國立台灣大學資訊工程研究所　　博士論文

指導教授 ： 陳文進博士

建構虛擬大學之多媒體電子教室與教務資訊系統之設計與實作

Design and Implementation Issues of Multimedia Digital Classroom and Academic Information System for a Virtual University

研究生 ： 俞旭昇 撰

學　號 ： D82506014

中華民國八十八年五月

中文摘要

二十世紀末葉的高等教育面臨了成本高漲,地域失衡與行政效率低落三大問題。高等教育必須要提昇其生產效率來解決這些問題。而這樣的效率提昇不能全然藉由傳統的預算刪減來達成,而是必需提供更多教育機會給有心向學者。

本論文探討以 Java 來實作多媒體電子教室與教務資訊系統之技術問題。建構在這兩個系統上的虛擬大學可以藉由提供網路化的學習與資訊收集環境來舒減甚至解決目前高等教育效率不彰的問題。本論文主要的貢獻可分為三部份:

(1) 提出一混合事件共享與分散物件之應用程式共享模型。此模型可作為學習環境中各種不同互動模式應用程式之建構基礎。講解、討論與考試是主要的學習活動, 而這些活動從應用程式共享的角度可歸類為"單一輸入多人觀看"、"多人輸入"與"個人觀看"等三種模式。我們所提出的混合式模型可以階層式的將不同互動模式的應用整合在一起。

(2) 在我們提出的應用程式共享模型上實作了各種遠距教學支援工具。這些工具包括了多媒體瀏覽器、語音會議、共享白板與聊天室。

(3) 實作一高效能、易用、低價與具延展性之教務資訊系統。此系統採用以 RMI 為基礎之三層式架構以應用任何可能的最佳化技巧。支援日常教務活動之各項功能均包含在本系統中。

本論文內的所有實作都是以 JDK1.1 完成並有實際之應用。

Design and Implementation Issues of Multimedia Digital Classroom and Academic Information System for a Virtual University

Student: Shiuh-Sheng Yu

Advisor: Professor Wen-Chin Chen

Department of Computer Science and Information Engineering,

National Taiwan University, Taipei, Taiwan, R.O.C.

May 1999

Abstract

The university education in the last decade of the twentieth century faces escalating costs, uneven demographics, and administration inefficiency. The university education must become more productive to solve these problems. This productivity advance cannot be achieved wholly through the traditional approach of reducing the inputs, but also through greater attention to the learner.

In this dissertation, we will investigate technical issues on multimedia digital classroom and academic information system implemented in Java. A virtual university based on these two systems can alleviate or even solve the problems of current university education by providing a networked learning and information retrieval environment. Our major contributions can be described in three aspects:

(1) A hybrid application-sharing model of event sharing and distributed object to support different interaction patterns in learning environment. Activities in learning environment include presentation, discussion and examination. These activities can be classified as single-input-multiple-views, multiple-inputs and local-view. Our hybrid model can combine applications with different interaction patterns transparently in a hierarchical manner.

(2) Various tools based on our proposed application-sharing model to support distance learning. Multimedia browser, audio conference, shared whiteboard and chat room has been implemented to construct a multimedia digital classroom.

(3) A high performance, easy to use, low cost and scalable academic information system. The system uses a RMI based three-tier architecture to explore any optimization techniques. Miscellaneous functionality has been provided to support daily academic information.

Our implementation is based on JDK1.1 and has been deployed in real world cases.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Motivation and Background

The university has been a place for education and research over centuries. The traditional approach uses face-to-face interaction for teaching. To support this scenario, we need the general affair office to build and maintain campus; the academic affair office to administrate academic information; and the student affair office to manage students. Other offices, such as library and computer center, are also required to support educational activities. Although a university has so many facilities and staffs, the students have to live inside or near campus for daily courses, and the teachers need to carry teaching materials to courses.

The university education in the last decade of the twentieth century faces escalating costs, uneven demographics, and administration inefficiency. The university education must become more productive to solve these problems. This productivity advance cannot be achieved wholly through the traditional approach of reducing the inputs, but also through greater attention to the learner.

As the rapid development of Internet and computer systems, there are more and more multimedia applications in markets. Some of these applications help geographically dispersed people working for their common goals [BUR91] [CRA92] [KUM94] [CUT96] [KOU96]. As one of these applications, the distance learning plays an important role of education popularization. It has been predicted that the distance learning would be one of the most important commercial applications in next decade [MEL96]. It breaks the limitation of traditional courses so that teachers and students do not need to be in the same classroom. These geographically dispersed teachers and students in the digital classroom can learn and share knowledge with less effort and cost.

Currently, the Minister of Education of R.O.C. has adopted three ways to promote distance learning:

- Broadcast video on TV. This approach currently has the greatest penetration, but there is no interaction between teachers and students at all. Questing, on-line practicing, and discussion are not possible in this environment. The number of courses offered to the public is also limited by the available chan-

nels.

- Build videoconference rooms in universities, and provide distance courses between these universities. This approach is expensive and hence not pervasive. Only students near a university with a conference room can attend classes. The only interaction between teachers and students is through audio conference. Due to current bandwidth limitation and compression techniques, teaching materials can not be presented clearly on screen.
- Put learning materials on the WWW, and let students study themselves through the Internet. This approach provides multimedia presentation, but lacks of interaction. It alone can only be thought as an assistant method for distance learning.

None of them is ideal to help people who can not attend the classes, and they do not address the administration problem of current universities. We need new approaches to solve the problems of escalating costs and uneven demographics.

We define a virtual university as *an educational environment on computer network to support distance learning and administration*. An ideal virtual university should obtain the educational quality of traditional face-to-face interaction method, and be available to as many learners as possible. To achieve these goals, a virtual university should have the following characteristics:

- A virtual university should support multimedia and different interaction patterns in distance learning. Multimedia presentation, audio conference, on-line practice, and shared whiteboard are essential tools for good presentation and discussion.
- The software of a virtual university should support heterogeneous environments to cover learners on different platforms.
- The runtime environment of a virtual university should require very low network bandwidth so that it can be accessed from as many users as possible on the Internet.
- The architecture of a virtual university should be extendible for the requirements of evolving educational methods. Different media, applications, and administration rules should be easily modified and added into a virtual university to support the changing requirements of education and administration.
- Setting up the environment of a virtual university should be easy or even in-

volves no human intervention. This is essential to let the virtual university reach every potential user.

- The implementation of a virtual university should have high performance, so that current hardware can support universities of medium to large scale.
- The cost of deploying a virtual university should be low so that most learners are affordable.
- The user interface of a virtual university must be intuitive and smart for novice users. Related functions should be put together to let users complete their works as fast as possible.

Being robust, secure, platform neutral, easy to use, easy to understand, and automatically downloadable on a network, Java is a powerful tool for building the virtual university environment. We use Java Development Kits (JDK) 1.1 to implement the virtual university. Packages related to this thesis are explained as follows [JDK98] for better understanding:

- Remote Method Invocation (RMI). RMI 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. A Java program can make a call on a remote object once it obtains a reference to the remote object. A remote reference is obtained either by looking up the remote object in the bootstrap-naming service provided by RMI, or by receiving the reference as an argument or a return value. A client can call a remote object in a server, and that server can be a client of other remote objects too. RMI uses Object Serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism.
- Java Database Connectivity (JDBC). JDBC is a standard Java API for executing SQL statements. It is easy to send SQL statements to virtually any relational database by using JDBC. In other words, with the JDBC API, it isn't necessary to write one program to access a Sybase database, another program to access an Oracle database, another program to access an Informix database, and so on. One can write a single program using the JDBC API, and the program will be able to send SQL statements to the appropriate database. With an application written in the Java programming language, one doesn't have to worry about writing different applications to run on different platforms. The

combination of Java and JDBC lets a programmer "write it once and run it anywhere". Installation and version-control are greatly simplified. A programmer can write an application or an update once, put it on the server, and everybody has access to the latest version.

- Object Serialization. Object Serialization extends the core Java Input / Output classes with support for objects. Object Serialization supports the encoding of objects and the objects reachable from them into a stream of bytes. It also supports the complementary reconstruction of the object graph from the stream. Serialization is used for lightweight persistence and for communication via sockets or RMI. The default encoding of objects protects private and transient data, and supports the evolution of the classes. A class may implement its own external encoding and is then solely responsible for the external format.
- Abstract Window Toolkit (AWT). The AWT consists of APIs responsible for building the graphical user interface (GUI) for Java programs. JDK 1.1 uses a peer model to provide a truly native look-and-feel by delegating the look-and-feel of the components to a set of underlying "peer" classes. Each platform has a different set of peer classes to wrap the native system's widgets. Java programs use these peer classes to output graphical commands to the underlying window system. Event types are encapsulated in a class hierarchy rooted at *java.util.EventObject*. An event is propagated from a "Source" object to a "Listener" object by invoking a method on the listener and passing in the instance of the event subclass which defines the event type generated. A Listener is an object that implements a specific EventListener interface extended from the generic *java.util.EventListener*. An EventListener interface defines one or more methods, which are to be invoked by the event source in response to each specific event type handled by the interface. An Event Source is an object that originates or "fires" events. The source defines the set of events it emits by providing a set of set<EventType>Listener (for single-cast) and/or add<EventType>Listener (for multi-cast) methods which are used to register specific listeners for those events. The event source is typically a GUI component and the listener is commonly an "adapter" object that implements the appropriate listener (or set of listeners) in order for an application to control the flow/handling of events. The listener object could

also be another AWT component that implements one or more listener interfaces for hooking GUI objects up to each other.

- Java Native Interface (JNI). The JNI is a native programming interface. It allows Java code that runs inside a Java Virtual Machine (VM) to inter-operate with applications and libraries written in other programming languages, such as C, C++, and assembly. The most important benefit of the JNI is that it imposes no restrictions on the implementation of the underlying Java VM. Therefore, Java VM vendors can add support for the JNI without affecting other parts of the VM. Programmers can write one version of a native application or library and expect it to work with all Java VMs supporting the JNI.
- Reflection. The Java Core Reflection API provides a small, type-safe, and secure API that supports introspection about the classes and objects in the current Java Virtual Machine. If permitted by security policy, the API can be used to construct new class instances and new arrays, access and modify fields of objects and classes. It can also be used to invoke methods on objects and classes, access and modify elements of arrays, and reflect information about the underlying member or constructor.

1.2 The Architecture of Multimedia Digital Classroom and Academic Information System for a Virtual University

Building a virtual university with all the characteristics mentioned in section 1.1 on Java is challenge. This thesis describes the design and implementation of two systems for a virtual university. One of the systems is a multimedia digital classroom supporting distance learning, the other is an academic information system supporting administration. We expect these two systems can enhance the productivity of a university and let the public attend classes through the Internet. Figure 1-1 shows the relationships between different roles of users and these two systems.

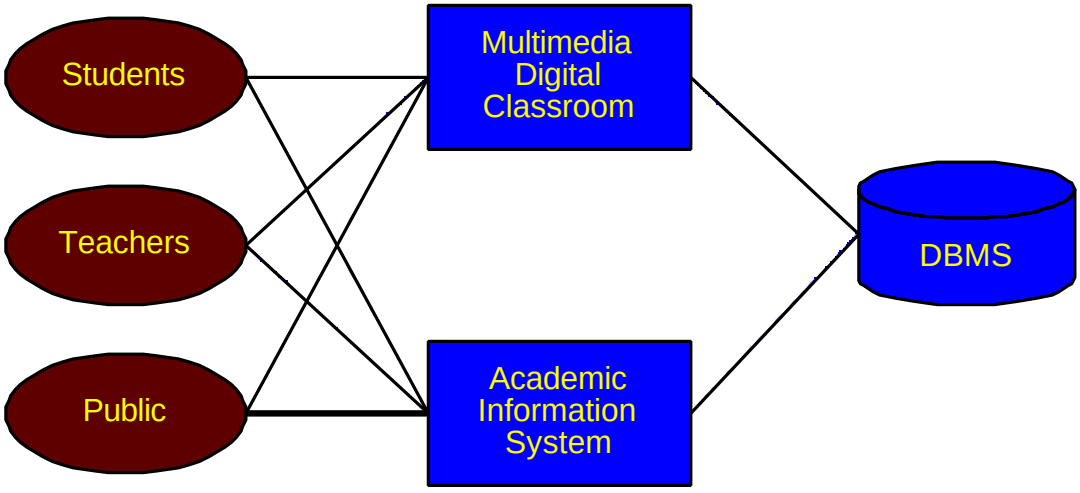


Figure 1-1. Multimedia digital classroom and academic information system for a virtual university.

The architecture of the multimedia digital classroom is shown in Figure 1-2. There can be many digital classrooms at the same time. All available classrooms register themselves to the classroom server, such that users can discover active classrooms from the classroom server. Each classroom consists of one teacher and many students. Each user uses the client software to perform educational activities. The client software should provide tools such as a multimedia Internet browser, an audio conference, and a shared whiteboard for material presentation and on-line discussion. All these tools are based on the underlining share manager to do application sharing, communication, synchronization, and access control.

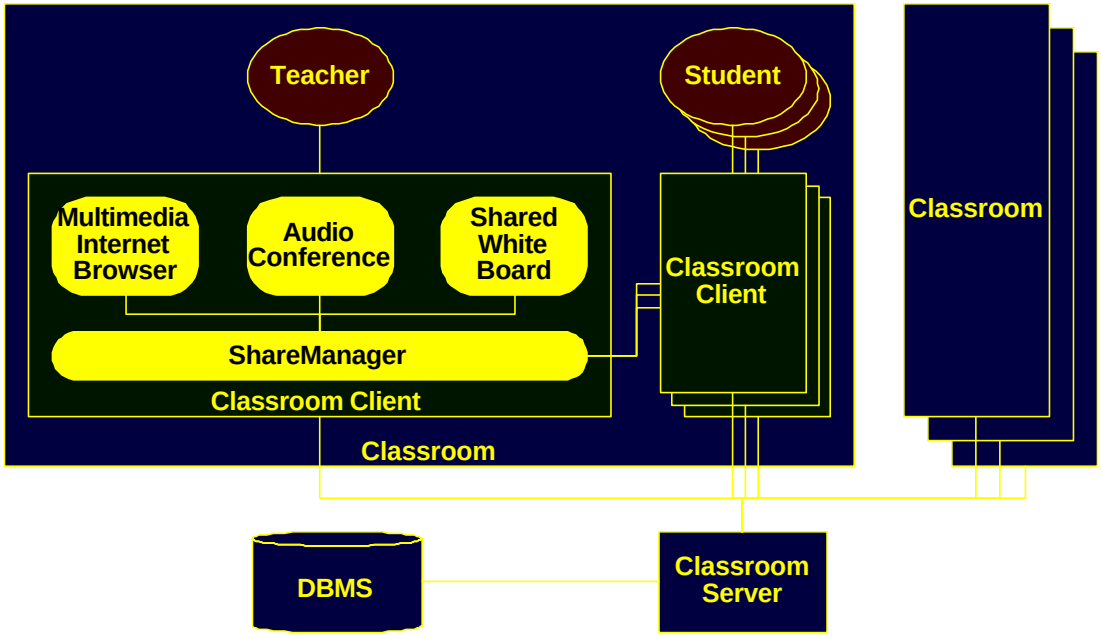


Figure 1-2. The architecture of a multimedia digital classroom.

Major issues about the design and implementation of multimedia digital classroom addressed in this thesis are:

- How to share different applications between teachers and students to provide better interaction. These applications provides various interaction patterns such as single-input-multiple-view, multi-input and local-view-only for distance learning.
- How to reduce network bandwidth requirement and still support multimedia presentation, audio conference and whiteboard to users from modem.
- How to design an extendible architecture to adapt new media and presentation methods.

The architecture of the academic information system is shown in Figure 1-3. Each user is served by a personalized agent to access academic information. The academic information system should support daily academic activities, such as:

- Query information and statistic of courses opened in each semester;
- Select and drop courses in a specific period of a semester;
- Automatically select mandatory courses for students;
- Update personal contact information;
- Query student information of a particular class;
- Send mail to students of a class;
- Query historic study status of a particular student;
- Query the scores of a student for a particular semester;
- Query the class time table of a student for current semester;
- Input score and syllabus of a class.

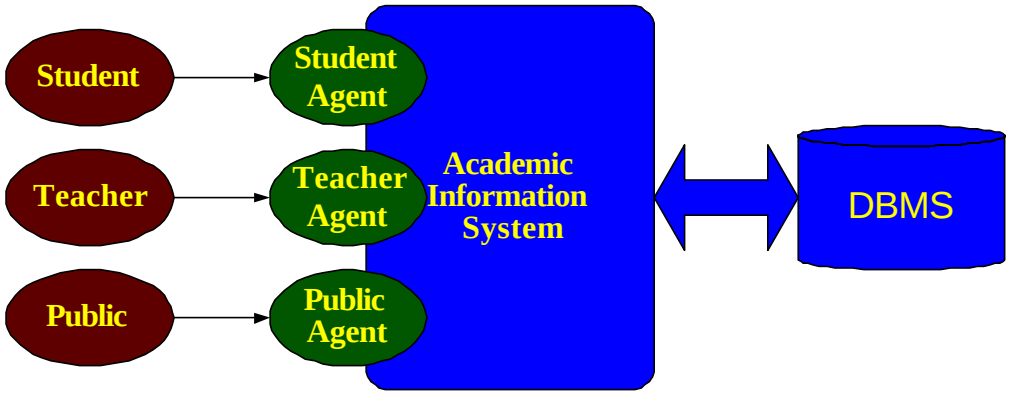


Figure 1-3. The architecture of an academic information system.

Major issues about the design and implementation of academic information system addressed in this thesis are:

- How to select courses in most efficient way to save everybody's time;
- How to provide instant information for everyone;
- How to provide sub-second response time in contemporary hardware;
- How to build a scalable system to support thousands of accesses in an hour;
- How to reduce the cost of development, deployment, administration and maintenance.

1.3 Organization of the Thesis

This thesis contains five chapters. Chapter 2 discusses the application sharing frameworks on Java. The Chapter proposed an application-sharing framework allowing single-user Java applications to be shared on network without modification, and a set of easy-to-use APIs for quickly writing complex cooperative applications. The hybrid of these two methods results a framework to build distance learning applications with different interaction patterns. Chapter 3 implements a multimedia digital classroom based on the work of Chapter 2. The digital classroom uses an extendible architecture to accommodate evolving teaching applications to support distance learning. It now consists of a multimedia browser, an audio conference, and a shared white board. Chapter 4 describes an academic information system on a Java-based three-tier architecture. The academic information system provides functions supporting academic administration activities of a university. The academic information system also shows great performance feasible for a large university. Conclusions and future works are given in Chapter 5.

Chapter 2 Application Sharing Frameworks

2.1 Introduction

There are two views about how to realize a CSCW (Computer Supported Cooperative Work) environment [MIN95]. One is based on the model of interaction between distributed objects [ITU95]. Although this distributed-objects model provides the most powerful mechanism to build complex cooperative applications, it usually requires building cooperative applications from scratch. This approach is too complex and expensive to make itself pervasive.

The other model is an application sharing system based on single-user applications. Running single-user applications on an application sharing system automatically makes these applications cooperative, with a user control the applications at any time. It is a bridge between the single-user applications and the environments of CSCW. This approach is cheap and intuitive, but its runtime environment limits the concurrent inputs from users.

There are three models concerning the architecture of application sharing systems, names *request-sharing*, *image-sharing* and *event-sharing*. In the *request-sharing* model, an application is started on a server and its graphical outputs are multiplexed to all the participants' window systems. The events from the window system holding the token are passed to the application, and all other window systems' inputs are blocked. Figure 2-1 shows the scenario of the request-sharing model.

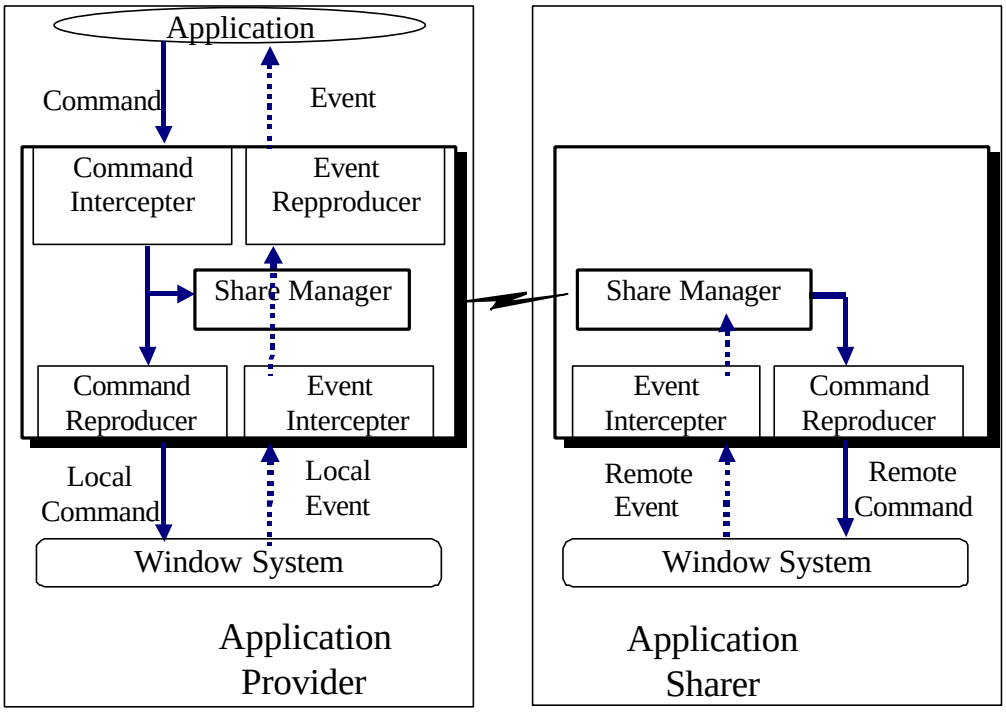


Figure 2-1. The request-sharing model

This model is able to share any single-user applications, and has been adopted in several commercial products and academic projects [ADB94][SHI96]. Although this model is quite successful, it can not support many users in heterogeneous environments due to the following reasons:

- Since the graphical outputs are duplicated to every session participants, this model consumes tremendous network bandwidth. This phenomenon is even worse for multimedia applications.
- The system is responsible for translating graphical outputs for different hardware configurations, e.g. different color models and resolutions. This translation also consumes much CPU time.
- It is difficult to translate graphical commands between different window systems.

Due to these performance and implementation problems, the request-sharing system can only support a session with several participants in a homogeneous environment [MIN94]. It cannot be scaled up for a large conference or an educational environment.

In the *image-sharing* model, an application is started on a server and its graphical image is periodically captured and multiplexed to all the participants' window systems. The events from the window system holding the token are translated then passed to the application, and all other window systems' inputs are blocked. Figure 2-2 shows the

scenario of the image-sharing model.

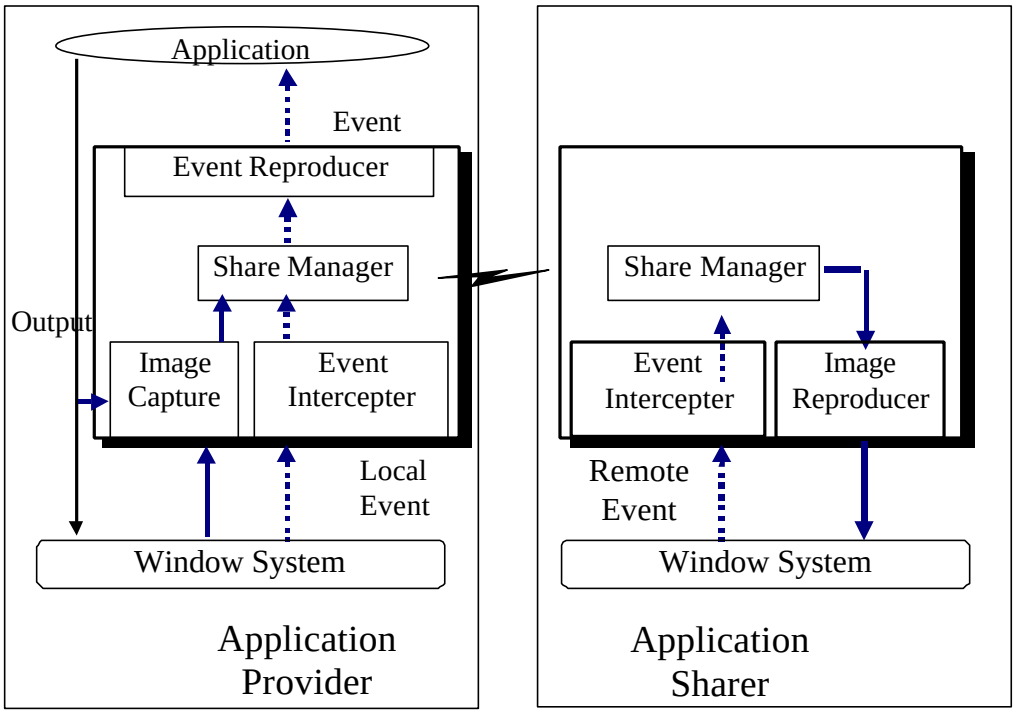


Figure 2-2. The image-sharing model

This model is also able to share any single-user applications, and has been adopted in Microsoft NetMeeting. Its main drawback is the consumption of tremendous network bandwidth, which makes the model unable to support many users at the same time. To reduce the requirement of network bandwidth, we can output an image only when it is modified, and group multiple graphical updates in a single image update.

The *event-sharing* model, as shown in Figure 2-3, presumes that the same application is started locally by all session participants. Then the input events from the current token holder are intercepted then sent to all participants. Every participant processes the remote input events as they are from local user and updates its local execution status.

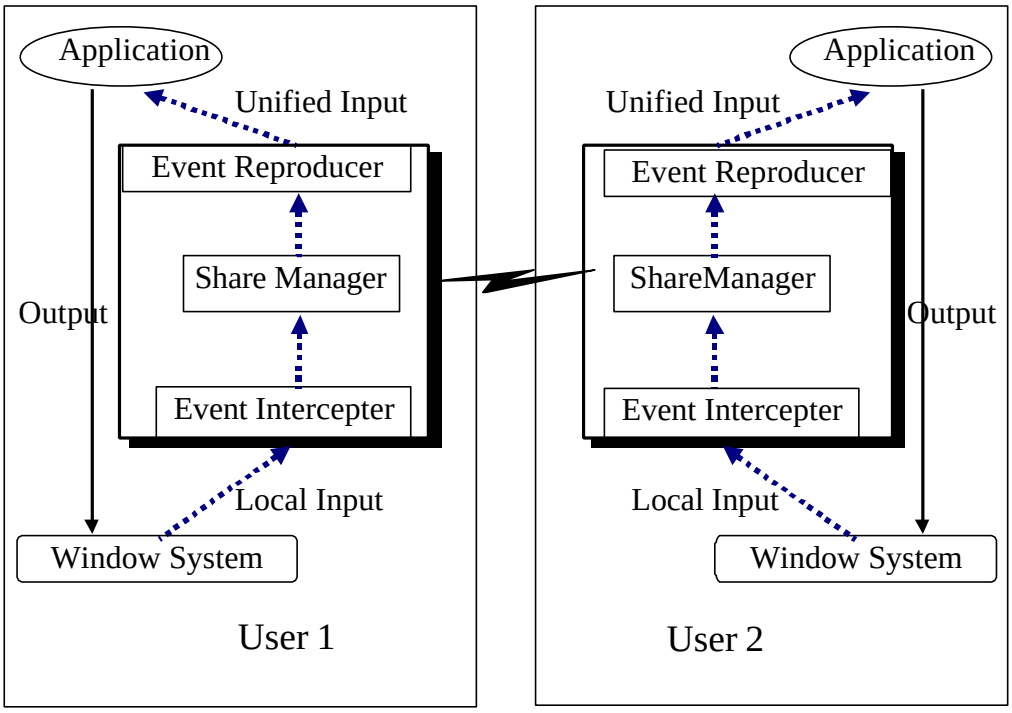


Figure 2-3. The event-sharing model

This model requires least network bandwidth, but has the following limitations and implementation difficulties:

- The shared application must be deterministic, in other words, the shared application must reach the same status on the same initial conditions and input events. Any time dependant functions can't generate the same results in this model.
- Every participant must have the same execution environment, for each shared application may access local environment variables or directories.
- Every participant must have its local copy of the same execution program.
- The shared applications are not allowed to update global data, for the update semantic may be ruined by many updates instead of one.
- The implementation must prevent the input events from being lost, since the processes running on some participants may not be ready to accept the input events from the current token holder due to slower execution speed.

The above constraints make the event-sharing system useful only for a limited scope of applications. It is often integrated with a distributed-object model to extend the functionality of an event-sharing system.

The multimedia digital classroom should provide presentation tools, audio conference, whiteboard, and examination tool etc. to support distance learning. These tools

have different interaction patterns in nature:

- Presentation can be thought as a single-input-multiple-view process. Only a user at the same time can generate inputs for this kind of applications and other users see the execution results.
- Discussion through whiteboard and chat room can be thought as a multiple-inputs process. Many users can generate inputs at the same time to express their idea and see results of all users.
- Examination can be thought as a local process. Each user manipulates and sees its own application context.

To support these interaction patterns, we proposed a hybrid model of event sharing and distributed objects. Section 2.2 discusses the implementation of event sharing model on Java. Section 2.3 proposes a simple set of APIs for fast building distributed object applications. The mechanism combining components of different interaction patterns is addressed at the end of Section 2.3.

2.2 Event-Sharing System

In CSCW, a *session* is a collection of users sharing an application. The user who starts up the shared application of a session is termed the *session owner*. All other users in the same session are called the *session participants*.

There are three issues to be addressed to implement the event-sharing model:

- How to setup and discover a session?
- How to manage the input token?
- How to intercept input events?

Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, multithreaded, and dynamic language. With the combination of an application sharing system and the Java language, it is possible to build a single-user Java application, then execute the application cooperatively in a heterogeneous environment without any modification.

The Abstract Window Toolkit (AWT) consists of APIs responsible for building the graphical user interface (GUI) for Java programs. JDK 1.1 uses a peer model to provide a truly native look-and-feel by delegating the look-and-feel of the components to a set of underlying “peer” classes. Each platform has a different set of peer classes to wrap the native system’s widgets. Java programs use these peer classes to output

graphical commands to the underlying window system.

In our implementation of the event-sharing model, a centralized scheme is adopted for session setup, discovery, and management, as shown in Figure 2-4. A register server running on a host provides all session participants the information of currently opened classes. On opening a class, the session owner registers the class to the register server, such that other users can find where to join the session and what applications should be started. Once session participant sets up connection with session owner, it starts Java applications by using reflection APIs. The detailed protocol for session setup is illustrated in Figure 2-5.

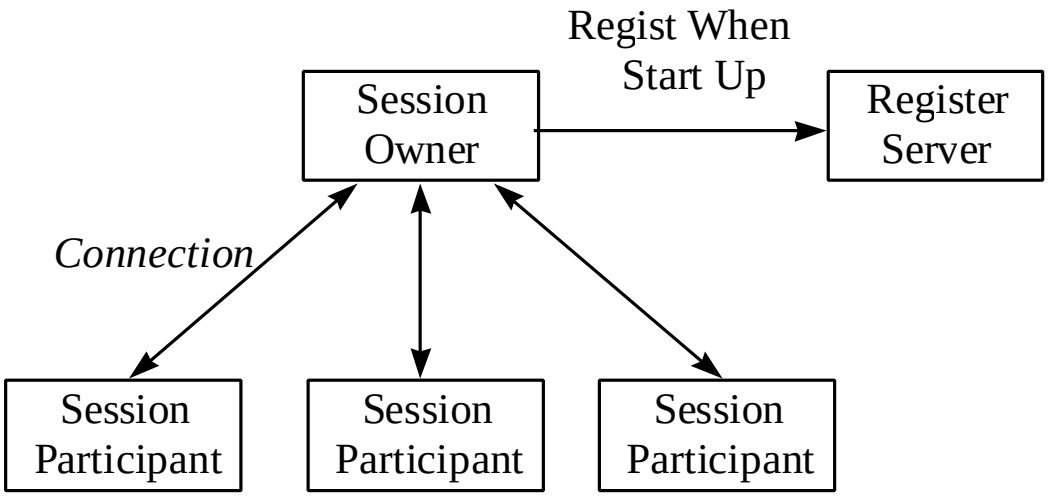


Figure 2-4. The architecture of event propagation

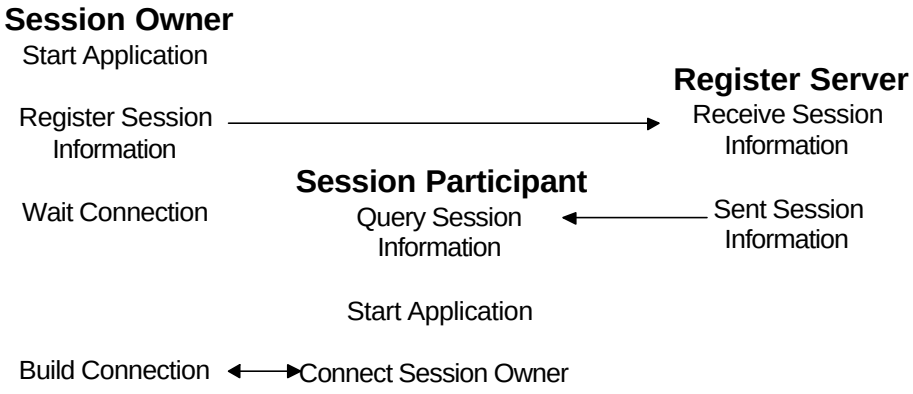


Figure 2-5. Session setup protocol

The session owner is responsible for the event propagation and the token management. At any instance, only one of the session participants holding the session token is allowed to generate input events. Once a session is initiated, the session owner holds the token in the first place. On receiving a token requesting command, the session owner immediately revokes the token from the current holder and grants it to the re-

quester. This protocol is designed to maintain the synchronization between the token and delayed events, since there can have events generated while token is revoked. If the current token holder leave the session normally or abnormally, the session owner gains the session token again. The session owner also has the privilege of taking back the input token at any time. The token management protocol is shown in Figure 2-6.

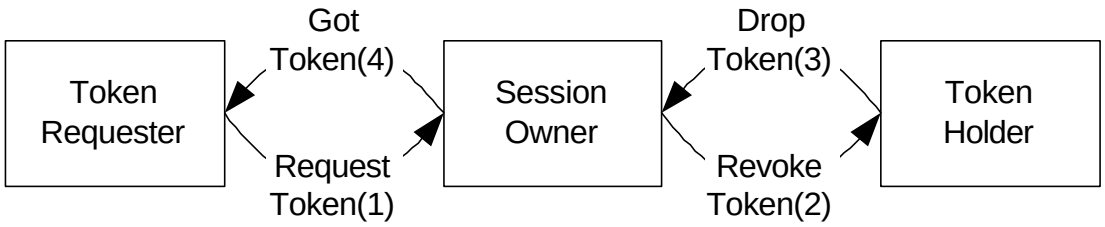


Figure 2-6. Token management protocol

In our implementation, two of the AWT classes - *Component* and *MenuComponent*, are rewritten for the event interception. The original classes of JDK can be replaced by putting our classes in the CLASSPATH environment variable instead of modifying the JDK package. This method avoids the break of JDK license agreement.

As being the roots of the class hierarchy of all visible AWT components, both of these two classes define the method *dispatchEventImpl* to dispatch events. We rewrite the *dispatchEventImpl* methods to intercept input events. On receiving an event, the rewritten *dispatchEventImpl* takes the following steps to process the events:

1. Checks if the event needs pre-processing. We have to take special actions for the following events to get correct behavior:
   - PAINT, UPDATE: ensures the content of the component is properly painted. These events are generated when user moves, resizes or exposes the component.
   - FOCUS_GAINED: ensures the component becomes the focus owner.
   - KEY_PRESSED, KEY_RELEASED: ensures hotkeys be properly processed.
   - MOUSE_PRESSED: ensures the component gets focus.
   - MOUSE_DRAGGED, MOUSE_RELEASED: checks if session control panel should be pooped out.
2. Check if the event should be processed locally. This includes COMPONENT_MOVED, COMPONENT_RESIZED, COMPONENT_SHOWN, COMPONENT_HIDDEN, FOCUS_GAINED, and FOCUS_LOST.

3. If the user holds the input token and the event is either consumed or a mouse-move event, the event is broadcast to all participants of the session. Otherwise, the event is discarded. The mouse events are always broadcast from token holder to all session participants for rendering a pseudo mouse cursor.

The propagated events are written to and reconstructed from network through Java Object Serialization API. For the transient event target, which can not be serialized by Object Serialization API, we encode it as the path from the root window to the target. Suppose we have a GUI hierarchy shown in Figure 2-7, the component "Button2" will be encoded as "WindowID, 0 0 1." The window ID is defined as a serial number of created windows since the application started. Because the constructor of class Component has also been modified, we can log any created windows for encoding event target.

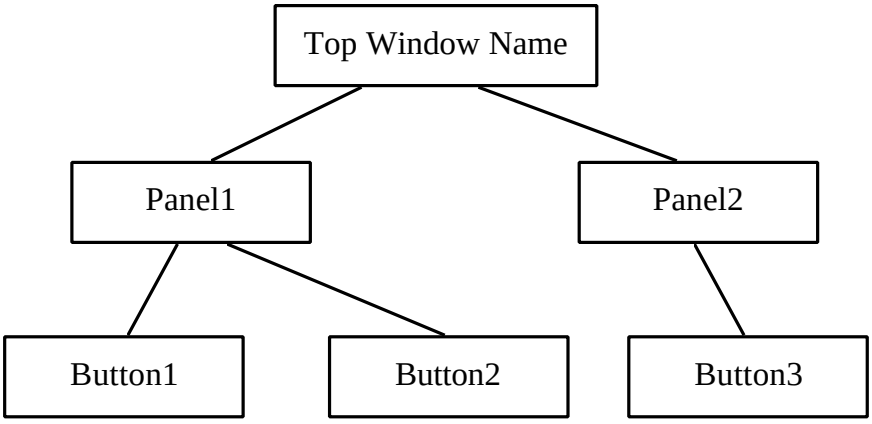


Figure 2-7. A GUI hierarchy example

Once a session participant receives a serialized event, it reconstructs the event and simulates the normal *dispatchEvent* method. The only difference is that either the target component does not exist or there is no component to consume the event. This happens when the program is not ready to accept the event. The participant thus will sleep for a while and try again until the event is consumed. The control flow of event interception and playback is shown in Figure 2-8.

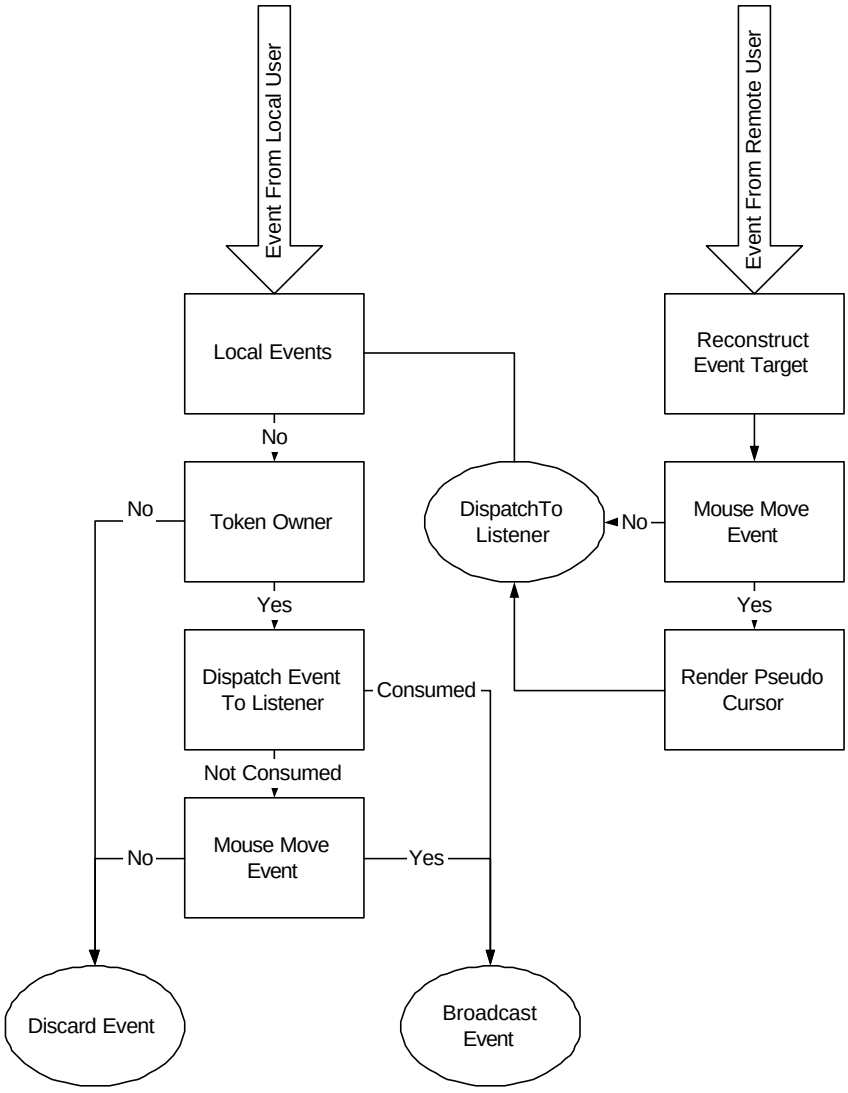


Figure 2-8. Control flow of event interception and playback.

In our implementation of event-sharing model, only two kinds of events are broadcast to each participant:

- Consumed event, which is necessary for program status synchronization.
- Mouse-move event, which is used to highlight token holder‘s moving of mouse.

We can further reduce the bandwidth requirement by grouping adjacent mouse-move events and still provide clear highlighting feature. Applications discussed in Chapter 3 shows our application-sharing framework consumes only about two hundred bytes per second.

Since all input events are passed to and recorded in the session owner, the architecture can easily support the late-join users. When a connection is established, before sending any new coming events, the session owner dumps to this connection the chains

of input events that have been recorded since the beginning of this session. Under such circumstance, the participant could be synchronized by fast playing back those input events.

2.3 APIs for Distributed Objects

In addition to the event-sharing model, we have also implemented a distributed-objects framework for Java programs. To modify a Java program for this framework, we first have to implement the *ShareAction* interface, which defines the only method *doAction (Object command)* for a *ShareAction* component. The *doAction* method is responsible for parsing the *command* argument to do certain application specific actions. Then we replace the related method invocations in the event handling routine with the *ShareManager.broadcastAction (Component target, String command)* method, where *command* is string encoding an application specific command and *target* is the component which implements the *ShareAction* interface. The *broadcastAction* method uses the network channels shown in Figure 2-7 to broadcast and synchronize the *ShareAction* command. When a *ShareManager* receives a *ShareAction* command, it calls the *doAction* method of the target component to manipulate the input command.

This design philosophy allows minimum efforts for converting single-user applications to cooperative applications. The only work is to implement the single method interface *ShareAction*, then modify the event handling routine to ask *ShareManager* broadcasting commands instead of calling application specific methods. Modules need to be added or modified are shown in Figure 2-9. Our experience about modifying a single-user whiteboard to a cooperative whiteboard shows that less than one hundred lines of code need to be modified.

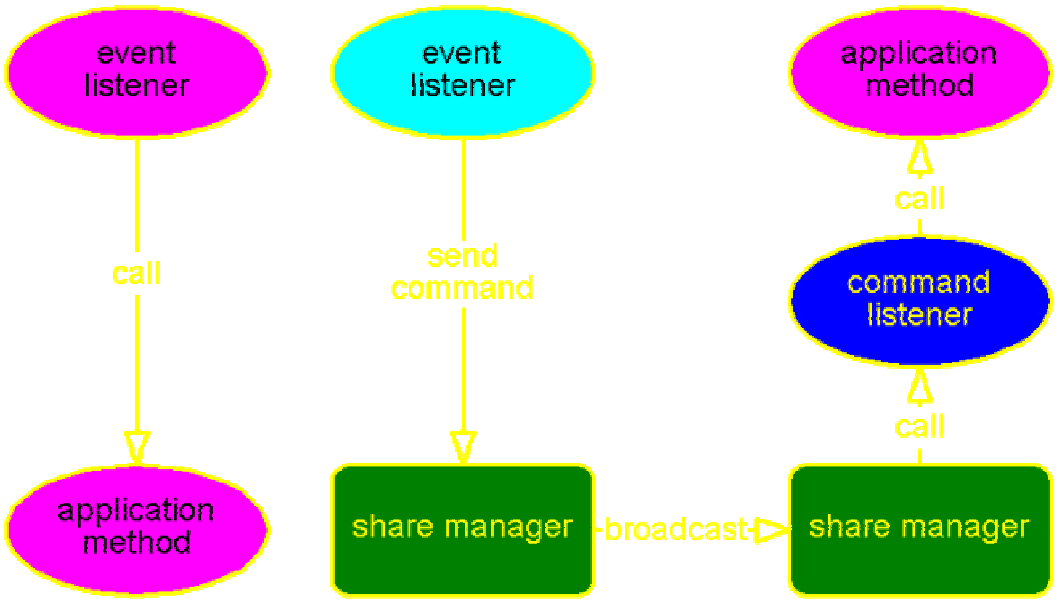


Figure 2-9. Modify single use application to distributed model. Only the event listener is modified and a command listener is added.

We also provide another interface called as "*PrivateCom*" for forcing applications sticking in single-user mode. If any object implements the *PrivateCom* interface, its input events will not be captured by the event-sharing system. It is useful for the work that is operated by the local participant only, for example, the examination system.

Our model allows users to implement their applications in a hybrid style. Users can implement the *ShareAction* or *PrivateCom* interface optionally for those components that can accept multiple input at one time or want to adhere to single-user mode.

When the *dispatchtEvent* method of a component receives an input event, it first checks if the target component is contained in any component of the type *ShareAction*. If the component is contained in a *ShareAction* component, no matter the user holds the token or not, the input event will be processed by the application. If the component is contained in a *PrivateCom* component, then the event is processed in single-user mode. Otherwise, only the user holds the token can process the input event. Such an event passing mechanism allows a single-user application to include multiple *ShareAction* and *PrivateCom* components. An example of hybrid components is shown in Figure 2-10.

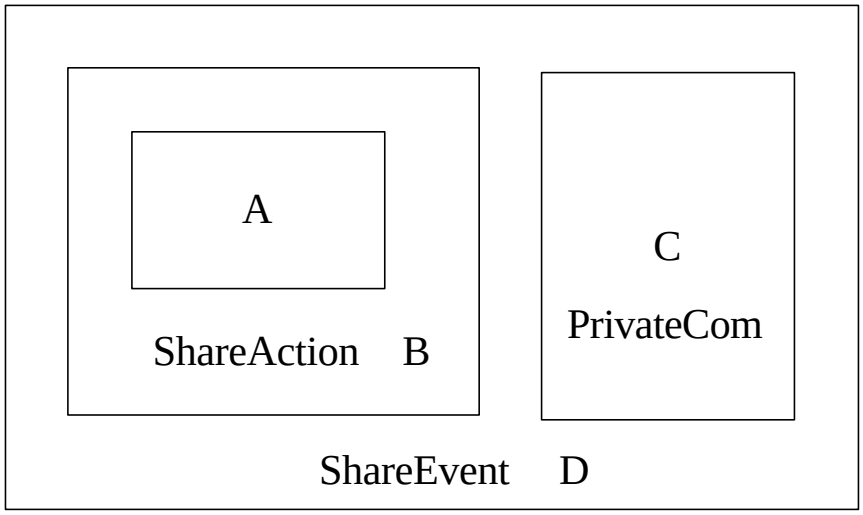


Figure 2-10. Container hierarchy of different components

Chapter 3 Tools for Multimedia Digital Classroom

3.1 Introduction

In the digital classroom, a variety of functionality should be provided to support all types of learning processes, such as lectures and discussions [Kou96]. Due to the unique characteristics of platform transparency and application protocol integration, WWW technologies have been adopted for information sharing on the cyberspace. Hence, we design and implement a WWW multimedia digital classroom and its applications using Java. The multimedia digital classroom includes many teaching tools, such as Multimedia Internet Browser, Shared White Board, and Multi-Channel Audio Conference. As the teaching materials can be any interactive applets for better presentation, the digital classroom uses the application sharing system in Chapter 2 to extend the capability of showing from static web pages to dynamic web pages.

3.2 Design of Tools for Multimedia Digital Classroom

The System architecture of our Multimedia Digital Classroom (MDC) is based on the application-sharing system discussed in Chapter 2. There is one teacher and a number of students in a classroom. The teacher first creates a classroom on a host and logs the class information to a register server. Whenever a student wants to find a classroom, he/she has to consult the register server. He/she then creates a connection with the teacher.

We build a package composed of many teaching application tools on Java, which are described as follows:

Multimedia Internet Browser

The WWW is very popular recently. Multimedia Internet Browser provides a bridge between digital classroom and WWW. A teacher uses this web browser to open a hypertext document that contains the contents of the current course. The teacher can lead the students to study on the WWW by sharing the same view of the navigated web pages. This approach lets teachers use global WWW resources to organize their teaching materials.

Figure 3-1 shows a detailed diagram of the architecture of the Multimedia Internet

Browser. In addition to the parsing capability of the HTML format, it also provides the ability to browse VRML documents and display some kinds of multimedia data, such as the MPEG-1 audio and video. As different applets are used to render different media, the Multimedia Internet Browser can be configured to dynamically load applets at run-time. This makes the Multimedia Internet Browser very flexible and extensible.

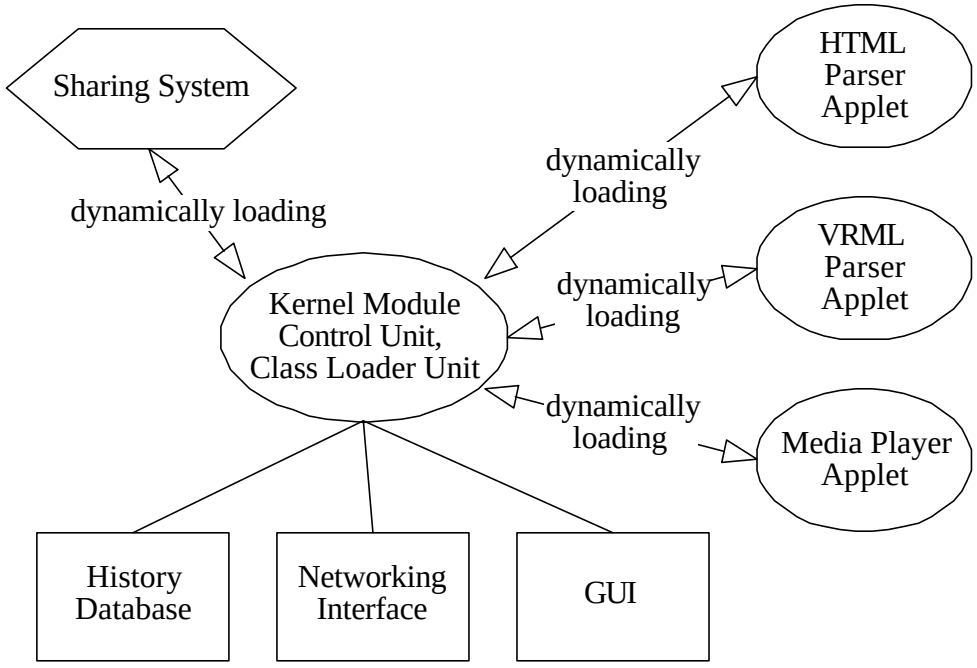


Figure 3-1. Architecture of Multimedia Internet Browser

Multi-Channel Audio Conference

Voice is a direct communication method traditionally. If teachers and students use voice to communicate with each other in the digital classroom, the learning process will be more efficient because students can ask questions directly about difficult or unclear teaching materials.

There are normally three ways to transmit audio data in an audio conference environment:

- Every participant broadcast compressed audio data to all the other participants. Every participant receives audio packets then decompresses and mixes them.
- Every participant send compressed audio data to the session owner. The session owner decompresses and mixes all audio data, then compresses and sends to all participants.
- Every participant send compressed audio data to the session owner. The session owner decompresses all audio data, but mixes individually for each par-

ticipant to make echo cancellation. Then the session owner sends back mixed audio to every participant.

Table 3-1 shows the overhead comparison for an audio conference with *n* users talking at the same time.

Method	Packets	Manager	Participant	Delay	Echo
Peer to peer	n*(n-1)	(1, n-1, 1)	(1, n-1, 1)	One way	No
Centralized echo	2*(n-1)	(1, n-1, 1)	(1, 1, 0)	Round trip	Yes
Centralized no echo	2*(n-1)	(n-1, n-1, n)	(1, 1, 0)	Round trip	No

Table 3-1. Comparison of audio transmission methods (compression times, decompression times, mixing times)

We choose the peer to peer model for the transmitting audio data in our audio conference for echo cancellation and lower network delay. As in the digital classroom environment there are few people talking at the same time, the $O(n^2)$ network bandwidth requirement won't be a critical factor. When the audio conference starts, voice uttered by all participants is recorded, compressed, and then sent to each participant. Each participant receives all audio source packets. The system then decompresses, mixes, and plays back. Scenario of the transmission of the audio data is show in Figure 3-2. Every participant uses UDP to transfer audio data. This may cause packet loss but has fastest response time and less jitter.

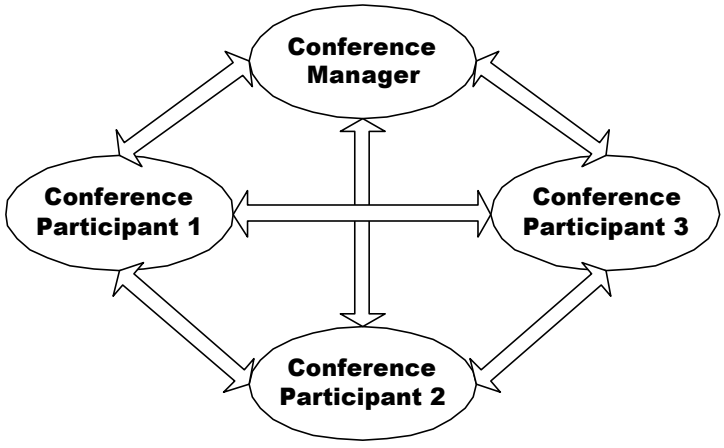


Figure 3-2. The transmission of audio data uses the peer to peer model.

On the other hand, we choose the client-server model shown in Figure 3-3 for the transmission of control data for synchronization and centralized management. This model uses TCP to prevent packet loss and ensure consistence.

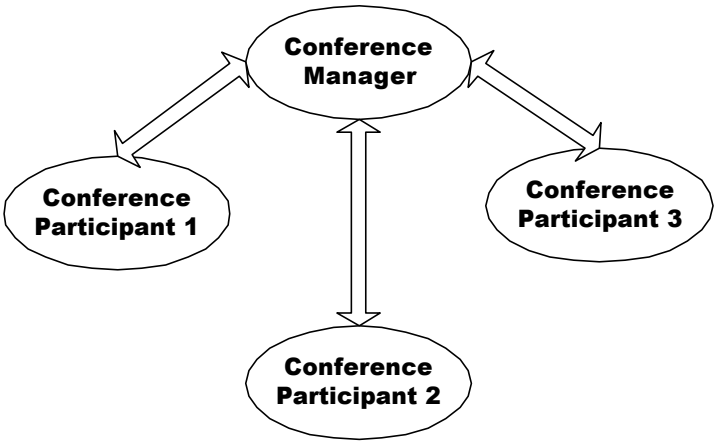


Figure 3-3. The transmission of control data uses the centralized model.

Shared White Board

Because of the event-sharing system described in Chapter 2, a simple drawing-panel Java application executed by the classroom manager becomes a shared whiteboard without any change. To enable concurrent access to a shared white board based on the command sharing model in Chapter 2, we implement a drawing panel with some basic functions, such as drawing basic graphics, writing texts, and multiple drawing pages. All people using the shared white board at the same time in the digital classroom do not interfere with each other.

3.3 Implementation of Tools for Multimedia Digital Classroom

The MDC system is implemented in a heterogeneous environment where workstations and PCs connected through an FDDI and an Ethernet network. We list the development environment as follows:

	PC	WORKSTATION
Operating System	Windows 95	Sun Solaris 2.5.1
Java Compiler	JDK 1.1	JDK 1.1
Native C Compiler	Microsoft Visual C++	GNU C Compiler
Network	10 Base-T Ethernet	10 Base-2 Ethernet

Table 3-2. Development environment of MDC

We have tested MDC system with participants from Windows 95 with Ethernet, Windows 95 with modem and SUN Solaris with Ethernet, respectively. The snapshot of the Multimedia Internet Browser is shown in Figure 3-4. The Multimedia Internet Browser supports functions for the HTML 3.0 tag, VRML 2.0 tag, MPEG audio/video

and several kinds of image formats. All modules of Multimedia Internet Browser described in Section 2 are applet-based and dynamically loaded by the browser kernel. They are functionally independent between each other so that performance enhancement can be achieved by using the multi-thread technique to execute different modules in parallel. With applet-based module design, we can get the maximum extension for future improvement. New modules, downloaded from web server as Java applets, could be added into the browser dynamically.

User can also add other functions to the browser without changing anything if these functions are written by Java applet class.

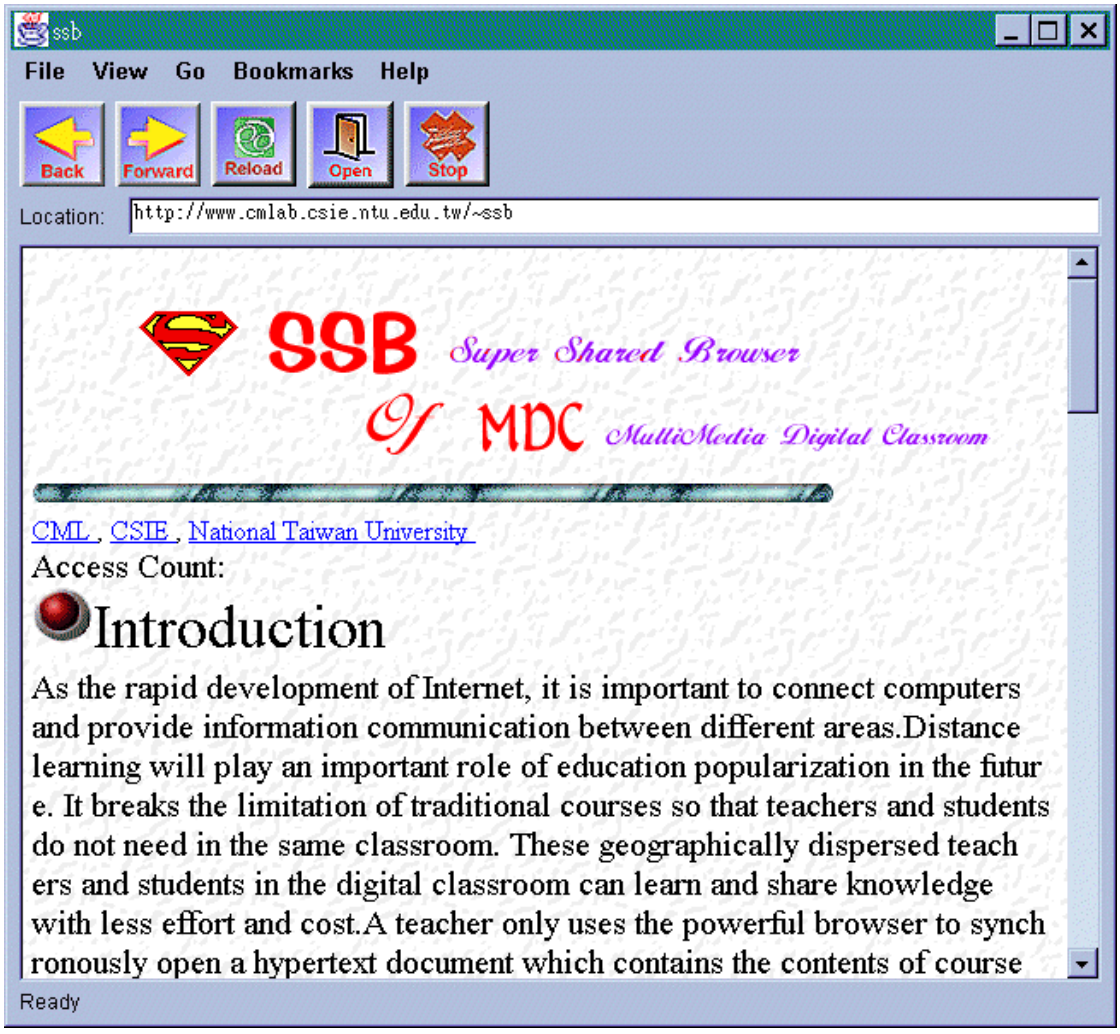


Figure 3-4. The Multimedia Internet Browser

When a teacher starts the Audio Conference, the control panel shown in Figure 3-5 will appear. The control panel shows how many participants are on this digital classroom, and who is speaking now. User can adjust the volume of recorder or player by dragging the controlling scrollbars.

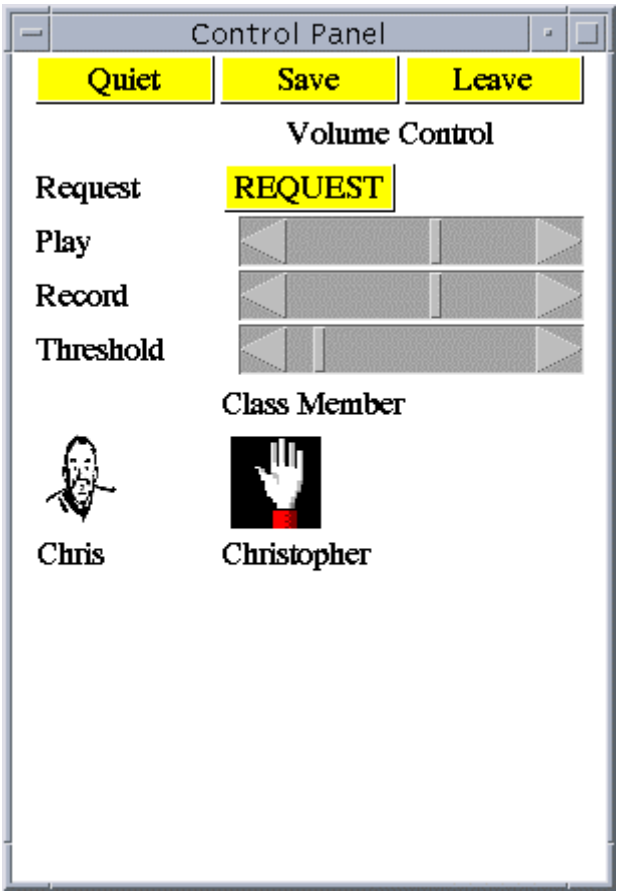


Figure 3-5. Control panel of the Audio Conference

We use ITU-T G.723.1-6.3 [ITU96-1] audio compression / decompression standard to reduce audio data size. The time frame is 60ms, and audio data is about 6.3kbits/sec*60ms ˜ 48bytes. We add a 16 bytes header to audio packet, and their meanings are listed in Table 3-2. In the environment of only one user keeps talking, the network bandwidth requirement will be 64bytes/60ms ˜ 8.53kbits/s.

Field name	Length(Byte)	Meaning
Size	1	Audio data size
Id	1	Participant's ID
Magic	1	Error detection
Reserved	5	Reserved for future extension
Seq	8	Time stamp

Table 3-3. Meaning of various fields in the header of an audio packet.

As the peer to peer transmission model of audio data needs n*(n-1) packets for a conference with n people talking at the same time, today's modem can support three users speak at the same time. With silence detection and access control, this method can support tens of students from various network configurations and still leaves enough bandwidth for multimedia presentation. Compressed audio packets are transmitted by

the UDP protocol. Besides, we use multiple threads for implementing the audio recorder, player, network handler, and central controller modules in order to enhance performance and avoid busy waiting situation.

We also provide a chat room and a shared whiteboard shown in Figure 3-6 and Figure 3-7 in the MDC system. These two applications are based on the distributed object APIs mentioned in Section 2.3. They are first written as single user application, and then modified to collaborative objects with less than 100 lines of code.

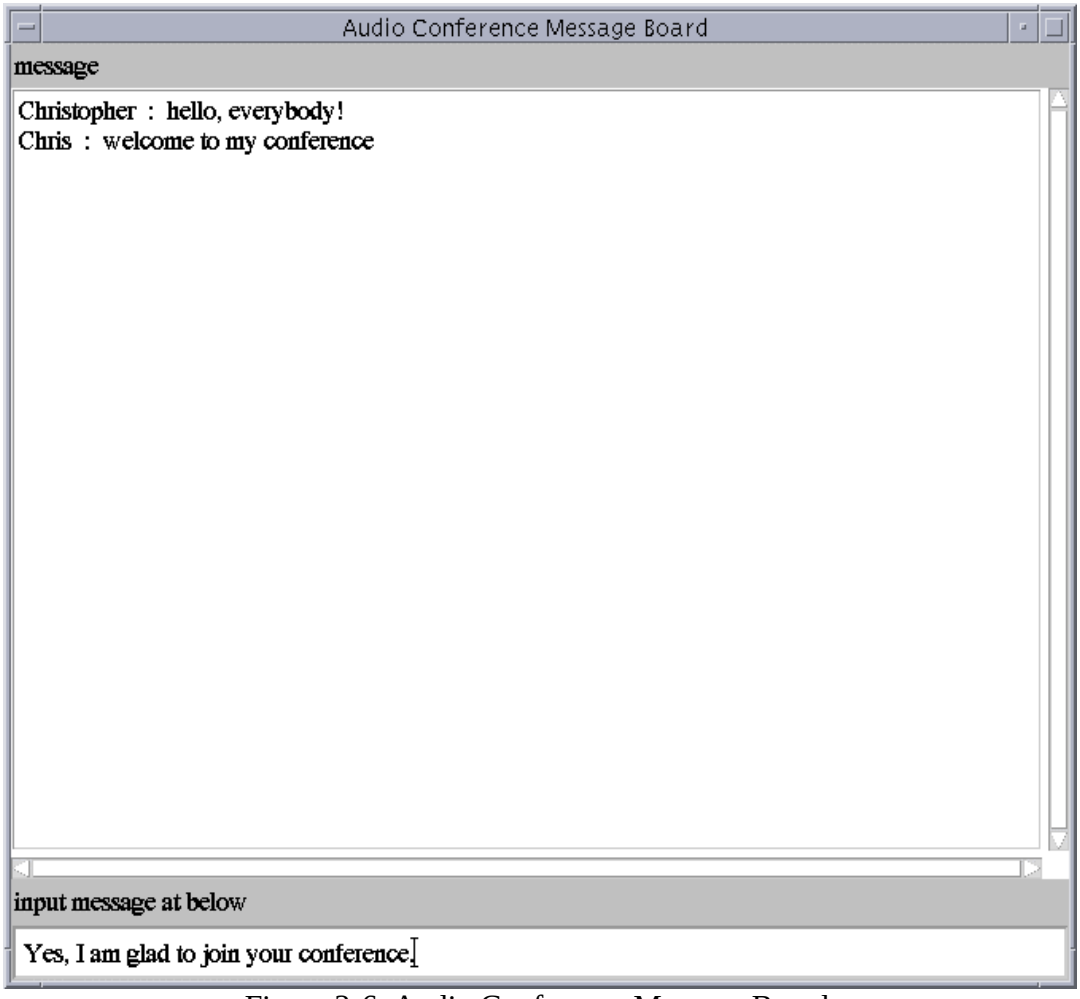


Figure 3-6. Audio Conference Message Board.

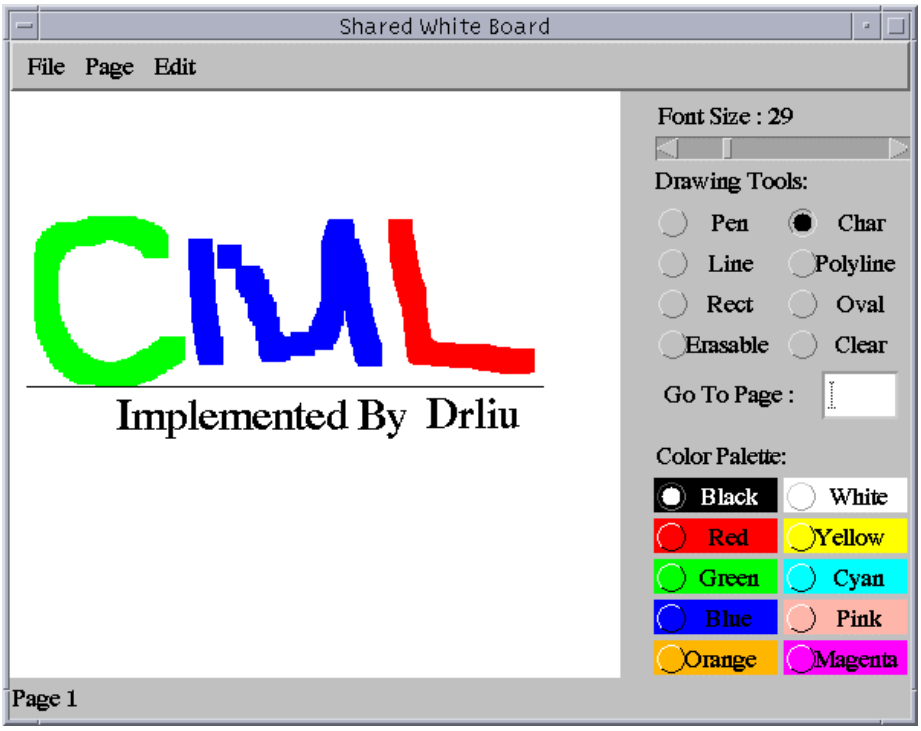


Figure 3-7. Graphics Shared White Board.

3.4 Discussions

This section discusses important implementation problems and the approaches we took to resolve these issues.

Performance Challenge

In order to achieve platform transparency, Java adopts the virtual machine technology. Each Java supporting platform executes the virtual machine first. Then the virtual machine loads the class file and interpreters its byte codes [Lin96]. Due to this kind of approach, performance of Java applications is usually slow in current environment. Although just-in-time (JIT) compiler has been introduced to alleviate the performance problem, we still need optimized native code to support real time and CPU intensive applications. Therefore, we use C native codes to record, compress, decompress, and mix audio data.

Weak Device Interface Supported by Java

Java lacks audio device interface in JDK1.1. We use native codes to develop audio device interface, including the recorder and player. The method increases the complexity of the overall system. Moreover, we must implement different device interfaces in order to sustain the cross-platform property. Java Media Framework (JMF), released with the Java2 platform extension, provides a better environment for multimedia

applications. As JMF evolves to support more multimedia formats and provides better performance, we can write pure Java audio conference in the near future.

Network Limitation

The ideal network environment of the MDC system should have the following properties:

- Low delay latency;
- Error free transmission;
- Multi-points communication.

However, Internet adopts the TCP/IP protocol to transmit the data packets. TCP/IP protocols guarantee error-free transmission, but do not support bounded transmission delay and multi-points communication. Therefore, we use UDP to send the audio data that need real-time transmission. The MDC server constructs many point-to-point TCP connections for each client to transmit shared events and actions. Although this approach solves the multi-point communication problem, it wastes too much network bandwidth for transmitting duplicate packets on these point-to-point connections.

Chapter 4 Academic Information System

4.1 Introduction

Chapters 2 and 3 have addressed the design and implementation issues of multimedia digital classroom for a distance learning environment on Internet. A complete virtual university needs an academic information system to address the problem of administration inefficiency in contemporary universities. The booming Internet industry has made WWW a major channel for publishing enterprise information. As there are many WWW technologies available, we must consider the usage patterns of an academic information system before choosing an ideal tool for it:

- Workload is very high in some periods. In the course selection period, most students try to login the system in the first hour to register hot courses. This makes performance a very critical factor to be addressed.
- Novice users need to do their jobs in a few minutes right after the first usage of the academic information system. How to design an easy and intuitive user interface is very important.
- Users need much information to make the decision of course selection and check the status of graduation. How to provide personalized information efficiently without paper is the key to reduce working time and administrative cost.

Most of the early implementations of the WWW publishing systems are done using the CGI technique. However, due to the poor performance and the lacks of interaction, the CGI is not an ideal tool to build an academic information system. Many technologies have been proposed to solve the performance problem of CGI, such as IDC, ASP and ISAPI. Their major contribution is using multi-thread instead of multi-process to reduce server load. The interaction problem at client side has been alleviated by dynamic HTML, VB script and Java script.

With the abilities to build cross-platform business logic into the browser and application server, Java gives WWW better performance and user interaction to meet the needs of academic information system [RAM97]. Our study results in a Java based 3-tier architecture shown in Figure 4-1. Under this architecture, once a client is connected to the system, the authenticator first checks the user‘s account and password

nected to the system, the authenticator first checks the user‘s account and password within a database table. If the password is valid, it then creates a corresponding user object to serve the client. Instead of building a database connection for each user object, all SQL commands are passed through the database connection pool. The connection pool is an array of pre-established JDBC *Connection* objects. It uses a round-robin allocation strategy when a user object requires a database connection. The user object uses the *synchronized* Java keyword to lock the database connection during a transaction to prevent contention on the database connection. This architecture has the following advantages over the other approaches:

- Java applet has the full power of programming language to build complex interactive applications. This could provide most efficient user interface to make the job to be done in optimal way.
- UI updates are handled by applet, and then can alleviate server load;
- RMI server is statefull and thus provides greater opportunity for performance tuning and security management.
- The traffic between client and middleware is further reduced;
- The database connection pool pre-allocates database connections to reduce response time;
- The database connection pool increases the utilization of database connections, thus reduces the license fee.

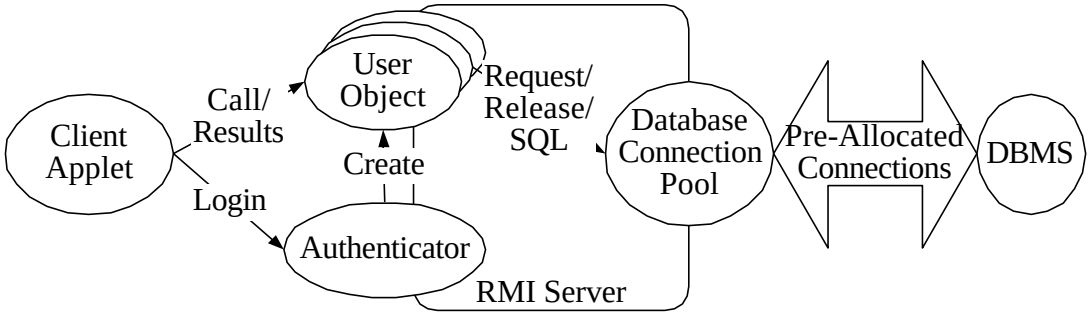


Figure 4-1. The architecture of RMI based 3-tier information system.

4.2 System Design and Implementation

Currently, our system provides the following functionality:

- Query courses;
- Select and drop courses;

- Automatically select mandatory courses for students;
- Update personal contact information;
- Query student information of a particular class;
- Send mail to students of a class;
- Query historic study status;
- Query scores in a semester;
- Query class time table of current semester;
- Input score and syllabus of a class;

System features, document for data analysis and design details are listed in Appendixes A and B. Compare to systems of other universities, our implementation has the following outperformed characteristics:

- Single login point. All functionality and information are integrated together by different roles of users. This provides a single authentication to all information.
- Smart course selection wizard. Our implementation provides the students personalized information to guide the course selection. All academic rules, such as mandatory courses, preliminary courses, double major and assistant major, are examined to filter available courses. Students see exactly what they can choose, no more and no less. Course selection can be done in several mouse clicks even for novice.
- Fixed interfaces for repetitive tasks. As we use Java applets, GUI of repetitive tasks are designed to let users complete their job in a minimum of navigation.

4.3 Performance Analysis

Performance is the most critical concern of current Java implementations. To evaluate the response time of our system, we setup a simulation environment with two servers for DBMS and RMI server whose configurations are shown in Table 4-1. These two servers are connected within the same local area network, and are not isolated from outside environment. Although this configuration may introduce some variances and make simulation results not accurately match theoretic prediction; it reflects the real runtime environment where many network applications competing with each other for network bandwidth.

CPU	Pentium 133
RAM	64MB
Hard disk	EIDE 4.0GB*1
Network adapter	10Base-T Ethernet
OS	Windows NT 4.0 + SP3
DBMS	SQL Server 6.5 + SP3
Language	JDK1.1.7
WWW server	IIS 3.0
Database connection software	JDBC-ODBC Bridge

Table 4-1. Server side configurations.

Workload is generated by a client running multi-threads to simulate many students connected at the same time. The client is an AMD K6-266 machine connected to a local area network with two routers away from servers. Each student executes the operations listed in Table 4.2 once per visit to our system, assuming that each operation is issued every 10 seconds. To evaluate the response time of our system, we use random distribution as our workload model. The response time is measured by the worst out of the best 90% time and illustrated in Figure 4-2. The simulation shows that the system can serve 200 visits per hour.

Operation	**Remote calls**	**Query complexity**
SelectCourse	Get credit hint	3 select statement
	Get core credit hint	8 select statement
	Get note	1 select statement
	Get selected course	7 select statement 1 prepared statement executed 8 times
	Get unselected course	7 select statement 2 prepared statement executed 150 times
QueryCourseD	Query course	1 select statement 5 prepared statements each executed 50 times
QueryCourse1	Query course	1 select statement 5 prepared statements each executed 40 times
QueryCourse2	Query course	1 select statement 5 prepared statements each executed 11 times
QueryCourse3	Query course	1 select statement 5 prepared statements each executed 22 times
QueryCourse4	Query course	1 select statement 5 prepared statements each executed 16 times
ListCourse	Query selected course	7 select statements 1 prepared statement executed 8 times
QueryScore	Query score	1 select statement
StuLogin	Login validation	1 select statement

Table 4-2. Operations and query complexity used in response time simulation.

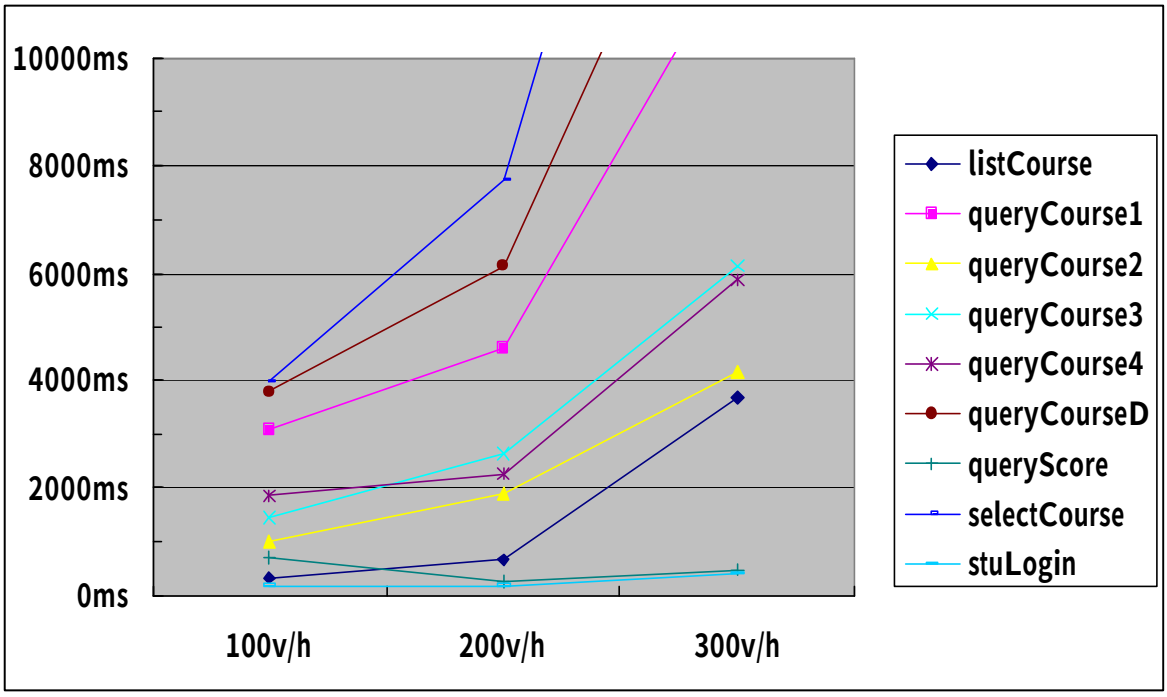


Figure 4-2. Response time vs. workload.

4.4 Performance Optimization

Most of the database overheads come from those prepared or heavily joined statements. These overheads normally come from complex database schema and application requirements. To optimize our system performance, we have adopted the following techniques:

- Database de-normalization. For information read frequently and updated rarely, we can use database de-normalization to prevent tables-joins or prepared statements. Course type, number of student and name of teachers are such kind of information in our system.
- Caching. There are information updated rarely in academic information system, which can be cached in RMI server to avoid database access. These kinds of information include preliminary requirement, mandatory course requirement, graduation rule, major department, double-major department, and assistant-major department. Data consistency between database and cache is maintained by administrator, and typically executed at most once per semester.
- Incremental status update. Some information generated in previous operations can be kept for later use instead of querying entirely from database. Credits passed or selected can be incrementally updated in RMI server instead of querying everything every time from database.
- Rewrite SQL commands. As we have cached much information in RMI server, some complex and slow SQL statements can be rewritten to a more concise

form. We use this technique to reduce the response time of course queries when query condition contains department's name.

- Query path heuristic. SQL is a declaration language. Its execution speed is highly affected by indexes and query optimizer. Although we have built proper indexes for frequent queries, the query optimizer in Microsoft SQL server may not be smart enough to choose the optimal access path. After careful examination of the execution plans generated by Microsoft SQL server, we give access hints to two select statements. This saves 250ms execution time totally on the database server.

The query complexity of each operation after optimization is listed in Table 4-2. All prepared statements have been eliminated from previous implementation. Every remote operation except one is now done at a single carefully examined SQL command.

Operation	**Remote operation**	**Query complexity**
SelectCourse	Get credit hint	
	Get core credit hint	
	Get note	
	Query selected course	1 select statement
	Query unselected course	2 select statement
QueryCourseD	Query course	1 select statement
QueryCourse1	Query course	1 select statement
QueryCourse2	Query course	1 select statement
QueryCourse3	Query course	1 select statement
QueryCourse4	Query course	1 select statement
ListCourse	Query table	1 select statement
QueryScore	Query score	1 select statement
StuLogin	Login validation	1 select statement

Table 4-3. Query complexity of each simulated operation after optimization.

The response times and DB/RMI CPU load ratios listed in Table 4-3 are generated by 100 repetitively runs of the same operation. It shows RMI server consumed about 100% to 200% of CPU time than database server. With new JIT (Just-In-Time) compiler technology reported in IBM JDK1.1.7 and Sun Hot Spot engine, our implementation becomes database bound. As we have off loaded many overheads from database to RMI server and the bottleneck is still on database server, we can claim that our implementation is optimal.

Remote operation	Response time	DB/RMI load ratio
Get credit hint	4ms	0
Get core credit hint	6ms	0
Get note	10ms	0
Query selected course	100ms	0.6
Query unselected course	570ms	0.6
Query courseD	350ms	0.5
Query course1	190ms	0.4
Query course2	90ms	0.8
Query course3	120ms	0.5
Query course4	100ms	0.6
List course	50ms	1.0
Query score	40ms	0.9
Login validation	50ms	0.2

Table 4-4. Response time and DB/RMI CPU load ratio of simulated operations after optimization.

Simulation showed in Figure 4-3 to Figure 4-11 predicts that the system could serve 1500 visits per hour with response time being less than 10 seconds.

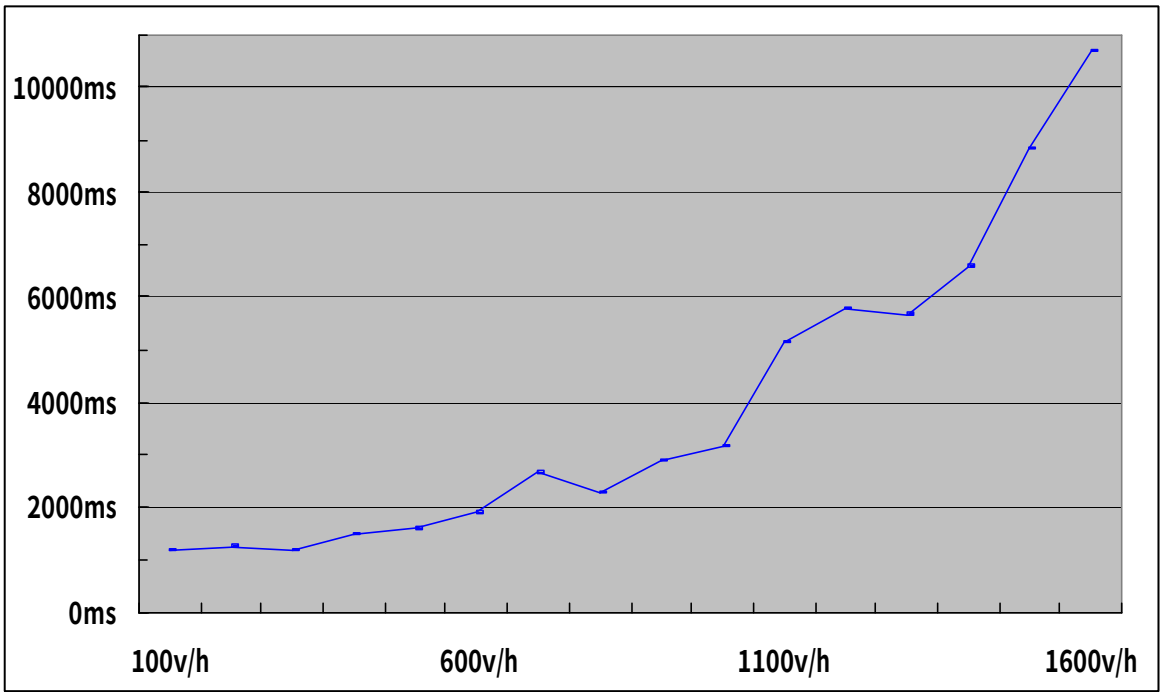


Figure 4-3. Response time vs. workload for selectCourse operation.

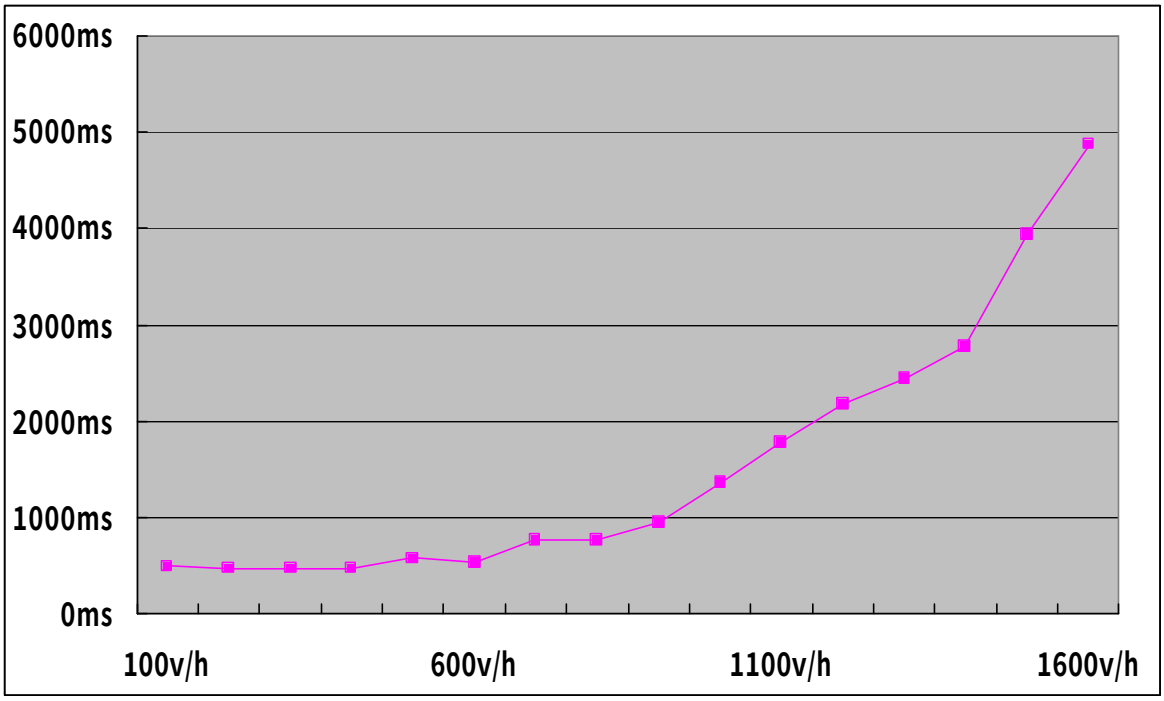


Figure 4-4. Response time vs. workload for queryCourse1 operation.

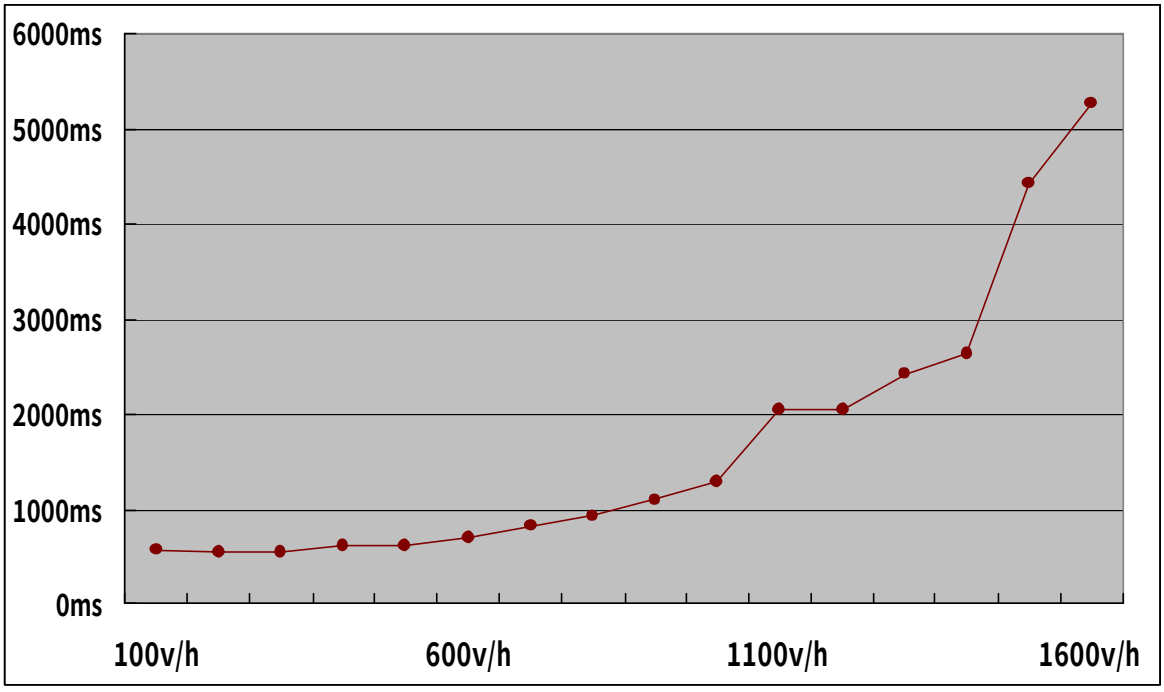


Figure 4-5. Response time vs. workload for queryCourseD operation.

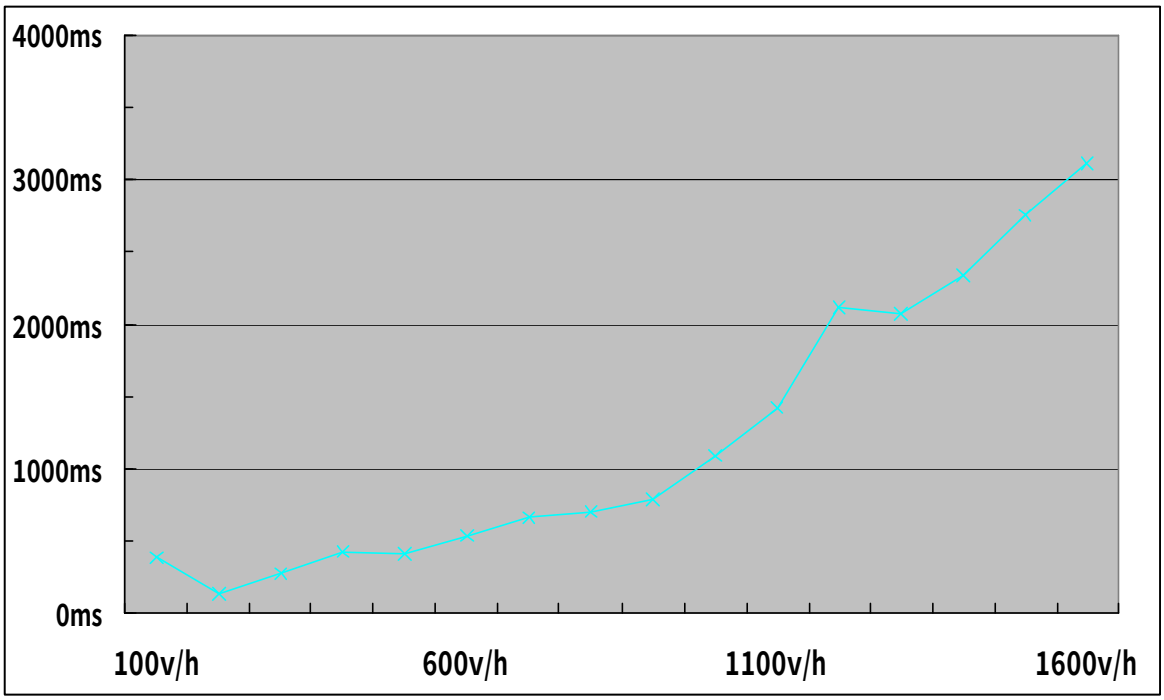


Figure 4-6. Response time vs. workload for queryCourse3 operation.

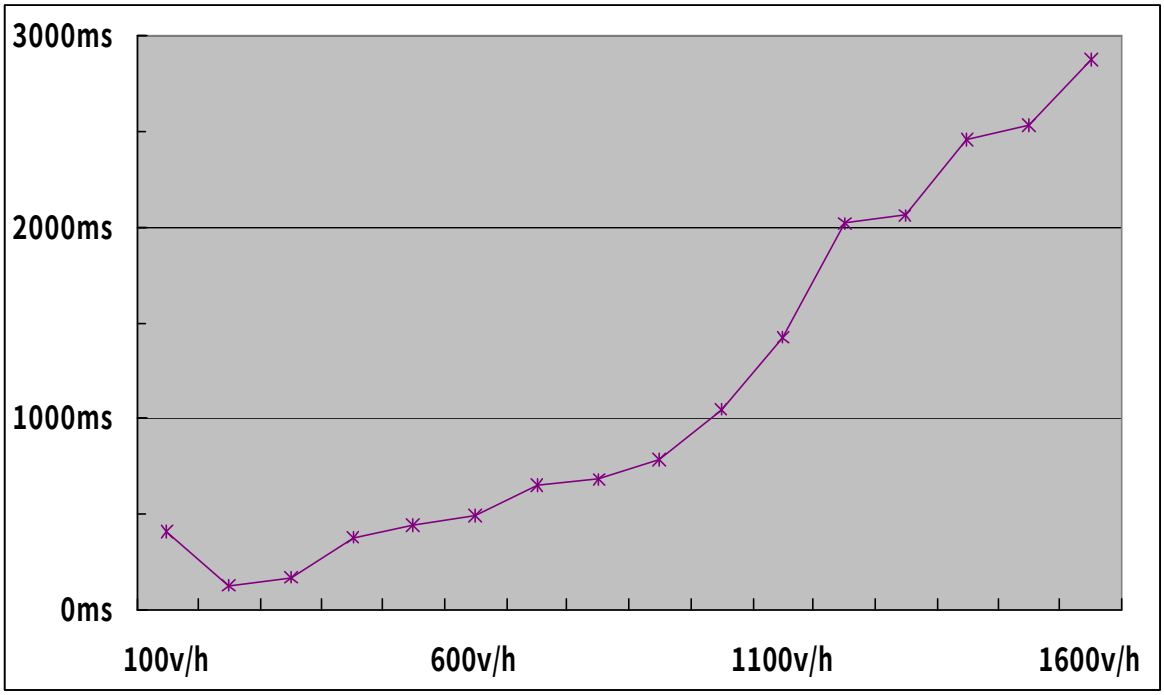


Figure 4-7. Response time vs. workload for queryCourse4 operation.

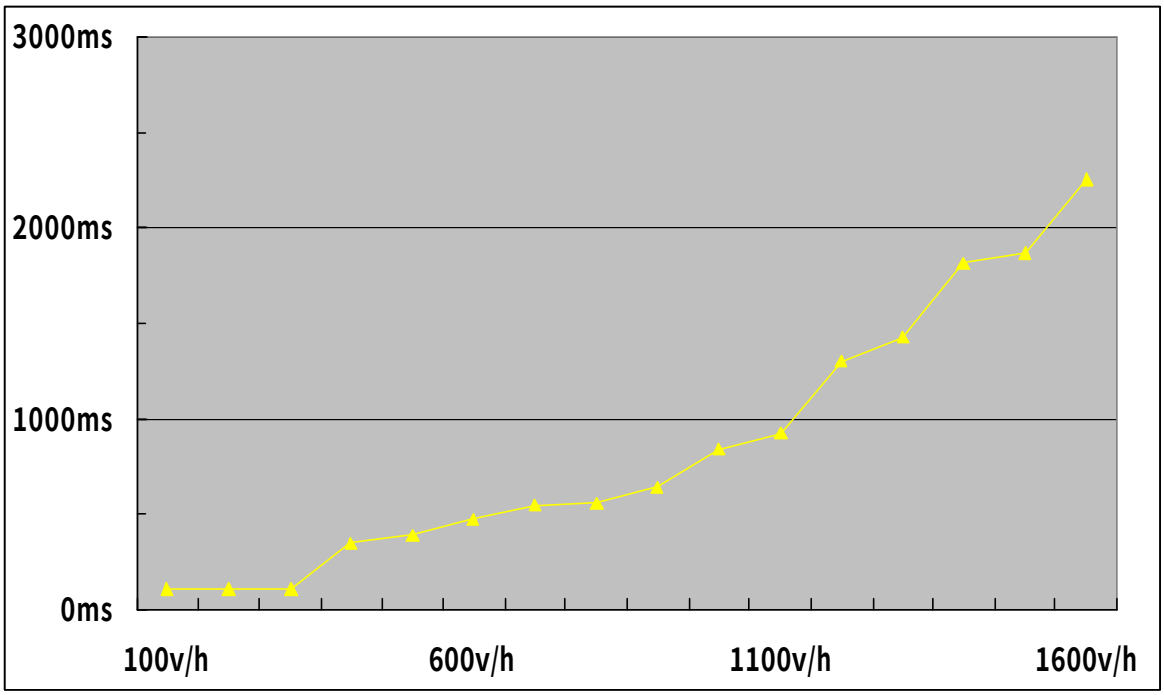


Figure 4-8. Response time vs. workload for queryCourse2 operation.

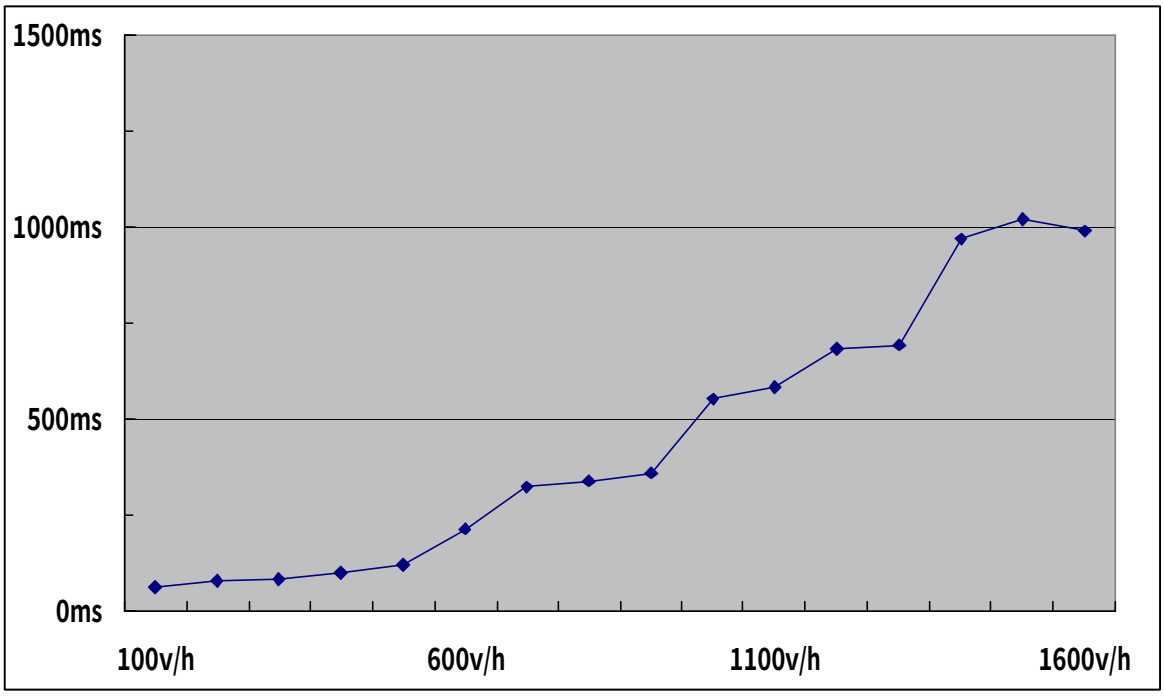


Figure 4-9. Response time vs. workload for listCourse operation.

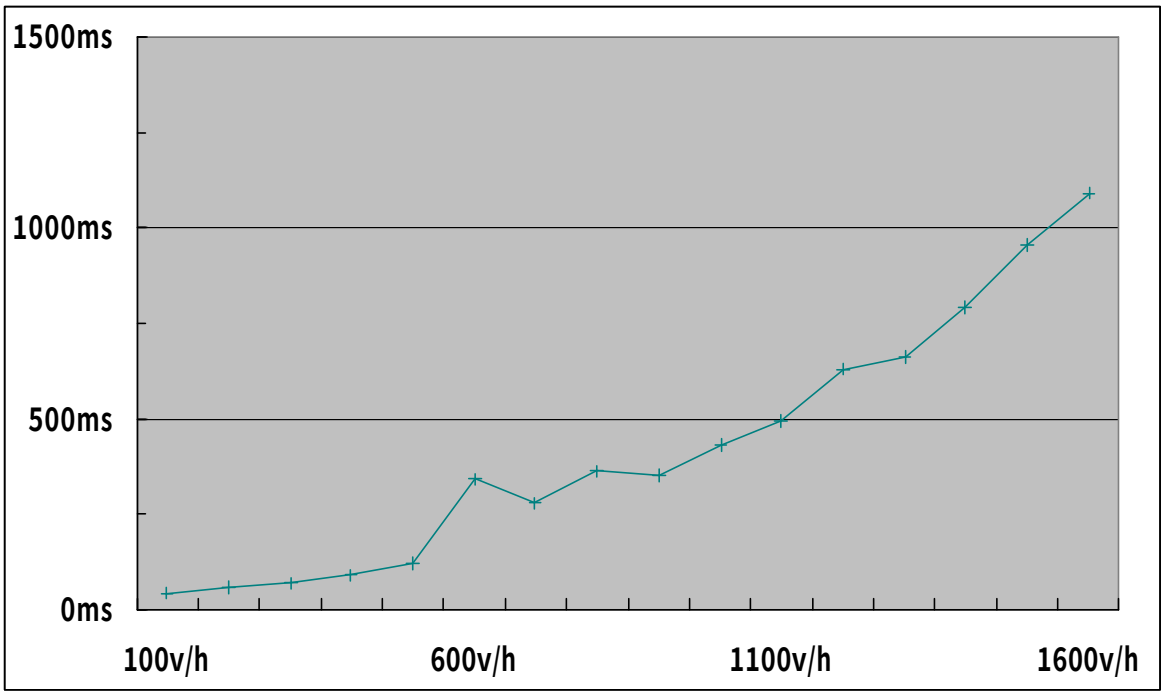


Figure 4-10. Response time vs. workload for queryScore operation.

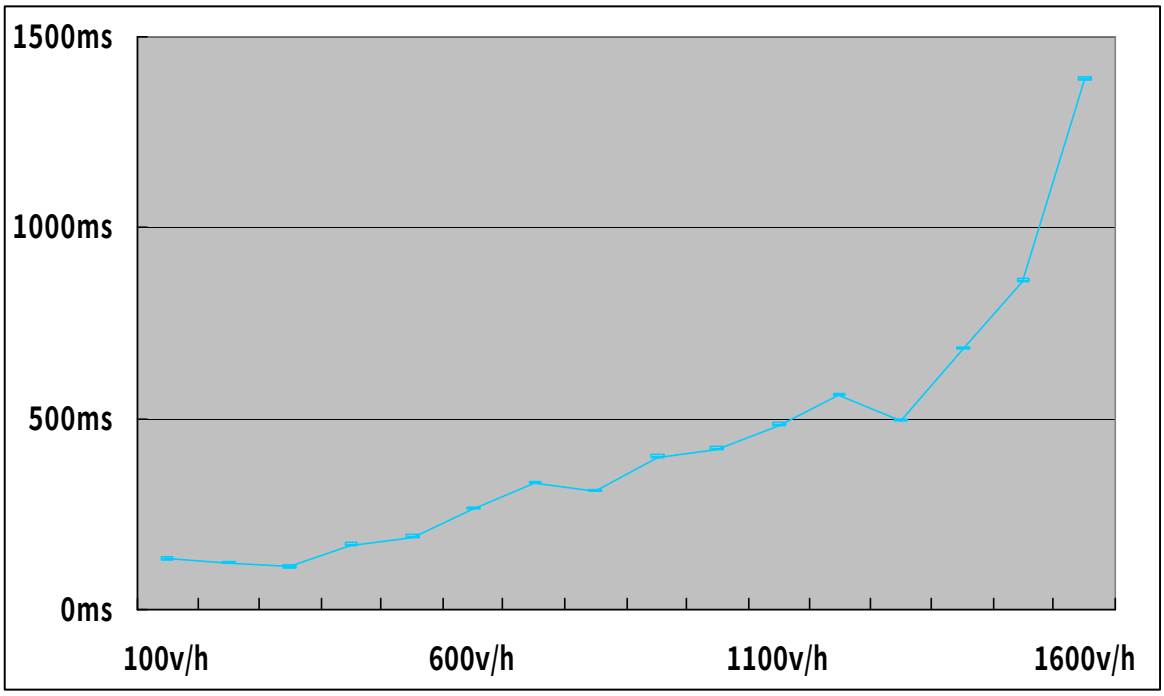


Figure 4-11. Response time vs. workload for stuLogin operation.

In course selection period, our experience shows that about 40% users will visit our system within the first hour. This makes our system running within the same environment as that in simulation feasible for a university with 4000 students.

4.5 Workaround Bugs in JDK1.1

There are some serious bugs in JDK1.1 that needs to be solved to make the aca-

demic information system works. Two tricky bugs and their solutions are discussed as follows:

JDBC driver internationalization problem

As most database servers can not store Unicode strings, JDBC drivers need to do encoding translation on the fly. The JDBC-ODBC bridge driver in JDK1.1 doesn't implement the correct internationalization specification. Instead, it truncates the high byte of Unicode when writes to database, and pads high byte with zero for each byte when reads from database. To work around this bug, we write a class SQL listed below. Then wrap code with SQL.toSQL() and SQL.fromSQL() before sending to and after reading from database.

```
import sun.io.*;
/**
 * The class is used to work around a bug in JDBC drivers. The driver truncate the leading
 * byte of Unicode character before sending to DBMS. This only works for ASCII character.
 * To handle strings to DBMS, you have to use <code>SQL.toSQL()</code> to wrap the output
 * string. JDBC drivers also treat the data from DBMS as ASCII code by adding 0 before every
 * bytes of the incoming data. This also produces errors for Non-ASCII characters. To work
 * around this bug, you have to wrap the incoming data with <code>SQL.fromSQL()</code>
 */
public class SQL {
    static ByteToCharConverter toChar;
    static CharToByteConverter toByte;
    static {
        try {
            SQL.toChar = ByteToCharConverter.getConverter("Big5");
            SQL.toByte = CharToByteConverter.getConverter("Big5");
        } catch(Exception ex) {
            ex.printStackTrace();
            System.exit(1);
        }
    }
    /**
    * convert a string to ASCII byte array
     */
   public static byte[] toAscii(String s) {
       if (s==null) return new byte[0];
        try {
            synchronized(toByte) {
                return toByte.convertAll(s.toCharArray());
            }
        } catch(Exception ex) {
            return new byte[0];
        }
    }
    /**
    * Convert a string for output to DBMS
    * @param s The string needs to be converted.
     * @return A string which will be truncated by JDBC to produce Big5 characters.
     */
```

```
    public static String toSQL(String s) {
        if (s==null) return "";
         byte[] orig;
         try {
              synchronized(toByte) {
                   orig = toByte.convertAll(s.toCharArray());
               }
             char[] dest = new char[orig.length];
            for (int i=0; i < orig.length; i++)
                  dest[i] = (char)orig[i];
              return new String(dest);
        } catch (Exception e) {
               e.printStackTrace();
              return s;
         }
     }
     /**
    * Convert an incorrect string produeced by JDBC drivers to correct Unicode string.
    * @param s The string needs to be converted.
    * @return A correct Unicode string.
      */
     public static String fromSQL(String s) {
          int i, j;
         if (s==null) return "";

         char[] orig = s.toCharArray();
         j = orig.length;
        byte[] dest = new byte[j];
        for (i=0; i < j; i++)
             dest[i] = (byte) orig[i];
         try {
              synchronized(toChar) {
                   return new String(toChar.convertAll(dest));
               }
         } catch (Exception e) {
               e.printStackTrace();
              return s;
          }
     }
}
```

RMI server can't be accessed cross sub-net on Windows NT

RMI server uses domain name instead of IP to publish its services. As the fully qualified domain name is not always available on every platform, this approach may cause cross sub-net problem. The implementation of java.net.InetAddress on Windows NT can't return fully qualified domain name. To solve this problem, we modify the line in java.net.InetAddress:

```
localHost.hostName = impl.getLocalHostName();
```

to

```
localHost.hostName = "admwww.cc.ncnu.edu.tw";
```

to give the RMI server a fixed domain name and place the modified

java.net.InetAddress in proper directory to replace that in JDK library.

Chapter 5 Conclusions and Future Work

5.1 Conclusions

The event-sharing framework on Java allows the possibility of building single-input collaboration tools from existing applications without any change. The distributed object API makes the building of multiple simultaneous input applications easy and fast. The hybrid model we propose can share most Java applications transparently, and can let programmers build sophisticated collaboration tools. Single-input, multiple-input and local AWT objects can be integrated together in a Java Virtual Machine. This framework can be used as a solid runtime environment for various tools for multimedia digital classroom.

Tools for multimedia digital classroom, such as browser, shared whiteboard and audio conference, are built on the application-sharing framework to support distance learning on Internet. The extendible architecture of our digital classroom makes new methods of material presentation possible whenever they are ready on Java.

Our implementation of the multimedia digital classroom can support most users doing distance learning on Internet. Multimedia presentation, audio conference, shared white-board, chat room and Java application sharing are all simultaneously available through modem connections on different platforms. The multimedia digital classroom is technically feasible for large applications of distance learning on Internet.

The academic information system exhibits features such as easy to use, high performance, robust, low bandwidth, and low cost. Our real world experience at National ChiNan University shows that even novice users on Internet can inquire academic information easily without any help. Our optimization efforts have made the academic information system feasible for a university of 15000 students on nowaday hardware. The cost estimation in Table 5-1 shows that hardware and software costing only 150000 NT$ is needed to support a large university.

Hardware/Software Cost	**Price(NT$)**
Dual Pentium II 350+256MB RAM	100000
Windows NT 4.0	24000
SQL Server 6.5	30000
Total	154000

Table 5-1 Cost to support a university of 15000 students.

The estimated benefits after using our academic information system, for a university of 15000 students, are summarized in Table 5-2.

Benefit	Amount	Unit price(NT$)	Return(NT$/year)
Paperless	300000 papers	0.5/paper	150000
No key in	150000 entries	0.5/entry	75000
Save student's time	15000 hours	75/hour	1125000
Save staff's time	1200 hours	250/hour	300000
		Total	1650000

Table 5-2 Returns of academic information system.

A virtual university based on our implementation of multimedia digital classroom and academic information system can alleviate or even solve the problem of escalating costs and uneven demographics.

5.2 Future Works

Our design and implementation of a virtual university are based on WWW and Java technologies. As these technologies are changing rapidly, new software and material presentation methods are introduced constantly. To satisfy new requirements and promote our system to more people, maintenance is always needed. Some of the future works are listed as following:

- Apply multimedia digital classroom to real distance learning courses. Although we have tested the multimedia digital classroom in various configurations and shown that it is feasible on Internet, experience from users is always valuable to improve the overall quality of the system. Audio jitter, functionality of shared whiteboard and supporting of HTML tags is to be evaluated in real distance learning courses.
- Port the application-sharing framework to JDK1.2 that has a new pure Java user interface framework called Swing. This effort can bring new applications written in Swing to multimedia digital classroom.
- Extend the functionality of academic information system, such as homework hand-in and course evaluation. Promotion of the RMI based 3-tier architecture will also be applied to other university information system, such as budget control and property management system.

Bibliography

[ABD91] H. Abdel-Wahab, and M. Feit, "XTV: A Framework for Sharing X Windows Clients in Remote Synchronous Collaboration", *Proceedings IEEE Conference on Communications Software: Communications for Distributed Applications & Systems*, Chapel Hill, NC, April 1991, pp. 159-167.

[ABD94] H. Abdel-Wahab and K. Jeffay, "Issues, Problems and Solutions in Sharing X Clients on Multiple Displays", *Journal of Internetworking Research & Experience*, Vol. 5, No. 1, pp.1-15, March 1994.

[ABD97] H. Abdel-Wahab, B. Kvande, O. Kim, J.P. Favreau, "An Internet collaborative environment for sharing Java applications," *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, Oct. 1997, pp. 112-117.

[ARC98] F. Arcelli, M. De Santo, A. Chianese, "Client-server architecture for distributed learning environments: a proposal," *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, Vol. 7, Jan. 1998, pp. 395-403.

[BAR97] D.B. Barsky, A.V. Shafarenko, "WWW and Java-based distributed examination system for distance learning applications," *Proceedings of Second Aizu International Symposium on Parallel Algorithms/Architecture Synthesis*, March 1997, pp. 356-363.

[BEG97] J. Begole, C. A. Struble, and C. A. Shaffer, "Leveraging Java Applets: Toward Collaboration Transparency in Java," *IEEE Internet Computing*, Vol. 1, No. 2, March-April 1997, pp.57-64.

[BER98] K. Bergner, A. Rausch, M. Sihling, "Casting an abstract design into the framework of Java RMI," *Proceedings of 1998 International Conference Software Engineering: Education & Practice*, Jan. 1998, pp. 278-285.

[BRO97] M.H. Brown, M.A. Najork, R. Raisamo, "A Java-based implementation of collaborative active textbooks," IEEE Symposium on Visual Languages, Sep. 1997, pp. 372-379.

[BUR91] A. Burger, B. Meyer, C. Jung, and K. Long, "The Virtual Notebook System", *Hypertext '91 Conference Proceedings*, November 1991.

[CHA96] K.M. Chandy, A. Rifkin, J. Mandelson, M. Richardson, W. Tanaka, L. Weisman, "A world-wide distributed system using Java and the Internet," *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing*, Aug. 1996, pp. 11-18.

[CHA97] I-C. Chang, L-L. Chen, J-J. Shen, K-C Hsu, J-H. Huang, "Design and Implementation of a Multimedia WWW-Based Note-Taking System for Distance Learning," *Proceedings of International Conference on Consumer Electronics*, June 1997, pp.10-11.

[CHE97] B-Y. Chen, T-J. Yang, and M. Ouhyoung, "JavaGL—A 3D Graphics Library in Java For Internet Browsers", *IEEE Trans. on Consumer Electronics*, Vol. 43, No. 3, Aug. 1997, pp.271-278.

[CRA92] E. CraigHill, R. Lang, and J.J. Garcia-Luna, "Environments to Enable Informal Collaborative Design Process," *4th Annual National Symposium on Concurrent Engineering*, CALS & CE, Washington ' 92, pp. 47-62.

[CRA97] T. Cramer, R. Friedman, T. Miller, D. Seberger, R. Wilson, M. Wolczko, "Compiling Java just in time," *IEEE Micro*, Vol. 17, No. 3, May-June 1997, pp. 36-43.

[CUT96] M.R. Cutkosky, J. Glicksman, and J.M. Tenenbaum, "MadeFast: Collaborative Engineering Over the Internet" *Communications of the ACM*, Vol. 39, No. 9, 1996, pp. 78-87.

[DIV98] S. Divjak, "Design of hypertext based courseware with the integrated Java and VRML modules," 9th Mediterranean Electrotechnical Conference, Vol. 1, May 1998, pp. 178-181.

[FRA87] K. Franze, O. Neumann, A. Schill, S. Stocker, "An infrastructure for collaborative teleteaching," *Proceedings of Sixth IEEE workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 1997, pp. 341-346.

[FOX96] G. Fox, W. Furmanski, "Towards Web/Java-based high performance distributed computing-an evolving virtual machine," *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing*, Aug. 1996, pp. 308-317.

[GOE98] K.M. Goeschka, J. Falb, W. Radinger, "Database access with HTML and Java-a comparison based on practical experiences," *Proceedings of The*

Twenty-Second Annual International Computer Software and Applications Conference, Aug. 1998, pp. 588-593.

[GOS97] J. Gosling, "The feel of Java," *IEEE Computer*, Vol. 30, No. 6, June 1997, pp. 53-57.

[GUA98] H. Guan, Y. Zhang, "Java-based approaches for accessing databases on the Internet and a JDBC-ODBC implementation," *Computing & Control Engineering Journal*, Vol. 9, No. 2, April 1998, pp. 71-78.

[GUP98] A. Gupta, C. Ferris, Y. Wilson, K. Venkatasubramanian, "Implementing Java computing: Sun on architecture and applications deployment," *IEEE Internet Computing*, Vol. 2, No. 2, March-April 1998, pp. 60-64.

[HAU98] M. Hauswirth, M. Jazayeri, A. Winzer, "A Java-based environment for teaching programming language concepts," *FIE 28th Annual Frontiers in Education Conference*, Nov. 1998, pp. 296-300.

[ITU96-1] ITU-T Recommendation T.120: "Data Protocols for Multimedia Conferencing," http://info.itu.ch/itudoc/itut/rec/t/t120_35511.html, July, 1996.

[ITU96-2] ITU-T Recommendation G.723, "Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbits/s," http://info.itu.ch/itudoc/itu-t/rec/g/g700-799/g723-1_32261.html, 1996.

[JDK98] JDK1.1.7B documentation, URL:http://www.javasoft.com/products/jdk/1.1/docs.html, 1998.

[KOS98] P. Kostur, "Building and managing an intranet," *Proceedings of 1998 IEEE International Professional Communication Conference*, Vol. 2, Sep. 1998, pp. 51-57.

[KOU96] R. T. Kouzse, J. D. Myers, W. A. Wulf, "Collaboratories: Doing Science on the Internet," *IEEE Computer Mag.*, pp.40-46, Aug. 1996.

[KOU97] A. Koutsoumbos, R. Arora, "Enterprise Java," Proceedings of Technology of Object-Oriented Languages and Systems, Nov. 1997, pp. 369-370.

[KUM94] V. Kumar, J. Glicksman, and G.A. Kramer, "A SHAREd Web to Support Design Teams," *Proc. Third Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Morgantown, West Virginia, April 1994, pp. 178-182.

[LAI97] F-P. Lai etc. "The integration of enterprise information—Intranet," *Communications of IICM*, vol. 1, No. 2, April 1997, pp. 43-54.

[LEE97] K-C. Lee, K-N. Chang, S-S. Yu, I-C. Chang, G-W. Shia, W-C. Chen, and J-H. Huang, "Design and Implementation of Important Applications in a Java-Based Multimedia Digital Classroom," *IEEE Transactions on Consumer Electronics*, Vol. 43, No.3, August 1997, pp. 264-270.

[LEW96] T. Lewis, "The big software chill," *IEEE Computer*, Vol. 29, No. 3, March 1996, pp. 12-14.

[LIN96] T. Lindholm and F. Yellin, "The Java Virtual Machine Specification," Addision-Wesley, MA USA, 1996.

[LIT98] M.C. Little, S.M. Wheater, "Building configurable applications in Java," Proceedings of Fourth International Conference on Configurable Distributed Systems, May 1998, pp. 172-179.

[LOO98] C. Loosley and F. Douglas, "High-Performance Client/Server, A Guide to Building and Managing Robust Distibuted Systems," Wiley, 1998.

[MAN94] R. Manohar and A. Prakash, "Replay by Re-execution: A Paradigm for Asynchronous Collaboration via Record and Replay of Interactive Multimedia Sessions," *ACM SIGOIS Bulletin*, Vol. 15, No. 2, pp.32-34, Dec. 1994.

[MEL96] H. Meleis, "Toward the Information Network," *IEEE Computer Mag.*, pp.59-67, Oct. 1996.

[MIN94] W. Minenko, J. Schweitzer, "An Advanced Application Sharing System for Synchronous Collaboration in Heterogeneous Environments," *SIGOIS Bulletin*, Vol. 15, No. 2, Dec. 1994, pp. 40-44.

[MIN95] W. Minenko, "The Application Sharing Technology," *The X Advisor*, Vol. 1, No. 1, June 1995.

[MIN98] D. Min, E. Choi, Y.-T. Han, D. Hwang, J.-H. Jung, "A distributed multimedia conferencing system for distance learning," *IEEE International Workshop on Multimedia Software Engineering*, April 1998, pp. 88-95.

[MIU98] M. Miura, J. Tanaka, "A framework for event-driven demonstration based on Java toolkit," *Proceedings of 3rd Asia Pacific Computer Human Interaction*, July 1998, pp. 331-336.

[NCN97] See http://admwww.cc.ncnu.edu.tw/index.html.

[RAM97] P. Ram, R. Abarbanel, "Enterprise Computing: The Java Factor," *IEEE Computer*, June 1997, pp. 115-117.

[ROS98] D. Rosenberg, "Bringing Java to the enterprise: Oracle on its Java server strategy," *IEEE Internet Computing*, vol. 2, no. 2, March-April 1998, pp52-59.

[SEL98] J. Sellentin, B. Mitschang, "Data-intensive intra- and Internet applications-experiences using Java and CORBA in the World Wide Web," *14th International Conference on Data Engineering*, Feb. 1998, pp. 302-311.

[SHI96] C-W. Shiah, J-C. Cheng, and W-C. Chen, "A Practical Point-to-point Filter-based Shared Window System," In *Conference of Eastern-Western Human Computer Interface* 96, Mosco, 1996.

[SHI98] S. Shirmohammadi, J.C. De Oliveira, N.D. Georganas, "Applet-based telecollaboration: a network-centric approach", *IEEE Multimedia*, vol. 5, no. 2, April-June 1998, pp. 64-73.

[SME97] C. Smeaton, I. Neilson, "Adapting the infrastructure provided by the World Wide Web for educational purposes," *Proceedings of the 23rd EUROMICRO conference*, Sep. 1997, pp. 72-77.

[TOY94] G. Toye, M.R. Cutkosky, L.J. Leifer, J.M. Tenenbaum, and J. Glicksman, "SHARE: A Methodology and Environment for Collaborative Product Development," *The International Journal of Intelligent and Cooperative Information Systems*, vol. 3, no. 2, June 1994, pp. 129-53.

[VAN97] A. Van Hoff, "The case for Java as a programming language," *IEEE Internet Computing*, Vol. 1, No. 1, Jan.-Feb. 1997, pp. 51-56.

[WOL97] A. Wollrath, J. Waldo, R. Riggs, "Java-Centric Distributed Computing," *IEEE Micro*, May/June 1997, pp. 44-53.

[XAV98] A. Xavier, A. Spanias, "An Adaptive System Identification Java Simulation for Internet based Courseware," FIE 28th Annual Frontiers in Education Conference, Nov. 1998, pp. 348-353.

[YU97] S-S. Yu, C-W. Shia, and W-C. Chen, "Application Sharing Frameworks on Java," *Proceedings of IEEE International Conference on Consumer Electronics*, June 1997, pp. 14-15.

[YU98] S-S. Yu, and W-C. Chen, "A Java Based Multi-Tier Architecture for Enterprise Computing: A Case Study from a University Academic Information System," *Proceedings of IEEE International Conference on Consumer Electronics*, June 1998, pp. 252-253.

[ZHA98] Z. Zhang, A. Karmouch, “Multimedia courseware delivery over the Internet,” IEEE Canadian Conference on Electrical and Computer Engineering, Vol. 2, May 1998, pp. 609-612.

Appendix A Functionality of Academic Information System

A.1 Server Side Interface Definitions

We defined four interfaces for objects on the RMI server:

- NCNUCourse for course queries;
- NCNU for authentication and global information;
- NCNUStudent for student related services;
- NCNUTeacher for teacher related services;

Their detailed definitions are listed below:

```
public interface NCNUCourse extends java.rmi.Remote {
   /**
    * Get current academic year.
    * @return Current academic year such as "871"
    */
   String getAcaYear() throws RemoteException;
   /**
    * Get courses satisfy query conditions.
    * @param year Academic year such as "871"
    * @param deptid Department short name, such as "資管系"
    * @param coreType Core course type, "0" for non core course, "1" to
"4" for legal core course type
    * @param name Course name. Any course name contains the string will
be selected
    * @param teacher Teacher name. Any Teacher's name contains the string
will be selected
    * @param grade Grade of the course designed for
    * @param time Courses hold in the time period. "5abcd" means Friday
morning
    * @param place Classroom where courses may take place.
    * @return array of (系所名,課號,課程名稱,班別,選別,學分數,時間,地點,教師,
年級,人數,上限,課綱,學年期)
    */
   String[][] queryCourse(String year, String deptid, String coreType,
String name, String teacher, String grade, String time, String place) throws
RemoteException;
   /**
    * Get syllabus of a course.
    * @param courseid Course ID
    * @param year Semester of the opened course
    * @param classid Class number of the opened course. "0" if only one
class is hold in that semester
    * @return Syllabus of the opened class, null if no such class
    */
   String querySyllabus(String courseid, String year, String classid)
throws RemoteException;
```

```
    /**
     * Query departments available in NCNU now
     * @return Array of department short name
     */
    String[] queryDepartment() throws RemoteException;
    /**
     * Send previous query result to an email account
     * @param email The email account where query result should be sent
     * @return "ok" if success, error message if fail
     */
    String sendQueryResult(String email) throws RemoteException;
    /**
     * Query rule about how to graduate from a department
     * @param deptName Short name of the queried department
    * @return Rule about how to graduate from the queried department, null
if no such department
     */
    String queryRule(String deptName) throws RemoteException;
    /**
     * Query if we are in prepare course selection period
     * @return true if in prepare period, false if not
     */
    boolean inPrepare() throws RemoteException;
}
/**
 * RMI Interface definition for Authentication server
 */
public interface NCNU extends NCNUCourse, java.rmi.Remote {
    /**
     * Request a student session.
     * @param studentID Student ID for NCNU
     * @param password Password
     * @return Remote NCNUStudent object for the session, null if fail
     */
    NCNUStudent loginStudent(String studentID, String password) throws
RemoteException;
    /**
     * Request a teacher session.
     * @param teacherID Teacher ID for NCNU
     * @param password Password
     * @return Remote NCNUTeacher object for the session, null if fail
     */
    NCNUTeacher loginTeacher(String teacherID, String password) throws
RemoteException;
    /**
     * Get error message for previous operation.
     * @return error message for previous operation
     */
    String getErrorMessage() throws RemoteException;
}
public interface NCNUStudent extends NCNUCourse, java.rmi.Remote {
    /**
     * Get welcome message for the student
     */
    String sayHello() throws RemoteException;
    /**
     * Change password for the student
     * @param oldPassword Old password
```

```
     * @param newPassword New password
     * @return "ok" if success, error message if fail
     */
   String changePassword(String oldPassword, String newPassword) throws
RemoteException;
    /**
     * Change personal information
     * @param attr (戶籍地址，通訊地址，電子郵件信箱，電話號碼，郵局帳號,郵遞區
號,監護人姓名)
     * @return 錯誤訊息，若成功則傳回"ok"
     */
    String changePerson(String[] attr) throws RemoteException;
    /**
     * Get personal information
     * @return (戶籍地址，通訊地址，電子郵件信箱，電話號碼，郵局帳號,郵遞區號,
監護人姓名)
     */
    String[] getPerson() throws RemoteException;
    /**
     * Get error message of previous request
     */
    String getErrorMessage() throws RemoteException;
    /**
     * Query scores of a semester
     * @param acaYear Semester
     * @return array of (課程名稱，學分數，成績,課程年級) appened with 修習
學分數,實得學分數,學期平均成績
     */
    String[][] queryScore(String acaYear) throws RemoteException;
    /**
     * Query courses which the student can but not yet select for current
semester
    * @return array of (課號，系所名，學分數，課名，時間，地點，核心類別，班別，
年級，必選修別，教師)
     */
    String[][] queryUnselectedCourse() throws RemoteException;
    /**
     * Query courses which has been selected for current semester
    * @return array of (課號，系所名，學分數，課名，時間，地點，核心類別，班別，
年級，必選修別，教師)
     */
    String[][] querySelectedCourse() throws RemoteException;
    /**
     * Add a course for this semester
     * @param cid Course ID
     * @param c Class ID
     * @return "ok" if success, error message if anything wrong
     */
    String addCourse(String cid, String c) throws RemoteException;
    /**
     * Drop a course
     * @param cid Course ID
     * @param c Class ID
     * @param credit credits of the dropped course
     * @param coretype coretype
     * @return "ok" if success, error message if anything wrong
```

```
     */
    String dropCourse(String cid, String c, int credit, int coretype) throws
RemoteException;
    /**
     * Check if enough credits has been selected by this student
     * @return null if credit is enough, warning message if needs more credit
     */
    String warnCredit() throws RemoteException;
    /**
     * Get hint about how many credits are needed to graduate
     */
    String getCreditHint() throws RemoteException;
    /**
     * Get hints about how many core course credits are neede to graduate
     */
    String[] getCoreHint() throws RemoteException;
    /**
     * Get note about the rules of course selection of this semester
     */
    String getNote() throws RemoteException;
    /**
     * Get name of the student
     */
    String getName() throws RemoteException;
    /**
     * Get grade of the student
     */
    String getGrade() throws RemoteException;
    /**
     * Check if we can selection now
     * @return true if can select course, false otherwise
     */
    boolean canSelect() throws RemoteException;
    /**
     * Send previous query score result to an email account
     * @param email Email account where score should be sent
     * @return "ok" if success, error message if fail
     */
    String sendQueryScoreResult(String email) throws RemoteException;
    /**
     * Query courses which has been selected for current semester
     * @return array of (課名，學分數，地點，教師，時間)
     */
    String[][] queryTableCourse() throws RemoteException;
}
public interface NCNUTeacher extends NCNUCourse,java.rmi.Remote {
    /**
     * Welcome message for the teacher session
     */
    String sayHello() throws RemoteException;
    /**
     * @return "ok" if success, error message if fail
     */
    String changePassword(String oldPassword, String newPassword) throws
RemoteException;
    /**
     * Get personal information of the teacher
```

```
     * @return (戶籍地址，通訊地址，電子郵件帳號，電話號碼，郵局帳號)
     */
    String[] getPerson() throws RemoteException;
    /**
     * Change personal information
     * @param array of (戶籍地址，通訊地址，電子郵件帳號，電話號碼，郵局帳號)
     */
    String changePerson(String[] attr) throws RemoteException;
    /**
     * Get error message of previous operation
     */
    String getErrorMessage() throws RemoteException;
    /**
     * Get courses opened by the teacher in this semester
     * @return array of (系所名,課號，學分數,中文課名，時間,地點,年級,班別，學
年期)
     */
    String[][] getOpenCourse() throws RemoteException;
    /**
     * Get information about students who have select the course in this
semester
     * @param courseid Course ID
     * @param classid Class ID
     * @param acaYear semester
     * @return array of (學號,姓名,成績,所屬系所)
     */
    String[][] getSelected(String courseid, String classid, String acaYear)
throws RemoteException;
    /**
     * Set scores for an opened course in this semester
     * @param course Course ID
     * @param class Class ID
     * @param score array of (學號，成績)
     * @return "ok" if all success, error message if fail
     */
    String setScore(String courseid, String classid, String[] scores)
throws RemoteException;
    /**
     * Get syllabus of a course in this semester
     * @param courseid Course ID
     * @param classid Class ID
     * @param acaYear semester
     * @return null if fail, syllabus if succeed
     */
    String getSyllabus(String courseid, String classid, String acaYear)
throws RemoteException;
    /**
     * Set syllabus for a course
     * @param courseid Course ID
     * @param class Class ID
     * @param acaYear semester
     * @param syllabus Syllabus of the course
     * @return "ok" if succeed, error message if fail
     */
    String setSyllabus(String courseid, String classid, String acaYear,
String syllabus) throws RemoteException;
    /**
```

```
   * Send mail to all students who has selected the course in this semester
     * @param courseid Course ID
     * @param classid Class ID
     * @param acaYear semester
     * @param returnAddress Email address of the teacher
     * @param subject Subject of the message
     * @param content Content to be sent to all student
     * @return "ok" if succeed, error message if fail
     */
    String sendMail(String courseid, String classid, String acaYear, String
returnAddress, String subject, String content) throws RemoteException;
    /**
     * Query students of a department according to their name,student id
and grade
     * @param name Name of a student
     * @param stuID Student ID
     * @param grade Grade of student
     * @return array of (studentid, name, grade)
     */
    String[][] queryStudent(String name, String stuID, String grade) throws
RemoteException;
    /**
     * Query historical score information about a student
     * @param stuID Student ID
     * @return A string which describes the historical information about
the studentid
     */
    String queryStudentCourse(String stuID) throws RemoteException;
    /**
     * Check if teacher can input score now
     * @return true if can input, false if can't
     */
    boolean canScore() throws RemoteException;
    /**
     * Query student's rank
     * @param dept Department's name
     * @param grade Student grade
     * @param semester
     * @return String of rank information
     */
    String queryRank(String dept, String grade, String semester) throws
RemoteException;
}
```

A.2 Applet Functions

Roles of users in our system are defined as student, teacher and the public. Their accessible functions are composed with the following features:

- Query course

Query conditions can be any AND combination of department name, core course type, course grade, classroom, course name, teacher name, semester and course time. Course information and syllabus are presented in the same page to ease usage. Query result can be mailed back to an Internet mail account.

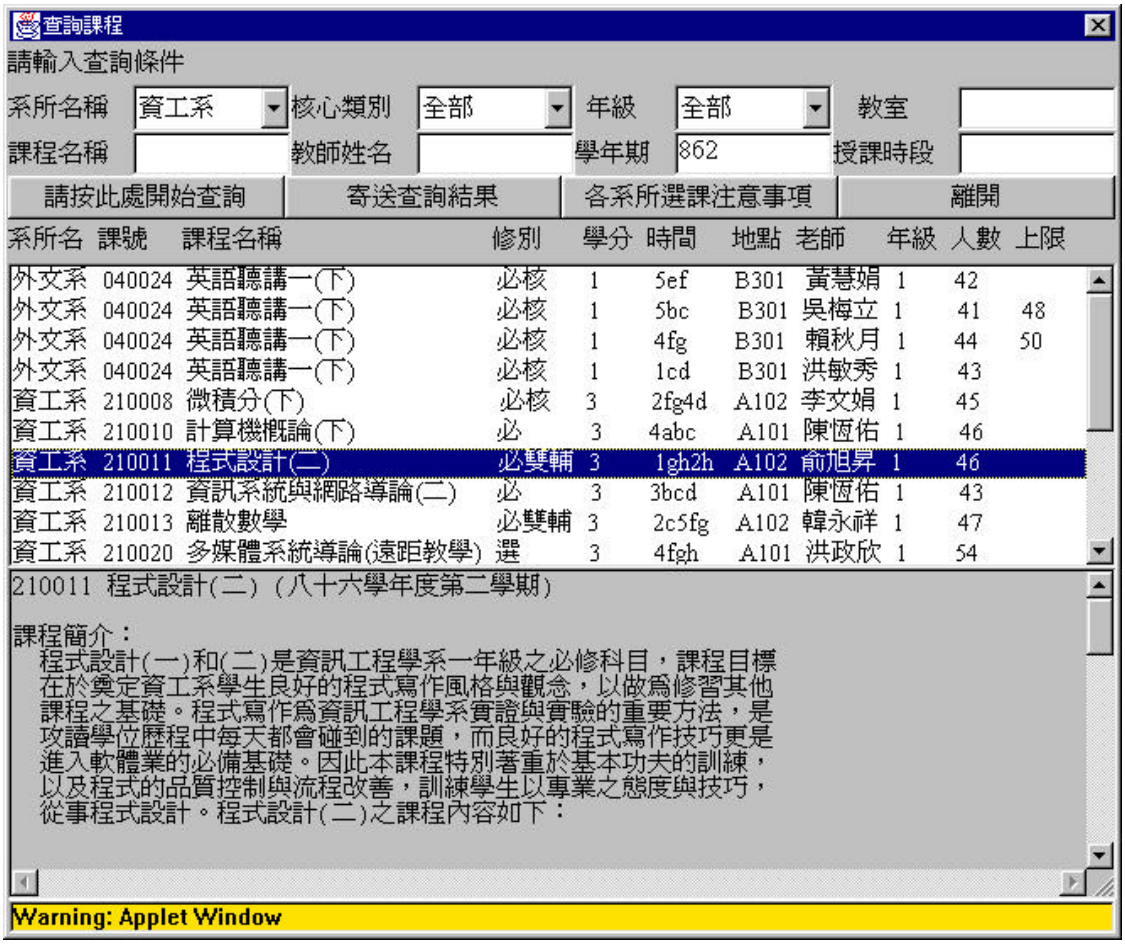


Figure A-1. Query course.

- Query score

Query scores of courses in a semester, and can mail query result to an Internet mail account. Total credits, got credits, and average score are listed below courses. Any fail scores will be marked as red numbers.

Figure A-2. Query score

- Course selection

The system groups courses in current semester by mandatory, optional, core and selected courses. The courses that have been selected or can‘t be selected by a student won’t be prompted. Other functions, such as selection suggestion, conflict notification, mandatory course auto selection, capacity limitation and selection priority, are also included.

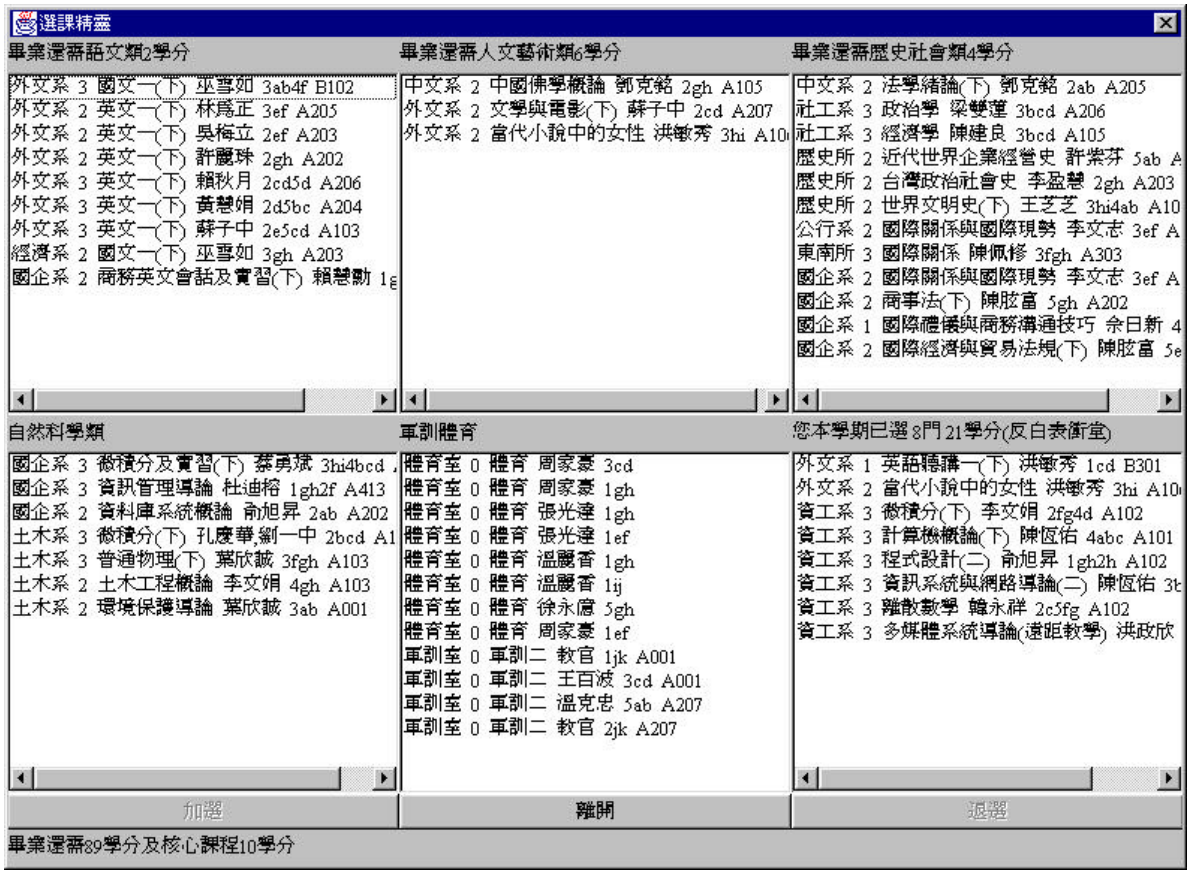


Figure A-3. Course selection.

- School timetable

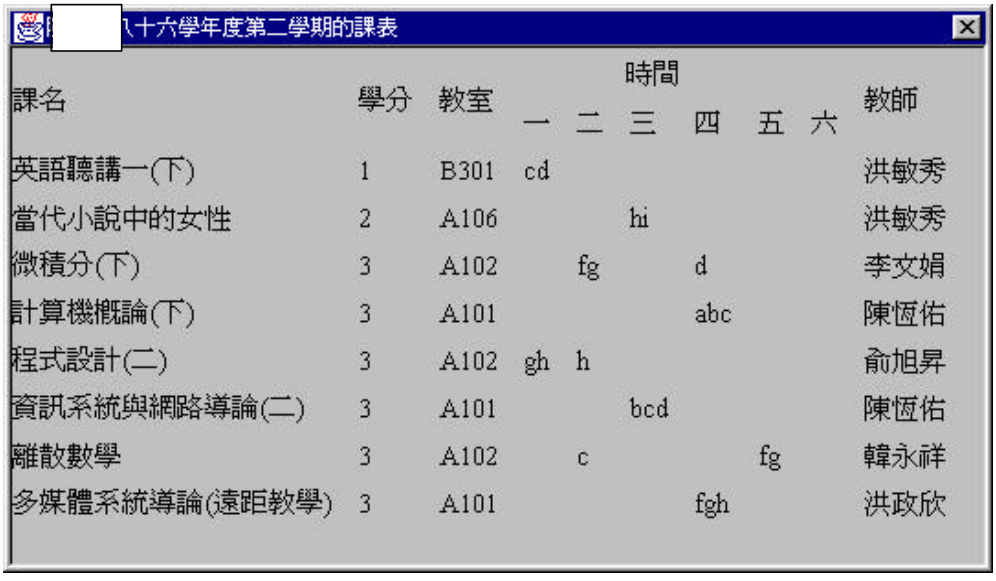


Figure A-4. School timetable

- Student personal information management.

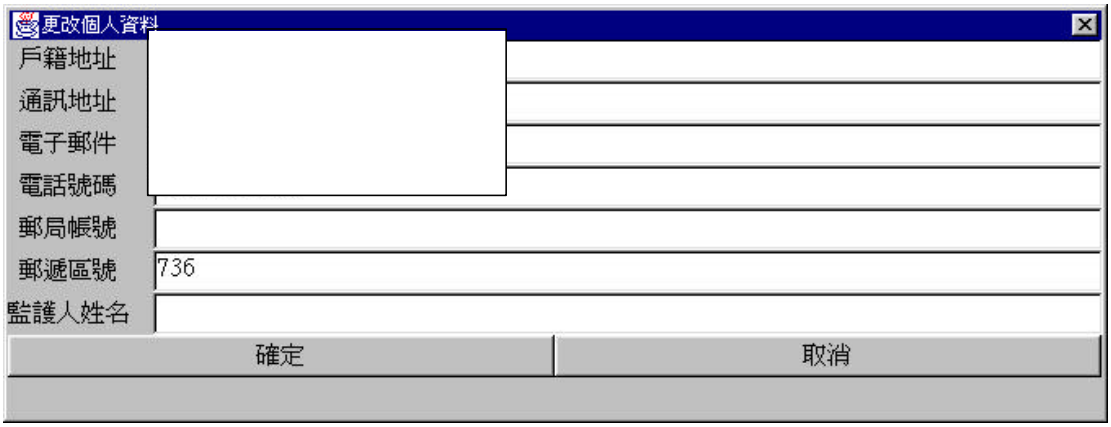


Figure A-5. Manage student personal information

- Change password

Figure A-6. Change password

- Course management

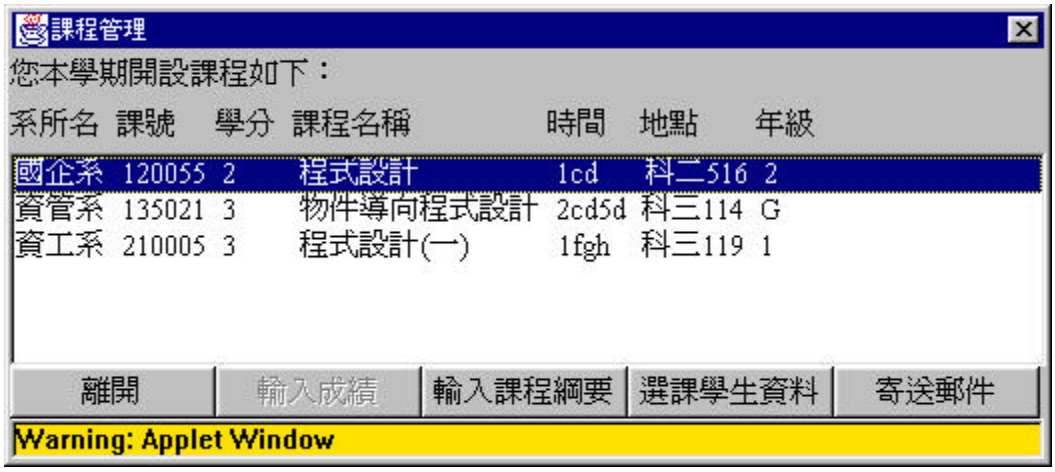


Figure A-7. Course Management

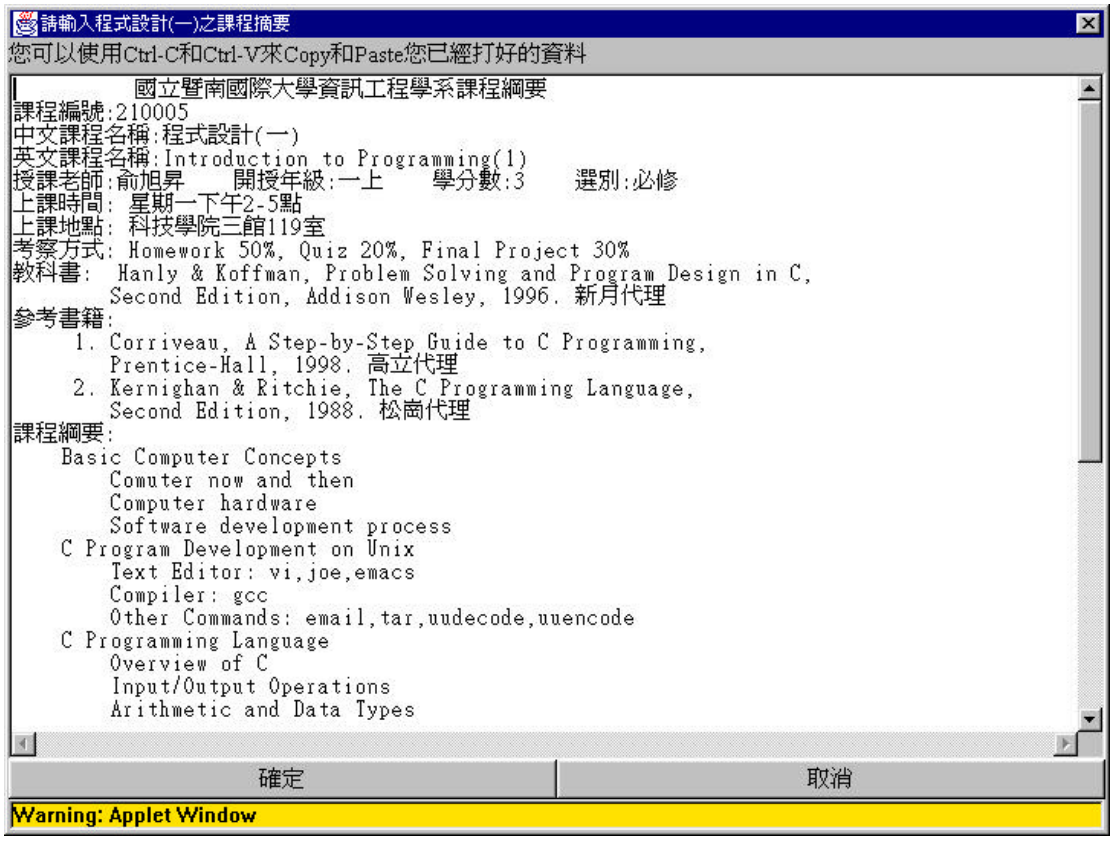


Figure A-8. Input syllabus.

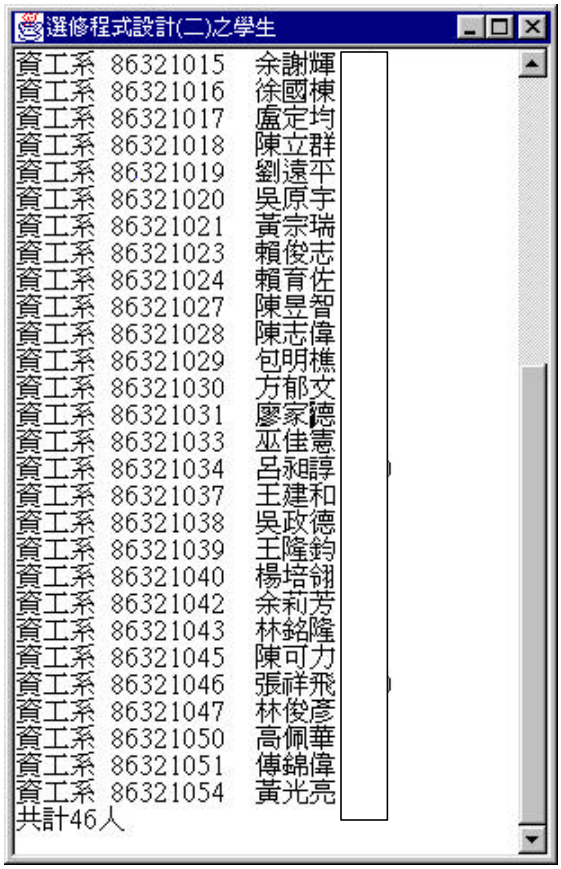


Figure A-9. Query information about students in a course

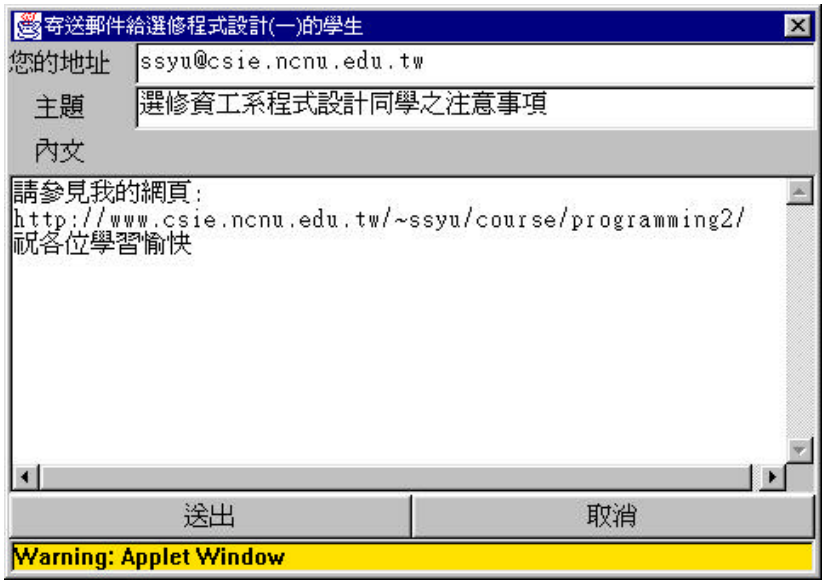


Figure A-10. Mail to students in a course

- Query study status of students in the same department.

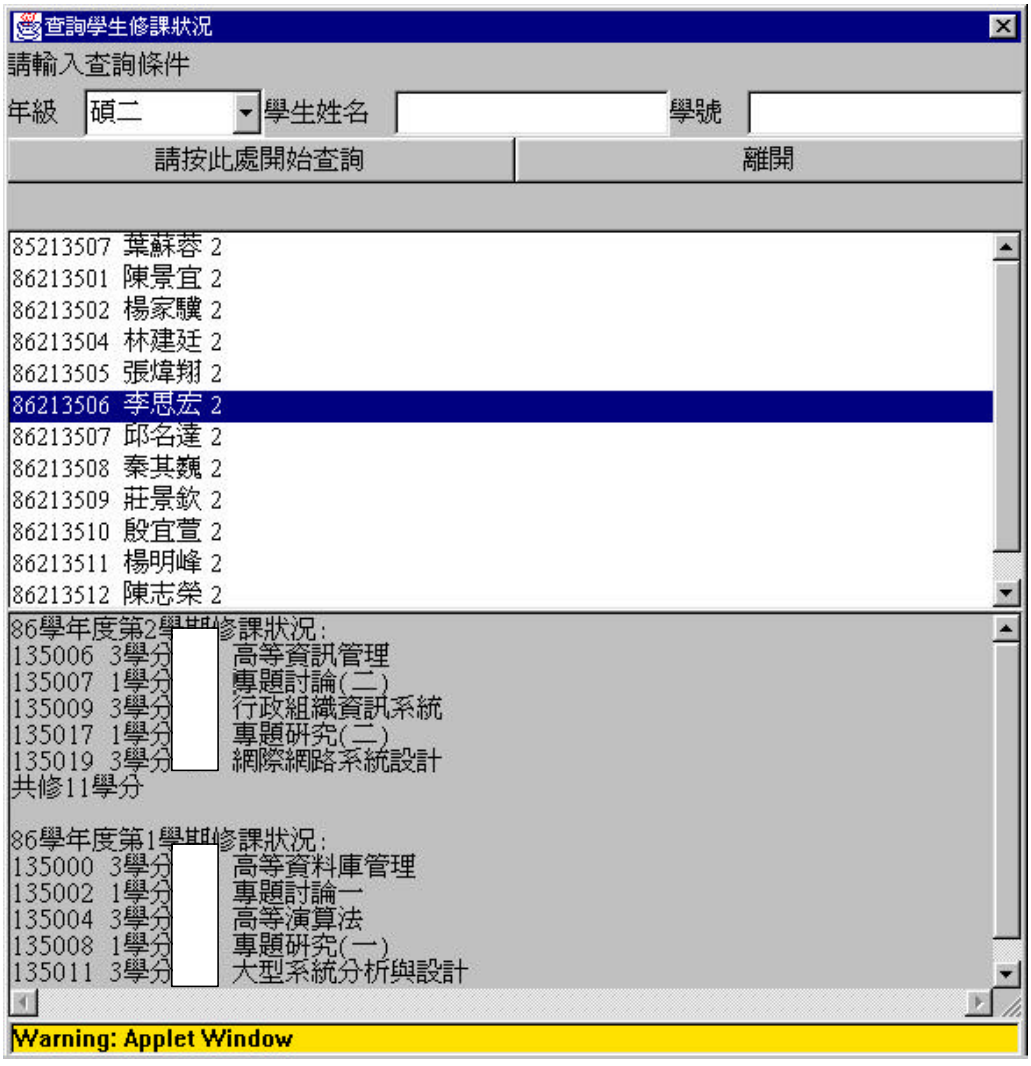


Figure A-11. Query study status of students in the same department.

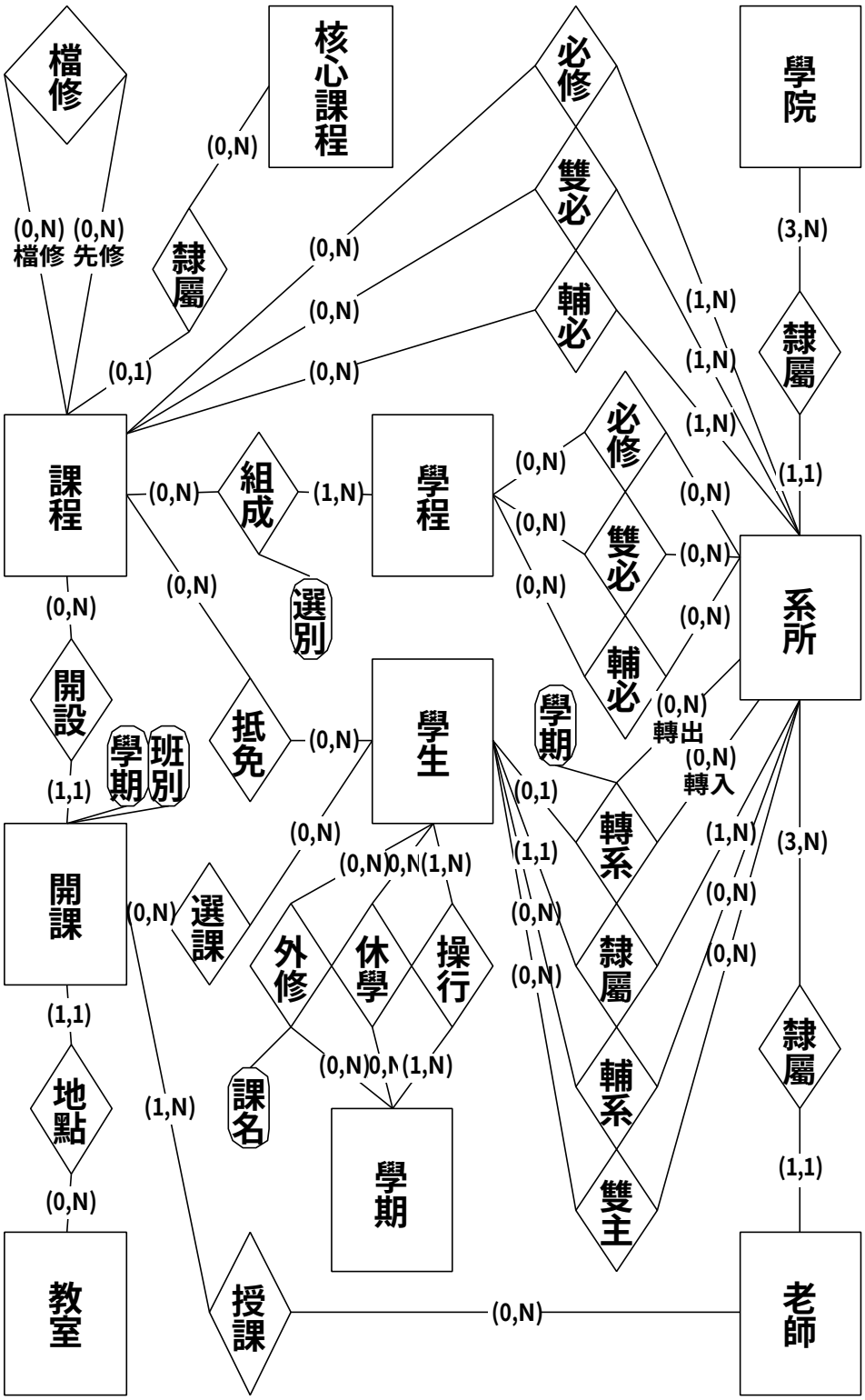
- Query ranks of students in a semester.

Figure A-12. Query ranks of students in a semester.

Figure A-13. Query graduate rule of a department.

Appendix B Data Analysis of Academic Information System

B.1 E-R Diagram



B.2 Relational Database Schema

The relational database schema of an academic information listed below follows the syntax of Microsoft SQL Server.

```
create table college /* 學院資料 */
(
  colid char(1) not null, /* 學院代碼 */
  name varchar(10) not null, /* 學院名稱 */
  primary key(colid)
)
go
create table department /* 系所資料 */
(
  deptid char(2) not null, /* 系所代碼 */
  colid char(1) not null, /* 所屬學院代碼 */
  degree varchar(20) not null, /* 學士學位名稱 */
  gdegree varchar(20) not null, /* 博碩士學位名稱 */
  cname varchar(40) not null, /* 系所中文名 */
  ename varchar(40) not null, /* 系所英文名 */
  sname varchar(10) not null, /* 中文簡稱 */
  mincredit numeric(3,0) not null, /* 大學部最低畢業學分 */
  gmincredit numeric(2,0) not null, /* 研究所最低畢業學分 */
  pmincredit numeric(2,0) not null, /* 博士班最低畢業學分 */
  dmincredit numeric(2,0) not null, /* 直攻生最低畢業學分 */
  note text null, /* 選課注意事項 */
  primary key (deptid),
  foreign key(colid) references college
)
create index dept_id ON department(deptid)
go
create table student /* 學生學籍資料 */
(
  studentid char(9) not null, /* 學號 */
  deptid char(2) not null, /* 所屬系所代碼 */
  name varchar(10) not null, /* 姓名 */
  birthday char(7) null, /* 生日 */
  sex char(1) not null, /* 性別 */
  grade char(1) not null, /* 年級 */
  type varchar(30) not null default '一般生', /* 身分別 */
  status char(1) not null default 0, /* 學生狀態 */
  leavedate char(7) null, /* 離校日期 */
  email varchar(40) null, /* 電子郵件信箱 */
  id char(10) null, /* 身分證字號 */
  tel varchar(14) null, /* 聯絡電話 */
  citizen varchar(12) null, /* 國籍 */
  addr1 varchar(80) null, /* 戶籍地址 */
  addr2 varchar(80) null, /* 通訊地址 */
  thesis varchar(120) null, /* 論文題目 */
  tscore numeric(5,1) null, /* 學位考試成績 */
```

```
  diplomaid char(4) null, /* 畢業證書號碼 */
  entrydoc varchar(40) null, /* 入學文號 */
  entrydiploma varchar(40) null, /* 入學學歷 */
  password varchar(20) null, /* 密碼 */
  entrydate char(5) not null, /* 入學年月 */
  ograduate char(1) not null default '1', /* 原學校畢肄業,1畢,0肄 */
  oentrydoc varchar(40) null, /* 原學校入學資格核准年月文號 */
  ograde char(1) null, /* 原學校肄業年級 */
  ograduatedoc varchar(40) null, /* 原學校畢業證書號碼 */
  equaldoc varchar(40) null, /* 比敘大專同等學力年月文號 */
  pracscore numeric(3,0) null, /* 實習成績 */
  leavereason varchar(30) null, /* 離校原因 */
  deptgroup varchar(20) null, /* 系所組別 */
  zipcode varchar(5) null, /* 郵遞區號 */
  pname varchar(20) null, /* 監護人姓名 */
  dormnum varchar(8) null, /* 宿舍房間 */
  dormfee numeric(6,0) not null default 0, /* 宿舍費用 */
  dormdeposit char(1) not null default 'N', /* 宿舍保證金 */
  account char(14) null, /* 郵局帳號 */
  age numeric(3,0) null, /* 年齡 */
  alive char(1) null default 'Y', /* 存歿 */
  tuition numeric(6,0) null default 0, /* 學費 */
  incidental numeric(6,0) null default 0, /* 雜費 */
  basic numeric(6,0) null default 0, /* 學雜費基數 */
  insurance numeric(5,0) null default 0, /* 平安保險費 */
  overseas numeric(5,0) null default 0, /* 僑保費 */
  ddfee numeric(5,0) null default 0, /* 住宿保證金 */
  creditfee numeric(6,0) null default 0, /* 學分費 */
  langfee numeric(5,0) null default 0, /* 語言實習費 */
  compfee numeric(5,0) null default 0, /* 電腦實習費 */
  primary key (studentid),
  foreign key(deptid) references department
)
create index student_id ON student(studentid)
go
create table teacher /* 教師基本資料 */
(
  teacherid varchar(10) not null, /* 身分證字號 */
  deptid char(2) not null, /* 所屬系所代碼 */
  name varchar(10) not null, /* 姓名 */
  sex char(1) not null, /* 性別 */
  birthday char(7) null, /* 生日 */
  isfulltime char(1) not null default '1', /* 是否專任 */
  position varchar(8) not null, /* 職稱 */
  parttime varchar(40) null, /* 兼職職務 */
  overtime numeric(2,0) default 0, /* 兼職核減時數 */
  hiredate char(7) not null, /* 起聘日期 */
  status char(1) not null default 0, /* 狀態 */
  password varchar(20) null, /* 密碼 */
  email varchar(40) null, /* email */
  account char(14) null, /* 郵局帳號 */
```

```
  addr1 varchar(80) null, /* 戶籍地址 */
  addr2 varchar(80) null, /* 通訊地址 */
  phone varchar(14) null, /* 電話 */
  primary key (teacherid),
  foreign key(deptid) references department
)
create index teacher_id ON teacher(teacherid)
go
create table course /* 課程基本資料 */
(
  courseid char(6) not null, /* 課號 */
  cname varchar(50) not null, /* 中文名稱 */
  ename varchar(80) null, /* 英文名稱 */
  grade char(2) null, /* 修習年級 */
  credit numeric(1,0) not null, /* 學分數 */
  coretype char(1) not null default 0, /* 核心類別 */
  practice char(1) not null default 0, /* 是否為實習課 */
  type varchar(8) null, /* 必選雙輔, query cache */
  primary key (courseid)
)
create index course_id ON course(courseid)
go
create table precourse /* 先修學分要求 */
(
  courseid char(6) not null, /* 課號 */
  precourseid char(6) not null, /* 先修課號 */
  start numeric(3,0) not null, /* 啟用年份 */
  stop numeric(3,0) not null default 999, /* 失效年份 */
  primary key (courseid,precourseid),
  foreign key (courseid) references course,
  foreign key (precourseid) references course
)
create index precourse_id on precourse(courseid)
go
create table require /* 必修學分 */
(
  courseid char(6) not null, /* 課號 */
  deptid char(2) not null, /* 要求必修系所代碼 */
  start numeric(3,0) not null, /* 啟用年份 */
  stop numeric(3,0) not null default 999, /* 失效年份 */
  primary key (courseid,deptid,start,stop),
  foreign key (courseid) references course,
  foreign key (deptid) references department
)
create index require_id on require(courseid)
create index require_deptid on require(deptid)
go
create table assist /* 輔系必修學分 */
(
  courseid char(6) not null, /* 課號 */
  deptid char(2) not null, /* 輔系系所代碼 */
  start numeric(3,0) not null, /* 啟用年份 */
  stop numeric(3,0) not null default 999, /* 失效年份 */
  primary key (courseid,deptid,start,stop),
```

```
  foreign key (courseid) references course,
  foreign key (deptid) references department
)
create index assist_id on assist(courseid)
create index assist_deptid on assist(deptid)
go
create table dmajor /* 雙主修必修學分 */
(
  courseid char(6) not null, /* 課號 */
  deptid char(2) not null, /* 雙主修系所代碼 */
  start numeric(3,0) not null, /* 啟用年份 */
  stop numeric(3,0) not null default 999, /* 失效年份 */
  primary key (courseid,deptid,start,stop),
  foreign key (courseid) references course,
  foreign key (deptid) references department
)
create index dmajor_id on dmajor(courseid)
create index dmajor_deptid on dmajor(deptid)
go
create table classroom /* 教室資料 */
(
  classroomid varchar(8) not null, /* 教室代碼 */
  capacity numeric(3,0) not null, /* 容量 */
  equipment char(1) default 0, /* 設備 */
  deptid char(2) null, /* 優先分配系所 */
  primary key (classroomid)
)
create index classroom_id on classroom(classroomid)
go
create table conduct /* 操行與學期成績 */
(
  studentid char(9) not null, /* 學號 */
  year char(4) not null, /* 學年期 */
  score numeric(3,0) not null default -1, /* 操行成績 */
  average numeric(6,2) null, /* 學期平均成績 */
  rank varchar(7) null, /* 學期排名 */
  deptid char(2) null, /* 當學期所屬系所 */
  grade char(1) null, /* 當學期之年級 */
  failpass numeric(2,0) null, /* 當學期為過學分數 */
  primary key (studentid,year),
  foreign key(studentid) references student
)
create index conduct_sid on conduct(studentid)
go
create table coremin /* 核心課程最低學分數 */
(
  coretype char(1) not null, /* 核心類別 */
  mincredit numeric(2,0) not null, /* 最低學分數 */
  primary key (coretype)
)
go
create table opened /* 開設課程資料 */
(
  courseid char(6) not null, /* 課號 */
```

```
  year char(4) not null, /* 學年期 */
  class char(1) not null default 0, /* 班別 */
  requirement char(1) default 0, /* 設備需求 */
  time varchar(8) null, /* 上課時間 */
  place varchar(16) null, /* 上課地點 */
  evaluation char(1) null, /* 評鑑結果 */
  hours numeric(1,0) null, /* 授課時數 */
  syllabus text null, /* 課程綱要 */
  limit numeric(3,0) not null default 0, /* 人數限制 */
  note varchar(40) null, /* 註記 */
  teachers varchar(42) null default '', /* 授課老師, query cache */
  students numeric(3,0) not null default 0, /* 修課人數, query cache */
  primary key (courseid,year,class),
  foreign key(courseid) references course
)
create index opened_all on opened(courseid,year,class)
create index opened_year on opened(year)
go
create table outcredit /* 外修學分資料 */
(
  studentid char(9) not null, /* 學號 */
  equals char(6) null,     /* 抵免課號 */
  credit numeric(1,0) null,     /* 抵免學分數 */
  school varchar(30) not null, /* 修習學校 */
  year char(4) not null, /* 修習學年期 */
  score numeric(3,0) not null default -1, /* 成績 */
  cname varchar(40) not null,     /* 中文課名 */
  ename varchar(80) null,     /* 英文課名 */
  primary key(studentid,year,cname),
  foreign key(studentid) references student
)
create index student_id ON outcredit(studentid)
go
create table rest /* 休學資料 */
(
  studentid char(9) not null, /* 學號 */
  year char(4) not null, /* 開始休學學年期 */
  period numeric(1,0) null, /* 修學期間(學期) */
  restdoc varchar(20) null, /* 休學文號 */
  primary key (studentid,year),
  foreign key (studentid) references student
)
go
create table selected /* 選課資料 */
(
  courseid char(6) not null, /* 課號 */
  year char(4) not null, /* 學年期 */
  class char(1) not null default 0, /* 班別 */
  studentid char(9) not null, /* 學號 */
  score numeric(3,0) null default -1, /* 成績 */
  mandatory char(1) not null default 'N', /* 是否必選 */
  primary key (studentid,courseid,year),
  foreign key (courseid,year,class) references opened,
```

```
  foreign key (studentid) references student
)
create index selected_who on selected(courseid,year,class)
create index selected_cid on selected(courseid,studentid)
create index selected_sid ON selected(studentid,year,class)
go
create table teaching /* 授課資料 */
(
  courseid char(6) not null, /* 課號 */
  year char(4) not null, /* 學年期 */
  class char(1) not null default 0, /* 班別 */
  teacherid varchar(10) not null, /* 教師身分證字號 */
  primary key (teacherid,courseid,year,class),
  foreign key(courseid,year,class) references opened,
  foreign key(teacherid) references teacher
)
create index teaching_tid on teaching(teacherid)
create index teaching_cid on teaching(courseid,year,class)
go
create table exempt /* 抵免學分資料 */
(
  studentid char(9) not null, /* 學號 */
  courseid char(6) not null, /* 課號 */
  primary key (studentid,courseid),
  foreign key(studentid) references student,
  foreign key(courseid) references course
)
create index exempt_sid on exempt(studentid)
go
create table transfer /* 轉系資料 */
(
  studentid char(9) not null, /* 學號 */
  year char(4) not null, /* 轉系學年期 */
  fromdep char(2) not null, /* 原系所 */
  todep char(2) not null, /* 轉至系所 */
  beforegrade char(1) not null, /* 轉系前年級 */
  aftergrade char(1) not null, /* 轉系後年級 */
  primary key (studentid,year),
  foreign key(studentid) references student
)
go
create table whoassist /* 輔系學生 */
(
  studentid char(9) not null, /* 學號 */
  deptid char(2) not null, /* 系所代碼 */
  primary key (studentid,deptid),
  foreign key (studentid) references student,
  foreign key (deptid) references department
)
go
create table whodouble /* 雙主修學生 */
(
  studentid char(9) not null, /* 學號 */
  deptid char(2) not null, /* 系所代碼 */
  primary key (studentid,deptid),
```

```
  foreign key(studentid) references student,
  foreign key(deptid) references department
)
go
```