interface.

mSQL-JDBC This is a JDBC driver of the third kind; A pure-Java JDBC driver for mSQL was created and is being maintained by George Reese from The Center for Imaginary Environments.

MySQL This is a true multi-user, multi-threaded SQL database server.

ODBC This stands for Microsoft's Open Database Connectivity; A standard or open application programming interface for accessing a database.

OO Language Object-Oriented Language: this is used to programming.

RDBMS Relational Database Management System.

RFC Request for Comments, in which the FTP commands are described.

RMI Remote Method Invocation: enables the programmer to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts.

SDK Software Development Kit: a programming package that enables a programmer to develop applications for a specific platform.

SDLC System Development Life Cycle.

SQL Standard Query Language is a simple, high-level language used for specifying transactions on a database.

UML This is a OO language and stands for Unified Modeling Language, which was designed by Grady Booch, James Rumburgh, and Ivar Jacobson, known as the three amigos.

URL Uniform Resource Locator; URLs are the mysterious text strings.

Chapter 1 Introduction

The aim of this project is to develop a client-server database system to keep track of various stages of courseware development and publishing. The system is to be used in connection with an existing project, the AudioGraph recorder [1-3], so that departments or groups of individuals can keep track of courseware, when preparing material and also provide interfaces for retrieving information when presenting it to students. The project is to be integrated on the Web, which provide the course information to both course developers and students.

1.1 The Description of the Project

This section discusses the background and goal of the project. It also describes briefly what the project should achieve.

1.1.1 Background of the project

The AudioGraph is a multimedia authoring tool for Web publishing, designed by researchers at Massey University and distributed by NZEdSoft [4], for recording audio-graphic presentation material for publication in an HTML reference environment. The tools comprise two Macintosh applications and two Netscape plug-ins. The applications are a multi-media authoring tool for recording lectures for publication on the Web and a previewing tool, which allows playback of the recorded material without having to publish it on a Web server. The AudioGraph has applications in all forms of teaching and training, such as for presenting formal university lectures, tutorials or training material on the web. Examples of the use of the tool can be found both at Massey University [5] and at Surrey University [6].

Using the recorder for preparing a web presentation is very simple. It can be based on a set of slides generated from Microsoft's PowerPoint as a Scrapbook file; this file format holds any Macintosh resource and in this case it holds one PICT image per slide. Such a file can be opened directly by the AudioGraph, and when saved it stores the slides and any annotations in the Macintosh AudioGraph format (*the lecture document*). When published as a web presentation, the same information in converted to a number of cross-platform files (the *presentation* file formats), which comprise (*.aep), and HTML files (*.htm). See AudioGraph documentation for more detail [7].

1.1.2 Purpose of the project

The sources to the web presentations are included in one large file with a number of slides in it, which makes it slow to load and save. So from the perspective of the AudioGraph recorder, what we want is a database in which we can store, or store references to, single slides of AudioGraph material. The database will store information about the structure of presentations or courses, which will combine all components required, namely the AudioGraph source file, the .aep and .htm files, which will need to

be maintained on a server before they go live for presentation to students. The database can also keep track of who developed or changed a particular presentation, in much the same way as a computer aided software engineering tool (CASE tool). This database will need to be relational in order to be able to maintain flexibility in creating and maintaining these tables and links. There will be additional data associated with each slide, including meta data describing the information on it, listing keywords etc, so students and lecturers can search for more information on a given topic.

Having the presentation built up indirectly or dynamically using information within a database enables lecturers to be able to modify presentations and then to update the web sites that include the presentation in the lectures. Thus the interface should also be able to store meta data about links from one slide to another so that when material is moved or updated the links will be updated as well.

In terms of interfaces we need to make a web interface which integrates a search engine so that students can find more information about other course components when they are browsing the lectures. Again there will be a requirement for an interface in the recorder to give access to the data concerning the stored presentations.

1.1.3 Components of the project

On the basis of the general nature of the project as described above, the components of the project can be specified as follows:

- To undertake a specification and requirements analysis of the system.
- To choose a database, preferably a public domain database, that can be downloaded or distributed with the AudioGraph software.
- To develop schemata for the database material so that course developers can find, edit, and create links between recorded AudioGraph material and so that students reading the AudioGraph can search for related material.
- To design a human computer interface to let the users (both developers and students) to interact with the database.
- Develop a web-based front-end interface to the database for developing courseware using the AudioGraph and for browsing the courseware.
- Link the front-end interface into the current AudioGraph project.
- Develop an html interface for searching the database from the web.

The following figure gives an intuitive view of the system:

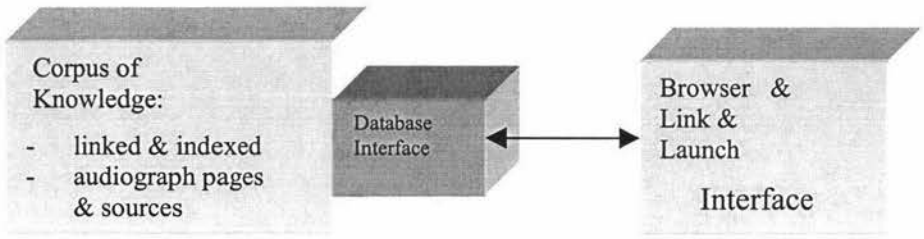

Figure 1.1 Conceptual view of the system

1.2 An Overview of the Project's Phases

The systems development life cycle (SDLC) is composed of five major phases: requirement specification, analysis, design, implementation, and test. The first four phases, as related to this project, are discussed in this section. The last phase will be discussed in chapters 5 and 6.

1.2.1 Requirements specification and analysis

Analysis and design begins with requirement analysis. A particular challenge, nowadays, for the analyst must also look at the business and real world needs rather than just the technical implementation. In effect, the requirements phase is a kind of front-end analysis.

Every method, tool, and practice has its own set of symbols and terminology, resulting in a lot of confusion and frustration. By establishing a common modeling technology that can be used for all kinds of systems, all phases of development, all scales of size, and with many different processes, object-oriented technology can take one step further to infiltrating the mainstream marketplace.

Object orientation technology is for producing models that reflect a domain, such as a business domain or a machine domain, in a natural way, using the terminology of the domain. Object-oriented models, when constructed correctly, are easy to communicate, change, expand, validate, and verify. Using it to analysis and design the system, our focus should not be on remembering in which direction the arrows are drawn in a specific tool or notation, but on creating high-quality models to help us build better and more effective software systems.

In terms of the application of object-oriented development, there are several different types of systems that can be analyzed. The book, *UML Toolkit* [8], categorizes them with their most common characteristics and is summarized below:

- *Information systems*: Store, retrieve, transform, and present information to users. Handle large amounts of data with complex relationships, which are stored in relational or object databases.
- *Technical systems*: Handle and control technical equipment such as telecommunications, military systems, or industrial processes. They must handle the

special interfaces of the equipment and have less standard software than information systems. Technical systems are often real-time systems.

- *Embedded real-time systems*: Execute on simple hardware embedded in some other equipment such as a mobile phone, car household appliance, etc. This is accomplished through low-level programming that requires real-time support.
- *Distributed systems*: Distributed on a number of machines where data is transferred easily from one machine to another. They require synchronized communication mechanisms to ensure data integrity and are often built upon object mechanisms such as CORMA (Common Object Request Broker Architecture), COM/DCOM (Component Object Model/Distributed Component Object Model), or Java Beans/RMI (Remote Method Invocation).
- *System software*: Defines the technical infrastructure that other software uses. Operation systems, databases and user interfaces perform low-level operations on the hardware, while presenting generic interfaces for other software to use.
- *Business systems*: Describe the goals, the resources, the rules, and the actual work in the business.

It is important to emphasize that most systems don't fit neatly into one of these categories, but belong to more that one of the system types, or as a combination of them. The system developed in this project, like many of today's information system, have both distributed and embedded real-time requirements.

The first step in analyzing a complex system is to divide it into a number of smaller, more manageable objects and then analyze each of these smaller objects separately. Applying that strategy to this project, it might be divided into the following components:

- *Database analysis*: In the server, the database should contain the information which provides support for academic staff to login, access their own courses and AudioGraph presentations and for students to read the AudioGraph and related materials flexibly.
- *Interface analysis*: In the client, the system should have a modern graphical user interface (GUI) which should contain all of the functions required for the various interactions with the users.
- *System mechanisms*: The System should have flexible mechanisms for browsing, downloading and uploading the various files. This will require the implementation or adoption of various protocols, such as ftp and http.
- *Multiple run environments*: The system should run on all popular technical environments (UNIX, Windows, Macintosh etc.).

ER (Entity-Relationship) modeling is used to analyze and to design a relational database in this project. UML is used to do both the requirements and domain analysis in the client-side system and the project uses an OO (Object-Oriented) language to implement the applications.

UML stands for Unified Modeling Language, which was designed by Grady Booch, James Rumburgh, and Ivar Jacobson, known as the three amigos. UML has the capability to model all of the above system types. UML can also be used in the different phases in the development of a system, from the requirement specification to the testing of a finished system.

1.2.2 Design schemata of the system

Once the requirements for the new system have been properly documented, it must be specified in sufficient detail to form a basis for later development.

One of the first issues in the specification of this project was the choice of a public domain database that should be relational as having a well define and easy to implement web-based interface.

The purpose of the design stage of the project is to specify a working solution that can be easily translated into programming code. The classes defined in the analysis are refined with additional detail, and new classes, which are created by object-oriented technology, are added to handle technical areas such as a database, a user interface, communication, devices and more.

The design can be divided into two segments: [8]

- *Architectural design*: This is the high-level design where the packages are defined, including the dependencies and primary communication mechanisms between the packages. Normally, the goals to make architecture clear and simple are few dependencies and avoid bi-directional dependencies.
- *Detailed design*: This stage gives details the contents in each package. It allows that all classes are described in enough detail to give clear specifications to the programmer. Dynamic models from the UML are used to demonstrate how objects of the classes behave in specific situations.

The following figure shows the architecture of the system:

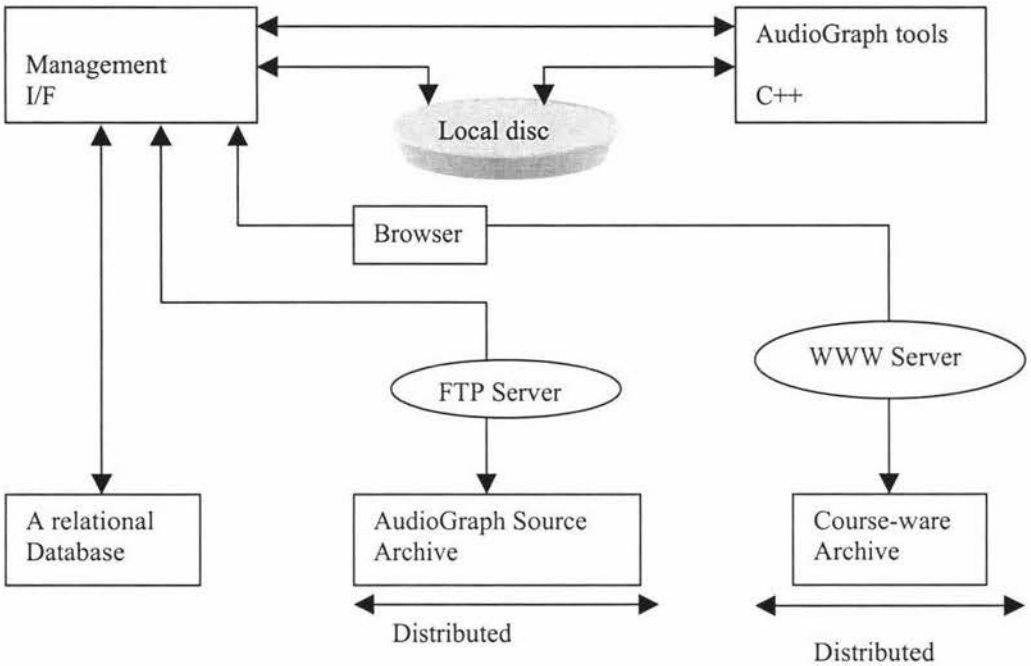

Figure 1.2 Architecture of the system

Database choice

Programming is all about data processing; data is central to everything we can do with a computer. Databases, like file systems, are the specialized tools for data storage. Access to the data into or out of the database depends on the implementation of the database and the protocols it provides to other systems. Most modern databases nowadays support a Standard Query Language, called SQL [9].

SQL is a comprehensive relational database language used for specifying transactions on databases. In fact, most of the simplest forms of database access and translate into equallysimple SQL statements. The database in our system needs to be able to run on Macintosh, PC and Unix systems, so it should be cross-platform, flexible and in addition make low use of memory.

mSQL and MySQL are two popular and robust database products, which can be downloaded from Internet and support key subsets of SQL on both LINUX and UNIX systems. With background of basic C, Java, Perl, or Python, we can write a program to interact with one of the databases, either as a stand-alone application or through a Web page [10].

MySQL is a real multi-user, multi-threaded SQL database server. It is a client/server implementation that consists of a server daemon mysqld and many different client programs and libraries. For Unix and OS/2 platforms, MySQL is basically free. However, for Microsoft platforms we must get a MySQL license after a trial time of 30 days [11].

Mini SQL, or mSQL, is a lightweight database engine designed to provide fast access to stored data with low memory requirements. The system itself is comprised of a database server and various tools that allow a user or a client application to communicate with the server. mSQL has been designed to provide exceptional data access performance on "small hardware" platforms. Because of these features, mSQL is well suited to the vast majority of data management tasks.

My research in comparing the two databases shows that MySQL is a bigger database while mSQL is a smaller one and ideal for systems that need a database operated with few system resources. Another different is that MySQL has several client API tools but the strongest is C API while for mSQL, the strongest is Java API. API stands for Application Programming Interface, which will be discussed later. This project employs mSQL because the system will use Java as the client programming code and the system needs fast access to data with low memory requirements. The last but not least reason, that mSQL is probably the most feasible solution for this particular project, is its free download for research purpose [12].

Mini SQL 2.0 is the second generation of the mSQL database system. The main focus of the second has been to extend tools to simplify the development and delivery of mSQL based applications. The performance increase for large applications has been achieved by the incorporation of flexible and powerful indexing to the database as well as sophisticated query execution optimization. One of the major applications of mSQL has been as a back-end database for World Wide Web sites.

Whatever our database choice, we must set up our database engine, create a database, and create the tables before we can begin writing client program. This client programming code needs to be able to access the database from remote computers belonging to lecturers developing courseware or students browsing the results of that development.

Languages such as Pascal, C, and C++ have brought a great deal of general functionality to programming, but Java's wide abilities make these languages seem incomplete and difficult. The features that Java possesses, such as its architecture independence, pure object-orientation and Internet-based, are more complete than the features of any of its predecessor [13]. So we have chosen Java as the development language of this project.

It is also evident that a suitable interface for developing client-server database systems is the Java JDBC (Java Database Connectivity) driver for that database engine. There are four basic types of JDBC drivers. The first one is simply a bridge between the JDBC API and the operating system dependent ODBC driver manager. ODBC stands for Microsoft's Open Database Connectivity, which is a C-based API for interfacing programs with relational databases. ODBC is popular mainly because it makes the underlying DBMS transparent to the client programs. DBMS stands for Database Management System and enables users to create, define and maintain the database and provides access controls to this database. The second one uses JNMI (Java Native Method Invocation) to call the functions of the database client library. Both of them are platform-dependent. The third one is the most interesting, a '100% pure' Java driver. Its important feature is that the

driver is written entirely in standard Java, and so it can be used on any platform that has a Java Virtual Machine.

The mSQL-JDBC is a JDBC driver of the fourth kind; using this we can pack all our class files together and deploy them on any platform we like. These files are added to 'the mixed a 100%' Java software installer and we can make our application as 'Run anywhere'.

User-interface Design

The past decade or two has seen the rise of a new approach, that of human-computer interaction (HCI). Given the number of incidents and accidents, which are attributed to 'human error', it is sensible to develop an approach to system design, which views humans as an essential element in the system [14]. Thus, an important aspect of designing systems is the study of the interaction between humans and the technology that they use.

As an example in this project, the roles are developers and students. It is obvious that people with a background in computer science will think of the system in quite different terms to people with a background in organizational psychology. In chapter 2, this thesis will discuss the User Role in more detail.

From the client application side of this project, there are two kinds of roles: one is academic staff who should be authorized and have a right to access to web and source material to do AudioGraph development. Another is the student who may also be authorized to access web but only to browse the presentations in their study.

A special activity carried out during the design phase of a project is the creation of the user interface and its 'look and feel'. The user interface in our client application is based on the user cases [8]. These interfaces provide the means of communication between the computer system and the people who use it. The interfaces, which access AudioGraph database, use 'search by course-plan' schema:

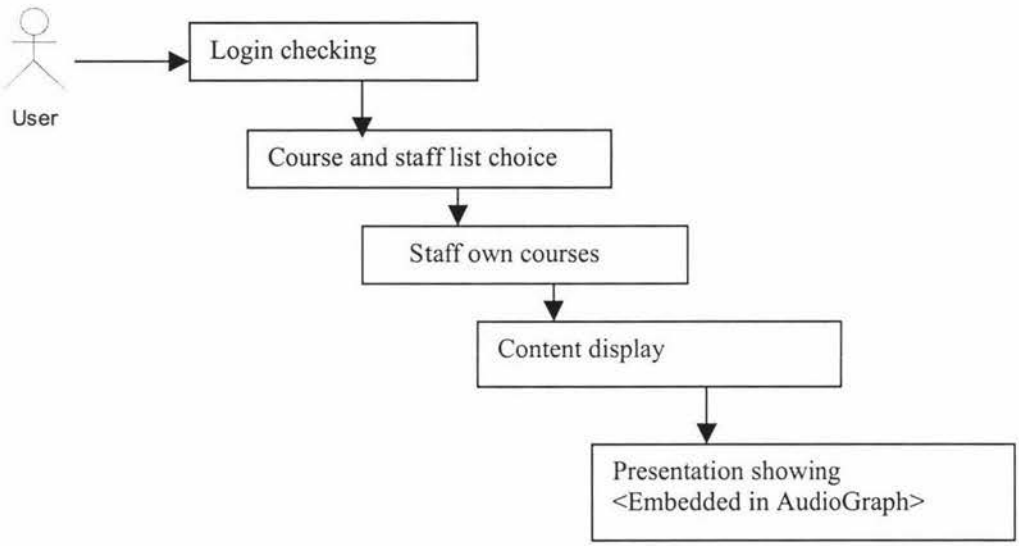

Figure 1.3 Search by 'course-plan' schema

1.2.3 Implementation of the system

The implementation phase is when the classes that have been developed in the analysis and design stages are programmed. The requirements specify that the system should be able to run on a number of different processors and operating systems that as a client server distributed application. As like mentioned above, Java was chosen to implement the system because of this.

But note, for this application, Java is being used as a general-purpose programming language. The project has not been implemented as a Java applet because no Java applet is permitted to access the file system on the client side [8]. Java makes mapping the logical classes to the code components easy, because there is a one-to-one mapping of a class to a Java code file (and a one-to-one mapping to a Java executable class file). Java also specifies that the name of the file be the same as that of the class it contains.

Developing schemata for the database material

The overall description of the database is called the database schema. There are three different types of schema in the database and these are defined according to the levels of abstraction of the three-level architecture. At the highest level, we have multiple external schemata, which correspond to different views of the data. At the conceptual level, we have the conceptual schema, which describes all the data items and the relationships between the data items, together with integrity constraints. There is only one conceptual schema per database. At the lowest level, the internal schema is a complete description of the internal model. It contains the definitions of stored records, the methods of representation, the data fields and the indexes and hashing schemes used, if any. In chapter 2, part 3 will give more detail about this topic.

Perhaps the most important component of the DBMS environment, certainly from the end-users' point of view, is the data. The data, in a database, acts as a bridge between the machine components and the human components. The database contains both the operational data and the meta-data, the 'data about data'. In our project, a subset of the final implementation, the database system schema consists of 9 files (tables) which contains data such as:

- Names, types, and sizes of data items.
- Names of linking relationships between the data items.
- Integrity constraints on the data.
- Names of authorized users who have access to the data.
- What indexes and what storage structures are being used, such as hashing inverted files, or B+-trees.

Implementing a front-end interface

The user interface application is "on top" of the other application. It presents the services and information in the system to a user. This package in our system is based on the standard Java AWT (Abstract Window Toolkit) class library for writing user-interface applications in Java, and can be executed on all Java platforms [15].

An Application Programming Interface (API) is the group of system tools that allows an application to access the system. The major types of application program interfaces include dynamic SQL interface, host language embedding interface, module language interface, and the callable function library interface [16].

In this project, SQL statements are embedded in the host language, Java. This is achieved using the JDBC API application interface.

Each approach to the design of systems has its own notion of what a system is, how it works, how to improve it, etc. At some stage during the requirements analysis, decisions must be taken on precisely what is to be included in the new system. Some of the requirements will turn out to be vital; others less so, and there will always be a trade-off between functionality and cost. Therefore, the implications of these decisions must be carefully evaluated.

Multiple running environments

This project worked under a client/server-distributed system environment, which is a popular technical environment nowadays and comprised of several platforms. We summarize briefly here.

In terms of the design, the application should also include an ftp client so that it can fetch and send back source files to a server anywhere with the ftp protocol.

The project chose Unix as a server system, which is installed with the mSQL database and can be distributed with the AudioGraph software. Because:

1) UNIX is an ideal and long history server system;
2) mSQL supports key subsets of SQL on UNIX;
3) Unix can interface to AudioGraph software, which understands local file names on Macintosh computer and
4) Unix can be a management interface to schemas for the information to be held on the database and protocols to link or transfer files between the various systems.

Both Windows NT and Macintosh act as clients in this system, as the AudioGraph software is implemented on Macintosh and will be implemented on Windows. In developing a front-end interface to the database for the AudioGraph software therefore the development of a distributed computing environment using a database running as a server with multiple clients accessing it using Java.

1.3 Layout of the Thesis

This thesis analyzes the design of the system described above and illustrated in Figure 2. Then it describes the detailed implementation in Java.

Chapter 2 begins by describing what the AudioGraph is and how it works, and then gives the background of the Java API, which includes JDBC and RMI. It also gives the details about installing and configuring the mSQL and about familiarizing the reader with Codewarrior, which offers the working environment of the Java application.

Chapter 3 provides the system schemata. ER modeling is used to analysis and to design our relational database in server and UML is used to analysis and to design the client application. Then follows a description of the architecture design. The chapter ends with by translating requirements into a schema.

The different methods and tools, which are used in this project, are introduced in chapter 4. The chapter also covers problems arising during implementation and how they were overcome.

Even through a Java application is supposed to be cross-platform and portable to any machine, there were still some issues about cross platform implementation when coding, in which differences in the behavior of the code created on a Windows NT computer, when run on Macintosh, became apparent. The paper discusses these problems and corresponding solutions in chapter 5 of this thesis.

The presentation of the completed system, including screen shots captured while running, is given in chapter 6. The performance issues are analyzed here quantitatively. The end of this chapter mentions the work still to be done.

The appendices, which included the Java code, follow the conclusion of this thesis.

Throughout the thesis, the Java programming language is used to show the corresponding Java code for different UML concepts and constructions.

1.4 Summary

This chapter is a brief introduction to the report. First it discusses the background and objectives of the project. Then it provides an overview of the project's phases in terms of the system development life cycle. ER modeling is used to analyses and to design our relational database in server and UML is used to analyses and to design the client application. Lastly, the chapter outlines the layout of the report, which is comprised of six chapters, a conclusion and appendices.

Chapter 2 Background

With the increasing popularity of the Internet, in 1995, the real world began using a new term, the Intranet. The Intranet is simply a part of the Internet. Specifically, an Intranet uses all of the popular Internet tools to let people in the same university or organization work together. Because of this desire for more efficient data processing and sharing data the across Internet or an Intranet, distance education technologies were developed; integrating information databases so that both internal users and Web navigators can view the same on-line course materials.

Most database systems, however, provide proprietary library calls for interfacing to them. Thus in addition to writing code that depends on the platform of the application and the database server, it also necessary to use a different set of library calls for each database we wish to connect with. To be suitable for data to be handled by corporate applications, a client/server development tool needs to have a powerful and simple interface for database access [13].

Application Programming Interfaces (APIs) use SQL for sending client requests to database servers. The major types of application program interfaces include dynamic SQL, embedded SQL, direct-all interfaces, and module language support. Many vendors support more than one API type. It is usually important to think of the ease of use or efficiency of the interface.

Java APIs, now provide an important new way of accessing databases that address these portability issues and offer functionality beyond that offered by traditional client/server development tools. This database access tool is the JDBC (Java Database Connectivity) API.

JDBC is the primary API for SQL database access. The JDBC has a uniform interface that connects to a variety of relational databases and endows developers with a fundamental base to create higher level tools and interfaces. With the availability of JDBC it is possible to access databases from a server-based application. In JDBC, Java classes are defined to represent database data, SQL statements, database connections, and results sets. JDBC is fully included in the JDK 1.1 software release.

This project concerns the computer technologies that have been mentioned above, namely the integration of distance education material into a flexible and portable database system. Before talking about the system in detail, the backgrounds of the project will be discussed. Firstly, the AudioGraph system will be described more detail. Secondly, the mSQL database system will be covered further. Thirdly, the features of Java, especially the use of Java in a distributed client/server database application will be covered. Fourthly, JDBC and one of its drivers, mSQL-JDBC will be discussed. Lastly, an outline of work environment concerning this project and relevant versions of the software, are explored.

2.1 AudioGraph System

AudioGraph is a concept proposed by C.R. Jesshope and his co-workers [1] in their project at Massey and Surrey universities, to dispatch a multimedia education package using the power of the Internet. From a technical point of view, an AudioGraph is a multimedia object that can be embedded in a reference HTML document. Using this concept, the target has been set up to use Internet resources for promoting distance education through the current community. Along this research line, the early results (Jesshope *et al.*, 1998) indicate that AudioGraph would be an effective learning media, which could offer the following advantages:

- Interactive feedback more than one dimensional textbook
- Multimedia visual and sound effects to assist verbal explanation of obscure points.
- Support lecturer to tailor studying materials, and to decrease learning curve
- Free learner from pressure, and by playback to solidify the understandings about the materials.

2.1.1 Principle for AudioGraph

The embedded AudioGraph can be triggered by a simple click. The logic behind this click event represents an activation of annotations using a sequence of pen strokes, images or sound, all overlaying a background image within the media object. The prerequisites for achieving this are:

- AudioGraph production: AudioGraph object generation, editing and packaging in a compressed form for internet distribution;
- Portable AudioGraph presentation across the Internet: compatibility between different browser i.e. Netscape Communicator or Internet Explorer and on Windows NT and Macintosh platform.

A set of software tools is required for the implementation of this strategy around these two issues.

2.1.2 Studio for AudioGraph production

- AudioGraph Recorder

This is a Macintosh application for authoring web-based multimedia presentation. This end-user oriented tool takes advantages of incorporating existing presentation with voice, hand-drawn text and highlight annotations.

- AudioGraph Player

A standalone Macintosh application is used for preview of web presentations. During the course, the user might be able to monitor the progress of the presentation in the capacity of manipulation at will to start, stop and playback.

2.1.3 *AudioGraph presentation across the Internet*

- Communicating AudioGraphs

Recorded sound and data are streamed at a low bandwidth through the use of advanced compression. The required data rate for compressed audio (13 Kbps) meets the requirements for most modems (14.4 Kbaud) without stalling the presentation and having to wait for download.

- AudioGraph plug-in player

Playback in a web browser requires a plug-in. The AudioGraph is embedded in an HTML reference environment and distributed across the Internet. Browsers that conform to the Netscape plug-in standard would recognize the file or MIME (Multi-Purpose Internet Mail Extensions) type of the embedded object and use the AudioGraph plug-in for playback of the multimedia presentation.

2.1.4 *A facade for developing AudioGraph application*

The inherent drawback of the static HTML referencing system is a mismatch in the rapid expansion of courseware using the AudioGraph application. The author must spend time to maintain all of the hyperlinks between documents when editing, modifying, adding or deleting materials. In particular, when multiple authors want to access the system for any modification, it could be chaos for the performance and maintenance of the integral system. Data redundancy and inconsistency could cripple the operation of the system.

A solution is required to deal with the above situation. Fortunately, a relational database management system (RDBMS) is an excellent remedy for these potential problems. The tight relation between data and storage in RDBMS provides the data integrity and consistency. Because of this the project was directed to seek a solution for the AudioGraph using RDBMS.

2.2 Mini SQL Database System

This section discusses the relational database model and introduces a relational database, the Mini SQL system (mSQL).

2.2.1 *Database system and relational database model*

In the book, *Database systems* [9], a database is defined as *a shared collection of logically related data (and a description of this data) designed to meet the information needs of an organization*. Databases provide applications with a more powerful data storage and retrieval system based on mathematical theories about data devised by Dr. E. F. Codd.

When we analyze the information needs of an organization, we attempt to identify entities, attributes and relationships. An entity is a distinct object (in this project, it is a staff member, a course or a slide etc.) in the organization that is to be represented in the database. An attribute is a property that describes some aspect of the object that we wish to record, and a relationship is an association between several entities. The database represents the entities, the attributes and the logical relationships between the entities.

With database systems, we separate the definition of data from the application programs that manipulate the data. That is similar to modern software development, where we provide an internal definition of an object and a separate external view. This approach is known as data abstraction and has one advantage, that is, if new data structures are added or existing structures are modified, then the application programs are unaffected.

A database system involves the data itself, the hardware on which the data resides, the software DBMS and the users themselves. A DBMS provides:

- A Data Definition Language (DDL) that enables users to define the database.
- A Data Manipulation Language (DML) that enables users to insert, update, delete and retrieve data from the database [9].

The popular query language SQL provides the functions for both DDL and DML.

Three different database models have achieved widespread popularity: hierarchical, network and relational. This project adopts the most popular of these models, the relational database model.

The relational database organization has many advantages over the hierarchical and network schemes.

- The tabular representation used in the relational scheme is easy for users to comprehend and easy to implement in the physical database system.
- It is relatively easy to convert virtually any other type of database structure into the relational scheme. Thus, the scheme may be viewed as a universal form of representation.
- The projection and join operations are easy to implement and make it easy to create the new tables needed for particular applications.
- Searches in a database can be faster than in schemes that require following a series of pointers.
- Relational structures are easier to modify than hierarchical or network structures. In environments where flexibility is important, this becomes critical.

The relational model developed by Codd is a logical representation of the data that allows the relationships between the data to be considered without concerning oneself with the physical implementation of the data structures [9].

A relational database is composed of tables. Any particular row of the table is called a record. Each column of the table represents a different field. Tables in a database normally have primary key even if it is not required. The primary key can be composed of more than one column (or field) in the database but cannot contain duplicate values. The detail description of the schema for this project will be found in next chapter.

2.2.2 The mini SQL system

Mini SQL (mSQL) [10] is a lightweight client/server relational database that earned great popularity as a back-end for web sites. The original goal in developing Mini SQL was to provide high speed and perform 100 basic operations per second on an average UNIX workstation. It should use very few system resources with small data sets. The software is becoming popular because this function has been achieved. mSQL is probably the most feasible solution for this project because it is also a free download for research purposes.

The philosophy of mSQL has been to provide a database management system capable of rapidly handling simple tasks, such as fast access to data with low memory and modest hardware. mSQL has also been used as a back-end database for web sites.

During mSQL's life, people have used it for applications beyond the scope of the original design. These high-end applications, containing up to 1 million rows of data, showed a need for better handling of complex queries and large data sets if the package was to be used in this way. The second generation of the mSQL server has been designed to suit these high-end applications while maintaining the original design goals of mSQL1. It has been designed to meet three main criteria.

- Provide better performance for simple operations than mSQL 1.x
- Provide rapid access to large databases and complex operation.
- Provide more of the functionality outlined in the ANSI SQL specification.

The mSQL distribution includes the mSQL server, client programs, a C programming interface for client software, and several tools. User contributed software is available including interfaces to mSQL from Perl, Tcl, REXX, Java and Python, WWW interfaces, a Windows port of the client library and much more.

The performance features of mSQL are:

- *Performance:* mSQL is quicker than MySQL at [10]:
    - Perform repeated connects and disconnects, running a very simple query during each connection;
    - INSET operations into very simple tables with few columns and keys;
    - CREATE TABLE and DROP TABLE;
    - A table scan is very easy (if SELECT on something that isn't an index.).

- *SQL Features:* mSQL does not support GROUP BY at all [12].

mSQL represents a simple client-server architecture. The server maintains the databases, receives commands from client programs and sends replies. The standard client programs that we will need to use are:

- `mSQL` -- the mSQL terminal monitor -- an interactive program for typing ad hoc queries.
- `relshow` -- the mSQL schema viewer -- shows what databases are available on a server and what tables are in a database.
- `msqldump` -- the mSQL database dumper -- writes out an ASCII version of a database as a series of SQL statements that can be reloaded into mSQL.

2.3 Applications in Java

Java is a popular, full-featured computer language that incorporates the best of modern thinking about object-oriented (OO) programming. The language's creator, Sun Microsystems, defines this flavor of Java as "a simple, robust, object-oriented, platform-independent, multithreaded, dynamic, general-purpose programming environment". These features brought Java 'Write once and run anywhere' [13].

2.3.1 Features of Java

Many Java books talk about the features of Java, which are generalized below on the basis of the definition above [13,15,17]:

- *Simple:* one of the goals when Java was designed was to eliminate those elements it didn't need, such as header files, unions, structures and multidimensional arrays etc. Java also contains a fairly small number of language constructs and has automatic garbage collection. The best feature with Java is the elimination of the most common causes of programming error due to the elimination of pointers from the user's view of the language.
- *Robust:* Java is robust because of its early background as a language for consumer devices, which demand a highly reliable, high-quality language. Java's pointer elimination increases its robust flavor. Java's automatic garbage collection and automatic memory management further reduces programming errors, which often happen in C/C++.
- *Object-oriented:* Programming in Java is all about defining classes and making objects. No other language commonly used, including C++, is as pure object-oriented as Java. This feature of Java makes it a stronger language that thoroughly employs encapsulation and reuse.
- *Platform-independent:* Java's platform-neutral architecture is the first element of its portability. The next is that it won't include 'implementation-dependent' aspects of

the language specification. The Java environment itself can also be easily ported to new operating systems and hardware platforms.

- *Multithreaded:* Multithreaded means the application has more than one thread of execution at a time. Java can priorities its threads so that threads of low importance - like garbage collection - are delegated to a low-priority mode.
- *Dynamic:* Java is dynamic because it can adapt to an evolving environment. Java programs are able to create dynamic new instance variables and methods in a library's object without affecting the dependent client objects. Even from across a network, Java's dynamic abilities will also permit the loading of classes.
- *Secure:* Java is an inherently secure language because of its excellent security. Its security system has four levels, and each of them is armed to prevent corruption. There are three features in Java's runtime environment that makes it secure. They are a runtime memory layout, the byte-code verifier and file access restrictions [13,14].

2.3.2 Java is the best for developing Internet-based applications

The features of Java bring a multitude of users to both the programming community and to the World Wide Web. Java's web-related strengths lie in its ability to create applets and applications for complex, distributed networks, including both the Internet and Intranet. Because of this support, it is defined as a distributed language. In Java there is no big difference between opening a local file and a remote file. Java's socket class also supports reliable stream network connections, which provides the ability to create distributed clients and servers [16].

As we can find from Sun's books on Java, Sun claims that Java's executable content is another key to its success. Java is set to become the language for transferring dynamic, executable content over the Web. Although applets are still Java's most obvious contribution to the Web, the union of Java with the Web heralds great things.

For example, it will no longer be necessary to create different versions of applications for every single platform that exists. This will have a positive impact upon individual programmers and small development groups that lack the resources to port application to multiple platforms [17].

However, after testing this project, we have found these claims to be not completely true. As can be seen from Chapter 5 and 6, we state some issues of the cross-platform and performance of Java.

Architecture independence is one of Java's most advantageous characteristics, so long as we avoid some pitfalls in writing Java. It's architectural neutrality is useful not only for networks, but also for software development under the JDK (Java Development Kit) or the Java Virtual Environment, UNIX, Apple, Microsoft, and the Java appliance world are all united, so Java development should move ahead cohesively, and hopefully, coherently [13].

Java is also a considerable language in Web-based client/server computing. Client/server computing means that information is easily shared across computer networks where some computers, called file servers, offer a common store of programs and data that may be used by client computers distributed throughout the network. In a client/server application, Java provides the user's interface to a program running on an Internet server.

Java can be used to create both applets and server programs because the standard Java library includes many classes that facilitate network connections. Some of the client/server applications available include chat systems, online games and databases. A Java client/server application can make use of one of three major database architectures. They are relational database, object database and object-relational database [17].

The majority of today's database applications use relational databases. At the heart of Java's enterprise computing philosophy lie the distributed computing and database access APIs - RMI and JDBC, respectively. Older languages require third party APIs to provide this kind of support. Java includes these features into the central Java distribution, which it will be find on every Java platform.

JDBC provides the programmer with access to relational databases and leaves the issue of object-to-relational mapping up to the programmer. As a developer, we can write distributed applications that run against relational databases and know that those applications will run on any system on which you deploy them [15].

All in all, Java is rapidly becoming the language of choice for Internet-based applications. As can be found from the Internet, many programmers have discovered that programming in Java helps them to be more productive than programming in C or C++ [18]. To think of these benefits, this project uses Java to write client side applications. However, when implementing with Java, we found that Java still has some cross-platform problems need to be improved.

2.4 JDBC and mSQL-JDBC Driver

JDBC is an SQL-level API that allows us to construct SQL statements and embed them inside Java interface calls. It is simple and takes advantage of the experience of existing database APIs. JDBC enables a smooth translate between the world of the database and the world of the Java application by:

- Enabling this to occur in a database-independent fashion, JDBC requires a database to implement a run-time version of its interfaces. The implementation routes SQL that calls to the database in a fashion it recognizes, which may be proprietary. A Java database application does not therefore care what its database engine is. No matter how many times the database engine changes, the application itself need never change.
- With the 'write once, compile once, run anywhere' power that JDBC offers, Java's database connectivity allows you to worry more about the translation of relational data into objects instead of worrying about how you are getting that data.

2.4.1 *The structure of JDBC*

Java's JDBC API gives a shared language through which applications can talk to database engines. Following in the tradition of its other multi-platform APIs such as the AWT, JDBC provides a set of interfaces that create a common point at which database applications and database engines can meet.

A traditional client server application requires an administrator responsible for the deployment of the client program on user's desktops. Java affects the way an application is distributed and maintained. The Java language employs the idea of the zero-install client. The object code for the entire application, client and server, reside on the network. Because the Java Virtual Machine (JVM) provides an application with a guaranteed runtime environment, no administration is needed for the configuration of that environment for individual applications. Java VM, which will be mentioned, is the component of the technology responsible for its hardware- and operating system-independence, the small size of its compiled code, and its ability to protect users from malicious programs.

Java programs consist of pieces called classes. Classes consist of pieces called methods that perform tasks and return information when they complete their tasks. Each piece can be programmed to form a Java program. However, we can take advantage of the rich collections of existing classes in Java class libraries. The set of classes that implement the JDBC interfaces for a particular database engine is called a JDBC driver. That will be described later in this chapter.

In order to build a database application, it does not have to worry about the implementation of these underlying classes at all; the whole point of JDBC is to hide the specifics of each database and let the programmer concentrate on our application alone. Figure 2.1 shows the classes and interfaces of the JDBC API package [15].

The JDBC API is divided into a user-level API for processing SQL statements and handling all application to JDBC Driver Manager communications [15], and a JDBC Driver API that database vendors use to interface their database drivers to Java. The JDBC Driver Manager is the core of the JDBC architecture and serves to connect the Java application with one of the four different driver types, as well as to ship any results back to the application.

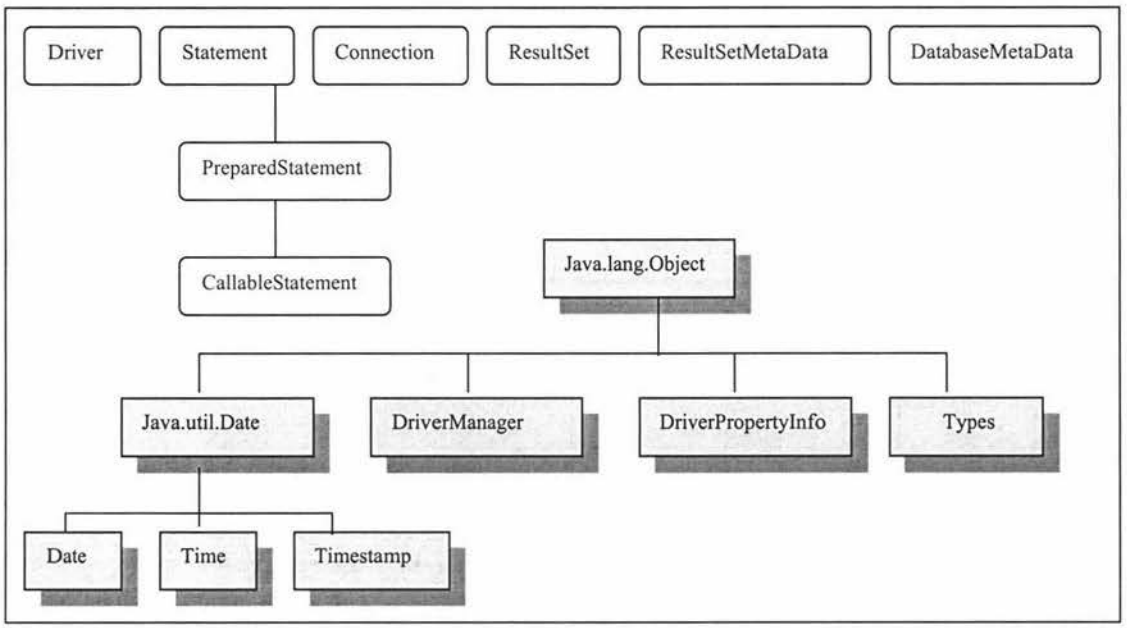

Figure 2.1 The classes and interfaces of the JDBC API package

2.4.2 Using JDBC for our database access

The set of classes that implement the JDBC interfaces for a particular database engine is called a JDBC driver. There are four basic types of JDBC drivers.

The first one is simply a bridge between the JDBC API and the operating system, which relies on ODBC driver manager. These kinds of drivers were created to allow quick development of applications without waiting for the database suppliers to create real JDBC drivers.

The second one uses JNMI (Java Native Method Invocation) to call the functions of the database client library. These are also platform-dependent, as the database supplier will have to provide both the client libraries and the JDBC driver specific for the desired platform.

The third one provides a client with a generic network API and the API is then translated into database specific access at the server level. The JDBC driver on the client uses sockets to call an application on a middleware of the server. This kind of driver is quite flexible because it requires no code to be installed on the client and a single driver is able to provide access to multiple databases.

The fourth type uses network protocols built into the database engine and talks directly to the database using Java sockets. This one is the most interesting, a '100% Pure' Java driver. It may not be certified as such, but the important feature is that the driver is

written entirely in standard Java, and then it can be used on any platform that has a Java Virtual Machine [20].
Some companies provide a variation of the third kind of driver in which the JDBC driver is just a stub, or a proxy, that forwards the JDBC calls to a server that has the logic to access the desired database [15].

There are many reasons for companies to create the last two types of driver, but the main ones are:

- An easy path to a 100% Java driver, without having to rewrite all database client logic in Java;
- To provide a unique JDBC driver that can be used with multiple databases, instead of installing one driver on each client for each database;
- To work around Applet security restrictions that won't let a Java Applet make TCP/IP connections to any server except the one from which the Applet has been downloaded [20].

This project uses the mSQL-JDBC driver, which belongs to the fourth type of JDBC drivers. It is described in detail below,

2.4.3 mSQL- JDBC Driver

This JDBC driver provides applets and applications with access mSQL database according to the Sun Microsystems JDBC database access API. Although mSQL cannot support all of the JDBC interfaces, this driver has an ability to let us write codes against JDBC and test this set of classes against a mSQL database. It also works well for any applications or applets, which we would like to write against mSQL database programming with JDBC and Java [15].

The mSQL-JDBC driver is a pure-Java JDBC driver for mSQL, which was created by and is being maintained by George Reese from the center of Imaginary Environments (http://www.imaginary.com/).

In addition to supporting the basic SQL syntax, mSQL-JDBC adds to support for prepared statements and multi-byte character sets. The mSQL-JDBC driver reads data from the database in separate threads. When we make a database query, the query will return immediately with an ResultSet class which provides the next() method to access data retrieved by a query. Details of the implementation will be discussed in next chapter. Figure 2.2 illustrates the architecture of the mSQL-JDBC driver in this project.

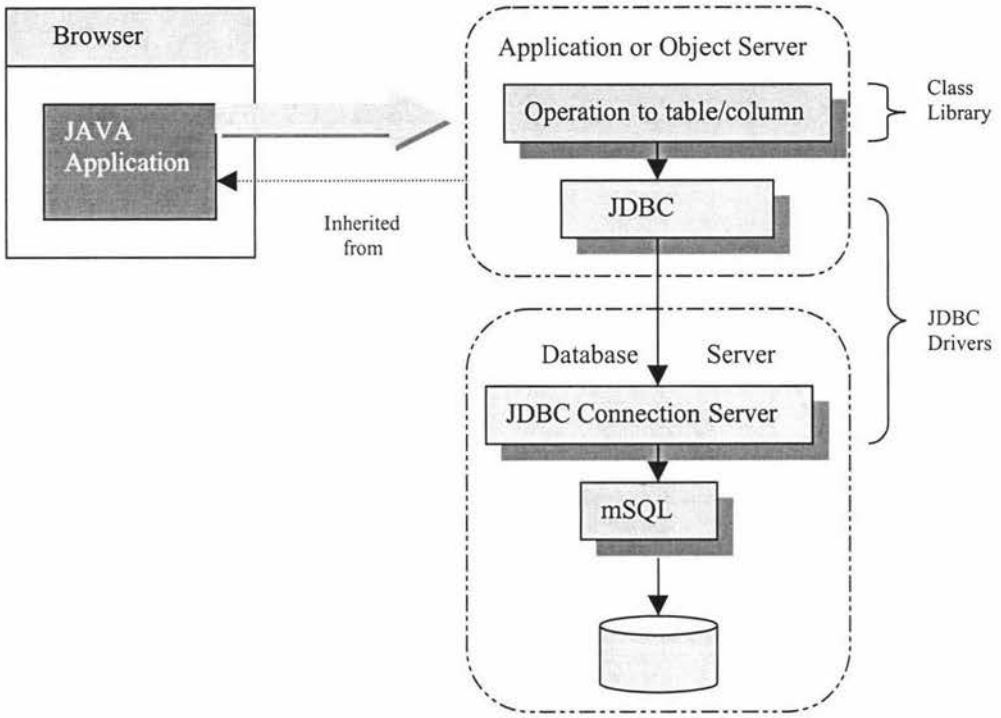

Figure 2.2 The architecture of the mSQL JDBC driver

The traditional application makes a clear distinction between the locations where processing occurs. That is, database access occurs on the server, and GUI processing occurs on the clients; the objects on the client machine talk to the database through a specialized database API [15]. In other case, the client might talk to the server through a set of TCP/IP or UDP/IP socket APIs. No matter what way it is employed, a wall of complex protocols divides the objects found on the clients from those on the server. Java using its Enterprise API, Remote Method Invocation (RMI) to helps break down this wall between the client and server.

Using RMI, an application can call methods in objects on remote machines as if those objects were located on the same machine. That is, any method call, whether on the same machine or across the network, uses the same Java method call syntax. This freedom to distribute objects across the network is called a distributed architecture [17].

Other languages use much more complex protocols like CORBA and DCE (Distributed Computing Environment). The former allows a Java program to communicate with programs written in other programming languages in a similar fashion to using Java's RMI package to communicate between Java programs. The later is a comprehensive suite of integrated products. It supports transparent file access and secure resource sharing in heterogeneous, networked computing environments.

For a client/server database application, a distributed architecture allows the various parts of the application to refer to the same physical objects when referring to particular objects in the data model. For example, take a course editing system that allows all staff who contributed to this course to update the lecture notes in this course. Current Web applications would have a user download a bunch of course information as an HTML page. If a staff updates some lecture notes that another staff or student is viewing at the same time, they will not see the staff updating the notes. This is because on each user screen we are simply seeing copies of data from the database.

If we reconstruct this Web application so that it is using RMI to retrieve data from a single lecture notes object on the server, we can allow any number of different staff and students to view the exact same lecture note object at the same time. In this way, we can be certain that all viewers see any change made to the lecture note object simultaneously. When staff update the lecture notes in a course, the course object makes an RMI call to all users viewing it to let them know the course has been updated.

All in all, this project will use JDBC for our database access, and RMI to distribute the objects that make up our application.

2.5 Working Environment with the Project

Experience has shown that the best way to develop and maintain a large program is to construct it from small, simple pieces or modules. Modules in Java are called methods and classes. The Java API provides a rich collection of classes and methods for performing common mathematical calculations, string manipulations, character manipulations, input/output, error checking and many other useful operations [18].

This client system, like typical client systems, consists of several parts: an environment, the language, the Java Applications Programming Interface (API), and various class libraries. The following discussion explains several class libraries, which support the client application.

2.5.1 Java Developer's Kit (JDK) 1.1.8

The Java API methods are provided as part of the (JDK). The JDK software and documentation is a free download from the web site [27]. JDK 1.1.8 is a development environment for writing applets and applications that conform to the Java 1.1 platform. Its compiler and other tools are run from a shell and have no GUI interface. Because the Macintosh Java development environment is not yet as up to date as the Window or UNIX versions at the time of project design. The MRJ (Macintosh Runtime for Java, which will be discussed in detail on the next step) 2.2 just supports JDK 1.1.8 and not Java 1.2. In order to 'write once, compiler once and run anywhere', the project uses JDK 1.1.8 which includes improvements in functionality, performance, and quality over version 1.0.2 of the Java platform, including bug fixes since the previous release of the 1.1 platform [27].

2.5.2 Collections

A collection represents a group of objects, known as its elements. The collections API includes a framework, which is a unified architecture for representing and manipulating collections of objects independent of their representation details. Collections enable interoperability between unrelated API, encourage software reuse, and make it easier to design, implement, or learn a new API. Collections were added in JDK 1.2. The InfoBus JDK 1.1 Collections API can be downloaded from the Web site [28].

2.5.3 Java Swing

Java Swing provides the Java foundation Classes and can be downloaded from Sun's web site at [29]. This project uses the *Java Swing GUI Components Package* from the web site [30]. The "Swing" refers to the new library of GUI controls (label, panel, etc.) that replaces the somewhat weak and inflexible AWT controls. These GUI components are written completely in Java and provide the Swing GUI components with little more flexibility. Because the GUI components can be customised to look like the user interface components of the computer platform on which the program executes, or they can use the standard *Java look-and-feel* that provides a user interface across some computer platforms. However, from the view of this project and some experts' comments on Java, it isn't like the Sun claiming "This provides the Swing GUI components with unparalleled flexibility". In fact, it still has some problems to fix properly in different platform and these issues of cross-platform will be discussed in Chapter 5.

2.5.4 MRJ on Macintosh

The Macintosh Runtime for Java is the Java Virtual Machine and associated class libraries for the Macintosh operating system. The MRJ has been a part of the default installation for MacOS since MacOS 8.0. However, MRJ is continually being updated and new versions are posted to Apple's Web Sites as they are available. In fact, there are essentially two parts to the MRJ. One is the MRJ itself, which is the runtime library needed if you are just running Java programs on a Macintosh. It is called the JRE (Java Runtime Environment) on other systems. The other is MRJ SDK that is the Java compiler and other developer tools. It is called the JDK (Java Developer Kit) on other systems.

2.6 Summary

This chapter is concerned with the computer technologies that are relevant to this project: the AudioGraph tool, the MiniSQL database system and the features of Java. Furthermore, JDBC and one type of its driver, mSQL-JDBC, are discussed in detail. The basics of computer work and the relevant software, including the system's environments are also explored in detail in this chapter.

Chapter 3 System Schema

In all engineering disciplines, the importance of models has been evident for a long time. A model is a description of some thing and can include a number of phases where each phase adds detail to the model [8].

As described in chapter 1, there are five phases of system development: requirement specification, analysis, design, programming, and testing. The requirement involves functionality, appearance, performance, and reliability. The analysis and design stages create a model that describes all the different aspects of the project. The model is often broken down into a number of views, each of which describes a specific aspect of the system under construction.

This project consists of three parts subjects: a relational database, user-interface application and work environment. The methodology of the Entity-Relationship (ER) is used to model and set up this relational database and UML is used to model the user-interface application. The work environment will be discussed in next chapter.

This chapter, first of all, introduces the concepts of ER and UML modeling. The second section contains the requirement analysis of the system. The system architecture design will follow in the third section. The chapter closes with a physical schema mapping.

3.1 Concepts of the ER and UML Modeling

This section introduces the ER and UML modeling methodologies that are used separately by the database and User-interface application.

3.1.1 Entity-Relationship modeling

A conceptual-level model is a convenient mechanism to allow the structure of a database to evolve over time. Conceptual models provide a framework for developing a database structure or schema from the top down. One very popular conceptual model is the entity-relationship model.

The entity-relationship model is a tool for analyzing the semantic features of an application. For a complex model, an entity-relationship diagram may be used to present a summary of the entities and relationships but it does not include the details of the attributes.

Some of the basic terms that are used for describing important entity-relationship model concepts are [9]:

- *Entity:* An entity is an object or concept that is identified by a name and a list of properties.

  Primary key: identifier used to uniquely identify one particular instance of an entity.

Foreign key: references a related entity through the primary key of that related entity.

- *Attributes:* An attribute is the property of an entity or relationship.

  Domain: conceptual definition of attributes.

- *Relationships:* the relationship is a set of associations between entities.

  Cardinality constraints: determines the number of possible relationships for each participating entity.

  Participation constraints: determines whether the existence of an entity depends upon it being related to another entity through the relationship.

A database normally contains many different entities. Although an entity has a distinct set of attributes, each entity has its own values for each attribute. The values held by attributes represent the main part of the data stored in the database. A relationship indicates the particular entities that are related.

An entity can be classified as a strong or a weak entity. A strong entity is an entity that is existence independent while weak entity is existence dependent on some other entity. An entity has some attributes that act as several keys such as candidate key, primary key or composite key. The most important key is a primary key, which is used to identify uniquely one particular instance of an entity [9].

The entity-relationship diagram provides a convenient method for visualizing the interrelationships among entities in a relational database. This tool has proven useful in making the transition from an information application description to a formal database schema. The entity-relationship diagrams are later turned into a conceptual schema in one of the other models in which the database is actually implemented [21].

3.1.2 OO methodology and modeling with UML

Object orientation is a technology for building models that reflect a domain, such as a project domain or a machine domain, in a natural way, using the terminology of the domain. When OO methodology is used correctly, the systems are flexible to change. The systems also have a well-defined architecture and provide the opportunity to create and implement reusable components. Requirements of the system are then traceable to code in the system.

With the release of UML, object-oriented developers have a common language for modeling and developing software systems. UML is a modeling language that consists of a notation and a set of rules directing how to use it. The notation is the symbols used in the models and the rules are syntactic, semantic, and pragmatic. Most modeling languages cover only syntax and semantics. Pragmatic is difficult to describe since it cannot be formalized; it can only act as a guideline. UML is much easier to comprehend than a natural language.

UML has a broad range of usage and is popular for software modeling in all phases of development and for all types of systems. The following overview describes the different parts of UML [8]:

- *Views*: Views show different aspects of the system and each view is described by a number of diagrams. The use case is the most popular view, which shows the functionality of the system as perceived by external actors.
- *Diagrams*: Diagrams are the graphs that describe the contents in a view. A diagram in a particular view should be simple enough to be easily communicated, yet coherent with the other diagrams and views. A diagram contains graphical symbols that represent the model elements of the system.
- *Model elements*: Model elements are used in the diagrams and they represent common object-oriented concepts such as classes, objects, and messages, and the relationships among these concepts. Several different diagrams can use the same model element, which always has the same meaning and symbol.
- *General mechanisms*: General mechanisms provide extra comments, information, or semantics about a model element. They also provide extension mechanisms to adapt or extend UML to a specific method process, organization, or user.

As an object-oriented modeling language, all the elements and diagrams in UML are based on the object-oriented paradigm. UML is phase independent, which means the same generic language and the same diagrams are used to model different aspects of the system in different phases of its development.

Use cases are used to capture the requirements of the user UML. Through use-case modeling, the external actors that have an interest in the system are modeled along with the functionality they require from the system. The actors and use cases are described in a UML use-case diagram.

In the analysis phase, the goal of the model is to capture the requirements of the system and to model the basic 'real-word' classes and collaborations. The classes that model these are identified, along with their relationships to each other, and described in a UML class diagram. Collaborations between classes to perform the use cases are also described, via any of the dynamic models in UML. In the analysis, only classes that are in the real-word concepts are modeled.

In the design phase, the purpose of the model is to expand the analysis model into a working technical solution with consideration for the implementation environment. New classes are added to provide the technical infrastructure: the user interface, database handling to store objects in a database, communication with other systems, interfacing to devices in the system, and others. The design results in detailed specifications for the implement phase.

In the programming phase, the model is the actual source code that is programmed and compiled into an object-oriented programming language. Depending on the capability of the language used, this can either be a difficult or an easy task.

The testing phase gives a description of how the system is deployed in the physical architecture. A system is normally verified in unit tests, integration tests, system tests, and acceptance tests. Different test teams use different UML diagrams as the basis for their work: unit tests use class diagrams and class specifications, integration tests typically use component diagrams and collaboration diagrams, and the system tests implement use-case diagram to validate that the system behaves as initially defined in these diagrams.

This system is analyzed and described in an analysis model with use cases and a domain analysis. It is then expanded into a design model that describes a technical solution. Finally, it is programmed in Java to create an application that is ready to run by the system tests.

The following sections give the requirements analysis, system design and physical mapping using the concepts of ER and UML described in this section.

3.2 Requirements Analysis

Correct and thorough requirement specifications are essential to a successful project. Requirements are a description of needs or desires for a product. The primary goal for the requirement phase is to identify and document what is really needed, in a form that clearly communicates to the user of this system.

3.2.1 Database requirement analysis

The requirement is a database in which it can store or refer to a presentation of AudioGraph material. The database will:

- Store information about presentations or courses, which will combine the AudioGraph source files;
- Maintain these source files on a server before they go live for presentation to students;
- Keep track of who developed or changed a particular presentation;
- Include meta data which describe the information on it, listing keywords with a presentation or lecture;
- Be a relational database that supports SQL.

This database system should analyze using the methodology of the database application lifecycle and build a conceptual data model from user view. To be simple, the staff acts as the user view for this requirement analysis.

It should provide support for the staff with the following support information:

- Staff information;
- Course information;
- Presentation information;
- Keyword in a presentation and course information;
- AudioGraph source file is edited record.

Figure 3.1 below, a simplified entity-relationship diagram shows the main entities and relationship of the system and identifies the functional areas of the system. The most common degree for relationships is binary and the cardinality ratios for binary relationships are one-to-one (1:1). One-to-many (1:M), and many-to-many (M:N).

This model needs applied extra entities to be applied to reinforce the structural constraint relationship that supports the logical design of the database. Detail operations will be discussed in next section.

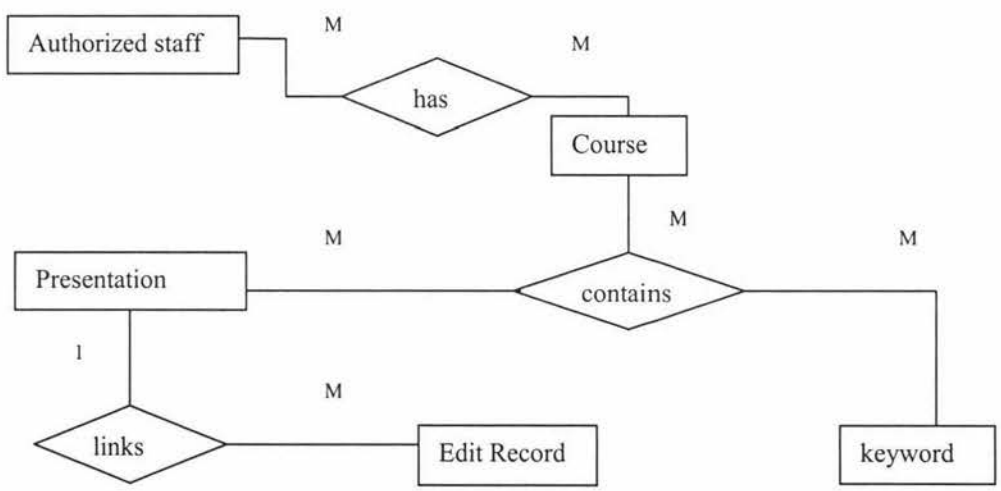

Figure 3.1 The conceptual Entity-Relationship model

Notes: M means multiple numbers of possible relationships for each participating entity.
1 means one number of possible relationships for each participating entity.

3.2.2 User-Interface application analysis

Many computer applications are interacted by humans. These applications must provide some mechanisms by which people can interact with the application. In the client side of

this project, it is this application system, which should contain some GUI functions to let the users access the database and related operations.

The first step in the requirement analysis is to define the use cases, which describe the system provided in terms of functionality. A use cases analysis involves reading and analyzing the specifications, as well as discussing the system with roles of the system.

This system should contain some functions, which provide support for authorized staff to access their own courses and AudioGraph presentations, for authorized staff and students to read the AudioGraph and related material flexibly in terms of keyword search.

One method used to identify use cases is actor-based, which:

- Identify the actors related to a system or organization;
- For each actor, identify the processes they initiate or participate in.

The actors in this system are identified as an administrator, the authorized staff and student. The administrator is responsible of developing and maintaining the database; while the staff, who have been authorized, have the privilege to add, insert, edit and delete their own course contents and these operations can be traceable to the database. The student can only have the right to display the presentation and related course information.

It should provide support for staff with the following functions:

- to login the system;
- to get a whole course number list and a whole staff number list;
- to choose one staff number and get the courses which relevant to this person;
- to choose one course and get the controller, contributors and structures relating to this course;
- to choose one presentation in this course structure and get the location of this presentation;
- to launch a browser and display the presentation using the AudioGraph multimedia tool;
- to download and upload the source file from the server and edit them in a local machine;
- to search for related material by keywords which reference to a presentation or a course;

The following is the use-case diagram, which shows the three actors and their connection to the use cases that the system provides.

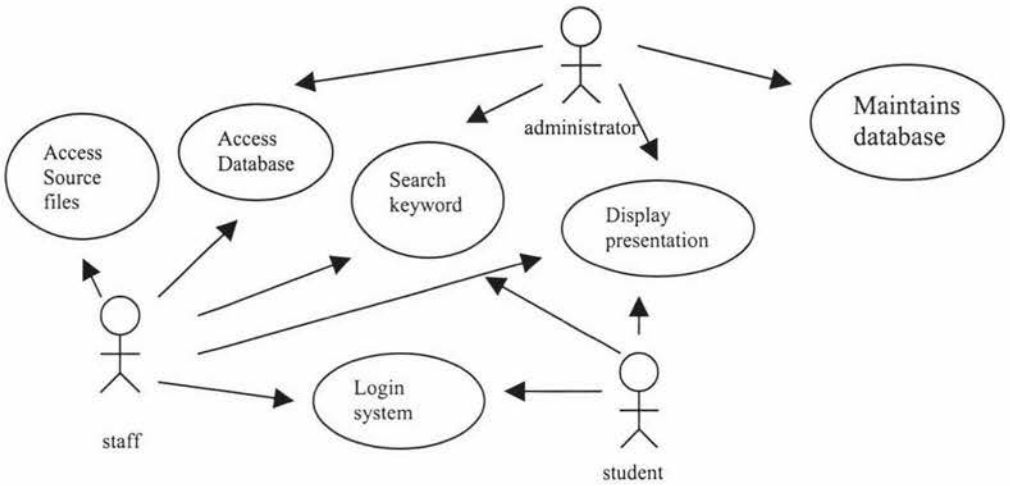

Figure 3.2 Use case diagram for the courseware system

3.3 System Schema Design

The design phase and the resulting UML model expand and detail the analysis model by taking into accounts all technical implications and restrictions. The purpose of the design is to specify a physical schema that can be easily mapped into the implementation phase.

3.3.1 Database design

Database design is the design of the database model that will support the operations and objectives of this courseware system. In the database design stage, the design process is divided into two main phases: logical database design and physical database design. The logical database design is the process of constructing a model of the information used in a project based on a special data model, but independent of a particular DBMS and other physical considerations. The physical database design is the process of producing a description of the implementation of the database on a secondary stage; it describes the storage structures and access methods used to gain access effectively.

In the logical database design, detail works, such as draw ER diagram with logical methodology, normalization, define integrity constraints and check for future growth, are included. The entities with their main attributes will be designed in this stage. From the conceptual ER model, which it is mentioned above, (See page 5 Figure 3.1) some entities should be add to reinforce relationships and map the local conceptual data model to local logical data model (See next page Figure 3.3).

3.3.2 User-Interface application design

The design is a technical expansion and adaptation of the analysis result. All the details of how things should work technically and the constraints of the implementation

environment are taken into consideration. The results of the analysis are carried over to the design and maintained in the center of the system.

The detailed design activity should include specification of all classes, including the necessary implementation attributes, their detailed interfaces, and descriptions of the operations for coding [8].

The design can be divided into two stages:

- *Architectural design:* This is the high-level design where the packages (subsystems) are defined, including the dependencies and primary communication mechanisms between the packages.
- *Detailed design:* This part details the contents in the packages. Dynamic models from the UML are used to demonstrate how objects of the classes behave in specific situations.

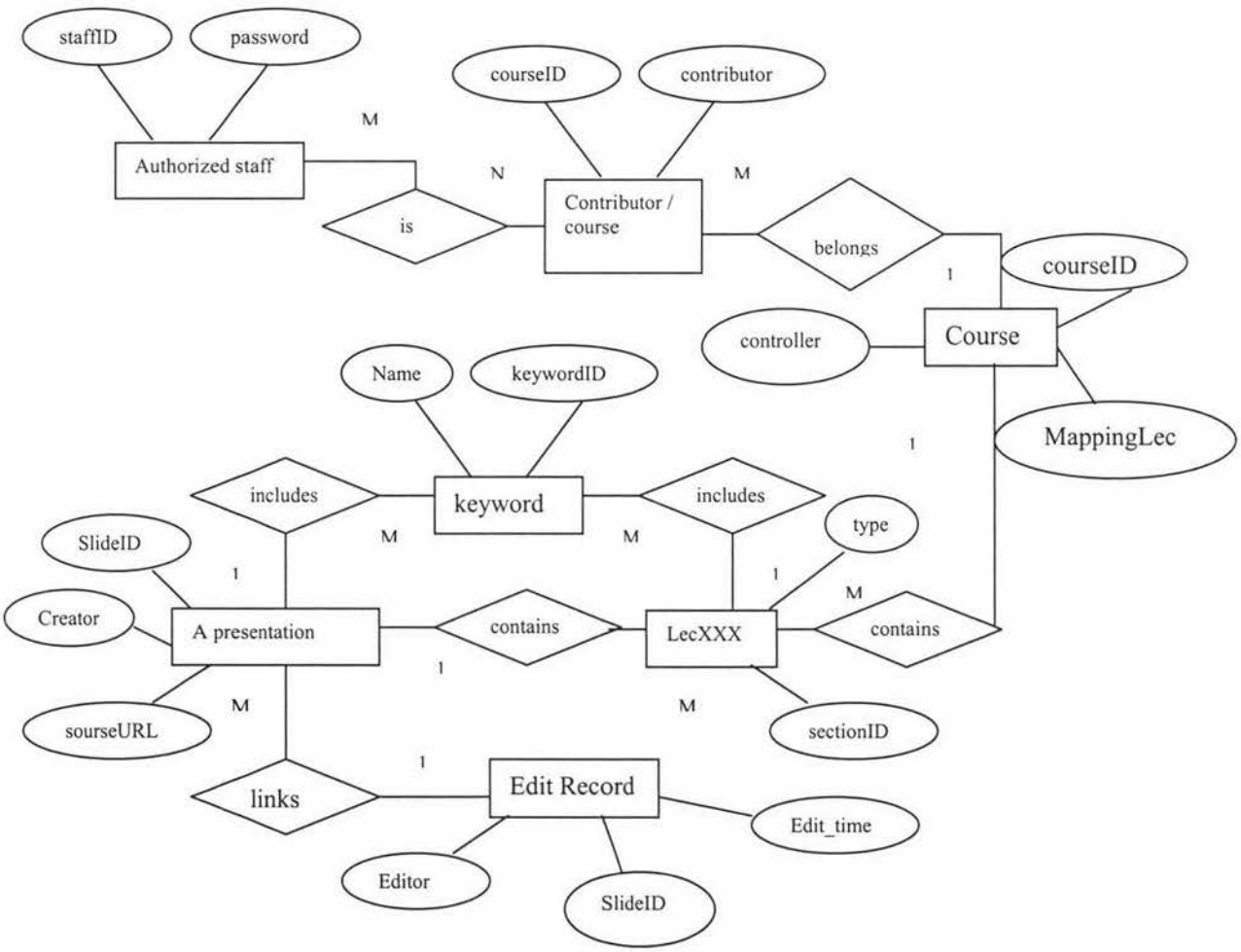

Figure 3.3 The Enhanced Entity-Relationship model

Notes: 1) the entity LecXXX is a particular course, which contains its content structure. The course entity uses LecXXX as the value of MappingLec, which is foreign key in Course table. Here, XXX stands for the lecture identification.

2) the controller of a course means the staff who is responsible for the course, while the contributor is the staff who is taking part in this course.

3) M means multiple numbers of possible relationships for each participating entity.
1 means one number of possible relationships for each participating entity.

Detailed Design

The purpose of the detailed design is to describe the classes in the user-interface and database packages and to expand and detail the descriptions of the utility packages, which are used by both of the interface package and the database package.

- User interface package: These classes present the services and information in the system to a user. A special activity carried out during the design phase is 'look and feel.' They should have functions such as information about this system, helping to choose a menu or accessing the system.
- Database package: These classes should have functions such as connecting to a server, accessing the data and closing the connection etc.
- Utility package: Because it is used in both the user-interface and database packages, these classes are embedded in both kinds of classes.

The class models can be seen in Figure 3.4 and 3.5, which are shown in the physical schema mapping section.

3.4 Physical Schema Mapping

This section presents the physical database design and the system schema mapping.

3.4.1 Physical database mapping

This is the process of producing a description of the implementation of the database on secondary storage; it describes the storage structures and access methods used to gain access effectively. It involves [9]:

- Designing the base relations and integrity constraints for the target DBMS;
- Selecting specific storage structures and access methods for the data to achieve good performance for the database activities;
- Designing any security measures required on the data.

This process requires intimate knowledge of the functionality offered by the target DBMS. The mSQL, which is used as the DBMS in this project, offers a significant subset of the features provided by ANSI SQL. It allows a program or use to store, manipulate and retrieve data in table structures. Although it does not support all the relational operations defined in the ANSI specification, such as the views and nested queries,

mSQL provides a significant subset of the ANSI SQL standard and is capable of supporting the vast majority of applications.

Some mSQL 2.0 SQL syntax, which it is relevant to this project, is summarized as below:

- *Index creation:* Indexes are created after the tables has been created, with separate CREATE INDEX statements. mSQL 2.0 provides much more sophisticated indexing support;
- *a unique identifier:* creates a SEQUENCE on a table and selects the _seq column;
- *the time for last modified:* uses the _timestamp column to get the time a column was last modified;
- *string comparisons:* performs in case-sensitive fashion with sorting in ASCII order; using CLIKE to do case-insensitive searching.
- *WHERE clauses:* evaluates everything from left to right. Some logical calculations with more than three arguments cannot be expressed in any way.
- *access control:* has a file mSQL.acl which can grant read/write privileges for users.
- *client connections:* on an average OS, mSQL 2.0 reconfigure itself to handle over 200 simultaneous clients.
- *Run-time configuration:* provides more flexibility. Both the server and any client application that talk to the server now included in a configuration file that is load all of configuration details at run-time.
- *Lite & W3-mSQL 2.0:* includes the Lite and W3-mSQL tools to aid in the development of applications. This tool can be used to develop sophisticated GUI based applications that are platform independent and available as shared resources on a network.

Each relation defined in the logical data model, using the DDL (Database Definition Language), can consist of:

- The name of the relation;
- A list of simple attributes in brackets;
- The primary key and, where appropriate, alternative keys and foreign keys;
- The integrity constraints for any foreign key identified.

Figure 3.4 is an example of the DDL for the Courses, Lec159703 and Slides:

```
# 1. Table structure for table 'Courses'
#
CREATE TABLE Courses (
  Course_id       CHAR(10)    NOT NULL,
  Course_name           CHAR(50),
  Controller            CHAR(30),
  Mapping_table         CHAR(10)
)

CREATE UNIQUE INDEX Course_idx ON Courses(Course_id)

# 2. Table structure for table 'Lec159703'
#
CREATE TABLE Lec159703(
  Section_id            CHAR(10)    NOT NULL,
  Subsec_id             CHAR(10),
  Type                  CHAR(10),
  Subsection_name CHAR(50)
)

# 3. Table structure for table 'Slides'
#
CREATE TABLE Slides (
  Slide_id        CHAR(10)    NOT NULL,
  Creator               CHAR(20),
  Slide_name            CHAR(50),
  Dura_time       Int,
  K_bytes         Int,
  Lecture_url           CHAR(100),
  Html_aep_url          CHAR(50)
  )

CREATE UNIQUE INDEX Slide_idx ON Slides(Slide_id)
```

Figure 3.4 The DDL for the tables of Courses, Lec159703 and Slides

3.4.2 Physical application mapping

Database package contains the classes about JDBC calls and how to use them.

First, it is the function of connecting to the server. This involves loading the driver and making the connection.

Second, they are the functions of creating JDBC statements, executing statements and retrieving values from result sets.

Lastly, it is the function of closing the connection.

The details will be discussed in the implement stage.

Interface package contains the classes about interacting with the users. They are based on the user cases, and have been divided into the following sections.

First, the main interface window has the functions:

- login to system,
- log-out of the system, and
- some information about what the system is and how to use it.

Second, the complete staff list for choosing a staff member and his relevant courses, the entire course list for choosing a course, and by default the course of login staff will be displayed on this window.

Third, the window gives the information of the course that the user chooses, such as the controller and contributors of the course and the contents of the course. With the latter, there are three choices: the presentations of the user with the course, all presentations with the course and some particular presentations of the staff, who is the contributor of the course.

Fourth, when the user chooses a presentation from the course contents, the browser will be launched on the screen to show the presentation, which uses the AudioGraph tool.

The resulting application user interface is composed of several windows with a hierarchy of the course-plan schema. The windows typically present the services of the system and are mapped to an initial use case. The windows are kept small, simple and so that the users can easily communicate with the computer.

The next page shows these class models (See next page, Figures3.5 and 3.6). They are the major classes involved in this project.

3.5 Summary

This chapter gives the details about the system analysis and design from two major parts: database server, user-interface application. The server side, focuses on the database creating process while the client side, is involved in three packages. The database package majors in the classes communicating with the server, the interface package majors in the classes communicating with the user while the utility package contains some mechanisms which provide the services for the system. The detail implementation will be discussed in the next chapter.

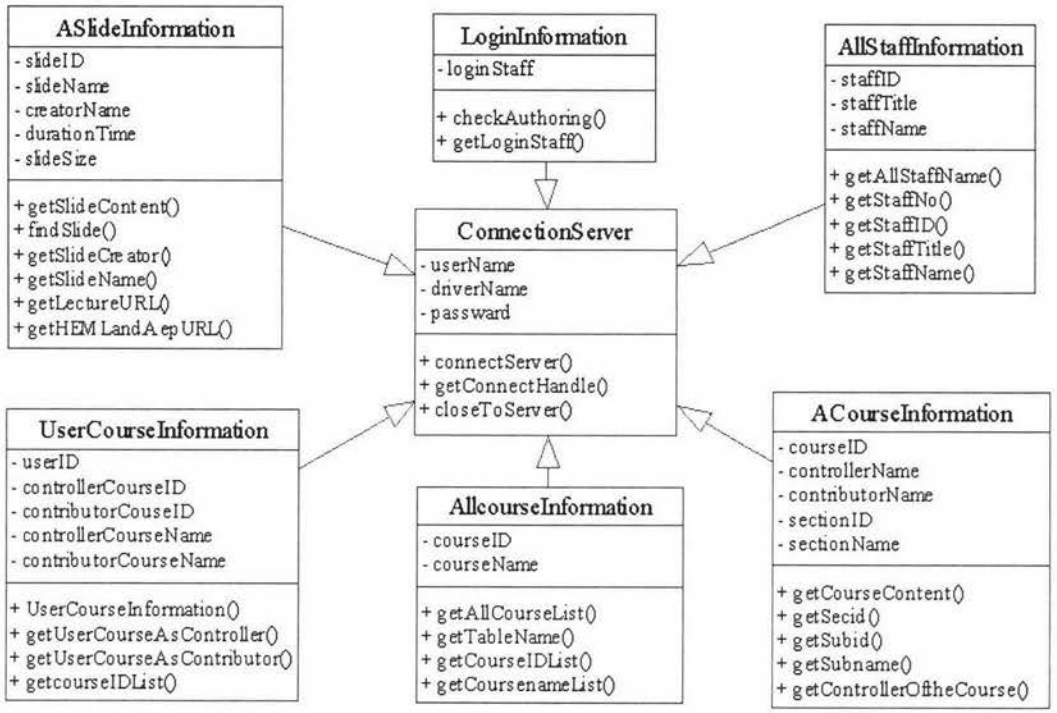

Figure 3.5 The database classes in the design model

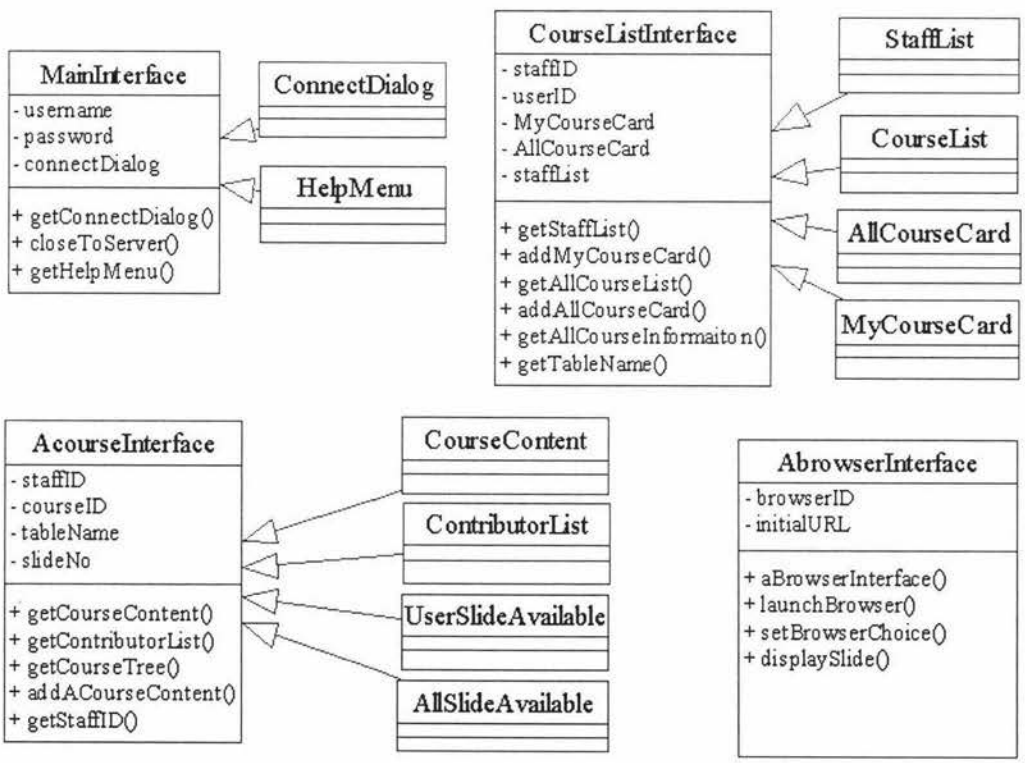

Figure 3.6 The interface classes in the design model

Chapter 4 Implementation

This chapter describes the implementation of the system and includes four sections. The first section discusses the methods employed on the server side. The second section discusses the methods employed on the client side. On the client side the relevant methods to implement the application system in Java are also discussed. The third section explains some tools used in the project, and the last section explores the problems that arose during the implementation and the solutions to these problems.

4.1 Methods Employed on the Server

This section describes the methods used in this project. The text is also separated into two parts; the server side and the client side. On the server side, the methods that have been employed in installing and configuring the database, and creating the tables are discussed.

The objective of the database implementation is to determine the file organization and access methods that will be used to store the base relations. That is the way in which relations and tuples will be held on secondary storage.

4.1.1 Installing mSQL 2.0 in the server system

The database used by this project, Mini SQL, is generally distributed in source code form to enable the widest possible use of the software. The software and the installation tools have been made as portable as possible. In general, the software will automatically configure itself to the capabilities of the operating system on which it is being compiled.

The following commands need to be used to complete the process of compiling and installing mSQL on UNIX platforms.

- make target: the process will create a new directory called targets in the top directory;
- ./setup: in targets directory, the compilation process can be configured;
- make all: compiles the software after the setup utility has been completed.
- make install: completes the installation.

A complete guide to the compilation, installation and configuration is included in the User Guide of the mSQL 2.0 documentation [32]. The important thing, in a normal installation, is to define certain values in an external run-time configuration file named msql.conf. In the msql.conf, using this project as an example, the following parameters were changed:

```
Inst_dir = /home/students/pjun/Hughes.

mSQL_User = pjun
Admin_User= pjun
Read_only = false
Remote_Access = True;
```

/home/students/pjun/Hughes is the install directory in this project and pjun is the username of the person who has the privilege of creating a database, and shutting down the server, etc.

4.1.2 Using the standard programs and utilities

The mSQL distribution includes the mSQL server, client programs, a C programming interface for client software, and several other tools. The administration program is msqladmin, which is used to perform administrative operations on an SQL database server. As mentioned in chapter 2, the standard client programs, which are used to set up the DBMS, are msql, msqldump and relshow.

Only the DBA (Database Administrator) can add or drop a database from mSQL. The DBA is Admin_User and is defined in the first step. To create or drop a new database, both actions are done with the msqladmin command on UNIX:

```
% msqladmin create dbproj
% msqladmin drop dbproj
```

When the new database, named 'dbproj' in this project, is created, it should add an entry in the file msql.acl, which defines access type restrictions to databases. This file should be set up with a 'default' access specification for the database (e.g., anyone can read, only root can write, access is permitted from any internet host, etc.).

The msqladmin command can also be used to shut down the server or cause it to reload the access control file:

```
% msqladmin shutdown
% msqladmin reload
```

The msql program, which acts as the mSQL terminal monitor is an interactive interface to the mSQL server. It can be used to submit SQL commands directly to the server. Any valid mSQL syntax can be entered at the prompt provided by the mSQL monitor. The mSQL monitor can also be used as a mechanism for creating the database tables and for submitting ad-hoc SQL queries to the server.

The msqldump produces an ASCII text file containing valid SQL commands that will recreate the table or database dumped when piped through the mSQL monitor program. The output will include all CREATE TABLE commands required to recreate the indices, and INSERT commands to populate the tables with the data currently contained in the tables. If sequences are defined on any of the tables being dumped, a CREATE SEQUENCE command will be generated to ensure the sequence is reset to its current value.

The relshow is a schema viewer and is used to display the structure of the contents of mSQL databases. If no arguments are given, relshow will list the names of the databases

currently defined. If a database name is given, it will list the tables defined in that database. If a table name is also given, then it will display the structure of the table (i.e. field names, types, lengths etc).

If an index name is provided along with the database and table names, relshow will display the structure of the specified index, including the type of index and the fields that comprise the index.

4.1.3 Turning the Entity Relational Model into a relational Database

A relational database is a set of tables containing data fitted into predefined categories. Each table, which is sometimes called a relation, contains one or more data categories in columns. Each row contains a unique instance of data for the categories defined by the columns.

There are two methods of defining or changing the database schema. One is using the msql client to issue SQL statements to the server to define or modify the schema. However, using this method, it is easy to make mistakes when typing. The other way is to enter all of the mSQL commands, which define the database, into a file (it is called proj-schema.msql in this project) and then use file redirection to call mSQL with the database name as input, for example:

```
% msql dbproj < proj-schema.msql
```

In proj-schema.msql of the project, the slide schema is defined as follows:

```
# This is the schema for dbproj database system
# Host: localhost    Database: dbproj
# Author: June Pan and Chris Jesshope  Last date:  21/05/2000
#--------------------------------------------------------------
#
# 1. Table structure for table 'Slides' (presentations)

DROP TABLE Slides \g       # in case this is a old one

CREATE TABLE Slides (
  Slide_id        CHAR(10)    NOT NULL,
  Creator         CHAR(20),
  Slide_name      CHAR(50),
  Dura_time       Int,
  K_bytes         Int,
  Lecture_url     CHAR(100),
  Html_aep_url    CHAR(50)
  ) \g                    # \g means go (Send query to database)

CREATE UNIQUE INDEX Slide_idx ON Slides(Slide_id)\g
```

Figure 4.1 The slide schema for the database system

A complete definition of the schema for the database will be found in Appendix One.

After defining the database schema, it is necessary to insert data into the database to test the display system. As with defining the database schema, there are two ways to enter data to each table. One is entering data with a series of INSERT commands. mSQL does support flexibility in the insert command. However, again, there is another way that is probably more convenient. That way is to enter all of the data in a batch mode by typing the mSQL INPUT statements into a file (it is called proj-data.msql in this project) and using it as input to the msql client:

```
% msql dbproj < proj-data.msql
```

In proj-data.msql of the project, the data of the slide table is entered like following:

```
# This is the data of the slide for dbproj database system
# Host: localhost    Database: dbproj
# Author: June Pan and Chris Jesshope  Last date:  21/05/2000
#--------------------------------------------------------------

#
# 1.insert data to table 'Slides'
#

DELETE FROM Slides\g
INSERT INTO Slides VALUES ('P11000','crjessho', 'The problem',
104, 253, 'http://www-
ist.massey.ac.nz/~crjessho/comp_arch/audiograph/Introduction.%C4/
Intro001.htm', 'http://html.aep' )\g
INSERT INTO Slides VALUES ('P12000','kdgrant', 'Background', 233,
556, 'http://www-ist.massey.ac.nz/~crjessho/comp_arch',
'http://html.aep' )\g
INSERT INTO Slides VALUES ('P13000','crjessho', 'Cache(1)', 233,
556, 'http://www-ist.massey.ac.nz/~crjessho/comp_arch',
'http://html.aep' )\g
```

Figure 4.2 The data of the slide table for the database system

A complete file of inserting data to the database will be found in Appendix Two.

Now using the command, `msqldump` that is described above, the administrator can submit the batch of queries and can use the file redirection strategy again, that is, just putting the queries in a file (it is called proj-query.msql in this project) and feeding the file to msql:

```
% msql dbproj < proj-query.msql
```

msqldump client can be used to generate a set of SQL commands to recreate a database:

```
% msqldump dbproj > proj-dump.msql
```

The following is an example of the proj-dump.msql to save a part of the database:

```
# This is the dumping for dbproj database system
# Host: localhost    Database: dbproj
# Author: June Pan and Chris Jesshope  Last date:  21/05/2000
#--------------------------------------------------------

#
# Table structure for table 'Slide_Keyword'
#
CREATE TABLE Slide_Keyword (
  Slide_id CHAR(10) NOT NULL,
  Keyword CHAR(20) NOT NULL
) \g

#
# Dumping data for table 'Slide_Keyword'
#

INSERT INTO Slide_Keyword  VALUES ('p1235','architecture')\g
INSERT INTO Slide_Keyword  VALUES ('p1236','Cache')\g
INSERT INTO Slide_Keyword  VALUES ('p1237','Introductory')\g
```

Figure 4.3 The dumping data of the Slide_Keyword to the database

A complete file of dumping data to the database will be found in Appendix Three.

It is useful to have a simple script like makedbproj to define, populate and dump a database. In this case, by using just one command, a database can be rebuilt:

```
% makedbproj
```

In makedbproj file, there are some commands as follows:

```
#! /bin/csh -f

## (re) defines the dbproj database in msql by (re)loading the
schema and the data
## and also dumping out the result in a standard form

msql dbproj < proj-schema.msql
msql dbproj < proj-data.msql
msqldump dbproj > proj-dump.msql
```

4.2 Methods Employed on the Client

The implementation phase on the client side requires the definition and implementation of a number of classes to access the data. The requirements specify that the system is able to run on a number of different processors and operating systems, so Java was chosen to implement the application on the client side. In Java, there is a one-to-one mapping from the class to the code file, which makes mapping the logical classes to be coded very easy. Java also specifies that the name of the file is the same as that of the class it contains.

It should be mentioned that Java package names are constructed in a hierarchical way, using a dot-separated naming convention. Package-name components therefore, construct

a unique path for the compiler and run-time systems to locate files. A package is a name for a group of related classes that share a visibility effect that the package has on access to variables and methods between classes. A class is declared to belong to a particular package with the package statement, which must appear as the first statement in a class. Classes within a package can refer to each other by their simple names. However, to locate a class in another package, Java supplies a qualifier, which is the import statement. One or more import statements can appear at the top of a compilation unit, beneath the package statement. The import statements list the full names of classes to be used within the file. Like a package statement, import statements apply to the entire compilation unit [22,23].

As described in the design stage, there are three packages on the client side. The methods developed for each of these packages are discussed below.

4.2.1 Database package implementation

The first function required is to connect the client to the server. An application uses JDBC to talk to one or more databases without having to know any of the details concerning the driver implementation for that database. When writing a Java database application, the only driver-specific information JDBC requires is the database URL(. Once a Driver recognizes the URL, it creates a database connection using the user name and password, which are the same as the administer identifiers configured in the database server. It then provides the DriverManager with a java.sql.Connection implementation representing that database connection. The DriverManager then passes that Connection object back to the application. With the project coding, the entire database connection process is handled by this one line of code:

```
Connection con = DriverManager.getConnection(url, uid, password);
```

As can be seen from the explanation above, JDBC uses one class (java.sql.DriverManager) and two interfaces (java.sql.Driver and java.sql.Connection) for connecting to a database.

The second function required is to prepare an SQL statement. After the application is connected to the database, it can begin making updates and queries. The most basic kinds of database accesses are updates (INSERT, UPDATE, or DELETE) or queries (SELECT). It is becoming important to know what type of SQL is sent to a database, because JDBC uses a different method for sending queries than for sending updates. The key difference is that the method for queries returns an instance of java.sql.ResultSet while the method for non-queries returns an integer. A ResultSet provides access to the data retrieved by a query.

Two basic JDBC Database Access Classes need to be explained in detail here:

java.sql.Statement: the Statement class is the most basic JDBC class representing SQL statements. It performs all of the basic SQL statements that have been mentioned so far.

java.sql.ResultSet: A ResultSet is a row of data returned by a database query. The ResultSet can manipulate the tables in a relational database. The class simply provides a series of methods for retrieving columns from the results of a database query. Because the ResultSet class handles only a single row from the database at any given time, the class provides the next() method for making a reference to the next row of a result set.

Making a database call requires nothing more complex than creating a Statement and passing via one of its execute methods. Unlike executeQuery(), however, executeUpdate() does not return a ResultSet (you should not be expecting any results). Instead, it returns the number of rows affected by the UPDATE, INSERT, or DELETE.

By default, JDBC commits each SQL statement as it is sent to the database; this is called *autocommit*. However, for more robust error handling, a Connection object can be set up, so it issues a series of changes that have no effect on the database until a commit is expressly sent.

The last function is closing the connection. Many of the objects are closed through a close() method. A given JDBC implementation may or may not be required to close these objects before reusing them. However, it is always a good idea to close the connections, as, while open, they may be holding precious database resources.

4.2.2 Utility package implementation

Two classes are involved: ExitListener and WindowUtilities. ExitListener is a listener that generally extends WindowAdapter and that we attach to the top-level Frame or Jframe of our application, so quitting the frame exits the application. WindowUtilities include a few utilities that simplify using Windows in Swing.

An Adapter provides trivial implementations of its corresponding Listener's methods. Generally, we should extend the Adapter class (if available) and override any methods we wish to handle.

4.2.3 Interface package implementation

The Java Application Programming Interface is the collection of classes that comes with every Java implementation. It encompasses all the public methods and variables in the classes that comprise the eight Core Java Packages listed in the table below [26]:

Package	Contents
Java.lang	Basic language classes
Java.io	Input and output
Java.util	Utilities and collections classes
Java.net	Sockets and URLs
Java.applet	The applet API
Java.awt	The Abstract Windowing Toolkit
Java.awt.image	AWT image classes

Table 4.1 The Core Java Package

When the design of the client interface has been satisfied, it must be converted into Java code. Unfortunately, because of the lack of a form designer in the JDK to generate code templates, everything must be written down in code. In particular, code is needed to:

- Make the components in the user interface look the way the user wants them to;
- Position the user interface components where the user wants them to be inside a window;
- Handle user input; in particular, program the components to recognize the events to which the user wants them to respond.

The Abstract Windowing Toolkit is the core of Java's windowing functionality. It is the package that provides a large collection of classes for building a GUI in Java. With AWT, we can create windows, draw, work with images, and use components like buttons, scrollbars, and pull-down menus. The classes that comprise the AWT are part of the java.awt package.

This project imported a number of classes in the java.awt package for designing user front-ends. These classes are introduced briefly below, before the implementation of the interface is discussed in detail.

Component and Container

A component is the fundamental user-interface object in Java. Everything you see on the display in a Java application is an AWT component. This includes things like windows, drawing canvases, buttons, checkboxes, scrollbars, lists, menus, and text fields. To be used, a component is normally placed in a container. Containers group components, arrange them for display, and sometimes associate them with a particular display device. The add() method of the Container adds a component to the container. A container has the same methods as other components, including the graphical and event-handling methods.

Layout manager

A layout manager is an object that controls the placement and sizing of components within the display area of a container. A layout manager is like a window manager in a display system; it controls where the components go and how big they are. Particular kinds of Layouts, such as Flow Layout, Card Layout and GridBagLayout, are involved in this project.

The CardLayout is not like the other layout managers, and is more appropriate for managing panels. It is, like Tabbed Dialog Boxes, a convenient way of organizing a lot of data on the screen without cluttering up the window. The CardLayout is a mechanism that allows us to stack up panels and display them as and when we need them (See below).

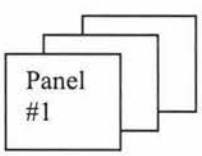

Figure 4.4 CardLayout

The GridBagLayout is not only one of the most flexible layouts, but also one of the most complicated. We will discuss it further in Section 4.4.

Button and event

The windows have been drawn in a visual environment, where components such as buttons and list fields can be visually and interactively incorporated into a window. The environment then generates the necessary code for creating a window with that appearance, including the addition of attributes that map to the components in the window.

It is also easy to add event handlers for user-generated events such as mouse clicks and menu choices, by selecting a specific component such as the okButton and then selecting an event on that button that needs to be handled, such as the Clicked event. The environment then generates an okButton_Clicked method that will be called when the OK button is clicked. An event that itself was generated by code is added to a standard handle Event method.

Choice and List

There are also many occasions where users should be given a finite set of choices. The List component is one such kind of choice component that takes up screen space and makes obvious to the user what can be chosen from it. A list component can also allow the user to make multiple selections, which can be any combination of the strings in the box.

In this project, List component is one of the most often employed components, such as StaffList, CourseList, or ContributorList. We also add the ActionListener event on List component to react to the user's DoubleClick operation. For a detailed explanation of the event, see below.

JDK 1.1 Event Hierarchy

After having a taste of how event handling works above, the next step is to turn to a more general discussion of event handling in Java 1.1. As a class of Java, event handling in Java is object-oriented, with all events descending from a common superclass, which is the EventObject in the java.util package. The action() method is an example of event-handing routines which cause an event to be sent whenever they are used.

When a source object needs to tell a listener object that an event has happened, the AWT:

- Calls the appropriate methods of the listener interface
- Passes it an object that descends from EventObject

This event object encapsulates the event and the event action happens automatically once the listener is registered with the source. When necessary, the event can be used to get methods to analyze the objects of type that were passed to the listener object. Some event types, such as the ActionEvent, are actually passed to listeners in this project.

The project also uses some classes that are part of the Swing GUI components for package javax.swing. These are the newest GUI components of the Java 2 platform, but

separate swing packages can be added to Java 1.1. "Swing" refers to the new library of GUI controls that replace the somewhat weak and inflexible AWT controls. There is an almost equivalent Swing component for most AWT components.

'Busy Cursor' for custom control

The problem with accessing databases in this project is that sometimes a query may take a long time to complete. This, in itself, is not a problem, but feedback to the user is always a comforting feature. In this case, we decided to implement a very simple progress bar as a custom control. The more detailed explanation of this implementation will be seen in Chapter 6.

String

Because strings are used so extensively throughout Java, the Java String class has quite a bit of functionality. Strings are immutable; once a string object is created, its value cannot be changed. Operations that would otherwise change the characters or the length of a string instead return a new String object that copies the needed parts of the original. Because of this feature, strings can be safely shared. This project, getting the benefits of the sharing safely, creates username and password as instances of the String class.

Vector

A Vector is a dynamic array and can grow to accommodate new items. Compared to Array, it can be easier to insert or remove elements at arbitrary positions within it. As with other mutable objects in Java, Vector is thread-safe. The Vector class works directly with the type Object, so it can be used with instances of any kind of class and even put different kinds of Objects in a Vector together. The project also uses the Vector array as the list class. Vector is also discussed in Chapter 6 for the performance issues.

Complex JTree

JTree is a class in Swing that was developed by Marty Hall [30] and the principle of it was introduced for constructing hierarchy of the Course Content in this project. The simplest and most common way to use JTree is to create objects of type DefaultMutableTreeNode to act as the nodes of the tree. The nodes that have no children will be displayed as leaves. It supplies a value, known as the 'user object', to the DefaultMutableTreeNode constructor, to act as the value at each node. The toString method of that user object is what is displayed for each node.

Once we have some nodes, we hook them together in a tree structure via parentNode.add(childNode). Finally, we pass the node to the JTree constructor. Note that, since trees can change size based upon user input (expanding and collapsing nodes), trees are usually placed inside a JScrollPane.

For more complicated trees like our project, it is more tedious and difficult to maintain if we just hook everything together 'by hand'. So we first make a simple tree-like data structure, then build nodes and hook them together automatically from that data structure. Our project, for example, uses nested arrays to define the data structure. For detailed explanation of JTree, please see CourseContent class of the application.

Exceptions and Error Classes

Dealing with errors in a language like C is the responsibility of the programmer. Java offers an elegant solution to the problems with exception handling. The Exception object is created by the code at the point where the error condition arises and is passed, along with the flow of control, to the handling block of code. This is where the terms 'throw' and 'catch' come from. The Exception object is thrown from one point in the code and caught by the other. The try/catch guarding statements wrap a block of code and catch designated types of exceptions that occur within it. For example:

```
try{
        con.close();
        dispose();
      }
      catch( Exception e ) {
          e.printStackTrace();
 }
```

The project also employs some of the classes in java.lang, java.util and java.net from the core language constructs.

4.3 Tools Involved in the Project

To be executed, Java programs normally go through several phases. This section discusses the tools that are needed to create, compile and run Java applications in this project. The client Java program of this project was developed under the Metrowerks CodeWarrior environment. This is a Java integrated development environment (IDEs), which includes the Java Developer's Kit 1.1x within the Windows NT and the Macintosh environments.

4.3.1 Create and save projects

The CodeWarrior tool can create a project environment and choose Java stationery. Project stationery files are pre-configured project files that include the basic Classes and Source files, which require creating a specific type of project. Creating a new project from stationery is an easy way to get a new project up and running quickly.

The CodeWarrior IDE automatically updates and saves project information whenever a project is closed, its preferences or target settings are changed, the files are added or deleted, any file is compiled, groups are edited, object code is removed, or the CodeWarrior IDE is quit.

4.3.2 Add source files and library class

To compile a project correctly, it needs certain components such as source code files, package files, and library files. Some build targets require additional files in order to compile properly. When the files are added, the locations of these files are automatically recorded in the Access Paths of the Target Setting panel that will be discussed later.

Java Applications are executables that usually run through a Java Interpreter. Java Applications can consist of multiple class files or a single Jar file (Java archive), which is bundled by multiple class files and the associated resources that can be easily downloaded, e-mailed, or stored in one convenient package. One of these classes must define a method main(), which is where the program starts running.

A Java Library is a collection of Java classes that can be used in many projects. Libraries are usually compiled into Jar files or Zip files (like classes.zip). The Zip file is similar to the Jar file in that it is a set of files compressed into a single file.

Java source files should be named using the .java extension. A source file is edited using the text edit tool and also needs to be added to the Sources type of the project. The application program involved some software packages to import some library classes, which are inherited by some subclasses in the source files. These software packages, such as swingall.jar or collections.jar, are added to the Classes type of the project.

4.3.3 Using Target Settings panel

The Target Settings panel is the most critical panel in CodeWarrior. This is the panel where the operating system and/or microprocessor will be picked for the project.

The Target Settings panel allows the name of the target to be set, as well as the target type (application or applet) and Main Class (the Class with main() method) in a project, etc.

In selecting a linker, it is necessary to specify the target operating system and/or chip; then the other panels available in this dialog box will be changed to reflect the choice. Because the linker choice affects the visibility of other related panels, it must be set to which linkers the project will use before specifying other target-specific options like compiler and linker settings. This project uses the Java Linker option.

4.3.4 Java Virtual Machine

The Java Interpreter is called the Java Virtual Machine. It is the component of the technology responsible for its hardware- and operating system- independence, the small size of its compiled code, and Java's ability to protect users from malicious programs. Figure 4.5 shows the location of Java Virtual Machine in a system.

The Java virtual machine is an abstract computing machine. Like a real computing machine, it has an instruction set and manipulates various memory areas at run time. It is reasonably common to implement a programming language using virtual machine. The Java virtual machine knows nothing of the Java programming language, only of a particular binary format: the class file format. A class file contains Java virtual machine instructions (or byte-codes) that are the platform independent codes interpreted by the Java interpreter, and a symbol table, as well as other additional information [33]. It is the Java byte-codes that make Java so popular as a program, and so portable.

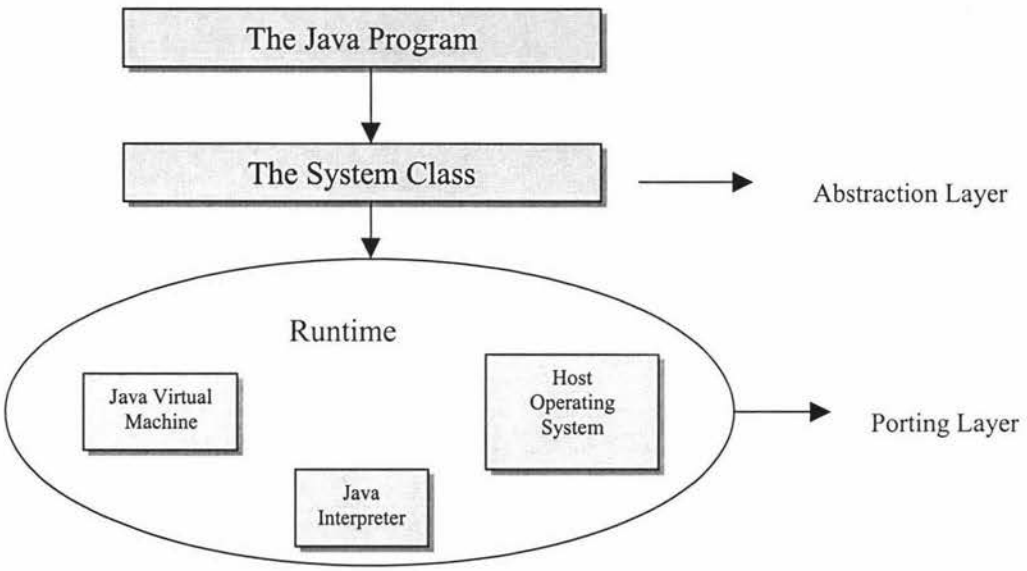

Figure 4.5 The location of Java Virtual Machine in a system

4.4 Problems Explored and Solutions

This section explores some important issues raised during the implementation phase and reasonable solutions for these problems. These problems included the proper access to data on the client/server architecture, writing the interface layout components and the handling of the program with a platform-dependent method. Now these issues will be discussed step by step.

4.4.1 Client/Server development

Client/server is an application architecture that divides processing among two or more processes, often on two or more machines. Any database application is a client/server application that handles data storage and retrieval in the database process and data manipulation and presentation somewhere else.

The architecture of a system depends on the application. Figure 4.6 shows how the project system gives the users access to centralized data. In this project, the user working in Windows NT or Macintosh retrieves data stored in mSQL at Unix, which is the server.

For a client/server system, security is one of the most important facts to consider. This project involved two kinds of security: the database security and Java security.

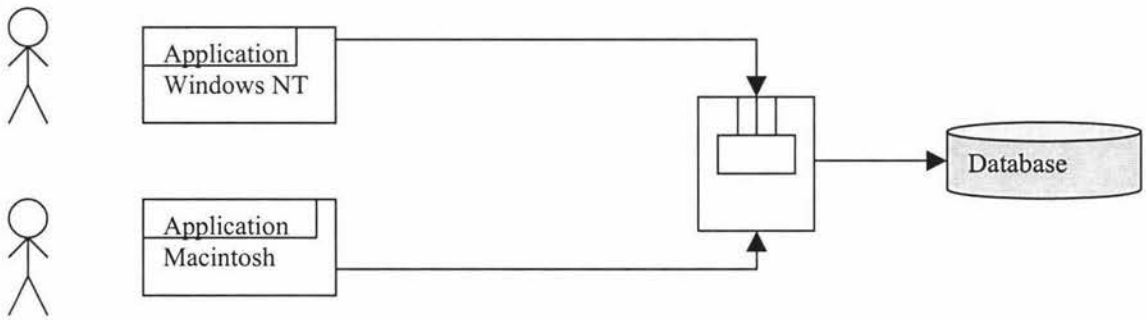

Figure 4.6 The diagram of the client/server system

Database access can be categorized into three main operations: querying, insertion and updating. Querying only requires the system to send information, and generally does not like insertion and updating, which affect the state of the database. Inserting is a non-destructive action and can be implemented in a similar manner to querying with the CGI interface and replying with the confirmation of the insertion, which will be discussed later.

Database transaction processing includes two highly interrelated components - concurrency control and recovery management.

Concurrent control is used by database systems to synchronize the operations of concurrent transactions and to ensure correct operations by guarding against mutual interface. The objective is to allow concurrent transactions to perform interleaving operations on shared database objects as if they are executed serially against the database. The DBMS solves these problems by using locking to avoid conflicts during the insertion or updating.

Recovery management allows a transaction to roll back changes in the case of concurrent conflict and also to redo changes in the case of system recovery. With the write ahead log protocol, the before images of the database object are recorded in the recovery log prior to the database object being updated in place. Furthermore, before a transaction is committed, its after images are recorded in the recovery log.

JDBC defaults all Connection objects to auto committing database transactions. The object locking is a mechanism to solve another side issue that can be encountered with persistent objects in a multi-user environment; that is, multiple people might try editing the same persistent object at the same time.

The project also considers the output which users get, the input which users enter, and the dialogues through which users and the system talk to each other. The layout of a screen and the consistency of the dialogue will be important to those users who have to spend a much time sitting in front of terminals, and who will base their evaluation of the whole system on these interfaces.

Access control lets the system determine who can access the server. Two attributes can be used for controlling access:

* *User-Group* requires users to enter a username and password before accessing the server.

* *Host-IP* requires the user to access the Compass Server from a specific computer, where the server recognizes the computer by either its hostname or its IP address.

User-Group Authentication is used in this project. The system requires the user's username and password for authenticating before getting access to the server. Figure 4.7 shows the authentication window with the message that the user typed.

Java API methods can also be used to incorporate security functionality into the programs, including cryptography services and security checks. The API framework enables the definition and integration of individual access permission (controlling access to specific resources), cryptography service implementations, security manager implementations, and policy implementation. In addition, classes are provided for public/private key pairs and public key certificates from authoring people. The details will be discussed further in chapter 6.

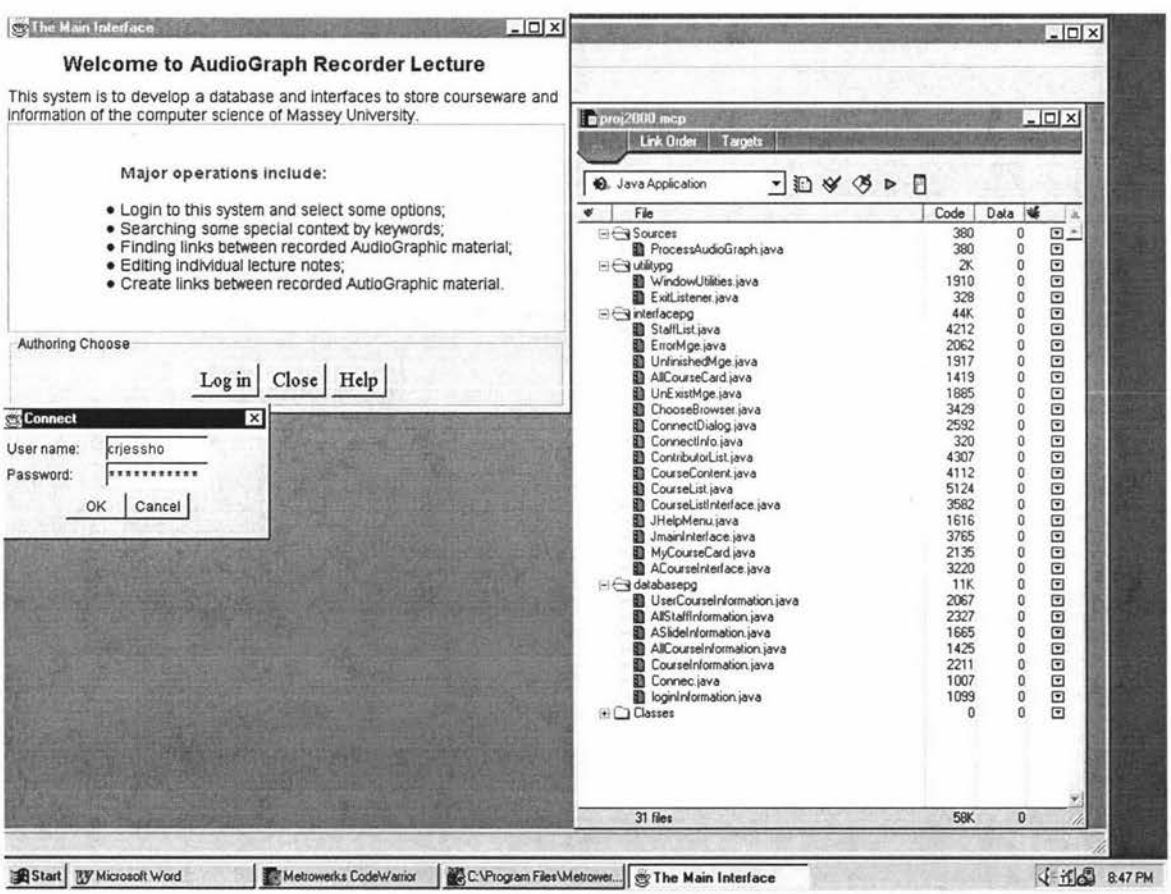

Figure 4.7 The screen shot of the Authentication Window

4.4.2 *Graphical User Interface Components*

A graphical user interface (GUI) presents a pictorial interface of a program. A GUI gives a program a distinctive 'look' and 'feel'. By providing different applications with a consistent set of intuitive user interface components, GUIs allow the user to spend less time trying to remember which keystroke sequences do what and spend more time using the program in a productive manner.

To build a user interface, obviously, one needs to decide how the interface should look. In particular, what components are needed and how should they appear? When the design of the client interface has been satisfied, it needs to be converted to Java code.

As mentioned above, because the JDK lacks a form designer in to generate code templates, everything should be written down in code. That means the design is often easiest done with old-fashioned paper and pencil in Java, since the JDK has no form designer like those in VB or Delphi.

AWT is a most seriously flawed tool in the Java package. It is very different from Sun announcing, as some believed that it absolutely allows cross-platform portability. In fact, it lacks some of the features other GUI environments provide. At least, from this project point of view, JDK 1.1.8 does. The next chapter discusses further pitfalls when writing multi-platform Java code.

The difference in GUI programming is most likely the greatest hurdle when programming for cross-platform portability, but there are still other issues to consider. Building a sophisticated graphical Java application is still required to do manual, error-prone and tedious coding, to manage every little aspect of every graphical object in the application, to worry about painting every line and even has to be bothered with the layout of a button [36].

For example, a layout manager should be used to layout an application screen and dialog. However, the basic layout managers, such as FlowLayout, BorderLayout have not given enough control to the layout. This project was aware of this problem and took the time to familiarize with the java.awt.GridBagLayout layout manager. It provides great flexibility and control over the layout and does so in a cross-platform manner; however, the layout manager takes a little more time to understand and master than the others.

Fortunately, there are some stronger GUI components supplying the original Java AWT, such as Java Swing, as mentioned above, which are the newest GUI components of the Java 2 (JDK 1.2) platform. Swing can also be imported to JDK 1.1 to create a more flexible interface. However, it cannot completely replace original Java AWT in JDK1.1.

The reason is that Swing is about as far from language neutral as possible. It must take the Java language and the other Java libraries with Swing, generally abandoning the perfectly good existing library [37]. So, there are still some different appearances and even different user interactions between Windows NT and Macintosh in this project.

4.4.3 Runtime.exec() invoking

The standard Java method Runtime.exec() launches new processes on the machine on which it is running. It can be used to pass a command line, either as a String of space-delimited arguments or as an array of String arguments in Windows NT. The project, for instance, uses it to open a URL in a Web browser. Runtime.exec() has been found in a platform-dependent part of the Java API set. It assumes the existence of a command-line interface to the OS and the ability to launch arbitrary applications that can accept arbitrary parameters. The Mac OS is at a disadvantage here because, unlike Unix or Windows, it is not a thin GUI veneer over a command-line-base OS. There is no such thing as a command line in the Mac OS itself; therefore, it is quite different for MRJ to support these kinds of uses. Now MRJ 2.1 supports a lot more of the typical uses of Runtime.exec() than its predecessors did [34]. This will be explored in detail in next chapter.

This project imports the MRJ 2.1 as the support to open a URL in a browser. First, it attempts to use openURL function of MRJ, and upon failure will revert to presenting a file dialog for the user to choose a browser to use, then launch the browser and open the specified URL. The detailed code can be seen at the ExceLaunchMc class of Application Classes' Source Files in the Appendices.

4.5 Summary

This chapter is concerned with the implementation of the system. It involves four parts. The first part illustrated installing mSQL, using the standard programs and utilities to set up the DBMS and then turning the entity relational model into a relational database on the server side. The second part explained the three packages that were implemented in the client side. The Database package contains the main classes and interfaces of JDBC and there is a more detailed explanation in the Appendices. The utility package implemented the Classes: ExitListen and WindowUtilities, which were borrowed from Swing. The interface package is comprised mostly of the Java AWT components that are the basic front-end mechanisms available to this project. The third part contains the tools that were involved in the project. It explores the work procedure under the Metrowerks CodeWarrior 4 environment, which also included JDK 1.1.8 and the Java interpreter, JVM. The last part involved in the problems that should be considered during implementing both sides of this system: the database security and Java API. The AWT package's sole responsibility is to provide all the interface and window controls to the developer. With Java being truly cross-platform, it can become an administrative nightmare to design and code a user interface that not only looks good, but also is consistent across the many different systems it may run. The next chapter will explore in more detail issues of the cross-platform in Java.

Chapter 5 Issues of Cross-platform Compatibility

After implementation, the application has to be tested. For this project, the original use cases [see Chapter 3 section 3.2] were tried in the finished application to determine whether they were supported by the application and could be performed as defined in the use-case descriptions. We used *walking the use case,* which is where different people in the model group role play the actors using the system in a specific use case. Normally, there are two ways popular to test, that is: testing an entire system if the system is small enough; and, testing a unit of each routine before a large system is integrated into the overall system. This project has adapted the technique above to test the entire system.

The project was tested on Windows NT (under CodeWarrior IDE 4.0) and Macintosh (under CodeWarrior IDE 3.2). The data was set up on the Unix. The results of the test are the project presentations that will be described in detail in Chapter 6. The application coding and the tested data have been discussed in Chapter 4 and the origin files can be found in the Appendices. The results of the test show that we have mostly achieved our objectives that are the requirements of the system at this stage. However, the system needs to supplement some functions in the next stage, which will be discussed in Chapter 6 as well.

This chapter discusses some of the cross-platform issues of Java, which was the major problem when the application was tested. The first section describes how to write multi-platform Java code. The second section explores cross-platform evolution. The last section gives some potential solutions to these problems.

5.1 Writing Multi-platform Java Code

The concept of producing a multi-platform application is to write a project from scratch, without having to port or redevelop the project later for another platform, but to create executable applications for all platforms at the same time. There are three routes one can take to create multi-platform software. The first is to write for a specific platform with porting in mind. The second is to use multi-platform libraries, and the third is to use a multi-platform language, such as Java, which should involve the least cost because it has only one source code to develop and maintain [35].

5.5.1 Architecture of Java's support for Platform Independence

Support for platform independence, like support for security and network-mobility, is spread throughout the architecture of Java. All the components of the architecture: the Java Platform, the language, and the class file play a role in enabling platform independence [37]. The following discusses the three parts of the architecture of Java in detail.

- *The Java Platform*

The architecture of Java supports the platform independence of Java programs in several ways, but primarily through the Java Platform itself. The Java platform acts as a buffer between a running Java program and the underlying hardware and operating system. The Java platform has two components: the Java Virtual Machine that runs the program, and Java API that gives the program access to the underlying computer resources. No matter where a Java program goes, it only needs to interact with the Java Platform. It need not worry about the underlying hardware and operating system. As a result, it should be able to be run on any computer that hosts a Java Platform.

- *The Java Language*

The Java programming language reflects the platform independence of Java in one principal way: the language defines the ranges and behaviors of its primitive types. In languages such as C or C++, the range of the primitive types is determined by its size, and its size is usually determined by the target platform's word size. By guaranteeing that primitive types behave the same on all platforms, the Java language itself should promote the platform independence of Java programs.

- *The Java Class File*

As mentioned in the first chapter, the Java class file defines a binary format that is specific to the Java virtual machine. Java class files can be generated on any platform. They can be loaded and run by a Java virtual machine that sits on top of any platform. Their format, including the big-endian order of multi-byte values, is strictly defined and independent of any platform that hosts a Java virtual machine.

Each Java program goes through two steps before it is applied. First, it is compiled, which happens only once. Compilation translates the Java program into an intermediate language called Java bytecodes, which are the platform-independent codes interpreted by the Java interpreter.

Due to its portable byte code, the same Java application should run anywhere that the Java Virtual Machine runs. Some books or articles on Internet, like Sun Microsystems Inc. [38], have claimed that Java is a programming language that is "platform independent". However, it is absolutely wrong to think that this means that, after the code is written once, it will work on any platform regardless of the hardware and software environment in which it runs. That is, Java does not solve all the problems of cross-platform development.

The availability of application software for a platform is directly proportional to the installed base of that platform. The higher the installed based, the greater the amount of available software. In the next section, the problem in cross-platform is explored step by step.

5.2 Problem Evaluation

Which factors really affect the platform independence when developing a Java application? Many articles and documents on Internet discuss this prevalent problem. A SunWorld contributor, Steven Gould, identifies 10 weak points to consider when exploring the problem evolution in writing multi-platform Java Code [40]. Comparing that with our system, we also found some cross-platform pitfalls, as listed below:

- *Troubles in graphical user interface development*

There are a few major limitations to Java's ability to do clean cross-platform execution. For example, platform independence is all well and good in command line environments with just text-based input and output. However, using Java AWT, the specific windowing toolkit relies on the native window system to implement all of its UI elements: on Unix it uses X/Motif, on the Macintosh it uses the toolbox, on Windows it uses the Windows API, and so on [39].

The key question in making this project on a cross platform programming is how to port the GUI. That is also a most interesting part of Java API. This project uses JDK 1.1.8 (Java 1.1) to implement the client application, which includes interface components, and is written on the Windows NT for multi-platform use. That means it is written, compiled and run on Windows NT first, then the same source files (except Runtime.exec, which is changed to suit to Macintosh and is mentioned in the last chapter) and the class packages are copied to Macintosh and run under Mac CodeWarrior IDE.

- *Use of platform-dependent routines*

When using any of the classes in the java.lang.Runtime, we should avoid any platform-dependent coding. Many of the methods in java.lang.System are also platform dependent in the use of their arguments, using platform-specific constants, anywhere in a code. For example, the static java.lang.System.exit method is intended to terminate the currently running JVM. It is intended for situations requiring abnormal termination.

- *File Input and Output*

For cross-platform on file input and output in the project, there are several considerations. First, these are frequently system-dependent, not only in the location of certain files, but also in the format of filenames and pathnames. So we should not use hard coded file paths in the coding. Second, because relative directory easily makes mistakes when transferring a project between different platforms, we also noted that we should use absolute directories as a parameter instead of relative directories in coding.

- *Java Database Connectivity (JDBC)*

For a Java to be cross-platform, everything it needs must also be cross-platform. It can write database clients in Java with Sun's JDBC package. But JDBC is less successful if it hasn't been ported to every platform that needs to be supported. So we either equip all Java Platforms with as many interfaces as possible, or we have to expect the cross-platform environment to be limited by the interfaces accessible to the Java environment.

Using JDBC and the DriverManager class in an application provides for great flexibility in loading the actual JDBC driver which can be made available to the DriverManager in two ways: Named in the jdbc.drivers system property or, in other way, explicitly loaded using the java.lang.Class.forName method. If the inaccurate JDBC driver name is chosen or entered, it may affect the portability of an application.

5.3 Solutions

How can platform independence be achieved when developing a Java application? The degree of platform independence of any Java program depends on several factors. As a developer, some of these factors are beyond our control, but most are within our control. Using the experience from this project, the following factors might be useful for maximizing the portability of a program:

- *Use of platform-dependent routines*

When using the java.lang.Runtime.exec methods, the string argument passed to these methods is platform specific. So in a platform neutral way, this project tries out the results on each of the target platforms as it goes. As mentioned in Chapter 4, when the code runs on Macintosh, we import the MRJ 2.2 package to support Runtime.exec() method(See the code for choosing a browser and launching it on Windows NT and Macintosh on this page.).

- *Choose an edition and version of the Java Platform that it is well enough distributed among the target hosts; Write the program to run on this version of the Java Platform*

As a developer, we can't control the release cycles or deployment schedules of the Java Platform, but we can choose the Java Platform edition and version that the programs depend upon. For example, as is mentioned in chapter 2, because the Macintosh Java development environment is not yet as up-to-date as the Window or UNIX versions at the time of project design, the MRJ 2.2 just supports JDK 1.1.8 but not Java 2. For keeping the platform independence, the program running on Windows NT has chosen JDK 1.1.8 as well.

- *Use some good methods for file and I/O handling*

It is better when file writing or reading to use the *println* method to handle the end of line, or the System.getProperty ("line.separator") method to print a line on a screen, and readLine method in java.io.BufferedReader class to read complete lines of text.

.

When coding, we also noted whether the file system in Windows NT or Macintosh is case sensitive. It helps if one comes up with a consistent naming convention here.

When using absolute directories as parameters in routines, we also noted different ways to express a path parameter when transferring the project from Windows NT to Macintosh. The detailed code to choose a path parameter and other differences in both machines can be seen (Figure 5.1 & 5.2) in last two pages of this chapter.

- *Strive to design a user interface that works better on all of the target hosts;*

The major variation between different Java Platform implementations is user interface. As is indicated from the description above, user interface is one of the more difficult issues in writing platform independent Java programs.

When designing a user interface, there are three factors should be considered for good "look and feel". These are the metaphors, the components, and the interaction between the components. It is better to ensure using only ASCII characters for the default text of messages, buttons, labels, and menu items. Furthermore, the project has considered using layout managers, such as FlowLayout and GridBagLayout, which are not absolute locations, to place items and to avoid assumptions about things like font metrics and screen size.

For those platforms that do support command-line arguments, be aware of the different conventions on the different systems. Java Foundation Classes (JFC) provides a clean way to handle property files. More detail about JFC will be discussed later in this chapter.

- *Choose the exact JDBC driver name in an application;*

To keep an application as portable as possible, we also choose or enter the exact JDBC driver name. This can be done by relying on the jdbc.drivers system property, or by using a propertied file to supply a name to pass to the java.lang.Class.forName method.

These are the rules to follow when writing a platform independent Java program, which involves portions of the Java virtual machine that can be implemented differently by different vendors. They address the variations allowed in the Java virtual machine specification for garbage collection and threads.

After testing the project on Windows NT and Macintosh, we solved the cross-platform problem step by step. This porting of the application did work, but some of the interfaces do not come out looking the same as they did on the Windows NT platform and had a slightly different appearance. So a Java program executed on different Java platforms has a different appearance and sometimes even different user interactions on each platform. The reason for this were explained earlier in this section, that is, using Java AWT, the specific windowing toolkit relies on the native window system to implement all of its UI elements.

- *Add Swing capability alongside Java AWT;*

The Swing classes eliminate Java's biggest weakness: its relatively primitive user interface toolkit. Swing provides many new components and containers that allow us to build sophisticated user interfaces, far beyond what was possible with AWT. This project has imported Swing to improve 'the look and feel' of the user interface. It hasn't completely replaced the AWT, however. The reason is that the JDK 1.1 event model, especially when combined with the use of inner classes, provides a much more object-oriented solution to the AWT event handling.

Figure 5.1 and Figure 5.2 (See page 65 & 66) are two pieces of the Java code on Windows NT and Macintosh separately for choosing a browser and launching it. Please note the differences, which used the **bold** font.

5.4 The Politics of Platform Independence

Java Platform vendors are allowed to extend the standard components of the Java Platform in non-standard and platform-specific ways, but they must always support the standard components. In the future, Sun Microsystems intends to prevent the standard components of the Java Platform from splitting into several competing, slightly incompatible systems, as happened, for instance, with UNIX [43].

The license that all Java Platform vendors must sign requires compatibility at the level of the Java virtual machine and the Java API, but permits differentiation in the areas of performance and extensions. There is some flexibility in the way vendors are allowed to implement threads, garbage-collection, and user interface look and feel. "If things go as Sun plans," someone said, "the core components of the Java Platform will remain a standard to which all vendors faithfully adhere, and the ubiquitous nature of the standard Java Platform will enable us to write programs that really are platform independent" [43].

5.4.1 Java Foundation Classes

As we mentioned above, there are some problems in developing graphical user interface in JDK 1.1. However, an improved Abstract Windowing Toolkit (AWT), contains the most significant cross-platform graphical user interface, the Java Foundation Classes (JFC) also includes the Swing classes, pluggable look and feel designs, and the Java Accessibility API. The JFC libraries allow the developer to quickly customize the look and feel of the application with a single line of code.

There are some differences in the JFC between JDK 1.1 and the Java 2 SDK. Both development kits contain the Abstract Window Toolkit, however JFC is part of Java 2 platform and included in SDK while JFC 1.1 does not belong to JDK 1.1 and must be utilized with JDK 1.1.

The Java 'look and feel' is the default interface class for applications built with the Java Foundation Classes. As Sun introduced in their web sites [44]: "Java Foundation Classes extends the original AWT by adding a comprehensive set of graphical user interface class libraries that is completely portable and delivered as part of the Java platform. In addition, JFC will also include many of the key features found in Netscape's Internet Foundation Classes." Because the JFC forms the core of the Java platform, there is no more need to download special classes at runtime and is also compatible with all AWT-based applications.

Java Foundation Classes have some new features that meet the designers' and developers' need for building consistent compatible, and easy-to-use Java applications. It

also offers significant performance improvements. With the JFC, we can reduce design and development time for creating and deploy large scale, mission-critical Intranet, Internet and Cross-ware applications. Because the JFC components provide visual elements underlying the Java 'look and feel', they promote flexibility and ease of use in cross-platform applications and support the creation of applications that are accessible to all users.

5.4.2 Other tools to support Java

There are also some potential cross-platform solutions to support Java. We are introducing two popular products here.

Sun's forte for Java [41], Community Edition Software is the entry–level version and development tool supported for creating Java cross-platform applications. It is an open Application Programming Interface (API) that allows developers to create plug-in modules and provides some functionality to allow us easily and quickly edit, compile and debug. We can also, with the Forte for Java IDE, create Internet services and solutions in Pure Java on several operating environment.

Inprise/Borland Jbuilder 3.5 is another popular Java development tool. It allows Java developers build quickly Internet and enterprise applications. Further more Jbuilder 3.5 support developers a wide choice of platforms, such as Linux, Windows and Solaris, on which to develop Java applications [57].

5.4.3 Write Java code on the Macintosh first

For the last thinking, it is better to write Java code first on the Mac if we want to achieve a better looking and feeling result. That is easier than ever to write a Java code on the Mac for multi-platform use. However, it should still be necessary to learn a lot about the way Java is implemented on each platform along the way.

Because the Mac is a good Pure Java starting point, if it works on the Mac it should work on other Java platforms. However, there are differences, often-subtle ones. These differences between Java on the Mac and Java on other platforms are often reflections of each underlying operating system [42].

In a word, developing the cross platform Java code on the Macintosh should involve some testing and some adjustments. Finding solutions to cross-platform problems will often reveal problem codes that slipped by the original compiler or VM and that need to be reworked anyway. Dealing with a few platform-specific problems and a wider variety of VM behaviors, we will have the satisfaction of developing the code on the Mac, and the ability to drop the result on an NT machine and find that it all just works.

5.5 Summary

Because this project was written on PC code first, several problems occurred during the test period time when it was imported to Macintosh. To solve the same problem as above, I rewrote some of the code, which is platform dependent on Macintosh. For example, the code imports the MRJ 2.2 and a particular class as the support to open a URL in a browser on Macintosh. Swing was used as a solution to replace part of Java AWT, which is suitable on PC but not on Macintosh. Finally, the objectives of the project have, for the main part, been achieved except that the user interface shows a slight different 'look and feel' on Windows NT and Macintosh.

```
package interfacepg;                                          // Code in Windows NT
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class ChooseBrowser extends JFrame implements ActionListener
{
      private String initialURL = new String();
      private String browserChoice = new String();
      private JRadioButton IEChoice, NSChoice;
    …
    public ChooseBrowser(CourseContent parent,String initialURL) {
      …
        this.initialURL = initialURL;
        IEChoice = new JRadioButton("Internet Explorer",true);
        NSChoice = new JRadioButton("Netscape",false);
       browserChoice = new String
                          ("C:\\Program\\Files\\Plus!\\Microsoft Internet\\Iexplore.exe");
      }
    public void actionPerformed(ActionEvent event)
    {
            String arg = event.getActionCommand();
          if(IEChoice.isSelected())
                browserChoice = new String     //choosing the path parameter of IE browser
                          ("C:\\Program\\Files\\Plus!\\Microsoft Internet\\Iexplore.exe ");
          else if(NSChoice.isSelected())
              browserChoice = new String  //choosing the path parameter of Netscape browser
                    ("C:\\Program\\Files\\Netscape\\Communicator\\Program\\netscape.exe ");
      }
    public boolean action(Event evt, Object arg)
     {
       if(arg.equals("ok"))
       {      dispose();
         parent.setBrowserChoice(browserChoice);
         try {            // launching the browser
             Process p = Runtime.getRuntime().exec(browserChoice+initialURL);
         }catch(IOException e)
        {          //handing error  }
      }
     }
}
```

Figure 5.1 The piece of Java code on Windows NT

```
Package interfacpg;                                                    // Code in Macintosh
import java.awt.Frame;
import java.awt.FileDialog;
import java.io.File;
import java.io.IOException;

import com.apple.mrj.MRJFileUtils;

public class ExecLaunch extends Frame
{
     private String url;
     public ExecLaunch(String url)
     {   this.url=url;
          try
          {
             MRJFileUtils.openURL(url); //Attempt to let MRJ to find the browser for us.
                    return;            //If this was successful, then we need not go on.
          }
          catch (IOException exc)
          {
                    //This can occur if problems arise while attempting to open the URL.
          }
          catch (NoSuchMethodError err)
          {
                   //This can occur when earlier versions of MRJ are used which do not
                  //support the openURL method.
          }
          catch (NoClassDefFoundError err)
          {
                     //This can occur under runtime environments other than MRJ.
          }

               //If we make it here, MRJ was unsuccessful in opening the URL, and
               //we need to do it the hard way, using Runtime.exec.
              String browserName;

               //Set up a FileDialog for the user to locate the browser to use.
              FileDialog fileDialog = new java.awt.FileDialog(this);
              fileDialog.setMode(FileDialog.LOAD);
              fileDialog.setTitle("Choose the browser to use:");
              fileDialog.setVisible(true);

                  //Retrieve the path information from the dialog and verify it.
              String resultPath = fileDialog.getDirectory();
              String resultFile = fileDialog.getFile();
              if(resultPath != null && resultPath.length()!= 0 && resultFile != null &&
                   resultFile.length() != 0)
              {
                    File file = new File(resultPath + resultFile);
                    if(file != null)
                    {
                          browserName = file.getPath();
                          try
                          {
                                //Launch the browser and pass it the desired URL
                              Runtime.getRuntime().exec(new String[]{browserName, url});
                          }
                          catch (IOException exc)
                          {
                                       exc.printStackTrace();
                          }
                    }
              }
        }
}
```

Figure 5.2 The piece of Java code on Macintosh

Chapter 6 The Results of the System

The project described in this report includes two subsystems. One is an application information system, which provides a service for academic staff so they can edit their multimedia lecture notes and keep track of any stages of courseware development and Web publishing. The second one is a web-based applet system, which is offered to the registered students, who can browse the courseware and access the lecture notes on the Web.

In this stage, the project just implements the staff side. This chapter involves a detailed narrative overview of the project. The first section describes the project presentations on the screen. The second section discusses the performance issues of Java and their relevance to our system. The last section gives the work needed to be developed in the near future.

6.1 Presentations of the System

In some applications, the code for the application interface is developed right along with the rest of the application. The windows are objects of classes created specifically to provide an interface to the application. In other cases, special tools are used to develop interfaces. Java provides its own interface development framework- the Abstract Windowing Toolkit (AWT). This project also uses supplemental tools, such as CodeWarrior, for interface development.

The resulting application user interface of this project is composed of a main window with which the author selects options and from which all the other windows in the system can be reached. The other windows typically present a service of the system. The presentation of the system can be divided into four interfaces. They are: the login authoring interface, the courses and staff lists interface, a course content interface, and a lecture note presentation interface.

6.1.1 The Login Authoring interface

The Login Authoring is the main interface in this system. There are three parts shown in this window. On the top of the panel is the welcoming information that introduces briefly what the system is. The middle area gives explanations of the major operations included in this system. The main part, with which the user can interact, is the Authoring Choose: Login to the system; Close the system and the Help function. Figure 6.1 gives the screen shot of the Login Authoring interface.

When the 'Close' button is clicked, the system closes the connection with the server and exits the application as well. When the 'Help' button is clicked, the help function is launched on the screen and it shows some information about how to browse this system.

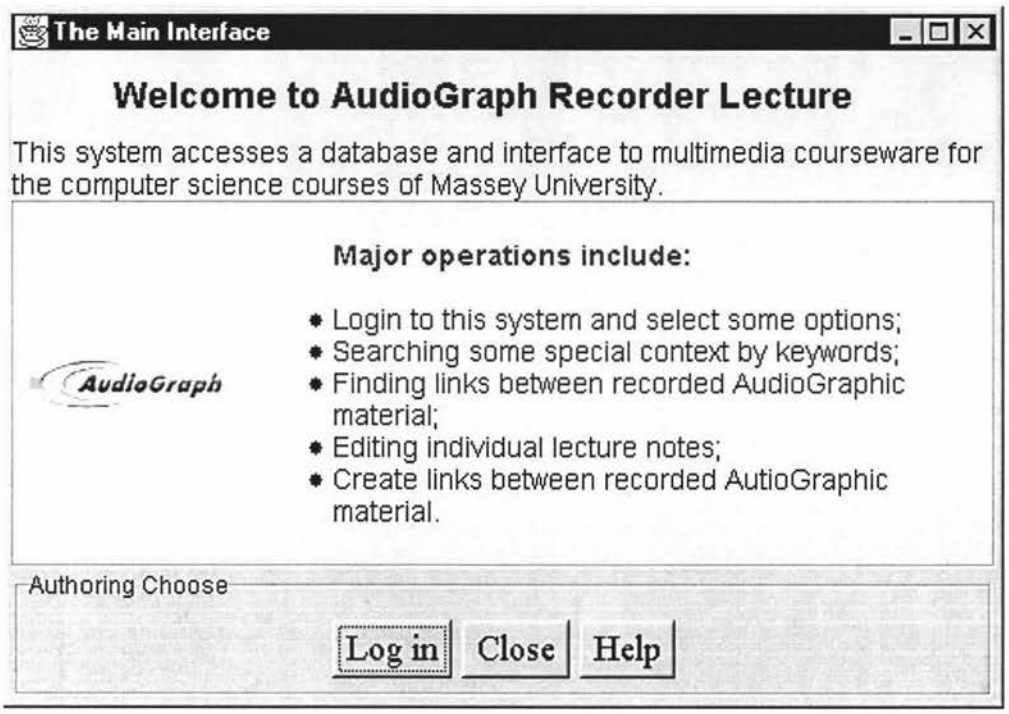

Figure 6.1 The screen shot of the Login Authoring interface

When the 'Log in' button is clicked, a dialog frame appears on the screen and asks the user for the user name and password. The user types their user name and password, the system connects the server and checks the author against records in the database. In this way the access to material can be managed and made secure.

If the login is successful, the system displays the staff and courseware information details. If the login fails, the system displays the 'wrong operation' message and offers the user the chance to try again. It allows the user to try four times. It can be exited automatically if the authoring checking hasn't been successful. Figure 6.2 gives the screen shot when the 'Login' button is clicked.

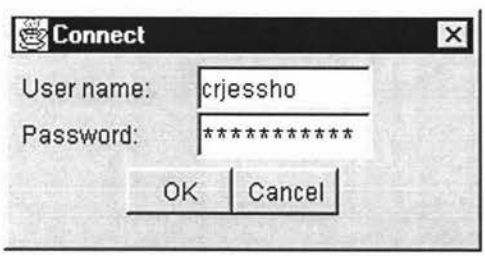

Figure 6.2 The screen shot of the Connect Authoring interface

6.1.2 The Courses and Staff Lists interface

The 'All Staff' List is always visible on the screen and located at the top right of this interface. The course lists occupy the left part of the interface. There are three kinds of course list presentations to choose: the user's teaching course lists, a particular staff teaching course lists and the all courses list in this system. Just one group of lists is visible on the screen.

The default visible course lists are the user's teaching course lists, which are divided into two parts: the user as controller, and the user as a contributor. Figure 6.3 gives the screen shot of the lists of the courses that the user own.

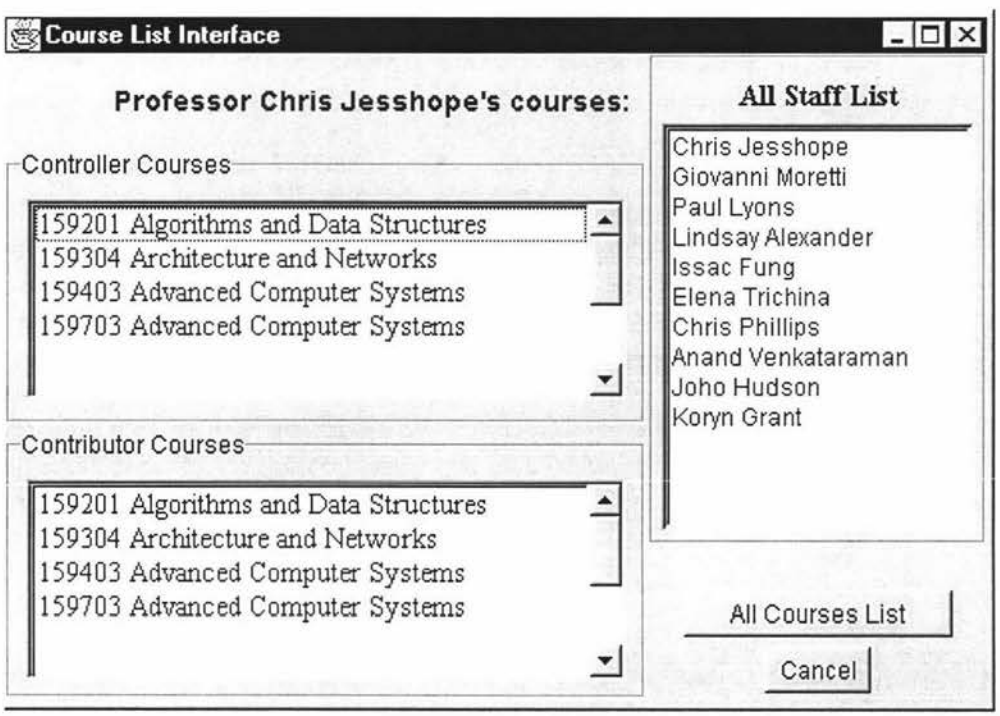

Figure 6.3 The Course List frame for the user

If some other staff member's course lists needs to be shown, just double click the person's name in the All Staff List and they will be displayed on the left-hand side. These course lists also have two parts; controller courses and contributor courses. Figure 6.4 gives the screen shot of the course lists interface of a particular staff.

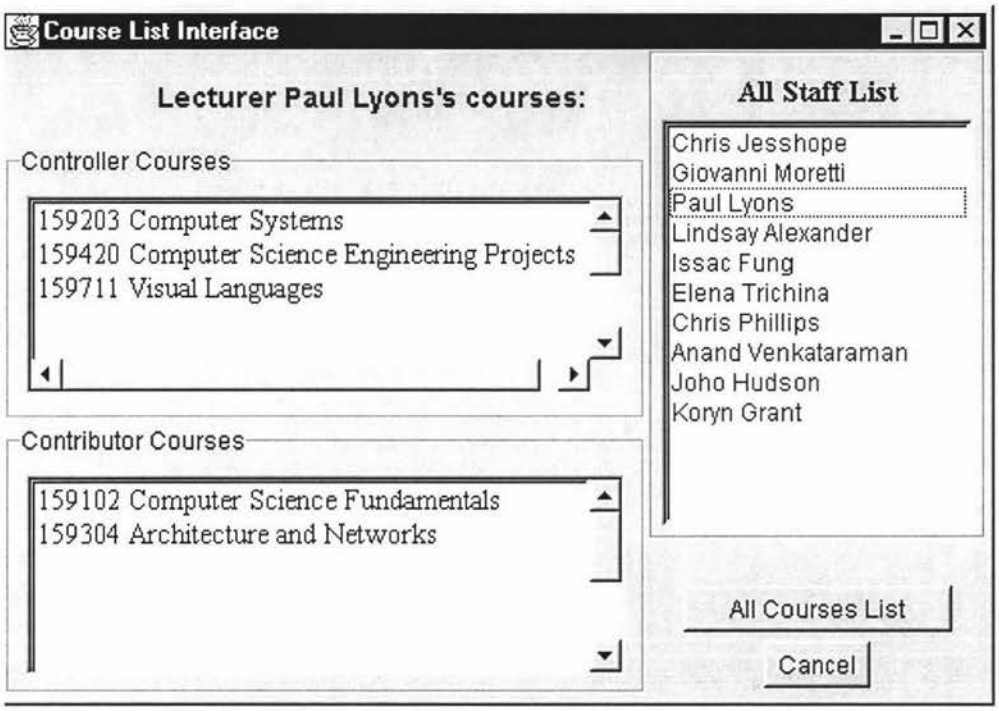

Figure 6.4 The Lists of Courses for the staff member 'Paul Lyons'

Under the 'All Staff List', there is a button named 'All Courses List'. When this button is clicked, the all course list in this system will be displayed on the left of the window. Figure 6.5 gives the screen shot of the all courses list interface.

To access the course content for any course, the user just double clicks any course name, which is in a list. The system will then extract the course content from the database server and then launches a new window, which includes the course content.

There is also a 'Cancel' button, which offers users the choice to rollback from this interface if they want to cancel this stage.

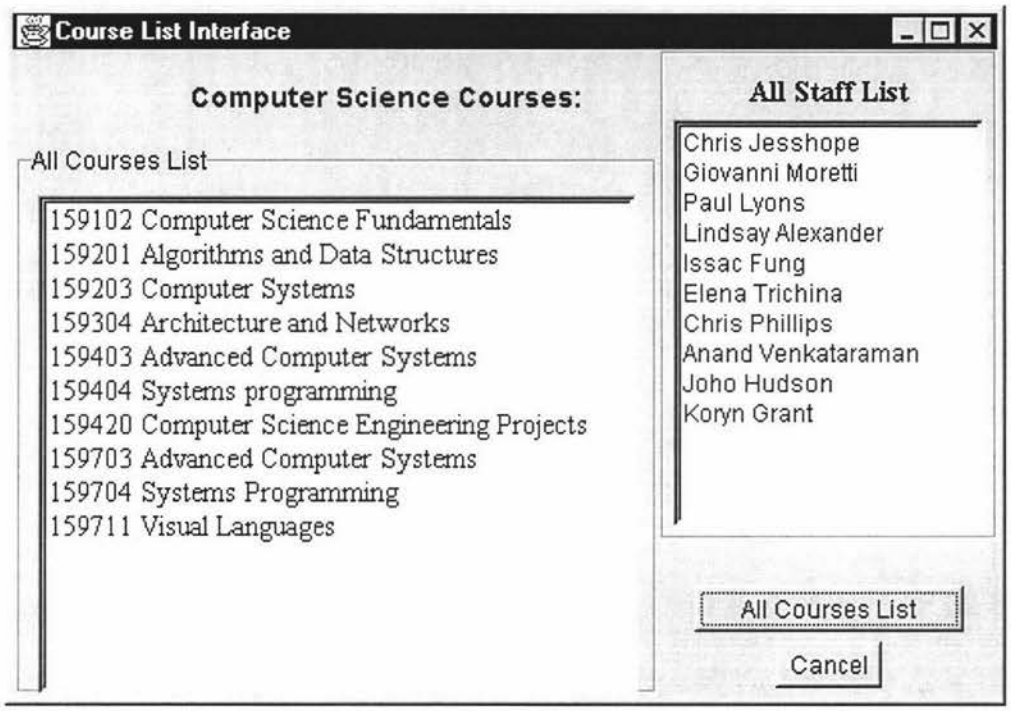

Figure 6.5 The Lists of Courses for Computer Science

6.1.3 The Courses Content interface

This window is launched by double clicking a course name in a list, which is one list on the previous windows. It is divided into two main parts. In the left part of the window the course content is embedded. The right part of the window has three layouts: the top layout shows the 'course controller' of this course, the middle layout lists the 'course contributor' in this course and the bottom layout has two buttons that are the 'All Slides Show' and the 'Cancel'. Figure 6.6 gives the screen shot of a course content interface.

The content of the course is a hierarchy or tree structure, which can hide all levels except the top level. One can click icons representing non-terminal nodes, which causes the structure below this to be exposed, and so on, until the bottom of the tree is reached. This is displayed with a different icon representing the leaf node. Each leaf is a lecture note, which links to an AudioGraphic, presentation slide. This interface also offers three choices: the lecture notes of the users, the lecture notes of a particular contributor and all lecture notes in the course.

The default visible slides are those of the user. Figure 6.7 gives the screen shot of the course content interface with the lecture notes of the user.

If another contributor's lecture notes are required, the user can just double click the person's name in the 'Course Contributor' board. The content of that lecturer's contribution to the course is then displayed.

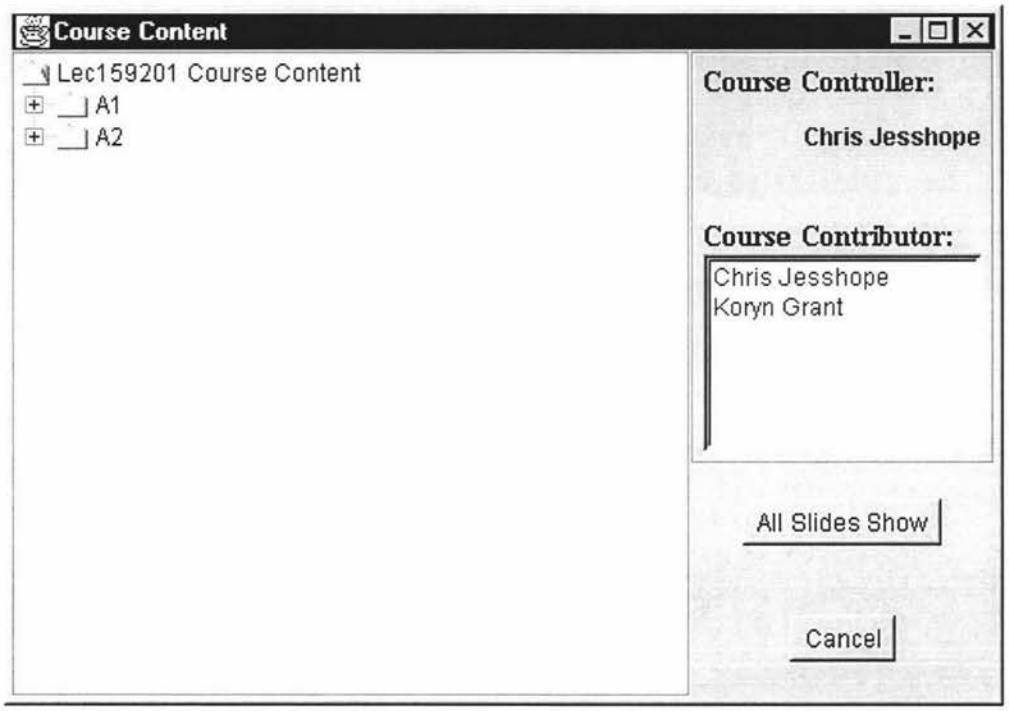

Figure 6.6 The Course Content interface of lecture 159201

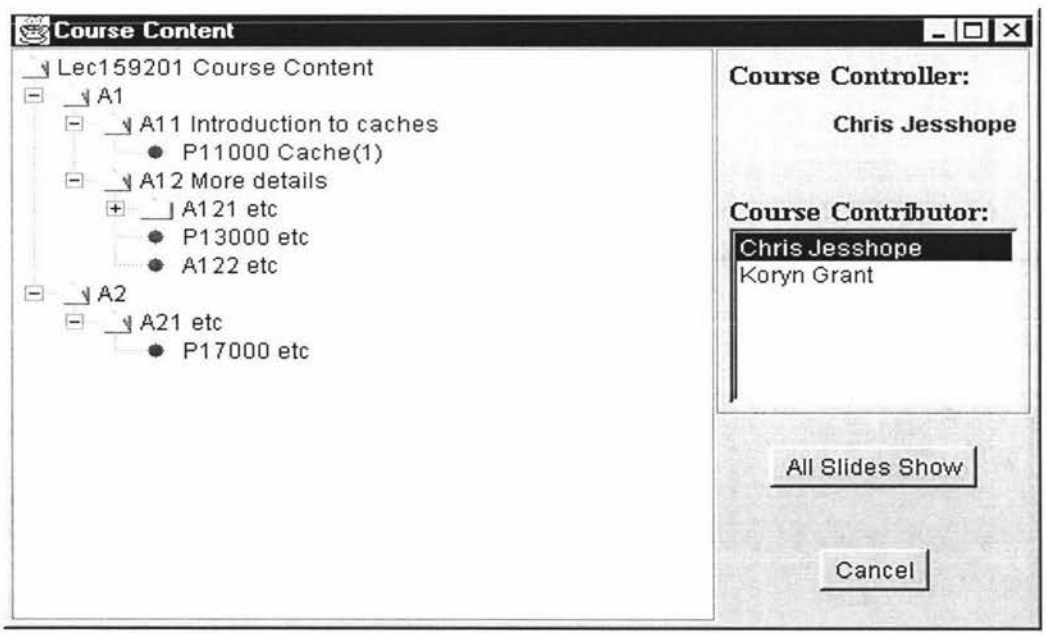

Figure 6.7 The Course Content with the lecture notes of the user

Clicking the 'All Slides Show' button shows all of the lecture notes for the course. Figure 6.8 gives the screen shot of all lecture notes available on interface.

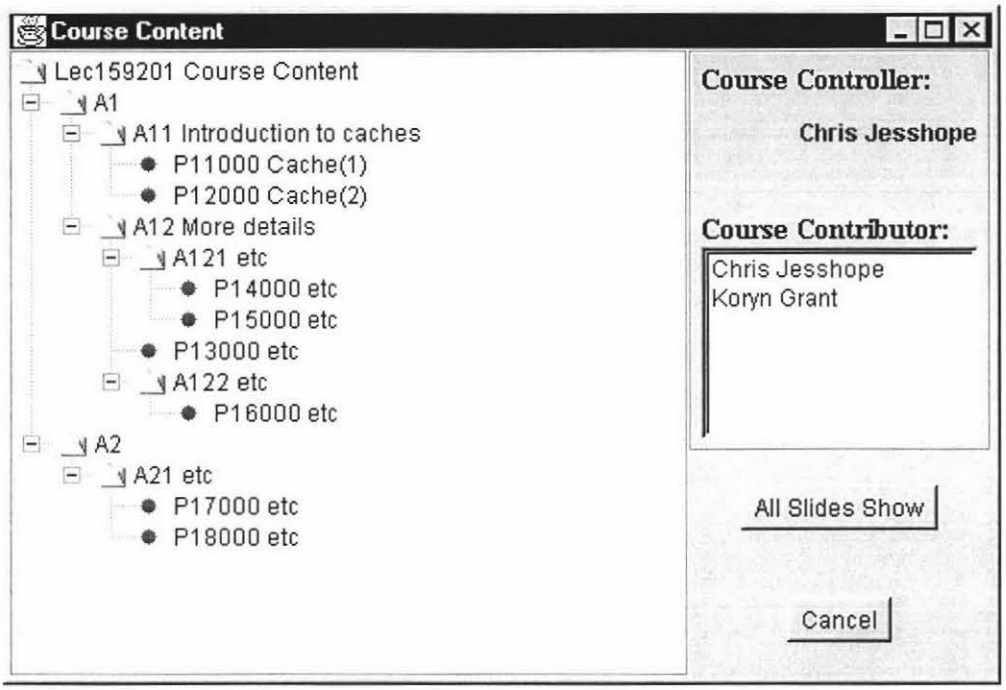

Figure 6.8 The interface of All Lecture Notes

To first view any lecture notes, the user just double clicks the relevant slide name that is in the course tree. The system will offer a choice between Netscape and Internet Explorer. Figure 6.9 gives the screen shot of the browser choice interface. You can view any lecture notes as many times as you like using the browser that you chose at the first viewing.

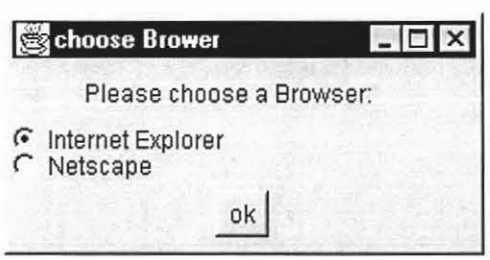

Figure 6.9 The interface for browser choice

There is also a 'Cancel' button on the Course Content Interface, which offers users the choice to rollback from this interface if they want to cancel this stage.

6.1.4 A Lecture Note Presentation interface

When the user selects a web browser tool at the first browsing in the previous window, the system looks up the location of the relevant source files, which has been stored in the database of the server, and then displays the lecture note on the web browser. The lecture notes are presented on the web browser that uses the AudioGraph plug-in tool.

Figure 6.10 gives the screen shot of a Lecture Note Presentation interface.

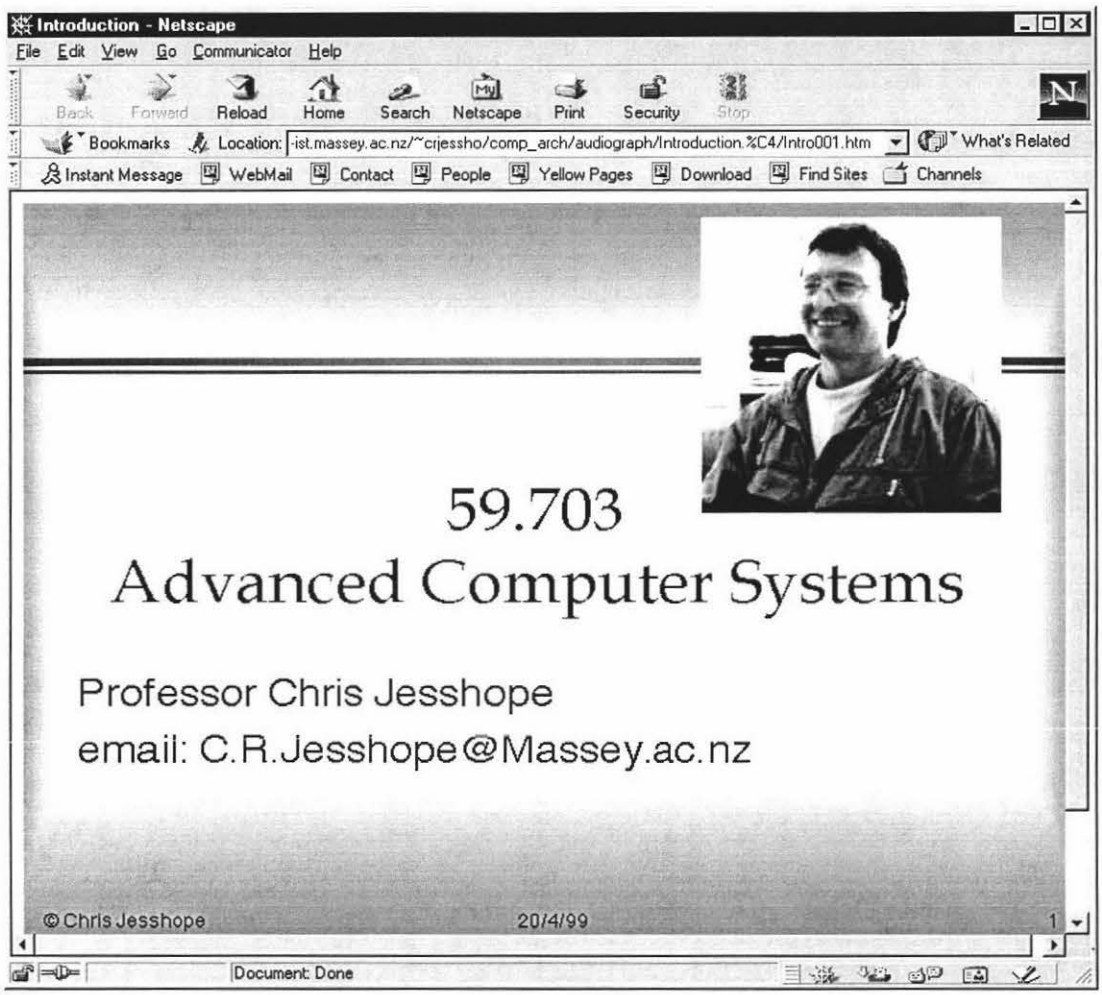

Figure 6.10 The screen shot of a Lecture Note Presentation

6.2 Java Performance Issues

Performance issues are the primary reason why most major corporations have not yet begun a wholesale revision of their existing computer systems using Java language. This section will discuss the performance of Java that includes the runtime performance and programming performance.

6.2.1 Runtime Performance

When speaking of performance in Java, people are actually referring to two very different problems. The first is the download performance of an applet. The second major issue behind performance is runtime performance. Because this project just focuses on Java application, the report ignores discussion of the download issue while concentrating on the runtime performance.

For both applets and applications, the speed with which Java computes is quite slow but it depends on what we are comparing its performance to. Compared to high-level, fully interpreted scripting languages, such as Perl or JavaScript, Java has a relatively high-performance. However, when compared to similar applications written in C++, Java is very much slower because of its weakish native compiler that is perhaps the only way to build high-performance, mission-critical applications. It is as Bob Lent, who is the distinguished engineer with Chordiant Software Inc. [52] said; "Native compilation destroys your cross-platform capabilities".

Java requires hardware and an operating system that is capable of executing 32-bit operations to run its exec. files. These allow Java development systems to match Windows 95 and NT, OS/2, Macintosh, and many UNIX systems. Java development systems generally expect a color monitor and a mouse, but do not require a huge amount of memory, or disk space.

In fact, it is possible to say that Java has the potential to become a high-performance language. Most Java applications cannot challenge the speed of C because the code is interpreted, not compiled, and there are many run time checking operations (e.g. a variable type check). However, just-in-time (JIT) compilation is closing the speed gap and it is used to implement a Java virtual machine. Through JIT compilation, a bytecode method is translated into a native method on the fly, so as to remove the interpretation overhead. Sun provided a specification for how and when JIT compilers should execute Java and vendors were left to implement their own JIT compilers as they chose. JIT compilation is much faster than regular bytecode, being between 10 to 50 times faster.[16].

A Just-In-Time compiler can be embedded in a software tool such as Metrowerks. It is available for both application and applets. For example, Metrowerks company recently launched CodeWarrior 11, hosted on Mac OS, one of the new Java Support is the final release of Metrowerks' PowerPC Java Just-in-time compiler, as well as an early release of Metrowerks' 68K Java JIT compiler. JIT compilers convert entire applications written in Java into native machine code at runtime and thus significantly increase the speed of applications when they are executed.

However, this project works under the CodeWarrior 4 that is not embedded in any JIT compiler. So the performance is still the most significant concern in my system. If the just-in-time compilers are used in the future, the performance might be improved. At this stage, we provide a 'busy cursor' as an appropriate feedback during user processing. This is often an easy and effective way to dramatically improve the perceived performance of a program. A busy cursor is simply a cursor icon that makes it obvious to the user that the system is busy doing something.
All I have done to change the cursor is to call the setCursor method on my component (e.g. CourseList) before setting up the operation that reacts to a user's click. I used the built-in Cursor: WAIT_CURSOR that has been customized for my application. To be safe, the project captures the current cursor first, then restores it when the operation is

complete. This wrapper approach allows the system to safely perform nested operations that might change the cursor, too.

6.2.2 Program Performance

The performance issue of Java is mainly meant to interpreting bytecode when the program running. However, programming in Java is also the technique and design practices that we can improve the runtime performance of the Java code [46]. Concerning our project, the following discusses the program performance for designing our Classes, the use of *GridBagLayout* and *Vector* Class. It also includes the other thin Java programming tips that this project has been paid attention to.

Class design

The tips for designing classes are to use aggregation rather than inheritance, and to reuse object instances [45].

Overusing inheritance can lead to unmanageable class hierarchies. In Java, object creation slows as inheritance hierarchies deepen. Reducing the depth of the hierarchy also reduces the number of classes that have to be downloaded and loaded into the virtual machine.

Object creation in Java is expensive. One solution is to design objects so that they can be reused. A method can be created which clears out an object's properties and prepares it for reuse. This works best for utility objects and GUI objects. However, we should make sure there are no dependencies on the object and that all stated information is removed as appropriate.

I was concerned the tips above, and wrote some classes myself for aggregation and reuse. For example, MyCourseCard object is aggregated by Contro and Contri components. AstaffCourseCard reuses MyCourseCard. Both MyCourseCard and AllCourseCard reuse CourseList Class as well.

Using GridBagLayout Properly

From a personal point of view, I must say that the GridBagLayout has been one of the most frustrating layout managers to use in my project. When comparing it to the other Java layout managers, it seems out of jplace, as if it is too complicated to use. Other layout managers such as BorderLayout and FlowLayout are easy to grasp intellectually and easy to use. The concepts presented in the Javasoft documentation described that the GridBagLayout is simply a grid that gets built dynamically based on the components and their desired position. The GridBagLayout class works in conjunction with the GridBagConstraints class [54]. Each component managed by an instance of GridBagLayout needs to have an associated instance of GridBagConstraints, which contains information for laying out the component. GridBagConstraints contains several properties, which include gridx, gridy, gridwidth, and gridheight. Using these properties properly to specify a component layout can also improve the performance of the program.

Analysis Vector Performance

The beauty of using a Vector object is that the Vector class hides all the underlying work needed to make sure that there is enough space allocated to safely add a new element. We simply needs to call the add() method, passing in the object to be added; the Vector class handles the rest. Along with the ability to dynamically resize itself, the Vector class offers a handful of useful methods to make a programming task easier. Retrieving an object from a Vector at a specific index is accomplished by calling the get() method, passing in the index of the object to be retrieved. A Vector gives us the added capability of removing all the elements it contains with a call to the clear() method. Also, several new elements can be added to a vector at once by calling the addAll() method, passing in a Collection.

This project uses Vector to take advantage of a dynamically growing container. Furthermore, the other interesting benefit is that the Vector class is synchronized, which makes it thread-safe. However, this is one of the facts that affects the programming performance, even if it is not serious, because virtually all of the performance cost associated with using a Vector is due to the synchronization overhead.

By the way, the Vector class has been updated in Java 2 and is included in the Collection classes. Vector now extends the AbstractList class. ArrayList is another one of the Collection classes that extends the AbstractList class. ArrayList is the most obvious alternative to using a Vector when we need a dynamically growing array in a nonthreaded environment.

Other thin Java programming tips

In our project, we also consider other Java programming tips as are mentioned below:

- Use shorter names for variables and methods

In Java, because of late binding, the names of classes, instances, variables, and methods are stored in the class file. This project chose the names that are short, but still meaningful, reducing the size of a class file.

- Declare as many members private as possible

To take the fullest advantage of code shrinkage, members should be given the most restrictive visibility possible. The coding of the project tries to make a member as private as possible. Only make a member public if it absolutely must be public.

- Reduce use and length of strings

In Java, two bytes are used to store each character in a string. To code this project, we try to reduce the use and length of strings in order to reduce the size of a Class and to avoid duplicating strings through multiple declarations of the same string.

6.3 Work Still Required

Given the time limit on this thesis, the following requirements, which have been analyzed and designed, but have not been implemented in this stage. They are the following:

- To download or upload the source files using FTP;
- Password encryption;
- Searching some presentations or courses by relevant keywords;

Let us talk about these issues step by step.

6.3.1 FTP technique and FTP in Java

FTP stands for File Transfer Protocol. It is basically a tool used for transferring and handling files across computers anywhere on the Internet – just like copying files over a network in an office.

To continue our work on this project, we should consider implementing the FTP Class in Client side and Sever side.

To create an FTP client application in Java, first, create a socket to establish a connection with the remote host by using a port number. Second, send the FTP commands described in RFC (Request for Comments) 959 [47] (user, pass, cwd, pwd etc) to the remote FTP server via the output stream of the socket.

Java JDK 1.1 lacks FTP feature. Sun has delivered some packages that support FTP clients in Java. We can find, download and import them for writing our own FTP client in Java. For example, the EOS FTP Bean is an implementation of client-side FTP [48], Sun claimed this package makes facilitates using FTP in Java applications. It can be used for interactively deleting files and listing, creating, or deleting directories.

The FTP in client side of our project has been designed to be capable of retrieving files both in ASCII and binary format. It will be capable of viewing, converting and browsing archive trees. The client side will be able to resume interrupted downloads. The client will also be based on the standards specified for the File Transfer Protocol in RFC 959.

On our server, we should also write an FTP application. It is required to check a valid login name and password to gain access to the machine by FTP. This is to prevent the files on the server being altered or deleted by other users.

6.3.2 Password Encryption

Not so long ago, the only way for a server to identify a client was to ask the user for a username and password. If the username and password matched those stored in the password file, the server granted the user access. This project, at this stage, allows

registered users to login to the system in this way. However, it still has security problems. For example, storing the user name and password in the database creates the risk of them being accessed by any user.

A better solution is for the user to generate a special kind of cryptographic key, called a public/private key pair. These keys work together and need an identity and a certification authority's signature. The combination of a public key, the identifying information and a certification authority's signature is called a certificate. The current generation of the standard for certificates is X.509v3 [49]. This technique is called encryption.

Encryption means transforming information into a meaningless bit stream that can later be changed back if the correct key is held. This definitely increases the security. For example, I can print the public key on my business card (or in my email signature block) and give it to someone. Then that person can send me message that only I can decrypt. Even though everyone else knows the public key, and even if they intercept all the messages coming to me, they cannot break the scheme and actually read the messages.

While Java finally provides support for several features crucial to secure transactions, it sorely lacks a cryptosystem implementation in JDK 1.1, especially one based on public-key encryption [50]. However, JDK 1.2 includes new APIs for parsing certificates and maintaining local databases of X.509v3 certificates.

For our project, I think we can use the RSA encryption method. RSA is a remarkable encryption method that was introduced by Ron Rivest, Adi Shamir, and Leonard Adleman in 1978. The detail information can be found in book *Applied Cryptography* [55].

6.3.3 Search by relevant keywords

To implement this function, firstly, the *Keyword* table should be fed by data, which are the meta-data that links to relevant lectures or presentations. The detailed procedure is as follows. When staffs are creating or editing one of their presentations, they are also asked to type some keywords, which are included in this presentation. The system then stored the keywords in the *Keyword* table of the database.

In the client application we should also write a code to let users search relevant presentations by keyword. The procedure is that a user inputs the keywords through a keyboard; the system connects to the server and searches these keywords through the *Keyword* table of the database and then displays all entries of the presentations that contain the keywords. The user can then double click on the entry of a presentation to look at the presentation.

6.4 Summary

This chapter evaluates the results of the system. It included a description of the interaction between a user and the system. Staff can follow and operate this system easily

as long as they have been registered. The most significant concern in this system is the performance issues, of which the major one is the runtime performance. However, if we run this system using some software tool that has a Just-In-Time compiler, runtime performance might be greatly improved. This chapter also analyses and designs some functions that need to be implemented later.

Chapter 7 Conclusion

The majority of educational systems today rely heavily on shifting from a traditional learning style that is based on a lecture-and-book model towards a non-traditional learning style where courses are provided through TV classes, CD-ROM and distributed computer systems.

This Masters project is part of a larger project that aims to build a seamless education environment that supports the preparation and delivery of electronic courses to distributed users over the network. Our courseware database system, using the AudioGraph multimedia authoring tools that are presented in this report, focused the staff's role in keeping track of any stages of courseware development and web publishing.

This thesis introduces the procedures of analysis, design and implementation of the project. It describes the utilities and tools for setting up this system. It discusses the project from two sides: a database server and a user-friendly client application. The advantages of Java are presented. However, the pitfalls of cross-platform software development and the performance of Java are also explored in this report.

7.1 Outline of the System

On the server side, mSQL, a relational database system, has been set up on Unix. Nine tables have been created using the database schema and fed by the test data, which can be seen in the Appendices (Source files).

On the client side, the client applications, written in Java, have been implemented on Windows NT and Macintosh. The application involved three parts: the user interface, the database processing and the local system operation.

Because Java is a platform-dependent language, we developed the client application on Windows NT and tried to port all the source files on Macintosh. However, Java is not completely cross-platform, we have had to slightly modified the code and have imported the MRJ 2.2 on Macintosh to run the system. We obtained the results, which we hypothesized, in both machines.

7.2 Achievement of the System

The original goals of this system have been achieved mostly through testing the results of the project. They are:

On the server side, the database can:

- Store the information about staff, courses, presentations and the keywords which are relevant to the AudioGraph system;
- Access the relationship between staff and course, course and presentation, or presentation and the keywords;
- Include meta-data which describe the information on it, for example, listing keywords with a presentation or lecture;
- Maintain a record on a server to track who has developed or edited a particular presentation.

On the client side, the client application can:

- Authorize login to the system;
- Obtain the user's courses (as controller and contributor);
- Access the whole courses and whole staff list;
- Download a particular member of staff's course (as controller and contributor);
- View a course's controller and contributors;
- Provide a course structure and links to a presentation;
- Launch a browser and display the presentation using the AudioGraph tool.

This system has also been designed so that it can be extended to:

- Download and upload the source file from a server and edit them in a local machine;
- Search for the materials by keywords with reference to a presentation or a course;
- Enforce security by employing Java's encryption method

7.3 Issues of the System

In this project, we also considered the issues relevant to the system. They are mainly in security, cross-platform, and performance.

The security of the database and Java are described in Chapter 4 and Chapter 6. Java is strong on security, especially on client applications. For example, when authorizing login to our system, the encryption method will be used to enforce security.

The platform-independent feature is one of Java's advantages. That is the major reason why Java is used in this client/server system. However, this is also the main source of problems in the system as Java's cross-platform still has many limitations. We discussed all of the problems and solutions in our system in Chapter 5.

Compared to C/C++, Java is weak on performance. The run time is so long sometimes that a user might get tired of waiting. However, we became aware of some of these issues during implementation and rectified the coding to achieve the performance. Furthermore, the Just-In-Time compiler is a significant solution to improve the performance.

7.4 Trends and Future of the System

The increased popularity of the Internet and its usage as a communication technology has offered a suitable opportunity to develop such a system in order to help lecturers to access courses from any access point at any time. From the results of this system, it can be seen that developing and maintaining this system will be a reasonably easy task.

In the future, the main focus should be on developing a web-based education delivery system for students. It should include a comprehensive knowledge base that answers student's questions and displays the relevant information, which the students search for by keyword. Furthermore, the meta-data management of this system should be refined to satisfy more user requirements.

The student users will need to be able to:

- Extend the database schema in storing the student authoring information.
- Develop an HTML interface for the students to search the database and for browser the courseware from the web.
- Keep track of any stages of courseware access and feedback from the register students.

Bibliography

[1] C. R. Jesshope, A. Shafarenko and H. Slusanschi (1998) *Low-bandwidth multimedia tools for web-based lecture publishing*, *IEE Engineering Science and Educational Journal*, **7 (4)**, pp148-154.

[2] C. R. Jesshope and H. Slusanschi (1998) *Distance Education Using the AudioGraph Authoring Tool*, *Proc. Intl. Conf. on Multi-Media Information Systems in Practice*, pp266-278, ISBN 981-4021-53-9, Hong Kong, December 1998 (Springer)

[3] C. R. Jesshope (1999) *Web-based Teaching - Tools and Experience*, *Australian Computer Science Communications*, **21,** (1), pp27-38, ISBN 981-4021-54-7, Proc Australasian Computer Science Conference, ACSC99, Auckland, Jan 1999, (Springer).

[4] NZEdSoft (2000) *Research and development of new Technology*
Web site: http://www.nzedsoft.com/

[5] M. Pearson and C. R. Jesshope (1998) *Multi-campus teaching using computer networks*, *Proc. of the Third Australasian Conference on Computer Science Education* pp 106 - 111, July 1998 (Association for Computing machinery Inc.).

[6] Segal, J. (1997) *An evaluation of a teaching package constructed using a web-based* lecture recorder, *Assoc. for Learning Technolgy Journal*, pp32—42.

[7] NZEdSoft (2000) *AudioGraph Documentation*
Web site: http://www.nzedsoft.com/documentation.html

[8] H. E. Eriksson & M. Penker (1998) *UML Toolkit* John Wiley & Sons, Inc. c1998

[9] T. M. Connolly, C. E. Begg & A. D. Strachan (1998) *Database Systems*
Harlow, England; Reading, Mass.: Addison-Wesley, c1999

[10] Randy JayYarger, Georgg Reese & Tim King (1999) *MySQL & mSQL*
Web site: http://www.oreilly.com/catalog/msql/

[11] MySQL AB (2000) *What is MySQL* Web site http://www.mysql.com/info.html

[12] Hughes Technologies (Aug 1999) *Mini SQL*
Web site: http://www.Hughes.com.au/product/msql

[13] E.Tittel & B. Brogden(1997) *Discover Java* IDG Books Worldwide, Inc. CA.

[14] D. Yeates, M. Shields & D. Helmy(1994) *Systems Analysis and Design*

London: Pitman, 1994

[15] Reese, G. (1997) *Database Programming with JDBC and Java.* O'Reilly & Associates, Inc. U.S.A

[16] S. Khoshafian & A. Chen & A. Wong & H.K.T. Wong(1995) *Client/Server SQL Applications* Morgan Kaufmann Publishers, Inc. U.S.A

[17] P. Sridharan (1997) A*dvanced JAVA networking* Prentice Hall PTR

[18] H. M. Deitel & P. J. Deitel (1999) *JAVA: How to Program,* Deitel & Associates, Inc.

[19] IBM software (1998) *Distributed Computing Environment* Web site: http://www-4.ibm.com/software/network/dce/

[20] Fernando Lozano (1999) *Accessing Databases Using Java and JDBC* Web site: http://www.edm2.com/0607/msql3.html/

[21] T.H.Grayson(1999) *Relational Database Design* Web site: http://gis.mit.edu/classes/11.520/lectures/lecture-18Nov1999.html

[22] C. S. Horstmann & G. Cornell (1997) *Core Java 1.1 Valume 1 – Fundamentals, The SunSoft Press*

[23] P. Niemeyer & J. Peck (1996) *Exploring Java,* O'reilly

[24] P. V. D. Linden (1997) *Not Just Java* The SunSoft Press

[25] H. M. Deitel & P. J. Deitel (1999) *Java: How to Program Java 2* Upper Saddle River, N.J.; London: Prentice Hall, 1999

[26] M. Campione, K. Walrath, A. Huml and the Tutorial Team (1999) *The Java Tutorial Continued, The Rest of the JDK* Reading, Mass.: Addison-Wesley, c1999

[27] Sun Company (9/1999) *Java Development Kits (JDK)* Web site: http://java.sun.com/products/jdk/README-update.html

[28] Java Sun Company (9/1999) *JavaBeans, INFOBUS* Web site: http://java.sun.com/beans/infobus/#DOWNLOAD_COLLECTIONS

[29] Java Sun Company (9/1999) *Java Foundation Classes (JFC)* Web site: http://java.sun.com/products/jfc/download.html

[30] Marty Hall (Tutorial 1999) *Swing: A quick Tutorial for AWT Programmers* Web site: http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/

[31] Apple Company (1999) *Mac OS Runtime for Java*

Web site: http://developer.apple.com/java/

[32] Hughes Technologies (1999) *Mini SQL 2.0 User Guide*
Web site: http://www.twics.com/assist/webhost/msql/manual/

[33] T. Lindholm & F.Yellin (1999)*The Java Virtual Machine Specification*
Reading, MA: Addison-Wesley, c1999

[34] Apple Company (02/2000) *The Care And Feeding of Runtime.exec*
Web site: http://devworld.apple.com/technotes/tn/tn1168.html#Section4

[35] Wade Githrie *Introduction to PIGUI*
Web site: http://www.acoin.com/kelly/pigui/piguiint.htm

[36] D. Dyer(2000) *AWT: Java's Achilles Heel*
Web site: http://www.andromeda.com/people/ddyer/java/achilles.html

[37] Gregory S. Smith (12/1997) *Java World JDK 1.1 and Beyond*
Web site http://sunsite.compapp.dcu.ie/IJUG/javaone/javaone97/java1-97-jdk.html

[38] Sun Microsystems,Inc. (2000) *Java Technology*
Web site: http://java.sun.com/pr/1996/dec/pr961203-01.html

[39] P. Baxter (05/1999) *Cross-platform GUI toolkits*
Web site: http://www.cirl.uoregon.edu/mailman/listinfo/developers

[40] Steven Gould(06/05/2000) *Pitfalls for writing multi-platform Java*
Web site: http://www.sunworld.com/swol-11-1998/swol-11-javaapps.html

[41] Sun Microsystems Company (2001) *FORTE TOOLS Forte for Java*
Web site: http://www.sun.com/forte/ffj/index.html

[42] Ken Ryall (MacTech Magazine 14/5/2000) *Writing Java on the Mac for All Platforms,* Web site: http://www.mactech.com/

[43] B. Venners (03/2000) *Platform Independence*
Web site: http://gartner..jmu.edu/research/ras/78000/78022/78022.html

[44] Microsoft (1997) *JFC for Java*
Web site: http://www.microsoft.com/presspass/press/1997/Jan97/IACmompr.asp

[45] Fatbrain Company *Enhance your Java application with Java Native Interface (JNI)*
Web Site: http://www1.fatbrain.com/whatshot/sigs/javasig/javaworld1099.asp

[46] IBM Company *Performance Introduction* Web Site:
http://www-4.ibm.com/software/network/on-demand/library/publications/v10/toolkit/perform.htm

[47] WebCom (1999) *An Introduction to FTP*,
Web site: http://www.webcom.com/help/ftp/overview.shtml

[48] Eastfork Object Space (1997-1999) *FTP Bean Overview*
Web site: http://www.eos.dk/beans/ftp/

[49] E. Ladd & J. O'Donnell, et al. (1999 Que) *Using HTML 4, XML, and Java 1.2*
Indianapolis, IN: Que, c1999

[50] Jim Lowe (Feb. 1997) *Transaction security: How does Java measure up?*
Web Site: http://www.netscapeworld.com/netscapeworld/nw-02-1997/nw-02-bytecode.html

[51] Sun Microsystems, Inc. (1997) *Java Security*, Web site:
http://sunsite.sut.ac.jp/java/JDK1.1/docs/guide/security/CryptoSpec.html

[52] Antone Gonsalves (05/25/98) *Java performance still disappointing*
Web Site: http://www.zdnet.com/eweek/news/0525/25java.html

[53] Dave Mark [MacTech Magazine, Volume Number:12.01] *Java Talk,* Web site:
http://www.mactech.com/articles/mactech/Vol.12/12.01/Jan96FactoryFloor/

[54] Michael C. Daconta, E. Monk, J. P. Keller & K. Bohnenberger *Java Pitfalls: Time-Saving Solutions And Workarounds to Improve Programs*
New York: Wiley, 2000

[55] Bruce Schneier, (1994) *Applied Cryptography* John Wiley & Sons

[56] ACQWeb MDAP List (1999) *What is an IDE*
Web Site: http://ide.dsmc.dsm.mil/learning_modules/IDE/what_is_an_ide.htm

[57] Borland Software Corporation. Press Release (Feb./2001)
INPRISE/BORLAND BOOSTS CORSS-PLATFORM JAVA DEVELOPMENT WITH JBUILDER 3.5 Web Site:
http://www.inprise.com/about/press/2000/jbuilder35released.html

Appendices

Contents

Part A Server Database Files

A1 proj-schema.msql

```
#
# This is the schema for dbproj database system
# Host: localhost    Database: dbproj
# Author: June Pan and Chris Jesshope  Last date:  21/05/2000
#--------------------------------------------------------------


#
# 1. Table structure for table 'Slides' (presentation)
#

DROP TABLE Slides \g


CREATE TABLE Slides (
  Slide_id              CHAR(10)    NOT NULL,
  Creator               CHAR(20),
  Slide_name            CHAR(50),
  Dura_time             Int,
  K_bytes               Int,
  Lecture_url           CHAR(100),
  Html_aep_url          CHAR(50)
  ) \g

CREATE UNIQUE INDEX Slide_idx ON Slides(Slide_id)\g

#
# 2. Table structure for table 'Slide_Keyword'
#
DROP TABLE Slide_Keyword \g

CREATE TABLE Slide_Keyword (
  Slide_id              CHAR(10)    NOT NULL,
  Keyword               CHAR(20)    NOT NULL  ) \g

#
#3. Table structure for table 'Lec159201'
#
DROP TABLE Lec159201\g

CREATE TABLE Lec159201(
  Section_id            CHAR(10)    NOT NULL,
  Subsec_id             CHAR(10),
  Type                  CHAR(10),
  Subsection_name       CHAR(50)
  ) \g
```

```
#
#4. Table structure for table 'Lec159102'
#
DROP TABLE Lec159102\g

CREATE TABLE Lec159102(
  Section_id            CHAR(10)    NOT NULL,
  Subsec_id             CHAR(10),
  Type                  CHAR(10),
  Subsection_name       CHAR(50)
  ) \g

#
#5. Table structure for table 'Lec159203'
#
DROP TABLE Lec159203\g

CREATE TABLE Lec159203(
  Section_id            CHAR(10)    NOT NULL,
  Subsec_id             CHAR(10),
  Type                  CHAR(10),
  Subsection_name       CHAR(50)
  ) \g

#
# 6. Table structure for table 'Lec159304'
#
DROP TABLE Lec159304\g

CREATE TABLE Lec159304(
  Section_id            CHAR(10)    NOT NULL,
  Subsec_id             CHAR(10),
  Type                  CHAR(10),
  Subsection_name       CHAR(50)
  ) \g

#
# 7. Table structure for table 'Lec159403'
#
DROP TABLE Lec159403\g

CREATE TABLE Lec159403(
  Section_id            CHAR(10)    NOT NULL,
  Subsec_id             CHAR(10),
  Type                  CHAR(10),
  Subsection_name       CHAR(50)
  ) \g
```

```
#
# 8. Table structure for table 'Lec159703'
#
DROP TABLE Lec159703\g

CREATE TABLE Lec159703(
  Section_id      CHAR(10)    NOT NULL,
  Subsec_id       CHAR(10),
  Type            CHAR(10),
  Subsection_name CHAR(50)
  ) \g

#
# 9. Table structure for table 'Courses'
#

DROP TABLE Courses \g

CREATE TABLE Courses (
  Course_id       CHAR(10)    NOT NULL,
  Course_name     CHAR(50),
  Controller      CHAR(30),
  Mapping_table   CHAR(10)
  ) \g

#
#10.  Table structure for table 'Course_Contributor'
#
DROP TABLE Course_Contributor \g

CREATE TABLE Course_Contributor (
  Course_id       CHAR(10)    NOT NULL,
  Contributor     CHAR(20)    NOT NULL  ) \g

#
# 11. Table structure for table 'Course_Keyword'
#
DROP TABLE Course_Keyword \g

CREATE TABLE Course_Keyword (
  Course_id       CHAR(10)    NOT NULL,
  Keyword         CHAR(20)    NOT NULL  ) \g

#
# 12. Table structure for table 'Section_Keyword'
#
DROP TABLE Section_Keyword \g

CREATE TABLE Section_Keyword (
  Section_id      CHAR(10)    NOT NULL,
  Keyword         CHAR(20)    NOT NULL  ) \g
```

```
#
#13.  Table structure for table 'Staff'
#

DROP TABLE Staff \g

CREATE TABLE Staff (
  Staff_id        CHAR(10)    NOT NULL,
  Title           CHAR(10),
  First_name      CHAR(10),
  Last_name       CHAR(15),
  Password        CHAR(20)
  ) \g

CREATE UNIQUE INDEX Staff_idx ON Staff(Staff_id)\g

#
# 14. Table structure for table 'EditRd'
#

DROP TABLE EditRd\g

CREATE TABLE EditRd (
  EditRd_id       CHAR(10)    NOT NULL,
  Slide_id        CHAR(10),
  Editor          CHAR(10),
  Edit_Date       date,
  Edit_time       time,
  Commit          CHAR(10)
  ) \g

CREATE UNIQUE INDEX EditRd_idx ON EditRd(EditRd_id)\g
```

A2 proj-data.msql

```
#
# This is the data file for dbproj database system
# Host: localhost    Database: dbproj
# Author: June Pan and Chris Jesshope  Last date:  21/05/2000
#--------------------------------------------------------


#
# 1.Insert data for table 'Slides'
#

DELETE FROM Slides\g

INSERT INTO Slides VALUES ('P11000','crjessho', 'The problem', 104,
253, 'http://www-
ist.massey.ac.nz/~crjessho/comp_arch/audiograph/Introduction.%C4/Intro0
01.htm', 'http://html.aep' )\g
INSERT INTO Slides VALUES ('P12000','kdgrant', 'Background', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P13000','crjessho', 'Cache(1)', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P14000','kdgrant', 'Cache(2)', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P15000','crjessho', 'process', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P16000','kdgrant', 'Multithreads', 233,
556, 'http://www-ist.massey.ac.nz/~crjessho/comp_arch',
'http://html.aep' )\g
INSERT INTO Slides VALUES ('P17000','crjessho', 'Advance', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P18000','kdgrant', 'Computer System', 233,
556, 'http://www-ist.massey.ac.nz/~crjessho/comp_arch',
'http://html.aep' )\g

INSERT INTO Slides VALUES ('P21111','plyons', 'The problem', 104, 253,
'http://www-
ist.massey.ac.nz/~crjessho/comp_archaudiograph/Introduction.%C4/Intro00
1.htm', 'http://html.aep' )\g
INSERT INTO Slides VALUES ('P28111','gmoret', 'Background', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P31222','plyons', 'The problem', 104, 253,
'http://www-
ist.massey.ac.nz/~crjessho/comp_archaudiograph/Introduction.%C4/Intro00
1.htm', 'http://html.aep' )\g
INSERT INTO Slides VALUES ('P38222','gmoret', 'Background', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
```

```
INSERT INTO Slides VALUES ('P41333','plyons', 'The problem', 104, 253,
'http://www-
ist.massey.ac.nz/~crjessho/comp_arch/audiograph/Introduction.%C4/Intro0
01.htm', 'http://html.aep' )\g
INSERT INTO Slides VALUES ('P48333','crjessho', 'Background', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P51444','crjessho', 'problem', 233, 556,
'http://www-
ist.massey.ac.nz/~crjessho/comp_arch/audiograph/Introduction.%C4/Intro0
01.htm', 'http://html.aep' )\g
INSERT INTO Slides VALUES ('P58444','crjessho', 'Background', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g

INSERT INTO Slides VALUES ('P61555','crjessho', 'problem', 233, 556,
'http://www-
ist.massey.ac.nz/~crjessho/comp_arch/audiograph/Introduction.%C4/Intro0
01.htm', 'http://html.aep' )\g
INSERT INTO Slides VALUES ('P62555','crjessho', 'Background', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P63555','crjessho', 'Cache(1)', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P64555','crjessho', 'Cache(2)', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P65555','crjessho', 'process', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P66555','crjessho', 'Multithreads', 233,
556, 'http://www-ist.massey.ac.nz/~crjessho/comp_arch',
'http://html.aep' )\g
INSERT INTO Slides VALUES ('P67555','crjessho', 'Advance', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g
INSERT INTO Slides VALUES ('P68555','crjessho', 'System', 233, 556,
'http://www-ist.massey.ac.nz/~crjessho/comp_arch', 'http://html.aep'
)\g

#
# 2. Insert data for table 'Slide_Keyword'
#
DELETE FROM Slide_Keyword\g

INSERT INTO Slide_Keyword VALUES ('p1236', 'Cache')\g
INSERT INTO Slide_Keyword VALUES ('p1236', 'Cache')\g
INSERT INTO Slide_Keyword VALUES ('p1237', 'Introductory')\g
```

```
#
# 3. Insert data for table 'Lec159201'
#

DELETE FROM Lec159201\g

INSERT INTO Lec159201 VALUES ('A1','A11','top','Introduction to
caches')\g
INSERT INTO Lec159201 VALUES ('A1','A12','top','More details')\g
INSERT INTO Lec159201 VALUES ('A11','P11000','bottom','Cache(1)')\g
INSERT INTO Lec159201 VALUES ('A11','P12000','bottom','Cache(2)')\g
INSERT INTO Lec159201 VALUES ('A12','A121','intermed','etc')\g
INSERT INTO Lec159201 VALUES ('A12','A122','intermed','etc')\g
INSERT INTO Lec159201 VALUES ('A12','P13000','bottom','etc')\g
INSERT INTO Lec159201 VALUES ('A121','P14000','bottom','etc')\g
INSERT INTO Lec159201 VALUES ('A121','P15000','bottom','etc')\g
INSERT INTO Lec159201 VALUES ('A122','P16000','bottom','etc')\g
INSERT INTO Lec159201 VALUES ('A2','A21','top','etc')\g
INSERT INTO Lec159201 VALUES ('A21','P17000','bottom','etc')\g
INSERT INTO Lec159201 VALUES ('A21','P18000','bottom','etc')\g

#
# 4. Insert data for table 'Lec159102'
#

DELETE FROM Lec159102\g

INSERT INTO Lec159102 VALUES ('A1','A11','top','Introduction to
caches')\g
INSERT INTO Lec159102 VALUES ('A1','A12','top','More details')\g
INSERT INTO Lec159102 VALUES ('A11','P21111','bottom','Cache(1)')\g
INSERT INTO Lec159102 VALUES ('A11','P22111','bottom','Cache(2)')\g
INSERT INTO Lec159102 VALUES ('A12','A121','intermed','etc')\g
INSERT INTO Lec159102 VALUES ('A12','A122','intermed','etc')\g
INSERT INTO Lec159102 VALUES ('A12','P23111','bottom','etc')\g
INSERT INTO Lec159102 VALUES ('A121','P24111','bottom','etc')\g
INSERT INTO Lec159102 VALUES ('A121','P25111','bottom','etc')\g
INSERT INTO Lec159102 VALUES ('A122','P26111','bottom','etc')\g
INSERT INTO Lec159102 VALUES ('A2','A21','top','etc')\g
INSERT INTO Lec159102 VALUES ('A21','P27111','bottom','etc')\g
INSERT INTO Lec159102 VALUES ('A21','P28111','bottom','etc')\g

#
# 5. Insert data for table 'Lec159203'
#

DELETE FROM Lec159203\g

INSERT INTO Lec159203 VALUES ('A1','A11','top','Introduction to
caches')\g
INSERT INTO Lec159203 VALUES ('A1','A12','top','More details')\g
INSERT INTO Lec159203 VALUES ('A11','P31222','bottom','Cache(1)')\g
INSERT INTO Lec159203 VALUES ('A11','P32222','bottom','Cache(2)')\g
INSERT INTO Lec159203 VALUES ('A12','A121','intermed','etc')\g
INSERT INTO Lec159203 VALUES ('A12','A122','intermed','etc')\g
INSERT INTO Lec159203 VALUES ('A12','P33222','bottom','etc')\g
INSERT INTO Lec159203 VALUES ('A121','P34222','bottom','etc')\g
```

```
INSERT INTO Lec159203 VALUES ('A121','P35222','bottom','etc')\g
INSERT INTO Lec159203 VALUES ('A122','P36222','bottom','etc')\g
INSERT INTO Lec159203 VALUES ('A2','A21','top','etc')\g
INSERT INTO Lec159203 VALUES ('A21','P37222','bottom','etc')\g
INSERT INTO Lec159203 VALUES ('A21','P38222','bottom','etc')\g

#
# 6. Insert data for table 'Lec159304'
#

DELETE FROM Lec159304\g

INSERT INTO Lec159304 VALUES ('T1','T11','top','Introduction')\g
INSERT INTO Lec159304 VALUES ('T1','T12','top','More details')\g
INSERT INTO Lec159304 VALUES ('T11','P41333','bottom','Cache(1)')\g
INSERT INTO Lec159304 VALUES ('T11','P42333','bottom','Cache(2)')\g
INSERT INTO Lec159304 VALUES ('T12','T121','intermed','etc')\g
INSERT INTO Lec159304 VALUES ('T12','T122','intermed','etc')\g
INSERT INTO Lec159304 VALUES ('T12','P43333','bottom','etc')\g
INSERT INTO Lec159304 VALUES ('T121','P44333','bottom','etc')\g
INSERT INTO Lec159304 VALUES ('T121','P45333','bottom','etc')\g
INSERT INTO Lec159304 VALUES ('T122','P46333','bottom','etc')\g
INSERT INTO Lec159304 VALUES ('T2','T21','top','etc')\g
INSERT INTO Lec159304 VALUES ('T21','P47333','bottom','etc')\g
INSERT INTO Lec159304 VALUES ('T21','P48333','bottom','etc')\g

#
# 7. Insert data for table 'Lec159403'
#

DELETE FROM Lec159403\g

INSERT INTO Lec159403 VALUES ('S1','S11','top','Introduction')\g
INSERT INTO Lec159403 VALUES ('S1','S12','top','More details')\g
INSERT INTO Lec159403 VALUES ('S11','P51444','bottom','Cache(1)')\g
INSERT INTO Lec159403 VALUES ('S11','P52444','bottom','Cache(2)')\g
INSERT INTO Lec159403 VALUES ('S12','S121','intermed','etc')\g
INSERT INTO Lec159403 VALUES ('S12','S122','intermed','etc')\g
INSERT INTO Lec159403 VALUES ('S12','P53444','bottom','etc')\g
INSERT INTO Lec159403 VALUES ('S121','P54444','bottom','etc')\g
INSERT INTO Lec159403 VALUES ('S121','P55444','bottom','etc')\g
INSERT INTO Lec159403 VALUES ('S122','P56444','bottom','etc')\g
INSERT INTO Lec159403 VALUES ('S2','S21','top','etc')\g
INSERT INTO Lec159403 VALUES ('S21','P57444','bottom','etc')\g
INSERT INTO Lec159403 VALUES ('S21','P58444','bottom','etc')\g

#
#8. Insert data for table 'Lec159703'
#

DELETE FROM Lec159703\g

INSERT INTO Lec159703 VALUES ('S1','S11','top','Introduction')\g
INSERT INTO Lec159703 VALUES ('S1','S12','top','More details')\g
INSERT INTO Lec159703 VALUES ('S11','P61555','bottom','Cache(1)')\g
INSERT INTO Lec159703 VALUES ('S11','P62555','bottom','Cache(2)')\g
INSERT INTO Lec159703 VALUES ('S12','S121','intermed','etc')\g
```

```
INSERT INTO Lec159703 VALUES ('S12','S122','intermed','etc')\g
INSERT INTO Lec159703 VALUES ('S12','P63555','bottom','etc')\g
INSERT INTO Lec159703 VALUES ('S121','P64555','bottom','etc')\g
INSERT INTO Lec159703 VALUES ('S121','P65555','bottom','etc')\g
INSERT INTO Lec159703 VALUES ('S122','P66555','bottom','etc')\g
INSERT INTO Lec159703 VALUES ('S2','S21','top','etc')\g
INSERT INTO Lec159703 VALUES ('S21','P67555','bottom','etc')\g
INSERT INTO Lec159703 VALUES ('S21','P68555','bottom','etc')\g

#
#9. Insert data for table 'Courses'
#
DELETE FROM Courses\g

INSERT INTO Courses VALUES ('159201','Algorithms and Data Structures',
'crjessho','Lec159201')\g
INSERT INTO Courses VALUES ('159102','Computer Science Fundamentals',
'gmoret','Lec159102')\g
INSERT INTO Courses VALUES ('159203','Computer Systems',
'plyons','Lec159203')\g
INSERT INTO Courses VALUES ('159304','Architecture and Networks',
'crjessho', 'Lec159304')\g
INSERT INTO Courses VALUES ('159403','Advanced Computer Systems',
'crjessho', 'Lec159403')\g
INSERT INTO Courses VALUES ('159404','Systems programming', 'gmoret',
'null')\g
INSERT INTO Courses VALUES ('159420','Computer Science Engineering
Projects', 'plyons', 'null')\g
INSERT INTO Courses VALUES ('159703','Advanced Computer Systems',
'crjessho', 'Lec159703')\g
INSERT INTO Courses VALUES ('159704','Systems Programming', 'gmoret',
'null')\g
INSERT INTO Courses VALUES ('159711','Visual Languages', 'plyons',
'null')\g

#
# 10. Insert data for table 'Course_Contributor'
#
DELETE FROM Course_Contributor\g

INSERT INTO Course_Contributor VALUES ('159201', 'crjessho')\g
INSERT INTO Course_Contributor VALUES('159201', 'kdgrant')\g
INSERT INTO Course_Contributor VALUES('159102', 'gmoret')\g
INSERT INTO Course_Contributor VALUES ('159102', 'araman')\g
INSERT INTO Course_Contributor VALUES('159102', 'plyons')\g
INSERT INTO Course_Contributor VALUES('159102', 'etrich')\g
INSERT INTO Course_Contributor VALUES('159203', 'gmoret')\g
INSERT INTO Course_Contributor VALUES('159203', 'johudson')\g
INSERT INTO Course_Contributor VALUES ('159304','crjessho')\g
INSERT INTO Course_Contributor VALUES ('159304', 'plyons')\g
INSERT INTO Course_Contributor VALUES('159403', 'crjessho')\g
INSERT INTO Course_Contributor VALUES('159404', 'gmoret')\g
INSERT INTO Course_Contributor VALUES ('159703', 'crjessho')\g
INSERT INTO Course_Contributor VALUES('159704', 'gmoret')\g
```

```
#
# 11. Insert data for table 'Course_Keyword'
#
DELETE FROM Course_Keyword\g

INSERT INTO Course_Keyword VALUES ('159403', 'Parallel computers')\g
INSERT INTO Course_Keyword VALUES ('159703', 'Distributed memory')\g
INSERT INTO Course_Keyword VALUES ('159102', 'Introductionay')\g

#
#12. Insert data for table 'Section_Keyword'
#
DELETE FROM Section_Keyword\g

INSERT INTO Section_Keyword VALUES ('S1', 'Parallel computers')\g
INSERT INTO Section_Keyword VALUES ('T1', 'Distributed memory')\g
INSERT INTO Section_Keyword VALUES ('A1', 'Introductionay')\g

#
# 13. Insert data for table 'Staff'
#
DELETE FROM Staff\g

INSERT INTO Staff VALUES ('crjessho','Professor','Chris','Jesshope',
'23wsknmdnbg')\g
INSERT INTO Staff VALUES ('gmoret','Mr','Giovanni','Moretti',
'12qwertyuiop')\g
INSERT INTO Staff VALUES ('plyons','Lecturer','Paul','Lyons',
'12asdfghjkl')\g
INSERT INTO Staff VALUES ('lalexand','Mr','Lindsay', 'Alexander',
'23zxcvbnm')\g
INSERT INTO Staff VALUES ('Issac','Dr','Issac','Fung',
'34qwertyuiop')\g
INSERT INTO Staff VALUES ('etrich','AProfessor','Elena','Trichina',
'34asdfghjkl')\g
INSERT INTO Staff VALUES ('cphillips','Dr','Chris','Phillips',
'12wsknmdnbg')\g
INSERT INTO Staff VALUES ('araman','Dr','Anand','Venkataraman',
'56qwertyuiop')\g
INSERT INTO Staff VALUES ('johudson','Dr','Joho','Hudson',
'45asdfghjkl')\g
INSERT INTO Staff VALUES ('kdgrant','Dr','Koryn', 'Grant',
'34zxcvbnm')\g

#
# 14. Insert data for table 'EditRd'
#
DELETE FROM EditRd\g

INSERT INTO EditRd VALUES ('jessho1234','P61555','crjessho', '01-Dec-
1999', '23:12:00', 'TRUE')\g
INSERT INTO EditRd VALUES ('jessho1235','P68555','crjessho','02-Dec-
1999', '00:24:00', 'FALSE')\g
```

A3 proj-dump.msql

```
#
# This is the dumping for dbproj database system
# Host: localhost    Database: dbproj
# Author: June Pan and Chris Jesshope  Last date:  21/05/2000
#--------------------------------------------------------------


#
# Table structure for table 'Slide_Keyword'
#
CREATE TABLE Slide_Keyword (
  Slide_id CHAR(10) NOT NULL,
  Keyword CHAR(20) NOT NULL
) \g


#
# Dumping data for table 'Slide_Keyword'
#

INSERT INTO Slide_Keyword  VALUES ('p1236','Cache')\g
INSERT INTO Slide_Keyword  VALUES ('p1236','Cache')\g
INSERT INTO Slide_Keyword  VALUES ('p1237','Introductory')\g

#
# Table structure for table 'Courses'
#
CREATE TABLE Courses (
  Course_id CHAR(10) NOT NULL,
  Course_name CHAR(50),
  Controller CHAR(30),
  Mapping_table CHAR(10)
) \g

#
# Dumping data for table 'Courses'
#

INSERT INTO Courses  VALUES ('159201','Algorithms and Data
Structures','crjessho','Lec159201')\g
INSERT INTO Courses  VALUES ('159102','Computer Science
Fundamentals','gmoret','Lec159102')\g
INSERT INTO Courses  VALUES ('159203','Computer
Systems','plyons','Lec159203')\g
INSERT INTO Courses  VALUES ('159304','Architecture and
Networks','crjessho','Lec159304')\g
INSERT INTO Courses  VALUES ('159403','Advanced Computer
Systems','crjessho','Lec159403')\g
INSERT INTO Courses  VALUES ('159404','Systems
programming','gmoret','null')\g
INSERT INTO Courses  VALUES ('159420','Computer Science Engineering
Projects','plyons','null')\g
```

```
INSERT INTO Courses  VALUES ('159703','Advanced Computer
Systems','crjessho','Lec159703')\g
INSERT INTO Courses  VALUES ('159704','Systems
Programming','gmoret','null')\g
INSERT INTO Courses  VALUES ('159711','Visual
Languages','plyons','null')\g

#
# Table structure for table 'Course_Contributor'
#
CREATE TABLE Course_Contributor (
  Course_id CHAR(10) NOT NULL,
  Contributor CHAR(20) NOT NULL
) \g

#
# Dumping data for table 'Course_Contributor'
#

INSERT INTO Course_Contributor  VALUES ('159201','crjessho')\g
INSERT INTO Course_Contributor  VALUES ('159201','kdgrant')\g
INSERT INTO Course_Contributor  VALUES ('159102','gmoret')\g
INSERT INTO Course_Contributor  VALUES ('159102','araman')\g
INSERT INTO Course_Contributor  VALUES ('159102','plyons')\g
INSERT INTO Course_Contributor  VALUES ('159102','etrich')\g
INSERT INTO Course_Contributor  VALUES ('159203','gmoret')\g
INSERT INTO Course_Contributor  VALUES ('159203','johudson')\g
INSERT INTO Course_Contributor  VALUES ('159304','crjessho')\g
INSERT INTO Course_Contributor  VALUES ('159304','plyons')\g
INSERT INTO Course_Contributor  VALUES ('159403','crjessho')\g
INSERT INTO Course_Contributor  VALUES ('159404','gmoret')\g
INSERT INTO Course_Contributor  VALUES ('159703','crjessho')\g
INSERT INTO Course_Contributor  VALUES ('159704','gmoret')\g

#
# Table structure for table 'Slides'
#
CREATE TABLE Slides (
  Slide_id CHAR(10) NOT NULL,
  Creator CHAR(20),
  Slide_name CHAR(50),
  Dura_time INT,
  K_bytes INT,
  Lecture_url CHAR(100),
  Html_aep_url CHAR(50)
) \g

CREATE UNIQUE  INDEX Slide_idx ON Slides (Slide_id) \g

#
# Dumping data for table 'Slides'
#

INSERT INTO Slides  VALUES ('P11000','crjessho','The
problem',104,253,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch/audiograph/Introduction.%C4/Intro0
01.htm','http://html.aep')\g
```

```
INSERT INTO Slides  VALUES
('P12000','kdgrant','Background',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P13000','crjessho','Cache(1)',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P14000','kdgrant','Cache(2)',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P15000','crjessho','process',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P16000','kdgrant','Multithreads',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P17000','crjessho','Advance',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES ('P18000','kdgrant','Computer
System',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES ('P21111','plyons','The
problem',104,253,'http://www-
ist.massey.ac.nz/~crjessho/comp_archaudiograph/Introduction.%C4/Intro00
1.htm','http://html.aep')\g
INSERT INTO Slides  VALUES
('P28111','gmoret','Background',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES ('P31222','plyons','The
problem',104,253,'http://www-
ist.massey.ac.nz/~crjessho/comp_archaudiograph/Introduction.%C4/Intro00
1.htm','http://html.aep')\g
INSERT INTO Slides  VALUES
('P38222','gmoret','Background',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES ('P41333','plyons','The
problem',104,253,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch/audiograph/Introduction.%C4/Intro0
01.htm','http://html.aep')\g
INSERT INTO Slides  VALUES
('P48333','crjessho','Background',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P51444','crjessho','problem',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch/audiograph/Introduction.%C4/Intro0
01.htm','http://html.aep')\g
INSERT INTO Slides  VALUES
('P58444','crjessho','Background',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P61555','crjessho','problem',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch/audiograph/Introduction.%C4/Intro0
01.htm','http://html.aep')\g
INSERT INTO Slides  VALUES
('P62555','crjessho','Background',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
```

```
INSERT INTO Slides  VALUES
('P63555','crjessho','Cache(1)',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P64555','crjessho','Cache(2)',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P65555','crjessho','process',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P66555','crjessho','Multithreads',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P67555','crjessho','Advance',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g
INSERT INTO Slides  VALUES
('P68555','crjessho','System',233,556,'http://www-
ist.massey.ac.nz/~crjessho/comp_arch','http://html.aep')\g

#
# Table structure for table 'Lec159201'
#
CREATE TABLE Lec159201 (
  Section_id CHAR(10) NOT NULL,
  Subsec_id CHAR(10),
  Type CHAR(10),
  Subsection_name CHAR(50)
) \g

#
# Dumping data for table 'Lec159201'
#

INSERT INTO Lec159201  VALUES ('A1','A11','top','Introduction to
caches')\g
INSERT INTO Lec159201  VALUES ('A1','A12','top','More details')\g
INSERT INTO Lec159201  VALUES ('A11','P11000','bottom','Cache(1)')\g
INSERT INTO Lec159201  VALUES ('A11','P12000','bottom','Cache(2)')\g
INSERT INTO Lec159201  VALUES ('A12','A121','intermed','etc')\g
INSERT INTO Lec159201  VALUES ('A12','A122','intermed','etc')\g
INSERT INTO Lec159201  VALUES ('A12','P13000','bottom','etc')\g
INSERT INTO Lec159201  VALUES ('A121','P14000','bottom','etc')\g
INSERT INTO Lec159201  VALUES ('A121','P15000','bottom','etc')\g
INSERT INTO Lec159201  VALUES ('A122','P16000','bottom','etc')\g
INSERT INTO Lec159201  VALUES ('A2','A21','top','etc')\g
INSERT INTO Lec159201  VALUES ('A21','P17000','bottom','etc')\g
INSERT INTO Lec159201  VALUES ('A21','P18000','bottom','etc')\g

#
# Table structure for table 'EditRd'
#
CREATE TABLE EditRd (
  EditRd_id CHAR(10) NOT NULL,
  Slide_id CHAR(10),
  Editor CHAR(10),
  Edit_Date DATE,
  Edit_time TIME,
```

```
  Commit CHAR(10)
) \g

CREATE UNIQUE  INDEX EditRd_idx ON EditRd (
      EditRd_id
) \g


#
# Dumping data for table 'EditRd'
#

INSERT INTO EditRd  VALUES ('jessho1234','P61555','crjessho','01-Dec-
1999','23:12:00','TRUE')\g
INSERT INTO EditRd  VALUES ('jessho1235','P68555','crjessho','02-Dec-
1999','00:24:00','FALSE')\g

#
# Table structure for table 'Course_Keyword'
#
CREATE TABLE Course_Keyword (
  Course_id CHAR(10) NOT NULL,
  Keyword CHAR(20) NOT NULL
) \g

#
# Dumping data for table 'Course_Keyword'
#

INSERT INTO Course_Keyword  VALUES ('159403','Parallel computers')\g
INSERT INTO Course_Keyword  VALUES ('159703','Distributed memory')\g
INSERT INTO Course_Keyword  VALUES ('159102','Introductionay')\g

#
# Table structure for table 'Staff'
#
CREATE TABLE Staff (
  Staff_id CHAR(10) NOT NULL,
  Title CHAR(10),
  First_name CHAR(10),
  Last_name CHAR(15),
  Password CHAR(20)
) \g

CREATE UNIQUE  INDEX Staff_idx ON Staff (Staff_id) \g


#
# Dumping data for table 'Staff'
#

INSERT INTO Staff  VALUES
('crjessho','Professor','Chris','Jesshope','23wsknmdnbg')\g
INSERT INTO Staff  VALUES
('gmoret','Mr','Giovanni','Moretti','12qwertyuiop')\g
INSERT INTO Staff  VALUES
('plyons','Lecturer','Paul','Lyons','12asdfghjkl')\g
```

```
INSERT INTO Staff  VALUES
('lalexand','Mr','Lindsay','Alexander','23zxcvbnm')\g
INSERT INTO Staff  VALUES
('Issac','Dr','Issac','Fung','34qwertyuiop')\g
INSERT INTO Staff  VALUES
('etrich','AProfessor','Elena','Trichina','34asdfghjkl')\g
INSERT INTO Staff  VALUES
('cphillips','Dr','Chris','Phillips','12wsknmdnbg')\g
INSERT INTO Staff  VALUES
('araman','Dr','Anand','Venkataraman','56qwertyuiop')\g
INSERT INTO Staff  VALUES
('johudson','Dr','Joho','Hudson','45asdfghjkl')\g
INSERT INTO Staff  VALUES
('kdgrant','Dr','Koryn','Grant','34zxcvbnm')\g

#
# Table structure for table 'Lec159102'
#
CREATE TABLE Lec159102 (
  Section_id CHAR(10) NOT NULL,
  Subsec_id CHAR(10),
  Type CHAR(10),
  Subsection_name CHAR(50)
) \g


#
# Dumping data for table 'Lec159102'
#

INSERT INTO Lec159102  VALUES ('A1','A11','top','Introduction to
caches')\g
INSERT INTO Lec159102  VALUES ('A1','A12','top','More details')\g
INSERT INTO Lec159102  VALUES ('A11','P21111','bottom','Cache(1)')\g
INSERT INTO Lec159102  VALUES ('A11','P22111','bottom','Cache(2)')\g
INSERT INTO Lec159102  VALUES ('A12','A121','intermed','etc')\g
INSERT INTO Lec159102  VALUES ('A12','A122','intermed','etc')\g
INSERT INTO Lec159102  VALUES ('A12','P23111','bottom','etc')\g
INSERT INTO Lec159102  VALUES ('A121','P24111','bottom','etc')\g
INSERT INTO Lec159102  VALUES ('A121','P25111','bottom','etc')\g
INSERT INTO Lec159102  VALUES ('A122','P26111','bottom','etc')\g
INSERT INTO Lec159102  VALUES ('A2','A21','top','etc')\g
INSERT INTO Lec159102  VALUES ('A21','P27111','bottom','etc')\g
INSERT INTO Lec159102  VALUES ('A21','P28111','bottom','etc')\g

#
# Table structure for table 'Section_Keyword'
#
CREATE TABLE Section_Keyword (
  Section_id CHAR(10) NOT NULL,
  Keyword CHAR(20) NOT NULL
) \g
```

```
#
# Dumping data for table 'Section_Keyword'
#

INSERT INTO Section_Keyword  VALUES ('S1','Parallel computers')\g
INSERT INTO Section_Keyword  VALUES ('T1','Distributed memory')\g
INSERT INTO Section_Keyword  VALUES ('A1','Introductionay')\g

#
# Table structure for table 'Lec159203'
#
CREATE TABLE Lec159203 (
  Section_id CHAR(10) NOT NULL,
  Subsec_id CHAR(10),
  Type CHAR(10),
  Subsection_name CHAR(50)
) \g

#
# Dumping data for table 'Lec159203'
#

INSERT INTO Lec159203  VALUES ('A1','A11','top','Introduction to
caches')\g
INSERT INTO Lec159203  VALUES ('A1','A12','top','More details')\g
INSERT INTO Lec159203  VALUES ('A11','P31222','bottom','Cache(1)')\g
INSERT INTO Lec159203  VALUES ('A11','P32222','bottom','Cache(2)')\g
INSERT INTO Lec159203  VALUES ('A12','A121','intermed','etc')\g
INSERT INTO Lec159203  VALUES ('A12','A122','intermed','etc')\g
INSERT INTO Lec159203  VALUES ('A12','P33222','bottom','etc')\g
INSERT INTO Lec159203  VALUES ('A121','P34222','bottom','etc')\g
INSERT INTO Lec159203  VALUES ('A121','P35222','bottom','etc')\g
INSERT INTO Lec159203  VALUES ('A122','P36222','bottom','etc')\g
INSERT INTO Lec159203  VALUES ('A2','A21','top','etc')\g
INSERT INTO Lec159203  VALUES ('A21','P37222','bottom','etc')\g
INSERT INTO Lec159203  VALUES ('A21','P38222','bottom','etc')\g

#
# Table structure for table 'Lec159304'
#
CREATE TABLE Lec159304 (
  Section_id CHAR(10) NOT NULL,
  Subsec_id CHAR(10),
  Type CHAR(10),
  Subsection_name CHAR(50)
) \g


#
# Dumping data for table 'Lec159304'
#

INSERT INTO Lec159304  VALUES ('T1','T11','top','Introduction')\g
INSERT INTO Lec159304  VALUES ('T1','T12','top','More details')\g
INSERT INTO Lec159304  VALUES ('T11','P41333','bottom','Cache(1)')\g
INSERT INTO Lec159304  VALUES ('T11','P42333','bottom','Cache(2)')\g
INSERT INTO Lec159304  VALUES ('T12','T121','intermed','etc')\g
```

```
INSERT INTO Lec159304  VALUES ('T12','T122','intermed','etc')\g
INSERT INTO Lec159304  VALUES ('T12','P43333','bottom','etc')\g
INSERT INTO Lec159304  VALUES ('T121','P44333','bottom','etc')\g
INSERT INTO Lec159304  VALUES ('T121','P45333','bottom','etc')\g
INSERT INTO Lec159304  VALUES ('T122','P46333','bottom','etc')\g
INSERT INTO Lec159304  VALUES ('T2','T21','top','etc')\g
INSERT INTO Lec159304  VALUES ('T21','P47333','bottom','etc')\g
INSERT INTO Lec159304  VALUES ('T21','P48333','bottom','etc')\g

#
# Table structure for table 'Lec159403'
#
CREATE TABLE Lec159403 (
  Section_id CHAR(10) NOT NULL,
  Subsec_id CHAR(10),
  Type CHAR(10),
  Subsection_name CHAR(50)
) \g


#
# Dumping data for table 'Lec159403'
#

INSERT INTO Lec159403  VALUES ('S1','S11','top','Introduction')\g
INSERT INTO Lec159403  VALUES ('S1','S12','top','More details')\g
INSERT INTO Lec159403  VALUES ('S11','P51444','bottom','Cache(1)')\g
INSERT INTO Lec159403  VALUES ('S11','P52444','bottom','Cache(2)')\g
INSERT INTO Lec159403  VALUES ('S12','S121','intermed','etc')\g
INSERT INTO Lec159403  VALUES ('S12','S122','intermed','etc')\g
INSERT INTO Lec159403  VALUES ('S12','P53444','bottom','etc')\g
INSERT INTO Lec159403  VALUES ('S121','P54444','bottom','etc')\g
INSERT INTO Lec159403  VALUES ('S121','P55444','bottom','etc')\g
INSERT INTO Lec159403  VALUES ('S122','P56444','bottom','etc')\g
INSERT INTO Lec159403  VALUES ('S2','S21','top','etc')\g
INSERT INTO Lec159403  VALUES ('S21','P57444','bottom','etc')\g
INSERT INTO Lec159403  VALUES ('S21','P58444','bottom','etc')\g

#
# Table structure for table 'Lec159703'
#
CREATE TABLE Lec159703 (
  Section_id CHAR(10) NOT NULL,
  Subsec_id CHAR(10),
  Type CHAR(10),
  Subsection_name CHAR(50)
) \g


#
# Dumping data for table 'Lec159703'
#

INSERT INTO Lec159703  VALUES ('S1','S11','top','Introduction')\g
INSERT INTO Lec159703  VALUES ('S1','S12','top','More details')\g
INSERT INTO Lec159703  VALUES ('S11','P61555','bottom','Cache(1)')\g
INSERT INTO Lec159703  VALUES ('S11','P62555','bottom','Cache(2)')\g
```

```
INSERT INTO Lec159703  VALUES ('S12','S121','intermed','etc')\g
INSERT INTO Lec159703  VALUES ('S12','S122','intermed','etc')\g
INSERT INTO Lec159703  VALUES ('S12','P63555','bottom','etc')\g
INSERT INTO Lec159703  VALUES ('S121','P64555','bottom','etc')\g
INSERT INTO Lec159703  VALUES ('S121','P65555','bottom','etc')\g
INSERT INTO Lec159703  VALUES ('S122','P66555','bottom','etc')\g
INSERT INTO Lec159703  VALUES ('S2','S21','top','etc')\g
INSERT INTO Lec159703  VALUES ('S21','P67555','bottom','etc')\g
INSERT INTO Lec159703  VALUES ('S21','P68555','bottom','etc')\g

#
# Table structure for table 'Lec159101'
#
CREATE TABLE Lec159101 (
  Section_id CHAR(10) NOT NULL,
  Subsec_id CHAR(10),
  Type CHAR(10),
  Subsection_name CHAR(50)
) \g

#
# Dumping data for table 'Lec159101'
#

INSERT INTO Lec159101  VALUES ('A1','A11','top','Introduction to
caches')\g
INSERT INTO Lec159101  VALUES ('A1','A12','top','More details')\g
INSERT INTO Lec159101  VALUES ('A11','P1','bottom','Cache(1)')\g
INSERT INTO Lec159101  VALUES ('A11','P2','bottom','Cache(2)')\g
INSERT INTO Lec159101  VALUES ('A12','A121','intermed','etc')\g
INSERT INTO Lec159101  VALUES ('A12','A122','intermed','etc')\g
INSERT INTO Lec159101  VALUES ('A12','P3','bottom','etc')\g
INSERT INTO Lec159101  VALUES ('A121','P4','bottom','etc')\g
INSERT INTO Lec159101  VALUES ('A121','P5','bottom','etc')\g
INSERT INTO Lec159101  VALUES ('A122','P6','bottom','etc')\g
INSERT INTO Lec159101  VALUES ('A2','A21','top','etc')\g
INSERT INTO Lec159101  VALUES ('A21','P7','bottom','etc')\g
INSERT INTO Lec159101  VALUES ('A21','P8','bottom','etc')\g
```

Part B Client Application Java Classes

Main Class: ProcessAudioGraph.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import interfacepg.*;
import utilitypg.*;
import databasepg.*;

/**********************************************************************
 * This project is to develop a msql_jdbc database and
 * interface to store courseware and information of the
 * computer schence of Massey University.
 * ---------------------------------------------------------------
 * include package: (1) jdk 1.1.8; (2)msql-jdbc-1.1b1.jar;
 *                  (3) collections,jar; (4)swingall.jar.
 * ---------------------------------------------------------------
 * $author: June Pan          $supervisor: Chris Jesshope
 * Last modified $Date: 20/07/2000 10:02:44
 **********************************************************************/

public class ProcessAudioGraph extends JFrame
{

  public static void main(String[] args)
  {

    //create a main Interface for a user interaction with this system

         new JmainInterface();
  }


}
```

B1 Database Package

Class 1: Connect.java

```
package databasepg;
import java.sql.*;
import java.util.*;

 public class Connec
 {
  private Connection con;

  //the administrator information for connection the database server

  private final String url =
                  "jdbc:msql://fims-rimutaka.massey.ac.nz:1114/dbproj";
  private final String user = "user";
  private final String userName = "pjun";
  private final String driverName =
                  "com.imaginary.sql.msql.MsqlDriver";
  private final String password = "";

  Statement stmt;

  public final void Connec()        // connect to the database server
  {
       try{
                  Properties p = new Properties();
            p.put(user, userName);
            Class.forName(driverName);
            con = DriverManager.getConnection(url, userName, password);
        }
        catch( Exception e ) {
          e.printStackTrace();
        }
  }

  public final Connection getConHandle()  // get connection handle
  {

         return con;
  }

  public final void closeToServer()        //close the database sever
  {
    try {
       con.close();
    }
    catch( Exception e ) {
          e.printStackTrace();
    }
  }

}
```

Class 2: LoginInformation.java

```
package databasepg;
import java.sql.*;
import java.util.*;

public class LoginInformation extends Connec
{

   private static String loginStaff;  //the user's name
   private boolean found = false;

 public boolean checkAuthoring(String sid, String pwd)
 {

    try {
           Connection con = getConHandle();
           Statement stmt = con.createStatement();
           ResultSet rs = stmt.executeQuery("SELECT Staff_id, Password"
                          + "FROM Staff ORDER BY Staff_id");
           while (rs.next()){
              if(sid.equals(rs.getString(1))){
                if(pwd.equals(rs.getString(2))){
                   loginStaff = new String(sid);
                   found= true;
                   stmt.close();
                   rs.close();
                }
              }
           }
        }
        catch( Exception e ) {
            e.printStackTrace();

        }
        return found;
 }


 public final String getLoginStaff()       //get login staff's name
 {
    return loginStaff;
 }


}
```

Class3: AllStaffInformation.java

```
package databasepg;
import java.sql.*;
import java.util.*;

public class AllStaffInformation
                        extends Connec
{
  private String sid = new String();//staff's id
  private String stitle;       //a staff title
  private String sname;  //a staff name
  private Vector allStaffFirstname=new Vector();

  private Vector allStaffLastname=new Vector();

 public void getAllStaffName()
 {
    try{ Connection con = getConHandle();
         Statement stmt;
         ResultSet rs;
         stmt = con.createStatement();
         rs = stmt.executeQuery(
         "SELECT First_name, Last_name
                      "+"FROM Staff");
         while( rs.next() ) {
                 //loop for result set
         allStaffFirstname.addElement
                    (rs.getString(1));
         allStaffLastname.addElement
                    (rs.getString(2));
         }
            stmt.close();
            rs.close();
        }
        catch( Exception e ) {
          e.printStackTrace();
        }
 }

 public String getStaffNo(String sname)
 {
    try{  Connection con = getConHandle();
          Statement stmt;
          ResultSet rs;
          stmt = con.createStatement();
          rs = stmt.executeQuery
             ("SELECT Staff_id,
               Title,First_name, Last_name
               "+"FROM Staff ORDER BY
               Staff_id");
          while( rs.next() ) {
          if(sname.equals
            (rs.getString(3)+" "
             +rs.getString(4))){
               sid = rs.getString(1);
               stmt.close();
               rs.close();
               return sid;
            }
         }
            stmt.close();
            rs.close();
        }
           catch( Exception e ) {
          e.printStackTrace();
        }
        return sid;
 }

 public String getSid()   //get staff identity

 {
    return sid;
 }

 public String getStitle(String sNo)
                              //get staff title
 {
   try{    Connection con =
                 getConHandle();
           Statement stmt;
           ResultSet rs;
           stmt = con.createStatement();
           rs = stmt.executeQuery
                ("SELECT Staff_id, Title
                 "+"FROM Staff ORDER BY
                 Staff_id");
         while( rs.next() ) {
         if(sNo.equals(rs.getString(1))){
                this.stitle = new String(
                rs.getString(2));
                stmt.close();
                rs.close();
            }
         }
        }
        catch( Exception e ) {
           e.printStackTrace();
        }
         return stitle;
 }

 public String getSname(String sNo)
{
  try { Connection con = getConHandle();
        Statement stmt;
        ResultSet rs;
        stmt = con.createStatement();
        rs = stmt.executeQuery("SELECT
              Staff_id,First_name,
              Last_name "+"FROM
              Staff ORDER BY Staff_id");
        while( rs.next() ) {
        if(sNo.equals(rs.getString(1))){
           this.sname = new
           String(rs.getString(2)+
           " "+rs.getString(3));
               stmt.close();
               rs.close();
            }
           }
        }
        catch( Exception e ) {
          e.printStackTrace();
        }
        return sname;
 }

  public Vector getAllStaffFirstname()
  {
   return allStaffFirstname;
  }

  public Vector getAllStaffLastname()
  {
   return allStaffLastname;
  }
}
```

Class 4: AllStaffInformation.java

```
package databasepg;
import java.sql.*;
import java.util.*;

public class AllCourseInformation  extends Connec
{
  private Vector cid = new Vector(); //course identity
  private Vector cname = new Vector();       //course name
  public AllCourseInformation(){}

 public void getAllCourseList()
 {
    try {
          Connection con = getConHandle();   //get connection handle
          Statement stmt;
          ResultSet rs;
          stmt = con.createStatement();
          rs = stmt.executeQuery("SELECT Course_id, Course_name "+
                                 "FROM Courses ORDER BY Course_id");
          while( rs.next() ) {
             cid.addElement(rs.getString(1));
             cname.addElement(rs.getString(2));
          }
            stmt.close();
            rs.close();
        }
        catch( Exception e ) {
          e.printStackTrace();
        }
  }

  public String getTableName(String courseId)
  {                                          //get a lecture table name
      String ct = new String();
      try{
            Connection con = getConHandle();
            Statement stmt;
            ResultSet rs;
            stmt = con.createStatement();
            rs = stmt.executeQuery("SELECT Course_id,Mapping_table "+
                                   "FROM Courses ORDER BY Course_id");
            while(rs.next()){
              if(courseId.equals(rs.getString(1))){
                 ct = rs.getString(2);
              }
            }
              stmt.close();
              rs.close();
      }catch(Exception e ) {
            e.printStackTrace();
      }
      return ct;
 }

 public Vector getCidList()          //get the list of course's identity
 {
      return cid;
 }

 public Vector getCnameList()         //get the list of course's name
 {
      return cname;
 }
}
```

Class 5: UserCourseInformation.java

```
package databasepg;
import java.net.URL;
import java.sql.*;
import java.util.*;


public class UserCourseInformation
                              extends Connec
{
  private String staffid;
                          //user's identity
  private Vector cid1 = new Vector();
                    //controller course ID
  private Vector cname1 = new Vector();
                  //controller course name
  private Vector cid2 = new Vector();
                    //contributor course ID
  private Vector cname2 = new Vector();
                 //contributor course name

  public UserCourseInformation(String
                                    staffid)
  {
    this.staffid = staffid;
  }

 public void getUserCourseAsController()
    //get the course user is a controller
 {
    try {
          Connection con =
          getConHandle();
                 // get connection handle
           Statement stmt;
           ResultSet rs;

            stmt = con.createStatement();
            rs = stmt.executeQuery("SELECT
       Course_id, Course_name, Controller
        "+"FROM Courses ORDER BY
                        Course_id");
     while( rs.next() ) {

     if(staffid.equals(rs.getString(3))){
      cid1.addElement(rs.getString(1));
      cname1.addElement(rs.getString(2));
     }
      }
      stmt.close();
      rs.close();
     }
           catch( Exception e ) {
          e.printStackTrace();
        }
}

public Vector getcourseIdList()
    //get the list of course's identities
{
    Vector courseId = new Vector();
    try {
          Connection con =
                            getConHandle();
           Statement stmt;
           ResultSet rs;
            stmt = con.createStatement();
            rs = stmt.executeQuery("SELECT
                 Course_id, Contributor "+
               "FROM Course_Contributor");
    while( rs.next() ) {
    if(staffid.equals(rs.getString(2))){
    courseId.addElement(rs.getString(1));

    }
}
                    rs.close();
                    stmt.close();
                    return courseId;
                }
                 catch( Exception e ) {
                 e.printStackTrace();
                }
                return courseId;

}

public void getUserCourseAsContributor()

//get the user's course as a contributor
{
   try {
          Vector courseNo =
                getcourseIdList();
          Connection con =
                getConHandle();
          Statement stmt;
           ResultSet rs;
          stmt = con.createStatement();
          rs = stmt.executeQuery("SELECT
               Course_id, Course_name "+
               "FROM Courses ORDER BY
                Course_id");
  while( rs.next() ) {
   for(int i=0; i<courseNo.size(); i++){

   if(courseNo.elementAt(i).equals
     (rs.getString(1)))
   {

     cid2.addElement(rs.getString(1));

     cname2.addElement(rs.getString(2));

     courseNo.removeElementAt(i);
   }
  }

  }
  stmt.close();
           rs.close();
  }
          catch( Exception e ) {
         e.printStackTrace();
  }
 }

public Vector getCid1List(){
//get the course's identity list that
//user are as controller
     return cid1;
}

public Vector getCname1List(){
//get the course's name list that user
//are as controller
     return cname1;
}

public Vector getCid2List(){
//get the course's identity list that
//user are as contributor
     return cid2;
  }

public Vector getCname2List(){
//get the course name list that user
//are as contributor
     return cname2;
  }
}
```

Class 6: AcourseInformation.java

```
package databasepg;
import java.sql.*;
import java.util.*;
import javax.swing.tree.*;
//import ULogin.*;

public class ACourseInformation extends
Connec{

  private String ctlname = new String();
       //name of the control
  private Vector ctbname = new Vector();
       //name list of the contributors
  private String courseId;
              //course identity
  private Vector secid = new Vector();
     //the section identity of the course
  private Vector subid = new Vector();
  //the subsection identity of the course
  private Vector loctype = new Vector();
       //the location type of the course
  private Vector subname = new Vector();
      //the sebsection name of the course

public void getCourseContent(String tabId)
{
   try{
       Connection con = getConHandle();
       Statement stmt =
                    con.createStatement();
       ResultSet rs =
       stmt.executeQuery("SELECT
              Section_id, Subsec_id, Type,
              Subsection_name "+
               "FROM "+tabId+" ORDER BY
                 Section_id");
     while(rs.next()) {
        secid.addElement(rs.getString(1));
        subid.addElement(rs.getString(2));
      loctype.addElement(rs.getString(3));
      subname.addElement(rs.getString(4));
      }
    stmt.close();
    rs.close();
   }
   catch( Exception e ) {
          e.printStackTrace();

   }

 }

  public Vector getSecid()
       //get section identity
  {
    return secid;
  }

  public Vector getSubid()
       //get subsection identity
  {
    return subid;
  }

  public Vector getLoctype()
   //get loction type (bottom or top)
  {
    return loctype;
  }

public Vector getSubname()
              //get subsection name
  {
    return subname;
  }

public String getControllerOftheCourse(
      String cid)   //get controller name
{
    try {
          Connection con =
                      getConHandle();
          Statement stmt =
                con.createStatement();
          ResultSet rs =
          stmt.executeQuery("SELECT
          Course_id, Controller "+"
          FROM Courses ORDER
                   BY Course_id");

    while( rs.next() ) {

         if(cid.equals(rs.getString(1)))
         {
            ctlname = rs.getString(2);
            stmt.close();
            rs.close();
            return ctlname;
          }
      }

    }
    catch( Exception e ) {
          e.printStackTrace();
    }
        return ctlname;
}

public Vector getContributorOftheCourse
             (String cid)
     //get name list of the contributors
{
    try {
          Connection con =
                 getConHandle();
          Statement stmt =
                 con.createStatement();
          ResultSet rs =
                 stmt.executeQuery("SELECT
                 Course_id, Contributor "+
                 "FROM urse_Contributor");

         while( rs.next() ) {

          if(cid.equals(rs.getString(1)))
         {
     ctbname.addElement(rs.getString(2));
          }
       }
     stmt.close();
     rs.close();
    }
    catch( Exception e ) {
    e.printStackTrace();
    }
        return ctbname;
  }

 }
```

Class 7: AslideInformation.java

```
package databasepg;
import java.sql.*;
import java.util.*;
import javax.swing.tree.*;

public class ASlideInformation
                        extends Connec
{

  private boolean found = false;

  private String pid = new String();
              // presentation identity

  private String creator = new String();
            // author of the presentation

  private String pname = new String();
              // name of the presentation

  private int durationInSecond;
          // duration of the presentation

  private int sizeInKbytes;
          // Size of the presentation

  private String lurl = new String();
          // URL of the lecture

  private String hurl = new String();
          // URL of the html file

 public void getSlideContent(String
slideId)
{
    try{
         Connection con = getConHandle();
         Statement stmt;
         ResultSet rs;
         stmt = con.createStatement();
         rs = stmt.executeQuery("SELECT
              Slide_id,
              Creator,Slide_name,
              Dura_time, "+ "K_bytes,
              Lecture_url,
              Html_aep_url "+ "FROM
              Slides
              ORDER BY Slide_id");

          while(rs.next()) {
            pid =rs.getString(1);
            if(pid.equals(slideId))
            {
               found = true;
               creator =rs.getString(2);
               pname = rs.getString(3);
               durationInSecond =
                         rs.getInt(4);
               sizeInKbytes =
                         rs.getInt(5);
               lurl= rs.getString(6);
                  hurl= rs.getString(7);
            }
          }


           stmt.close();
           rs.close();
        }
        catch( Exception e ) {
            e.printStackTrace();

        }

 }

 public boolean getFoundSlide()

              //get found true or false
{
   return found;
 }

 public String getSlideCreator()
             //get author name of Slide
 {
   return creator;
 }

 public String getSlideName()
            //get the name of the Slide
 {
   return pname;
 }

 public String getLectureURL()
             //get URL of the Lecture
 {
   return lurl;
 }

 public String getLectureHTMLandAEPURL()
             //get URL of the lecture's
              //HTML file and AEP file
 {
   return hurl;
 }

}
```

B2 Utilities Package

Class 8: WindowUtilities.java

```
package utilitypg;
import javax.swing.*;
import java.awt.*;

/** A few utilities that simplify using
 *  windows in Swing.
 *  reference from Marty Hall, 1998-99
 *  http://www.apl.jhu.edu/~hall/java/
 *  Jun Pan     last modified: 05/21/2000
 */

public class WindowUtilities {

  /** Tell system to use native look and
   * feel, as in previous
   * releases. Metal (Java) LAF is the
   * default otherwise.
   */

public static void
       setNativeLookAndFeel()
 {
   try
   {
     UIManager.setLookAndFeel(UIManager.
      getSystemLookAndFeelClassName());
   } catch(Exception e){
     System.out.println(
     "Error setting native LAF: " + e);
   }

 }

public static void setJavaLookAndFeel()
 {
   try {
    UIManager.setLookAndFeel(UIManager.
    getCrossPlatformLookAndFeelClassName());
   } catch(Exception e) {
      System.out.println(
        "Error setting Java LAF: " + e);
   }
 }

 public static void setMotifLookAndFeel()
 {
   try {
         UIManager.setLookAndFeel(
         "com.sun.java.swing.plaf.
          motif.MotifLookAndFeel");

      } catch(Exception e) {
        System.out.println(
        "Error setting Motif LAF: " + e);
   }
}

/** A simplified way to see a JPanel or
 *  other Container.
 *  Pops up a JFrame with specified
 *  Container as the content pane.
 */

public static Jframe
openInJFrame(Container content,int width,
int height,String title, Color bgColor) {

    JFrame frame = new JFrame(title);

    frame.setBackground(bgColor);
    content.setBackground(bgColor);
    frame.setSize(width, height);
    frame.setContentPane(content);

    frame.addWindowListener(new
                        ExitListener());

    frame.setVisible(true);
    return(frame);

  }

/** Uses Color.white as the background
color. */

public static JFrame
openInJFrame(Container content,int width,
int height, String title)
{

   return(openInJFrame(content, width,
           height, title, Color.white));

}

/** Uses Color.white as the background
 *  color, and the
 *  name of the Container's class as
 *  the JFrame title.
 */

public static JFrame
openInJFrame(Container content,int width,
             int height)
{

    return(openInJFrame(content, width,
    height, content.getClass().getName(),

    Color.white));

  }

}
```

Class 9: ExitListener.java

```
package utilitypg;

import java.awt.*;

import java.awt.event.*;


/** A listener that I attach to the top-level Frame or JFrame of

 *  my application, so quitting the frame exits the application.

 */


public class ExitListener extends WindowAdapter {

  public void windowClosing(WindowEvent event) {

    System.exit(0);

  }

}
```

B3 Interface Package

Class 10: JmainInterface.java

```
package interfacepg;
import java.awt.*;
import java.sql.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import utilitypg.*;
import databasepg.*;


/** This main Interface to let a user:
 *  1) Login to this system;
 *  2) Get some info about this system;
 *  3) Exit from this system.
 */

public class JmainInterface extends
JFrame
{
  private ConnectDialog pd; // dialog box
  private String staffId = new String();
  private int LoginTime = 0;
  private Connec con;

  public JmainInterface()
  {
    super("The Main Interface");
    WindowUtilities.
                  setNativeLookAndFeel();
    addWindowListener(
                    new ExitListener());
    Container content = getContentPane();
    Font font = new Font("Serif",
                    Font.PLAIN, 30);
    content.setFont(font);
    String labelText =
      "<html><B><UL><FONT SIZE=4
       COLOR=PURPLE>Welcome to "+
         "AudioGraph Recorder
          Lecture</B></FONT> "+
         "<p>This system accesses a
          database and "+
         "interface to multimedia
          courseware for the computer " +
         "science courses of Massey
          University." + "</UL></html>";
    JLabel titleLabel =
          new JLabel(labelText,
    JLabel.CENTER);
    content.add(titleLabel,
            BorderLayout.NORTH);
    labelText =
      "<html><UL>" +
      "<FONT COLOR=GREEN><B>Major
        operations include:</FONT>"+
      "<p><LI>Login to this system and
        select some options;" +  "
       <LI>Searching some special
           context by keywords;" + "
       <LI>Finding links between recorded
           AudioGraphic material;" +   "
       <LI>Editing individual lecture
           notes; " +"
       <LI>Create links between recorded
          AutioGraphic material.</FONT>"+
       "</UL></html>";
    JLabel bodyLabel = new
                    JLabel(labelText,new
    ImageIcon("Images/
                AudiographButton.gif"),
    JLabel.CENTER);
    bodyLabel.setBorder(BorderFactory.createT
    itledBorder(""));
    content.add(bodyLabel,
    BorderLayout.CENTER);
    JPanel controlArea = new JPanel(new
    FlowLayout());
    controlArea.setBorder(BorderFactory.creat
    eTitledBorder("Authoring Choose"));
    Font nFont = new Font("Serif",
                          Font.PLAIN, 18);
    controlArea.setFont(nFont);
    controlArea.add(new Button("Log in"));
    controlArea.add(new Button("Close"));
    controlArea.add(new Button("Help"));
    content.add(controlArea,
    BorderLayout.SOUTH);
    con = new Connec();
    setSize(500, 350);
    setVisible(true);
    }

    public boolean action(Event
                          evt, Object arg)
    {
       if(arg.equals("Log in"))
       {
         LoginTime = LoginTime + 1;
         {
           connectInfo in = new
                        ConnectInfo("", "");
           pd = new ConnectDialog(this, in,
                                  LoginTime);
           pd.setLocation(20,300);
           pd.setLayout(new
           FlowLayout(FlowLayout.LEFT));
           pd.show();
         }
       }
       else if(arg.equals("Close"))
       {
          con.closeToServer();//system exits.
          System.exit(0);
       }
       else if(arg.equals("Help"))
       {
           // create a help information
          JHelpMenu wd = new
                          JHelpMenu(this);
          wd.setLocation(150,300);
          wd.getContentPane().setLayout(new
               FlowLayout(FlowLayout.LEFT));
          wd.show();
       }
       else dispose();
            return true;
    }

    public boolean handleEvent(Event evt)
    {
      if (evt.id == Event.WINDOW_DESTROY
            &&  evt.target == this)
          System.exit(0);
      else
          return super.handleEvent(evt);
          return true;
    }


}
```

Class 11: ConnectDialog.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import java.awt.Cursor.*;
import javax.swing.*;
import javax.swing.border.*;
import utilitypg.*;
import databasepg.*;

/** This is a dialog for a user typing
 *  username  and password.
 *  It is created by the user's
 *  login action of the Main Interface.
 */

class ConnectDialog extends Dialog {
 private JTextField username;
 private JPasswordField passwd;
 private JmainInterface parent;
 private String staffId = new String();
 private LoginInformation li;

 private static final int TryTime = 4;
   // the user can try 4 times to login
 private int tryLoginTime;

 ConnectDialog(JmainInterface  parent,
      ConnectInfo u, int tryLoginTime)
 {
   super(parent, "Connect", true);
   this.tryLoginTime = tryLoginTime;
   li = new LoginInformation();
   this.parent = parent;
   Panel p1 = new Panel();
   p1.setLayout(new GridLayout(2,2));

   p1.add(new Label("User name:"));
   p1.add(username = new
   JTextField(u.username, 8));
  // the maxLength of the user's name = 8

   p1.add(new Label("Password:"));
   p1.add(passwd = new
   JPasswordField(u.userpd,12));
     //the maxLength of the password = 12

   add("Center", p1);
   JPanel p2 = new JPanel();
   p2.setLayout(new GridLayout(1,3));
   p2.add(new Label(" "));
   p2.add(new Button("OK"));
   p2.add(new Button("Cancel"));
   add("South", p2);
   resize(240,120);
 }

 public boolean action(Event
                       evt, Object arg)
 {
   boolean authoring = false;
   ErrorMge em;
   if(arg.equals("OK"))  //ok button
   {
      if (tryLoginTime >= maxTime)
      {
           em = new ErrorMge(parent,
            "Sorry, you try too much.");
      em.setLocation(150,300);
      em.getContentPane().setLayout(new
          FlowLayout(FlowLayout.LEFT));
      em.show();

      try{
          //refuse the user enter the system

          li.closeToServer();
          dispose();
          }
          catch( Exception e ) {
              e.printStackTrace();
          }
      }
      else
            // TryTime < tryLoginTime
      {
        dispose();  // remove the dialog box
        parent.setCursor(new
           Cursor(Cursor.WAIT_CURSOR));
           authoring = li.checkAuthoring
                     (username.getText(),
                      passwd.getText());

       if(authoring)//the user is registered
           new CourseListInterface(staffId);
       else{
         //authoring=false, show message box

            em = new ErrorMge(parent,
           "Wrong information, try again");
            em.setLocation(150,300);
            em.getContentPane().setLayout(
               new FlowLayout
                   (FlowLayout.LEFT));
            em.show();
          }
          parent.setCursor(new
          Cursor(Cursor.DEFAULT_CURSOR));
       }
      }
      else         // cancel button is clicked
        if(arg.equals("Cancel"))
            dispose();
        else return super.action(evt,arg);
        return true;
      }

      public boolean handleEvent(Event evt)

                //the user destroy the window
      {
         if (evt.id == Event.WINDOW_DESTROY)
            dispose();
         else return super.handleEvent(evt);
         return true;
      }

}
```

Class 12: ConnectInfo.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;


/**************************************
 * This is the child of the Dialog box
 * (see ConnectDialog.java)
 *  Just get user name and password
 *  information.
 * /

class ConnectInfo {

 String username;
 String userpd;
 ConnectInfo(String u, String p)
   { username = u; userpd = p; }
}
```

Class 13: JHelpMenu.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

class JHelpMenu extends JDialog {
JHelpMenu(JmainInterface parent) {
    super(parent, "Help", true);
    Container content = getContentPane();
    Font font = new Font("Serif", Font.PLAIN, 30);
    content.setFont(font);

    String labelText =
      "<html><FONT SIZE=2 ALIGN=CENTER>" +
      "<FONT COLOR=Green>Major operations include:</FONT>" +
      "  <LI>Select [Connect] to Login to this system;" +
      "  <LI> some options" +
      "  <LI>Searching some special context by keywords " +
      "  <LI>Finding links between recorded AudioGraphic material" +
      "  <LI>Editing individual lecture notes" +
      "  <LI>Create links between recorded AudioGraphic material" +
      "</UL></html>";

    JLabel coloredLabel = new JLabel(labelText, JLabel.CENTER);
    content.add(coloredLabel, BorderLayout.CENTER);

    JPanel okButton = new JPanel();
    okButton.add(new Button(" ok "));
    content.add(okButton, BorderLayout.SOUTH);
    resize(400,220);
    }

    public boolean action(Event evt, Object arg)
    {
      if(arg.equals(" ok "))    // ok button is clicked
          dispose();
      else return super.action(evt,arg);
       return true;
    }

  public boolean handleEvent(Event evt)  //the frame is forced to exit.
  {
     if (evt.id == Event.WINDOW_DESTROY)
        dispose();
     else
        return super.handleEvent(evt);
     return true;
  }
}
```

Class 14: CourseListInterface.java

```
package interfacepg;
import java.awt.*;
import java.awt.Cursor.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import databasepg.*;
import utilitypg.*;

public class CourseListInterface extends
      JFrame implements ActionListener
{
  private Panel cards;
  private CardLayout layout;
  private String staffId;
  private String userId=new String();
  private MyCourseCard mc,uc;
  private AllCourseCard ac;
  private StaffList sl;
  private List staffList;
  private String cid;
  private JPanel bPanel;
  private TextField infoText;

  public CourseListInterface(String
                             staffId)
 {
    super("Course List Interface");
    WindowUtilities.
          setNativeLookAndFeel();
    addWindowListener(new
                  ExitListener());
    this.staffId = new String(staffId);
    Container content = getContentPane();
    content.setLayout(new
              BorderLayout());
    setCursor(new
        Cursor(Cursor.DEFAULT_CURSOR));
       //set the waiting cursor
    JPanel staffPanel = new JPanel();
    sl = new StaffList(this);
    staffPanel = sl.getStaffList();

    cards = new Panel();
    layout = new CardLayout();
    cards.setLayout(layout);
    mc = new MyCourseCard(staffId,this);
    cards.add("My courses list", mc);
    content.add("West",cards);

    JPanel mPanel = new JPanel();
    mPanel.setLayout(new BorderLayout());
    mPanel.add("Center",staffPanel);
    bPanel = new JPanel();
    bPanel.setLayout(new BorderLayout());
    bPanel.add("North", new Label(" "));
    bPanel.add("West",new Label(" "));
    Button cl = new
           Button("All Courses List");
    cl.addActionListener(this);
    bPanel.add("Center",cl);
    bPanel.add("East",new Label(" "));
    JPanel cPanel = new JPanel();
    cPanel.add(new Label(" "));
    cPanel.add(new Button("Cancel"));
    cPanel.add(new Label(" "));
    bPanel.add("South",cPanel);
    mPanel.add("South",bPanel);
    content.add("Center",mPanel);
    setSize(500, 350);
    setVisible(true);
  }

  public boolean action(Event
                        evt, Object arg)
  {
    if(arg.equals("Cancel"))
       dispose();
    else return super.action(evt,arg);
    return true;
  }

public void actionPerformed(ActionEvent
                                      e)
// the action for all courses list
  {
    String arg = e.getActionCommand();
    if (arg.equals("All Courses List")){
       setCursor(new
             Cursor(Cursor.WAIT_CURSOR));
       // the waiting cursor action
       ac = new
            AllCourseCard(staffId,this);
       cards.add("All courses list",ac);
       layout.last(cards);
       setCursor(new
          Cursor(Cursor.DEFAULT_CURSOR));
       // the waiting cursor destroy
    }
  }

  public void addAStaffCourseCard(String
                                  userId)
     // for choosing a particular staff
  {
    mc = new MyCourseCard(userId,this);
    cards.add("The staff course list",
                                   mc);
    layout.last(cards);
  }


  public String tableIsExist(String cd)
 // the message for the inexistent slide.
  {

    AllCourseInformation aci = new
                  AllCourseInformation();
    String tabId = new
    String(aci.getTableName(cd));
    if(tabId.equals("null"))
    {
     UnExistMge uem= new
       UnExistMge(this,"The presentations
                haven't created yet.");
       uem.setLocation(150,300);
       uem.getContentPane().setLayout(new
            FlowLayout(FlowLayout.LEFT));
       uem.show();
    }
    return tabId;
  }

}
```

Class 15: StaffList.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import java.awt.Cursor.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import utilitypg.*;
import databasepg.*;

/** This is a Staff List panel for
 *  displaying all staff name in the
 *  database. The panel is embedded in
 *  CourseList Interface
 */

class StaffList extends JList
 {
    private Vector sfname = new Vector();
    private Vector slname = new Vector();

    private AllStaffInformation asl;
      // the information from staff table

    private List staffName;

    private JPanel titlePanel;
      //the title of the staff list panel

    private CourseListInterface parent;

   StaffList(CourseListInterface parent)
   {
     this.parent = parent;
     this.staffName = new List();
     Font titleFont = new Font("Serif",
                        Font.BOLD, 14);

     JLabel titleLabel = new JLabel("All
             Staff List", JLabel.CENTER);
     titleLabel.setFont(titleFont);
     titlePanel = new JPanel();
     titlePanel.add(titleLabel,
               BorderLayout.CENTER);

     staffName = new List();
     asl = new AllStaffInformation();
     asl.setAllStaffName();
     sfname = asl.getAllStaffFirstname();
     slname = asl.getAllStaffLastname();

     for (int i=0; i<sfname.size();i++)
     staffName.add((String)
           (sfname.elementAt(i)+"
            "+slname.elementAt(i)));
     staffName.addActionListener
             (new listDoubleClick());

   }

   public JPanel getStaffList()
   {
     ScrollPane aListPane = new
                          ScrollPane();
     aListPane.add(staffName);

     JPanel aListPanel = new JPanel();
     aListPanel.setLayout(new
                    BorderLayout());
     aListPanel.setBackground(
                    Color.lightGray);

     Border aListPanelBorder =
            BorderFactory.
                  createTitledBorder("");

     aListPanel.setBorder(
                  aListPanelBorder);
     aListPanel.add(titlePanel,
                   BorderLayout.NORTH);
     aListPanel.add(aListPane,
                   BorderLayout.CENTER);

     return aListPanel;
   }

   private class listDoubleClick implements
                             ActionListener
   {
    public void actionPerformed(ActionEvent
                                      evt)
    {
       parent.setCursor(new
              Cursor(Cursor.WAIT_CURSOR));
       String arg = evt.getActionCommand();

       String sName = arg.toString();

       String userId = new
            String(asl.getStaffNo(sName));

       parent.addAStaffCourseCard(userId);

       staffName.deselect(
                    getSelectedIndex());
       parent.setCursor(new
           Cursor(Cursor.DEFAULT_CURSOR));
    }
   }

}
```

Class 16: CourseList.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import java.awt.Cursor.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import databasepg.*;


class CourseList extends JList
{
   private JPanel titlePanel;
   private Vector cid = new Vector();
        //the list of course identities
   private Vector cname = new Vector();
       //the list of course names
   private CourseListInterface parent;
        //CourseList is dependent on
        //CourseListInterface
   private List aList;
   private AllCourseInformation acl;
   private String listTitle,userId;
   private String CNo = new String();
       //CNo is the number of course
   private String[] theCourseList;
   private String staffNo;

 CourseList(CourseListInterface parent,
             String staffNo)
 {
    this.parent = parent;
    this.staffNo = staffNo;
 }

public JPanel getTitlePanel(String title)
{
   setLayout(new BorderLayout());
   String labelText =
    "<html><B><UL><FONT SIZE=3
         COLOR=PURPLE>"+title+
      "</B></FONT></UL></html>";
     JLabel titleLabel = new
     JLabel(labelText, JLabel.CENTER);
     JPanel titlePanel = new JPanel();
     titlePanel.add(titleLabel,
                BorderLayout.CENTER);
     return titlePanel;
}

public JPanel getListPanel(String
      listTitle,Vector cid, Vector cname)
{

    JScrollPane.
    this.listTitle = listTitle;
    this.aList= new List(10);

    for (int i=0; i<cid.size();i++)
    aList.add((String)(cid.elementAt(i)+
     " "+cname.elementAt(i)));
     Font displayFont = new Font("Serif",
     Font.PLAIN, 14);
     aList.setFont(displayFont);
     aList.addActionListener(new
     listDoubleClick());

    ScrollPane aListPane = new
                          ScrollPane();
    aListPane.add(aList);
if (listTitle.equals("All Courses List")
{
       aListPane.setSize(300,250);
}
else
{
     aListPane.setSize(300,100);
}
     JPanel aListPanel = new JPanel();

aListPanel.setBackground(Color.lightGray);
Border aListPanelBorder =
        BorderFactory.createTitledBorder
                              (listTitle);

aListPanel.setBorder(aListPanelBorder);
aListPanel.add(aListPane,
rderLayout.CENTER);
return aListPanel;

}

private class listDoubleClick implements
                            ActionListener
{
   public void actionPerformd
                       (ActionEvent evt)
   {
        //set waiting cursor for feedback
        //a user's action

      parent.setCursor(new
            Cursor(Cursor.WAIT_CURSOR));
      String arg = evt.getActionCommand();
      CNo = arg.toString().substring(0,6);
      String tid= new
      String(parent.tableIsExist(CNo));
      if(!tid.equals("null"))
        new ACourseInterface(
                       staffNo,CNo,tid);
      else
         System.out.print(
                  "this is null table.");
      // recover the cursor to default one
      parent.setCursor(new
           Cursor(Cursor.DEFAULT_CURSOR));
      // remove the mark of the select
      //item on a list
      aList.deselect(getSelectedIndex());
   }

  }


  public String getCourseId()
                  //get a course identity
  {
    return CNo;
  }
 }
```

Class 17: AllCourseCard.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import utilitypg.*;
import databasepg.*;

/** This is a panel for displaying all course (List)
 *  name with course id number in the database.
 *  The panel is embedded in CourseList Interface and
 *  it is vailable by click the "all course list" button
 *  of the CourseList Interface
 */

class AllCourseCard extends Panel
 {
   private Vector cid = new Vector();
   private Vector cname = new Vector();

   // set the fist six rows are visible on the list
   private List allList= new List(6);

   private AllCourseInformation aci;
   private CourseList cl;
   private CourseListInterface parent;

   AllCourseCard(String loginStaff, CourseListInterface parent)
   {
     setLayout(new BorderLayout());
     this.parent = parent;
     cl = new CourseList(parent,loginStaff);
     add("North",cl.getTitlePanel("Computer Science Course:"));
     aci = new AllCourseInformation();
     aci.getAllCourseList();
     cid = aci.getCidList();
     cname = aci.getCnameList();
     JPanel listPanel = cl.getListPanel("All Courses List",cid,cname);
     add("Center", listPanel);

   }

}
```

Class 18: MyCourseCard.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import databasepg.*;


 class MyCourseCard extends Panel
 {
     private Vector cid1 = new Vector();
     //the identity list of the courses with which user are as a controller

     private Vector cname1 = new Vector();
     //the name list of the coursex with which user are as a controller

     private Vector cid2 = new Vector();
     //the identity list of the course with which user is as contributor

     private Vector cname2 = new Vector();
     //the name list of the course with which user are as a contributor

     private AllStaffInformation asl;
     private CourseList contro, contri;
     private UserCourseInformation uci;
     private CourseListInterface parent;

     MyCourseCard(String staffNo, CourseListInterface parent)
     {
       JPanel titlePanel, list1Panel, list2Panel;
       asl = new AllStaffInformation();
       String title = asl.getStitle(staffNo);
       String sname = asl.getSname(staffNo);

       setLayout(new BorderLayout());

       this.parent = parent;
       contro = new CourseList(parent,staffNo);
       add("North",contro.getTitlePanel(title+" "+
                                        sname+"'s courses: "));
       uci = new UserCourseInformation(staffNo);
       uci.getUserCourseAsController();
       cid1 = uci.getCid1List();
       cname1 = uci.getCname1List();
       list1Panel = contro.getListPanel("Controller Courses",
                                        cid1,cname1);
       add("Center", list1Panel);
       uci.getUserCourseAsContributor();
       cid2 = uci.getCid2List();
       cname2 = uci.getCname2List();
       contri = new CourseList(parent, staffNo);
       list2Panel = contri.getListPanel("Contributor Courses",
                                        cid2,cname2);
       add("South", list2Panel);
     }

 }
```

Class 19: AcourseInterface.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import java.awt.Cursor.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.event.*;
import databasepg.*;
import utilitypg.*;

public class ACourseInterface extends
JFrame
{
  private Panel cards;
  private CardLayout layout;
  private String presentationChoice =
                        new String("all");
  private ContributorList sl;
  private String PNo, tabName;
  private String staffId, myId;
  public ACourseInterface(String myId,
              String cid, String tabName)
 {
  super("Course Content");
  WindowUtilities.setNativeLookAndFeel();
  addWindowListener(new ExitListener());
  this.myId = myId;
  this.staffId = myId;
  this.tabName = tabName;
  ontainer content = getContentPane();
  content.setLayout(new BorderLayout());
  cards = new Panel();
  layout = new CardLayout();
  cards.setLayout(layout);
  CourseContent cc = new
  CourseContent(this,tabName,
                presentationChoice);
  JScrollPane jsp = new
         JScrollPane(cc.getCourseTree());
  cards.add("All Slides", jsp);
  content.add("Center",cards);
  JPanel staffPanel = new JPanel();
  sl = new ContributorList(this,cid);
  staffPanel = sl.getContributorList();

  JPanel mPanel = new JPanel();
  mPanel.setLayout(new BorderLayout());
  mPanel.add("North",staffPanel);

  JPanel bPanel = new JPanel();
  bPanel.setLayout(new GridBagLayout());
  GridBagConstraints gbc = new
                    GridBagConstraints();
  gbc.fill = GridBagConstraints.NONE;
  gbc.weighty = 100;
  gbc.gridx = 0;
  gbc.gridy = 0;
  gbc.gridwidth = 1;
  gbc.gridheight = 1;
  bPanel.add(new Button("All Slides
             Show"),gbc);
  gbc.fill = GridBagConstraints.NONE;
  gbc.weighty = 100;
  gbc.gridx = 0;
  gbc.gridy = 1;
  gbc.gridwidth = 1;
  gbc.gridheight = 1;
    bPanel.add(new Button("Cancel"),gbc);
    mPanel.add("Center",bPanel);
    content.add("East",mPanel);
    setSize(500, 350);
    setVisible(true);
  }

  public boolean action(Event evt,
                        Object arg)
  {
    if(arg.equals("All Slides Show"))
    {
       //set wait coursor for a user's
       //action feedback
     setCursor(new
             Cursor(Cursor.WAIT_CURSOR));

     presentationChoice = "all";
   //all slides in this course are visible
     CourseContent cc = new
              CourseContent(this,tabName,
                     presentationChoice);

    //show all slides entry in the course
    JScrollPane jsp = new
    JScrollPane(cc.getCourseTree());
    cards.add("All Slides", jsp);
    layout.last(cards);

    // recover the cursor to original one
     setCursor(new
          Cursor(Cursor.DEFAULT_CURSOR));
    }
    else if(arg.equals("Cancel"))
      dispose();
    else return super.action(evt,arg);
    return true;
  }

  public void addACourseContentCard(String
                                    userId)
  {
     presentationChoice = userId;
     CourseContent cc = new
              CourseContent(this,tabName,
                     presentationChoice);
     JScrollPane jsp = new
         JScrollPane(cc.getCourseTree());
     cards.add("user Slides", jsp);
     layout.last(cards);
  }

  public boolean handleEvent(Event evt)
  {
   if (evt.id == Event.WINDOW_DESTROY
         &&  evt.target == this)
         System.exit(0);
   else
     return super.handleEvent(evt);
     return true;
  }

  public String getStaffId()
  {
     return staffId;
  }

}
```

Class 20: ContributorList.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import java.awt.Cursor.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import utilitypg.*;
import databasepg.*;

/** This is a list panel for the
 * contributors of the course which
 * the user choice from the course list.
 *  It also displays the controller of
 *  the course. The panel is embedded in
 *  ACourseInterface.
 */

class ContributorList extends JList
 {
     private String ctlname =
                              new String();
     private Vector ctbname =
                              new Vector();
     private ACourseInformation aci;
     private AllStaffInformation asl;
     private List staffName;
     private ACourseInterface parent;

   ContributorList(ACourseInterface
                    parent, String cid)
   {
     this.parent = parent;
     this.staffName = new List();

     staffName = new List();
     aci = new ACourseInformation();
     ctlname =
       aci.getControllerOftheCourse(cid);
     ctbname =
      aci.getContributorOftheCourse(cid);
     asl = new AllStaffInformation();
     for (int i=0; i<ctbname.size();i++){
         staffName.add(asl.getSname(
         (String)ctbname.elementAt(i)));
     }
     staffName.addActionListener(
            new listDoubleClick());
   }

   public JPanel getContributorList()
   {
     ScrollPane aListPane =
                  new ScrollPane();
     aListPane.add(staffName);
     Font titleFont =
       new Font("Serif", Font.BOLD, 14);
     JLabel ctlLabel =
        new JLabel("Course Controller:");
     ctlLabel.setFont(titleFont);
     JPanel aPanel = new JPanel();
     aPanel.setLayout(
              new BorderLayout());
     aPanel.add("North", ctlLabel);
     String sName =
          asl.getSname((String)ctlname);
String labelText =
      "<html><B><UL><FONT SIZE=2
COLOR=PURPLE>"+sName+
      "</B></FONT></UL></html>";
     aPanel.add("Center", new
JLabel(labelText));

JPanel aListPanel = new JPanel();
aListPanel.setLayout(new BorderLayout());
JLabel ctbLabel = new JLabel("Course
                              Contributor:");
ctbLabel.setFont(titleFont);
aListPanel.add("North",ctbLabel);
aListPanel.add("Center",aListPane);

JPanel staffPanel = new JPanel();
staffPanel.setLayout(new BorderLayout());
staffPanel.setBackground(
                         Color.lightGray);
Border aListPanelBorder =
    BorderFactory.createTitledBorder("");

staffPanel.setBorder(aListPanelBorder);
staffPanel.add("North", aPanel);
staffPanel.add("Center", new Label(" "));
staffPanel.add("South", aListPanel);

   return staffPanel;
}

   // this method displays a particular
   // staff presentations
   // which included in the courses.

private class listDoubleClick implements
                          ActionListener
{
   public void  actionPerformed(
                         ActionEvent evt)
   {
     parent.setCursor(new
             Cursor(Cursor.WAIT_CURSOR));
     String arg = evt.getActionCommand();

     String sName = arg.toString();
     String userId = new
           String(asl.getStaffNo(sName));

    parent.addACourseContentCard(userId);
    parent.setCursor(new
          Cursor(Cursor.DEFAULT_CURSOR));

     }
   }

}
```

Class 21 CourseContent.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import java.lang.*;
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.event.*;
import databasepg.*;
import utilitypg.*;

/** This is tree panel which displays
 *  the Content of the course.
 *  which has three states:
 *  1) All slides are visible
 *  2) the user slides are visible
 *  3) a particular staff's slides are
 *     visible
 *  The panel is embedded in
 *  the ACourseInterface.
 */

public class CourseContent extends JFrame
implements TreeSelectionListener
{
  private String courseId = new String();
  private String PNo = new String();
  private JTree tree;
  private JScrollPane sp;

  private String listTitle, staffId;
  private String slidesChoice =
                            new String();
  private ACourseInterface parent;

  private boolean browserShow = false;
  private String browserChoice
                          = new String();

  CourseContent(ACourseInterface parent,
       String tabId, String slidesChoice)
  {
    this.parent = parent;
    this.slidesChoice = slidesChoice;

    courseId = tabId;
    ACourseInformation ci =
              new ACourseInformation();

    ci.getCourseContent(tabId);
    DefaultMutableTreeNode root =
          GetContent(ci.getSecid(),
          ci.getSubid(), ci.getLoctype(),
                        ci.getSubname());

    tree = new JTree(root);
    tree.addTreeSelectionListener(this);
  }

/** Small routine that will make node
 *  out of the first entry in the
 *  array, then make nodes out of
 *  subsequent entries and make them
 *  child nodes of the first one. The
 *  process is repeated recursively for
 *  entries that are arrays.
 */

public JTree getCourseTree()
{
    return tree;
}

public DefaultMutableTreeNode
    GetContent(Vector secid,Vector subid,
    Vector loctype,Vector subname)
{
  DefaultMutableTreeNode courseRoot =
        new DefaultMutableTreeNode(
            courseId+" Course Content");

  String topname = new String();

  for(int i=0; i<secid.size(); i++) {
   if(loctype.elementAt(i).equals("top"))
   {
      topname = (String)
                secid.elementAt(i);

      DefaultMutableTreeNode section =
        new DefaultMutableTreeNode(
                                topname);

      DefaultMutableTreeNode subSection;

      while(secid.elementAt(i).equals(
            topname))
      {
        if(!loctype.elementAt(i).equals(
                                "bottom"))
          subSection = displaySubSection(
          secid,subid,loctype,subname,i);
        else
          subSection = new
                DefaultMutableTreeNode(
                    subid.elementAt(i)+
               " "+subname.elementAt(i));

        section.add(subSection);
        i++;
        if(i >= secid.size())
        {
         courseRoot.add(section);
         return(courseRoot);
        }
      }
      courseRoot.add(section);
   }
  }
    return(courseRoot);
}
```

Class 21 CourseContent.java (continuted)

```
private DefaultMutableTreeNode
    displaySubSection(Vector secid,Vector
    subid,Vector loctype,Vector subname,
                              int index)
{
   int k;

   DefaultMutableTreeNode section = new
        DefaultMutableTreeNode(
        subid.elementAt(index)+
        " "+subname.elementAt(index));

   String subId =new String((String)
                 subid.elementAt(index));

   for(k=index;
       !secid.elementAt(k).equals(subId);
       k++)

    DefaultMutableTreeNode subSection;

  while(secid.elementAt(k).equals(subId))
  {
    if (!loctype.elementAt(k).equals(
                              "bottom"))
    {
        subSection =
           displaySubSection(secid,
           subid,loctype,subname,k);

        section.add(subSection);
    }
    else
    {
        ASlideInformation si = new
                ASlideInformation();

        si.getSlideContent((String)
                    subid.elementAt(k));

      if(this.slidesChoice.equals(
        "all")||this.slidesChoice.equals(
        si.getSlideCreator()))
      {
          subSection = new
             DefaultMutableTreeNode(
               subid.elementAt(k)+
               " "+subname.elementAt(k));

          section.add(subSection);
      }

    }
    k++;

      if(k >= secid.size())
           return(section);
  }
    return(section);
  }
```

```
  public void valueChanged(
              TreeSelectionEvent evt)
  {
    PNo =
     tree.getLastSelectedPathComponent().
              toString().substring(0,6);
    ASlideInformation si = new
                      ASlideInformation();
    si.getSlideContent(PNo);

    if(!si.getFoundSlide())
    {
      UnfinishedMge um =
         new UnfinishedMge( parent,
            "It can't found the slide.");

      um.setLocation(150,300);
      um.getContentPane().setLayout(
        new FlowLayout(FlowLayout.LEFT));
      um.show();
    }
    else{
      String url = new
              String(si.getLectureURL());
      if ( browserShow == false )
      {
        browserShow = true;
        ChooseBrowser cb = new
                ChooseBrowser(this,url);
      }
      else{
        try {
         Process p =
             Runtime.getRuntime().exec(
             browserChoice+url);
        }catch(IOException e){
             System.err.println("Cannot
                        start browser");
             System.exit(1);
           }
         }

    }

  }

  public void setBrowserChoice(String
                            browserChoice)
  {
     this.browserChoice = browserChoice;

  }

}
```

Class 22 ErrorMge.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import javax.swing.*;
import javax.swing.border.*;

/** This is a message for the unfound username or password, which the user typed
 *  It is created by the user's login action of the Main Interface.
 *
 */

class ErrorMge extends JDialog
{
  private JmainInterface parent;

  ErrorMge(JmainInterface parent, String info)
  {
    super(parent,"Login Error", true);
    this.parent = parent;
    Container content = getContentPane();
    content.setLayout(new BorderLayout());
    Font font = new Font("Serif", Font.PLAIN, 30);
    content.setFont(font);
    String labelText =
      "<html><B><FONT SIZE=2 ALIGN=CENTER><FONT COLOR=PURPLE>"+
      info + "</FONT></B>"+ "</UL></html>";
    JLabel coloredLabel = new JLabel(
                          labelText,new ImageIcon("Images/WarnButton.gif"), JLabel.CENTER);
    content.add(coloredLabel, BorderLayout.NORTH);

    JPanel okButton = new JPanel();
    okButton.setLayout(new GridLayout(1,5));
    okButton.setFont(new Font("Serif", Font.PLAIN, 14));

    okButton.add(new Button(" ok "));
    content.add(okButton,BorderLayout.SOUTH);
    resize(240,120);
  }

  public boolean action(Event evt, Object arg)
  {
      if(arg.equals(" ok "))
      {
         try
        {
              dispose();
         }
         catch( Exception e ) {
             e.printStackTrace();
         }
      }
      else return super.action(evt,arg);
       return true;
  }

    public boolean handleEvent(Event evt)
  {
     if (evt.id == Event.WINDOW_DESTROY)
        dispose();
     else
        return super.handleEvent(evt);
     return true;
  }
}
```

Class 23 UnfinishedMge.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

/**************************************************************************
 *  This is a message for some unfinished information that a user asked.
 *  It is created for protected ask some unproperty action by the user.
 */

class UnfinishedMge extends JDialog
{
  UnfinishedMge(ACourseInterface parent, String info)
  {
    super(parent,"Information frame", true);
    Container content = getContentPane();
    content.setLayout(new BorderLayout());
    Font font = new Font("Serif", Font.PLAIN, 30);
    content.setFont(font);
    String labelText =
      "<html><B><FONT SIZE=2 ALIGN=CENTER><FONT COLOR=PURPLE>"+
      info + "</FONT></B>"+ "</UL></html>";
    JLabel coloredLabel = new JLabel(labelText,new
                          ImageIcon("Images/MWI.gif"), JLabel.CENTER);
    content.add(coloredLabel, BorderLayout.CENTER);

    JPanel okButton = new JPanel();
    okButton.setLayout(new GridLayout(1,5));
    okButton.setFont(new Font("Serif", Font.PLAIN, 14));
    okButton.add(new Button(" ok "));

    content.add(okButton, BorderLayout.SOUTH);
    resize(300,120);
    }

    public boolean action(Event evt, Object arg)
    {
      if(arg.equals(" ok "))
          dispose();
      else return super.action(evt,arg);
      return true;
    }

    public boolean handleEvent(Event evt)
    {
       if (evt.id == Event.WINDOW_DESTROY)
          dispose();
       else
          return super.handleEvent(evt);
       return true;
  }
}
```

Class 24 UnExistMge.java

```
package interfacepg;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

/** This is a message for answer some unexist information
 *  that a user typed.
 */

class UnExistMge extends JDialog
{
  UnExistMge(CourseListInterface parent, String info)
  {
    super(parent,"Unexist Table Information", true);
    Container content = getContentPane();
    content.setLayout(new BorderLayout());
    Font font = new Font("Serif", Font.PLAIN, 30);
    content.setFont(font);
    String labelText =
      "<html><B><FONT SIZE=2 ALIGN=CENTER><FONT COLOR=PURPLE>"+
      info + "</FONT></B>"+
      "</UL></html>";
    JLabel coloredLabel = new JLabel(labelText,new
                          ImageIcon("Images/MGW.gif"), JLabel.CENTER);
    content.add(coloredLabel, BorderLayout.CENTER);

    Panel okButton = new Panel();
    okButton.setLayout(new GridLayout(1,5));
    okButton.setFont(new Font("Serif", Font.PLAIN, 14));
    okButton.add(new Button(" ok "));
    content.add(okButton, BorderLayout.SOUTH);
    resize(300,120);
    }

    // the action method for "ok" button
    public boolean action(Event evt, Object arg)
    {
      if(arg.equals(" ok "))
          dispose();
      else return super.action(evt,arg);
      return true;
    }

    public boolean handleEvent(Event evt)
    {
     if (evt.id == Event.WINDOW_DESTROY)
        dispose();
     else
        return super.handleEvent(evt);
     return true;
  }
}
```

Class 25-1: ChooseBrowser.java

```
package interfacepg;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

/** A small interface to choose a "Web
 *  browser" and multi-display the
 *  presentations on the browser.
 */

public class ChooseBrowser extends JFrame
implements ActionListener
{
  private String initialURL =
                              new String();
  private String browserChoice =
                              new String();
  private JRadioButton IEChoice,NSChoice;
  private JPanel choicePanel;
  private CourseContent parent;

  public ChooseBrowser(CourseContent
              parent,String initialURL)
  {
    super("choose Brower");
    this.initialURL = initialURL;
    this.parent = parent;

WindowUtilities.setNativeLookAndFeel();
   addWindowListener(new ExitListener());
   Container content = getContentPane();
   JPanel topPanel = new JPanel();
    topPanel.setBackground(
                        Color.lightGray);
    topPanel.add(new Label(
           "Please choose a Browser:"));
    content.add(topPanel,
                    BorderLayout.NORTH);
    IEChoice = new JRadioButton("Internet
                         Explorer",true);
    NSChoice = new
          JRadioButton("Netscape",false);
    ButtonGroup choiceBox =
                        new ButtonGroup();
    choiceBox.add(IEChoice);
    choiceBox.add(NSChoice);

    choicePanel=new JPanel(
                    new GridLayout(2,1));
    choicePanel.setBackground(
                        Color.lightGray);
    IEChoice.addActionListener(this);
    NSChoice.addActionListener(this);

    browserChoice = new
              String("C:\\Program Files\\
              Plus!\\Microsoft Internet\\
                        Iexplore.exe ");

    choicePanel.add(IEChoice);
    choicePanel.add(NSChoice);
    content.add(choicePanel,
                  BorderLayout.CENTER);

    JPanel okPanel = new JPanel();
    okPanel.add(new Button("ok"));
```

// the code is on window NT

```
content.add(okPanel,BorderLayout.SOUTH);

    resize(240,120);
    setVisible(true);
}

public void actionPerformed(ActionEvent
                                event)
{
   String arg = event.getActionCommand();

   if(IEChoice.isSelected())
        browserChoice = New String
           ("C:\\Program Files\\Plus!\\
                Microsoft Internet\\
                Iexplore.exe ");
    else if(NSChoice.isSelected())

         browserChoice = new String
          ("C:\\Program Files\\Netscape
            \\Communicator\\Program
             \\netscape.exe ");

}

 // the process launch the browser and
 //display the presentation

 public boolean action(Event evt,
                            Object arg)
 {
   if(arg.equals("ok"))
   {
     dispose();
     parent.setBrowserChoice(
                          browserChoice);
     try {
         Process p =
              Runtime.getRuntime().exec(
               browserChoice+initialURL);
     }
     catch(IOException e){
      System.err.println(
      "Cannot start the browser");
        System.exit(1);
      }
   }
   else return super.action(evt,arg);
   return true;
 }

  public boolean handleEvent(Event evt)
  {
     if (evt.id == Event.WINDOW_DESTROY
        &&  evt.target == this)
        dispose();
     else
        return super.handleEvent(evt);
     return true;
  }
}
```

Class 25-2: ChooseBrowser.java ***// the code is on Macintosh***

```
Package interfacpg;
import java.awt.Frame;
import java.awt.FileDialog;
import java.io.File;
import java.io.IOException;

import com.apple.mrj.MRJFileUtils;

public class ExecLaunch extends Frame
{
     private String url;
     public ExecLaunch(String url)
     {   this.url=url;
          try
          {
             MRJFileUtils.openURL(url); //Attempt to let MRJ to find the browser for us.
                    return;             //If this was successful, then we need not go on.
          }
          catch (IOException exc)
          {
                    //This can occur if problems arise while attempting to open the URL.
          }
          catch (NoSuchMethodError err)
          {
                   //This can occur when earlier versions of MRJ are used which do not
                  //support the openURL method.
          }
          catch (NoClassDefFoundError err)
          {
                     //This can occur under runtime environments other than MRJ.
          }

                //If we make it here, MRJ was unsuccessful in opening the URL, and
                //we need to do it the hard way, using Runtime.exec.
              String browserName;

                //Set up a FileDialog for the user to locate the browser to use.
              FileDialog fileDialog = new java.awt.FileDialog(this);
              fileDialog.setMode(FileDialog.LOAD);
              fileDialog.setTitle("Choose the browser to use:");
              fileDialog.setVisible(true);

                  //Retrieve the path information from the dialog and verify it.
              String resultPath = fileDialog.getDirectory();
              String resultFile = fileDialog.getFile();
              if(resultPath != null && resultPath.length()!= 0 && resultFile != null &&
                   resultFile.length() != 0)
              {
                    File file = new File(resultPath + resultFile);
                    if(file != null)
                    {
                          browserName = file.getPath();
                          try
                          {
                              //Launch the browser and pass it the desired URL
                              Runtime.getRuntime().exec(new String[]{browserName, url});
                          }
                          catch (IOException exc)
                          {
                            exc.printStackTrace();
                          }
                    }
              }
       }
}
```