GRADER - DESIGN AND IMPLEMENTATION OF A COMPONENT-BASED DISTRIBUTED SYSTEM

Daofeng Sun

A Major Report

In

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Science at
Concordia University
Montreal, Quebec, Canada

October 2000

0-612-59341-X

ABSTRACT

GRADER - Design and Implementation of a Component Based Distributed System

Daofeng Sun

This project focuses on the design and implementation of student grade management system using component-based middle-tier distributed applications. Object-oriented design and component technology paradigms are used to analyze and design this GRADER application. The implementation is based on the Java language Enterprise Java Bean (EJB) and J2EE application server. Three different distributed architectures: DNA, CORBA and J2EE, are introduced and compared to give an overview of the current distributing computing technologies. By partitioning three or more tiers to a Web application, each tier can be logically and may physically separated from each other to present better design patterns and improve overall Web application reliability and scalability. All application services are also distributed in each tier, including clients on the front end, data resources on the back-end, and one or more middle tiers between them where business logic of the application should stay. The middle tier implements as EJB component, and hence has good reusability and hides the client tier from the complexity of the enterprise, and takes advantage of rapidly maturing Internet technologies to minimize user administration and good portability. Transparent access allows clients to interact with remote servers regardless of physical component location. The underlying infrastructure takes care of locating any server resource. Java security mechanism is also discussed in this project.

Acknowledgments

I deeply appreciate my supervisor Professor ***Tao Linxin*** for the benefits I took from his courses and initiative of this interesting project. I am grateful to him for his guidance, valuable suggestions, encouragement and time he spent in my major project.

I really appreciate Dr. ***Mustillo Pardo*** for his many efforts to examine the report and gave me many valuable suggestions. Many thanks are to the system administrators at Computer Science Department especially ***Stan Swiercz*** who has provided superior system support. I also really appreciate graduate secretary ***Halina Monkiewicz*** who was always friendly and gave me many help.

Special thanks go to my wife for her constant support throughout my Master program study.

TABLE OF CONTENT

List of Figures

1 Introduction

1.1 Multi-tier programming model

Application programming models fall into two categories. The *classic model* entails the creation of an application as a single stand-alone entity, whereas the *component model* [1] allows the creation of an application as a set of reusable components. In the classic model, the same programmer typically authors the application framework and the business logic. The component model is quite different. A programmer writes code to implement specific business functions, while a standard container supplied by someone else handles the rest. Requests from outside the application are directed to the container, which is responsible for executing the correct code within the objects implementation. The container and the object communicate through some protocol. The current hot Web-based applications require a more scalable, flexible, and reliable software infrastructure. This has resulted in a necessity for the integration of back-end systems and services with the Web. Component features act as natural glue for integrating business objects in the middle tier of distributed applications. Distributed applications are typically implemented in multiple tiers in which many business components cooperate to form so called middleware. Figure 1 illustrates this.

The three-tier architecture has become quite pervasive, especially for Web-based applications [8]. The first tier represents the presentation and GUI layer, such as a web browser in an Internet-based application. The middle tier consists of the application business logic, which can be constructed as business component objects. These business objects can be new application modules or existing applications that are encapsulated so they can be integrated into the environment. The back tier includes data repositories, such

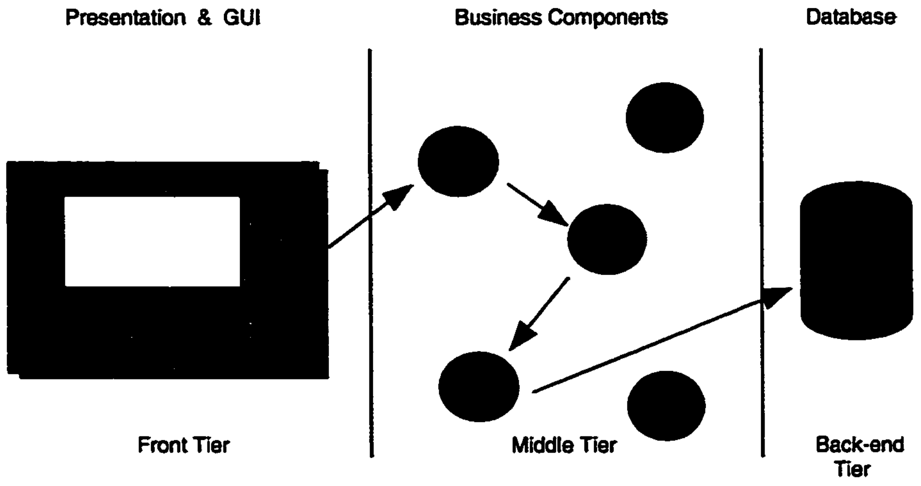


Figure 1: Distributed architecture diagram

as relational or object-oriented databases. The multi-tier component-based distributed system is superior to the traditional *Client/Server* system in the aspects of scalability, flexibility, security and productivity. Any third-party components can be plugged in to the middle-tier where a standard application server will host these components. This will demand component interfaces used in the distributed system must adhere to some standard by all parties involved with the creation and utilization of the business component objects. Examples of the component-based distributed systems include J2EE, DNA, and CORBA as described in Chapter 2.

1.2 Objectives

GRADER is designed and implemented in this kind of multi-tier component-based distributed system. It focuses on the features such as a three-tier component-based distributed instead of trivial stand-alone application. GRADER should provide basic functional capabilities to let teachers login and open a new course session, which can be used to store and retrieve students grades. GRADER is typically used for maintaining many course sessions and stores all transactions into a database aimed at Web access.

GRADER works on the main platforms (e.g. Unix, Windows) and with stable rich user interface. The Database also has good features like platform independency, and at least support for SQL standard. The Database hosts all the student data information and other related information. This system should be used by all groups of users – administrators, teachers and students. The system can be reliably accessed from the Internet by browsers like Netscape or Internet Explorer. The back-end database could be a relational database or object database, or even plain files. The storage is transparent to users, and the database will keep the persistence and connect via middle-tier business components. Teachers can record, modify/query/delete information for each student (ID, names, assignment/lab/test grades, exam grades, etc.) who belongs to teacher's courses, and registered students can know their own grades and other course information by logging in to the system and then modify his/her profile. Other facilities should be included in this application, like automatically computing course-weighted sums, averages, means and course final grades based on student basic marks. In addition, there should be some kind of facility to compute course statistics, and show all these course data and grade histograms to give very straightforward representation.

Based on the above considerations, the following design goals were chosen for this component-based distributed system [15][19]:

- Object-Oriented design

  OOD must be used to design GRADER in order to get the benefit of being easy maintain and extensible as well as reusability.

- Component based

  To explore the new technologies and follow the industrial trend, most persistence objects and business logic should be organized to be components, and allow these components interact each other to cooperate for providing the services to requests. The benefits of these components are the code reuse and service transparency. This goal will be reached by biding one of the current component design framework.

- Distributed computing

  GRADER should be implemented in a distributed computing environment, and act as a test bed of the adopted middleware application server.

- Internet based

  The system is to be used on the Internet, and should consider the Web accessibility from anywhere via Web browsers. This means that some kind of remote transport protocol must be used to provide the full communication.

- System usability

  GRADER must provide good user interfaces and be designed carefully to let users know what to do and get online help if needed.

- System portability

  GRADER should be designed and implemented to meet the multiple platform usage. The coding also needs to follow some kind of convention and have good documentation.

- System security

  Different group user such as teacher and student should have appropriate security levels to prevent student from reaching unauthorized content, and each teacher can only manage his/her own course materials.

2 Component-based distributed technologies

2.1 J2EE (Java 2 Platform Enterprise Edition)

The J2EE [2], which contains the Enterprise JavaBeans (EJB) [3] [4] server-side component architecture, was designed to meet the Web complex applications with the development, deployment and management of multi-tier enterprise solutions. It is used for the life cycle application development and deployment of component-based distributed business system, and is an open industry standard in the Java domain initiative led by Sun Microsystems. J2EE synchronizes the releases of the various Enterprise Java APIs, and defines the architecture to support both business logic components as well as data processing components, which are critical one to build a complex and reusable domain object model. J2EE also supports transaction load balancing and middle tier data caching mechanism. All of features make it easy to extend system scalability by using multi-tier deployment and load balancing. The Java language being platform independent makes it possible for J2EE application EJB components to be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification. A well-designed J2EE application can be independent of middleware, operating system, and hardware then can be deployed in a heterogeneous server-side environment. Middleware independence is possible because J2EE is a specification, not a middleware implementation. Operating system and hardware independence is possible because J2EE is built on the platform-independent Java virtual machine. J2EE provides full support for EJB components, Java Server Pages (JSPs) to create dynamic Web pages, and support for XML technology. J2EE also provides naming and directory services through the Java

Naming and Directory Interface, transaction services, and Java Messaging Service. Figure 2 describes this J2EE architecture [4][29].

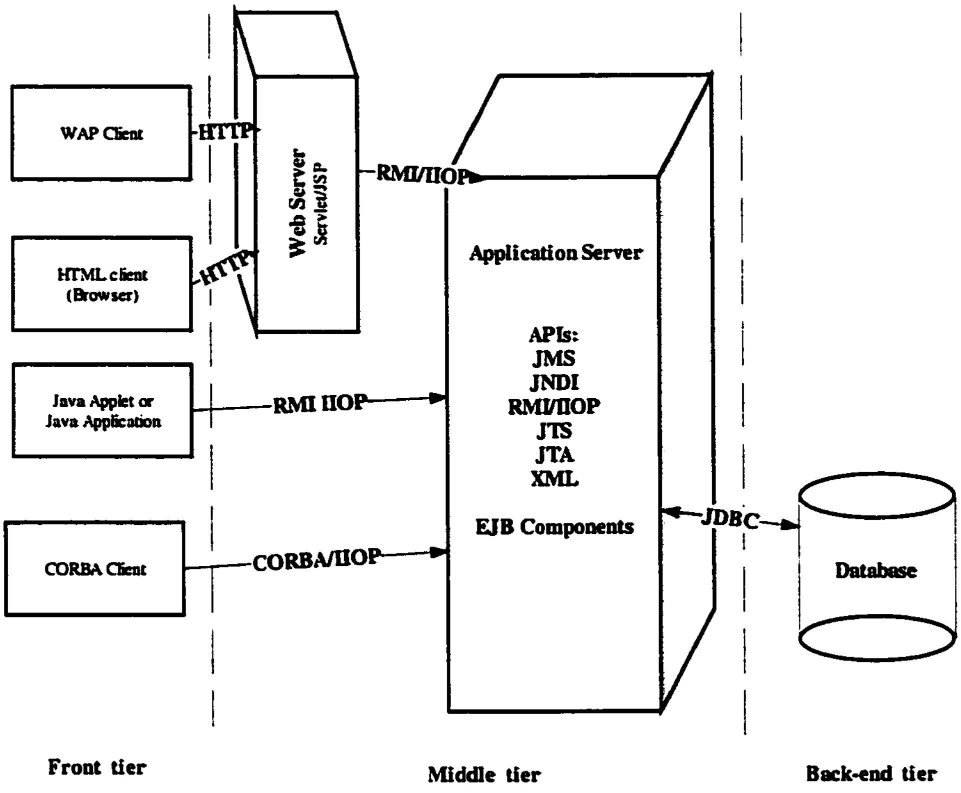


Figure 2: J2EE multi-tier architecture

The major benefit of the J2EE application model is in the middle tiers of multi-tier applications. In the J2EE platform, middle-tier business functions are implemented as Enterprise JavaBean components, as shown in FIGURE 2. These enterprise beans allow service developers to concentrate on the business logic and let the EJB server handle the complexities of delivering a reliable, scalable service. JSP(Java Server Pages) technology and servlets present middle-tier functions to the client tier as simple-to-access Internet-style services. JSP technology makes it easy for user interface developers to present dynamically generated pages to anyone with a browser. Servlets give more sophisticated developers of Java technology-based applications the freedom to implement dynamic presentations completely in the Java programming language. The following standard Java service APIs provide basic access to these systems:

JDBC - the standard API for accessing relational data from Java.

Java Naming and Directory Interface (JNDI) - the standard API for accessing information in enterprise name and directory services.

Java Message Service (JMS) - the standard API, which used to send and receive synchronized messages.

Java Transaction Service(JTS) - the standard API for transaction processing.

JavaMail (JMS) - the standard API for sending E-mail.

JavaIDL - the standard API for calling CORBA services.

J2EE is built on top level of Java 2 APIs and based on these flexible component configurations [7][18][30]. The J2EE application using J2EE architecture can achieve quicker development, easier customization and greater ability to develop powerful enterprise applications. The J2EE applications achieve all the benefits of Java technology: scalability, portability, and programming ease.

2.2 DNA (Distributed interNet Applications) architecture

Microsoft Windows DNA [23] is a framework and unifying architecture for distributed applications to access the rich set of services on the Windows platform. It is an extensible platform that builds on the Component Object Model (COM) [22] framework that allows developers to focus on solving business logic rather than on building basic Web solution plumbing. It consists of a combination of Microsoft products and technologies that help applications fully use the power of the Web construction. Windows DNA is a distributed architecture that describes how to build three-tier or n-tier applications for the Windows platform. Using DNA, developers just need to concentrate on maximizing overall application autonomy, reliability, availability, scalability, and interoperability. DNA adopts COM component technology and MTS role-based security to prevent clients from accessing critical resources directly and use MTS transactions to ensure accurate results in a multi-user environment. By implementing redundant hardware and software systems, which include using MSCS for redundancy solutions involving clustering DNA, can eliminate single points of failure. It also uses MSMQ's store and forward, guaranteed delivery, and dynamic routing features to simulate increased network availability. DNA architecture use MTS to share resources among users and to pool resources in short supply to minimize resource acquisition times. When connecting to a database, DNA uses ADO or OLE DB for universal data access and XML to share information with other applications. DNA includes these modules such as ADO, OLE DB, ODBC, MTS, DTC, XML, MSMQ, MSCS and COM+ Components. Within DNA, COM defines the basic component/object model and DCOM (Distributed COM) [21] allows components to be moved around various networks. MTS provides a component runtime environment for all the middle-tier components. DTC coordinates distributed transactions. MSMQ provides asynchronous communications and MSCS allows multiple application servers to work transparently form outside. Figure 3 illustrates this architecture.

The Windows DNA business tier contains business and data logic, encapsulated within COM+ components. COM+ components can be written in any language that supports COM+. All invocations to COM+ components are intercepted by the COM+ runtime, and delegated to the components. This gives the COM+ runtime the opportunity

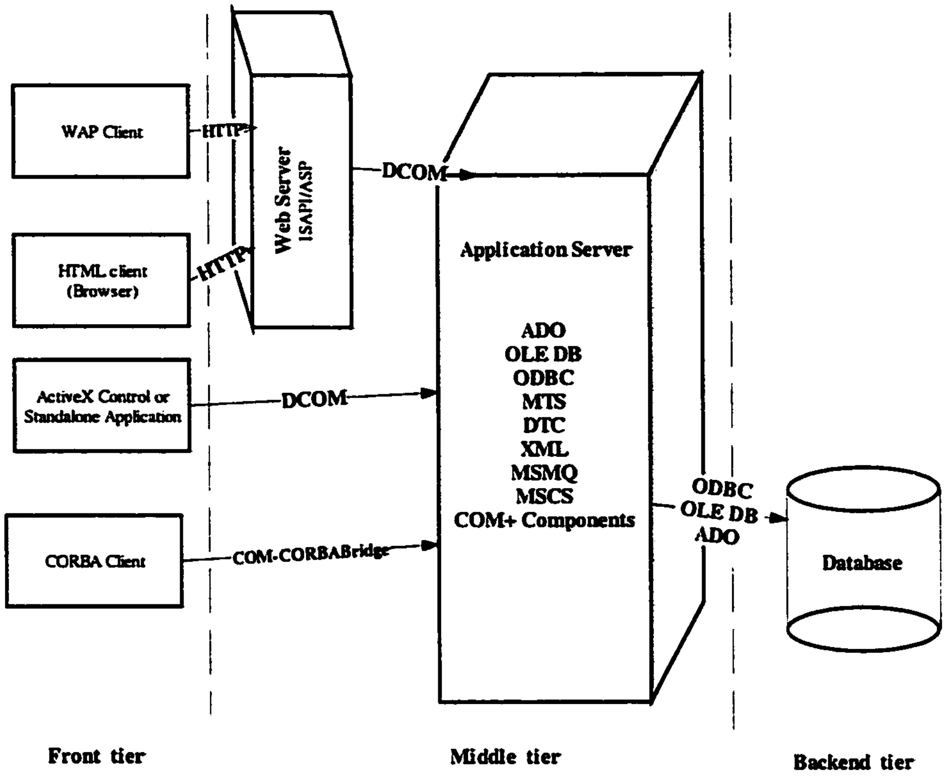


Figure 3: **DNA multi-tier architecture**

to perform middleware operations, such as transactions, security, and object lifecycle management. The technologies involving in DNA [23] [12] are:

Active Server Pages (ASP) - An open application environment in which HTML pages, server-side scripts, and ActiveX components are combined to create dynamic content Web-based applications.

Internet Server Application Programming Interface (ISAPI) - A set of functions for Internet servers included units, such as a Microsoft Windows NT Server running Microsoft Internet Information Server.

Distributed Component Object Model (DCOM) - An object protocol that enables ActiveX components to communicate directly with each other across a network. DCOM is language-neutral, so any language that produces ActiveX components can also produce DCOM applications.

ActiveX Data Objects (ADO) - A collection of data access objects within a hierarchical object library. ADO enables you to write a client application to access and manipulate data in a database server through a interface provider such as Microsoft IIS 4.0.

OLE DB - A set of OLE interfaces that provide applications with uniform access to data stored in diverse information sources, regardless of location or type. These interfaces allow data sources to share their data through common interfaces without having to implement non-native database functionality to the data store.

Open Database Connectivity (ODBC) -A standard programming language interface used to connect to a variety of data sources.

Microsoft Distributed Transaction Coordinator (DTC) - A system service that coordinates transactions. Work unit controlled by a DTC can be committed as an atomic transaction even if it spans multiple resource managers, potentially on separate computers. Microsoft DTC implements a two-phase commit protocol to ensure that the transaction outcome (either commit or abort) is consistent across all resource managers involved in a transaction. Microsoft DTC ensures atomicity, regardless of failures.

Microsoft Message Queue Server (MSMQ) - A technology that enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. Within an MSMQ enterprise, applications send messages to queues and read messages from queues.

Component Object Model (COM) - An architecture for defining interfaces among objects implemented by widely varying software applications. A COM object instantiates one or more interfaces, each of which exposes zero or more properties and zero or more methods. All COM interfaces are derived from the base class IUnknown. Technologies built on the COM include ActiveX, MAPI, and OLE.

Microsoft Transaction Server (MTS) - A COM-based transaction processing system that provides a run-time environment for objects to perform the business logic.

Component Services (COM+) - A set of services based on extensions of MTS and the COM that provide improved threading and security, transaction management, object pooling, and application administration and packaging.

2.3 CORBA (Common Object Request Broker Architecture)

The Common Object Request Broker Architecture (CORBA) [9] is an emerging open distributed object-computing infrastructure standardized by the Object Management Group (OMG). CORBA automates many common network programming tasks such as object registration, location, and activation; request de-multiplexing; framing and error-handling; parameter marshalling and de-marshalling; and operation dispatching. The goal is to move technology to a plug-and-play environment. The basis of the architecture is to recognize all components of a network's infrastructure as objects. The implication is that both the client and the server are now recognized as receiver and sender, not just only as receiver or just as sender. The core of the architecture is the Object Request Broker (ORB) through which all objects communicate. Objects are identified in four categories: applications, domains, services, and facilities. The main achievement of CORBA is the interoperability cross all heterogeneous platforms and all kind of program languages.

Clients access objects only through objects advertised interface, invoking only those operations that that object chooses to expose, with only those parameters (input and output) that are included in the invocation. In order to invoke the remote object instance, the client first obtains its object reference using Naming services. To make the remote invocation, the client uses the method calls just like it used in the local invocation, but substitutes the object reference for the remote instance. When the ORB examines the object reference and discovers that the target object is remote, it marshals the arguments and routes the invocation out over the network to the remote object's ORB. The CORBA object invocation process has two key levels: First, the client knows the type of object it is, and the client stub and object skeleton are generated from the same IDL. This means that the client knows exactly which kind of operations it may invoke and where to go in the invocation. Second, the client's ORB and object's ORB must agree on a common protocol, which is standard IIOP (Internet Inter-ORB Protocol). The typical CORBA distributed [9] application picture is as follows:

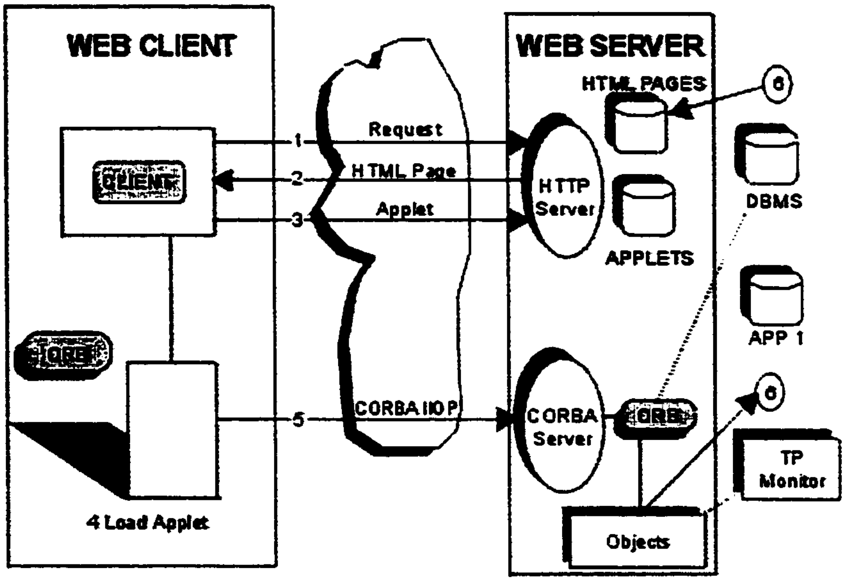


Figure 4: **CORBA application architecture**

The CORBA architecture defines many services including Object Services, Naming Service, Trading Service, Event Service, Query Service, Relationship Service, Security Service, Transaction Service and other common facilities.

2.4 Comparison

For serious enterprise applications deployed as three-tier or n-tier, the most important of requirements is scalability. The middle tier will be the component tier. It is on this tier that the business logic will run, packaged as software components. Instances of these components will be managing requests coming in from the client tier and will be updating back-end tier databases. Since the component instances are updating back-end databases, they require database connections. Database connections are valuable resources, and each component instance having a database connection is also a valuable resource. They too, therefore, must be managed carefully. In enterprise applications the number of clients using the system will usually far exceed the number of component instances the system can support. Any distributed architecture must therefore include some mechanisms for sharing instances among clients. All three of these architecture address component objects scalability by merging two different technologies [12][11]: Components and Transaction Processing Monitors (TPMs). Component coordination must support not only transactions explicitly expressed as sequences of service [10][6] invocations, but also the situation in which the internal implementation of one of those services in turn relies on an autonomous service. The issue of proprietary architectures within the CORBA is that it does not define a server side component coordinator framework. It does not deal with issues such as transactions, component packaging, or automatic state management, but a specification. Both J2EE and DNA have good middle-tier components objects caching mechanism, but DNA being a Microsoft member, sticks with only Windows, not a multiple platforms as most enterprise application deployed. It is not a pure platform independent solution to distributed objects computing. The strength of CORBA was deployed mainly in mainframe environments as a way to wrap legacy code for use as components. J2EE provides for interoperability with CORBA through The Java interface definition language, a CORBA IDL-to-Java compiler. J2EE makes distributed object technology more accessible through EJB technology and easier to use by offering an abstraction level that is higher, and therefore more efficient for software development. The complexities of distributed object communication are completely hidden. EJB can exploit object communication mechanisms such as CORBA and RMI without having to expose these underlying technologies to the application programmer. EJB uses RMI over IIOP - the CORBA wire protocol to allows the transparent mapping of RMI to CORBA. J2EE can work in the Java language, and still gain the possible interoperability benefits of CORBA. The most significant difference between Windows DNA and J2EE is that Windows DNA is a proprietary product supported by a single vendor, whereas J2EE is an open industry standard supported by a variety of middleware vendors, each of which are providing implementations for the standard. J2EE has both business and data components, whereas DNA only has business components. There is very weak support for stateful business processes in Windows DNA. COM+ [8] components cannot hold state beyond a transaction because they are activated and deactivated by the COM+ runtime upon transactional boundaries. J2EE fully support stateful business process by session EJB beans and persistent support by EJB entity beans. COM+ does not provide any form of automated persistence support, because there is no notion of a data component in COM+. The benefit of J2EE is obvious in a component-based heterogeneous distributed system.

3 High level Design of the three-tier GRADER

The basic requirement of GRADER is a three-tier, component-based distributed system, which is an expedition to a new computing paradigm. According to the analysis of the first part, J2EE is the best candidate to be applied to the GRADER online application. It has a sound architecture and mature APIs to satisfy all the needs of GRADER. By partitioning GRADER into three distributed tiers that are front end, middle tier, and back-end, we actually follow the MVC rule (Model, View, Control) [7][19] to design the system. Model is the persistent data stored in Database, and view is the user interface whereas Control is middle tier business component. The details of these three layers will be described in the following parts. The basic things are that front-end will provide a GUI interface to let teachers manage the course data and students have an easy way to browse their grade information. The middle tier is made of distributed components, which are EJB beans that may be deployed in several sites. This layer focuses on the business logic and performs the gateway to the database and front-end presentation GUI interface. The back-end layer plays a role to keep persistent objects, which will map to the records of some tables if using a relational database, or serializable binary files if using file storage.

This GRADER application also needs to consider the security problems like different access policies [24]. By using the J2EE well-defined security model, we can easily achieve this goal. J2EE can let the developer define access role for even every method call, and make sure the data will be consistently feed to users. Using different group identifiers, it is easy for the system to control the authentication of login user and present the different GUI with the proper access permission. The teacher has the privilege to register a new course section and set up what kind of grades that they want to prevent to student. Students can only view his/her own courses information restricted by the course lecturer and change their own access password or profile.

J2EE can let the application take full power from the Java programming language and hence it is a totally platform-independent solution. The mature EJB (Enterprise Java Bean) component technology guarantees the transactional database connection and remote invocation to the EJB components and good performance as well. It is the best choice for GRADER.

3.1 Front tier design

3.1.1 CLIENT SIDE TECHNOLOGY

J2EE provides rich front-end choices that are HTML/JavaScript, Java stand-alone, and Java applet within Java-enabled browser [30][27]. Let us analyze each of them to find which one is best solution for GRADER.

HTML with JavaScript uses the HTTP protocol to communicate with the Web server where JSP and Servlet technology from J2EE can provide dynamic content from the client request. In this mode, EJB components just talk to JSP or Servlet not HTML directly. All the GUIs have to be built using HTML and JavaScript that is very limited for GRADER like complex GUI demand. HTML also is static and this lead all the dynamic content request must go to JSP or Servlet back and forth and hence increase the Internet traffic. Each time the HTML page must be refreshed when student records are changed or course grade is calculated on the fly.

For the stand-alone Java front-end, full Java APIs like Swing and AWT can make a perfect user interface and very powerful to carry all the functionalities. The disadvantage is that the user must live with a JDK environment to run the GRADER application, and take care of all the Java environment set up and each time GRADER has new server side modification such we change some business logic, the client must have the same version of all modified class files to make the consistency to the server. Administration work is definitely evitable.

By adopting an Applet client solution, we can have the Java rich interface advantage, but avoid the administration problem. Users can use any Java-enabled browser like Netscape or IE to play as a client end point. We don't need to use JSP or Servlet to feed the dynamic content to users. The only disadvantage is that each time the applet needs to download the bunch of server class files to run the application. Comparing the above two solutions, it is worth to scarify this to get other benefits. Regarding the Applet security restriction, we can use a signed jar file or modify security policy to free the sand box limitation of the Applet.

3.1.2 FRONT TIER MODULES

From the user point of view, the GRADER GUI can be divided into three main groups, and each group has its own functional modules as shown in Figure5.

The individual modules can be further described to explain its functionality as follows. *Log in* – the entry login in a window to authenticate the user.

Monitor group:

- *Add/Modify/Query User*—registers new monitor/teacher/student, assigns login ID and password to each user, and can query a specific user to modify his/her profile or delete this user record from the database.
- *Batch add User*—provides a flexible batch mode way to add many users at the same time.

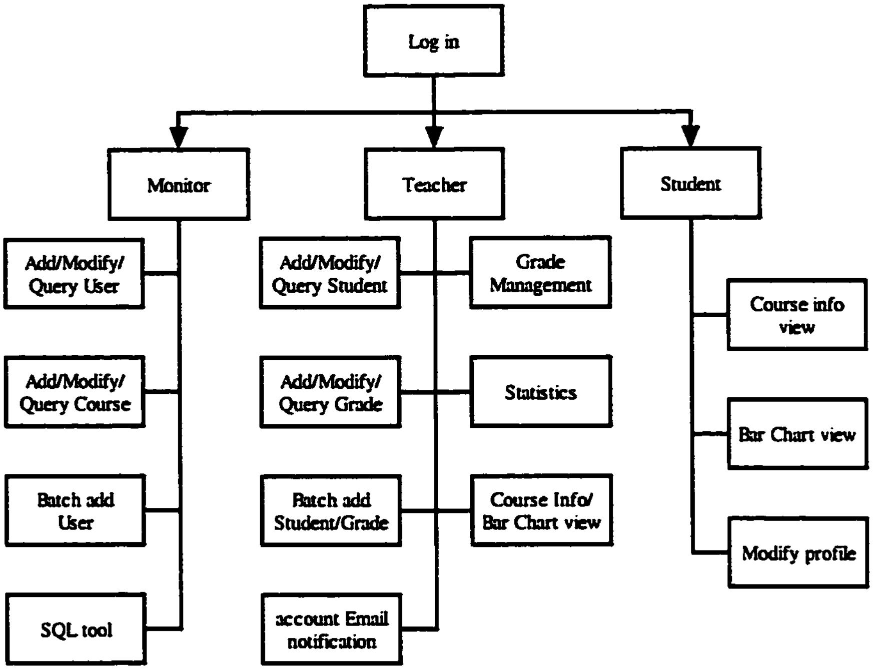


Figure 5: GRADER front-end GUI modules

- *Add/Modify/Query Course* – registers new course, assign course ID and teacher. Can query specific course and modify this course profile or delete this course from database.
- *SQL tool* – provides a GUI tool to query the database or build initial necessary tables.

<u>Teacher group:</u>

- *Add/Modify/Query student t*—registers new student, assign login ID and password to each student and can query specific student to modify/delete profile from database.
- *Add/Modify/Query Course grade* – adds new course grade item, registers students to this course and can query specific course and modify this course grade or delete this record from database.

- *Batch add student/grade* – provides flexible batch mode way to register many students to the course at the same time, and can add a whole class course grade instead of one by one.
- *Email notification* – provides a convenient Internet communication mechanism to send an E-mail of account information to individual students and student summary lists to teachers.
- *Grade Management* – provides way to let teacher author the course weight rule, set up student view permission, full mark assignment and final grade calculation.
- *Statistics* – provides way to let the teacher send command to middle tier component to calculate course items statistics.
- *Course Info/Bar Chart view* – lets teachers have the full choice to view course grade information and grade bar graphics chart.

Student group:

- *Course info view* –lets students view his/her courses grade summary information
- *Bar Chart view* – lets students have a full choice to view course grade bar graphics charts.
- *Modify profile* – lets a student modify his/her profile like password or e-mail address.

3.2 Middle tier design

3.2.1 ENTERPRISE JAVA BEAN INTRODUCTION

J2EE architecture depends heavily on the EJB components in the middle [25][26] tier to realize its business components distribution and data persistence database connection. Enterprise JavaBeans are well-designed reusable business logic components for use in distributed, multi-tier application architectures. EJB based applications can be get up and run quickly by just integrating all needed EJB components. For example, if you are developing a high-end e-commerce application, you might purchase one EJB component that performs on line credit card processing, another that manages a client database, and another that calculates shipping costs. You would then tie these together within your application server by setting up deployment properties of the beans, and there you would have it - an order processing system. An application server would automatically handle sticky issues like requests for balancing loads, maintaining security, monitoring transaction processes, object resources share, and ensuring data integrity.

EJB is different from the conventional Java bean at the point that it is not a GUI property and behavior of transactional server side Java bean components. Conventional beans are Java classes, typically graphical user interface (GUI) components, designed to conform to a series of programming conventions so that integrated development IDE environments. They follow conventions of interfaces using get and set to define class prosperities and aren't for use inside development. The purpose of Enterprise JavaBeans is to encapsulate business logic into server-side components. In the EJB paradigm, an application is implemented as a set of business-logic-controlling EJB components that have been configured in application-specific ways inside an "EJB container" such as an application server. Clients are then written to communicate with the EJB components and handle the results. The standardized interfaces exist to allow the EJB container to manage security and transactional aspects of the bean.

Application servers are runtime environments to host EJB containers because they automate the more low level features of multi-tier computing. Application servers manage resources on behalf of the components involved in the design. They also provide infrastructure services such as naming, directory services and security. And they provide bean-based applications with the benefit of scalability - most application server environments will let application clustered through the addition of new clusters of machines. EJB containers offer their beans a number of important services including Life cycle management, multithreading, security services, transaction, and persistence and state management services. This will keep the programmer focused on the business logic instead of system considerations.

3.2.2 HOW EJB WORKS

There are four types of EJB beans – stateless/stateful session beans and container-managed persistence/bean-managed persistence entity beans [27]. Session beans don't deal with data persistence and represent a process that will be performed on the server. Entity beans represent persistent objects and have to deal with data persistence. Each entity beans object might implicitly represent and mapping to a row in a relational database table and they will keep the data even system get crashed or reboot.

Each EJB component has two interfaces associated with it: EJBHome and [5] [4] EJBObject. The classes that represent these interfaces are created when the EJB is loaded into the container. EJBHome provides the way to create the EJB beans and have find methods for entity beans. Both session and entity beans have create methods that will return an instance of the EJBObject class. There may be a number of these create methods, each taking different parameters depending upon what data is required to initialize the EJB. In the case of entity beans, there will also be "finder methods," which take a primary key as a parameter and return a reference to a unique EJBObject instance or collection of EJBObjects matching find methods. EJBObject is an instance of the EJB bean remote interface class and is used by the client to access the methods provided by the EJB component. The EJBObject class acts as a proxy, putting the necessary infrastructure-specific code between the client and the EJB component. We can show the relationship as follows.

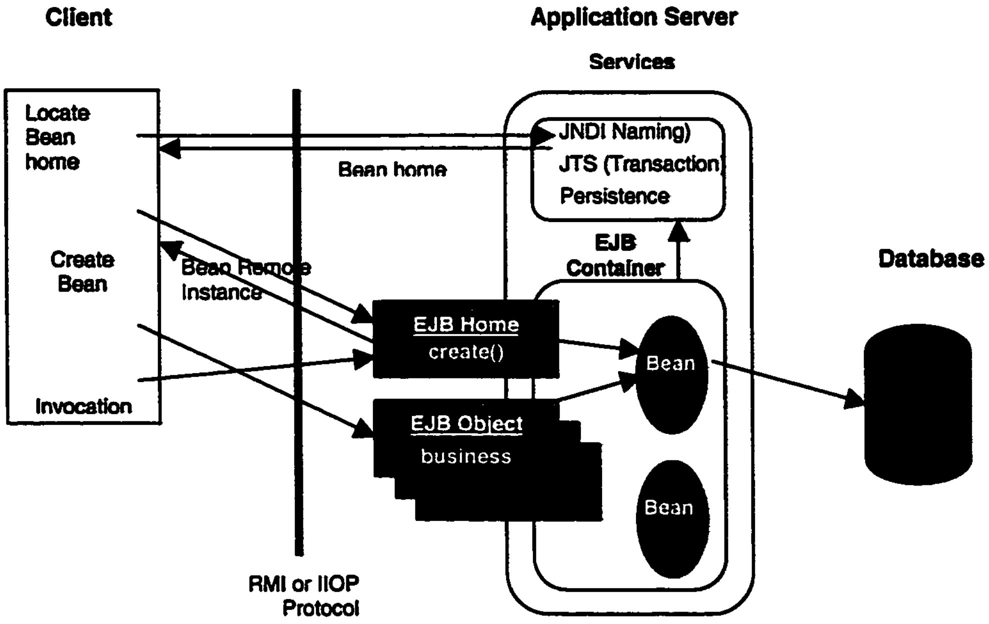


Figure 6: EJB work flow diagram

As shown above, a bean is installed into a container by supplying a deployment descriptor, which contains details about the bean's interfaces. EJB container tools analyze the EJB package, reading the deployment descriptor and generating implementations classes for the home and remote interfaces and client stubs. The bean's home object is then registered with the naming service, and the home and remote interfaces are then made available to clients. A client tries to find a bean's home interface by querying the Java naming service (JNDI). The client calls the create or find methods, as defined in the home interface, and receives in return a reference to the object that implements the remote interface. A session or entity EJB Object is a remote Java object that is accessible through the standard Java programming interfaces for remote invocation. The client simply calls business methods on the object. The container provides security, concurrency, transactions, swapping to secondary storage, and other services for the object. This is completely hidden from the client. As an example, consider the following client pseudo code for working with an EJB bean “User”:

```
UserHome userHome = <Naming Service>.lookup(“MyUser”);
User user = userHome.create(<creation parameters>);
(or User user = userHome.findByPrimaryKey(<primaryKey>));
methodReturnType result user.setName(“Bob”);
```

We first lookup EJB User remote home by naming service and then use returned User home object to create a new User remote interface or use *findByPrimaryKey()* method to find the remote interface object. After having EJB remote interface client can perform all the methods this User EJB registered just like a local object invocation. Note that the User bean implementation object is not exposed to the client. This is analogous to a CORBA object implementation.

The major difference between EJB and other distributed object protocols is in how servant objects are created and managed. In CORBA, servants are explicitly instantiated inside the server application and then registered with the ORB’s object adapter. The same programmer that writes the object implementation usually writes the server application. RMI is handled similarly. In the case of EJB, a bean programmer writes only the code for the bean implementation. The responsibility for executing the code is delegated to the EJB container, which in most cases is written by someone else.

The EJB container resides inside the EJB server, which interacts with the operating system and the system administrator.

EJB specifies a contract between containers and beans. The container contract is a protocol that defines how beans should be manipulated by a container. Part of the contract is a requirement for each bean to implement standard methods, for example *ejbCreate()* and *ejbFind()*. A bean also contains implementations of business logic methods. The container uses Java reflection to redirect calls made by clients to the remote interface for execution by the bean's methods. As an illustration of the complexity of bean management within a container, the following diagram shows the lifecycle of a stateful EJB session bean [3]:

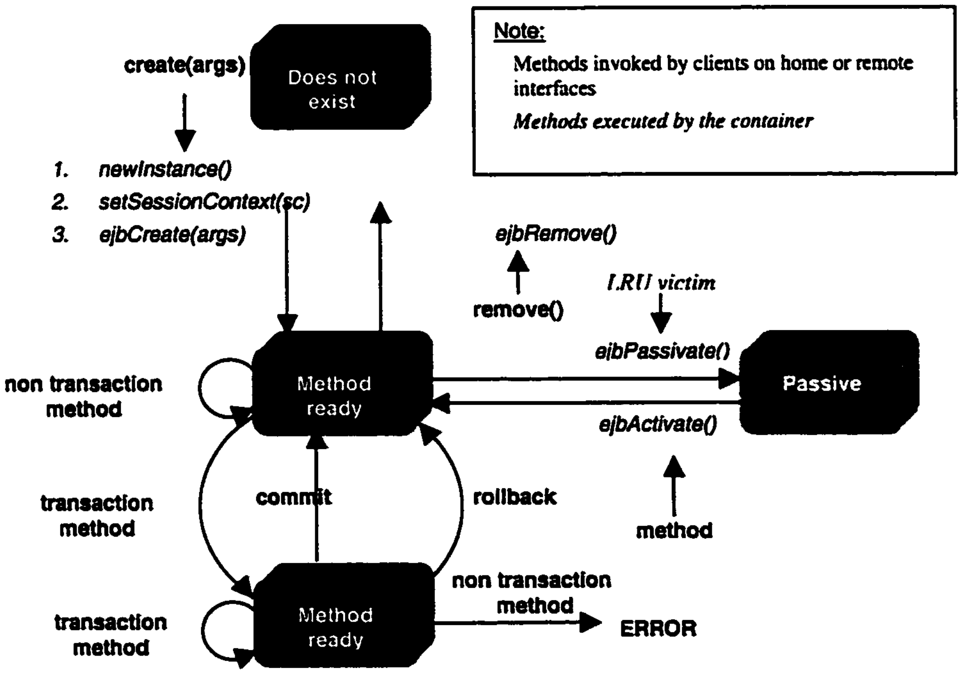


Figure 7: **EJB bean lifecycle management**

3.2.3 GRADER EJB COMPONENTS

In the GRDAER system, based on the requirement analysis, six entity beans and one session bean are chosen to play as middle layer components.

User entity bean – provides basic three groups with user authentication information, stores user ID and name, and password.

Student entity bean – stores student's extra specific information.

Teacher entity bean – stores teacher's extra specific information.

Course entity bean – stores course information such as course name, section, room, teacher, office hour, etc.

CourseGrade entity bean – stores course' grade information for all registered students.

Rule entity bean – stores course rules like weigh allocation, full mark, final grade schema and student viewing permission rule.

GraderManager session bean – provides basic methods for the grade summary calculation.

3.3 Back end tier design

Since EJB defines entity beans to represent persistence data. A single copy of a particular entity bean is shared among all clients that need it. Therefore, all entity beans must map to a underlying storage system such as database or file system. In the GRADER system, we use database to store the data. J2EE reference implementation provides a Java-based relational database called CloudScape. We can also use other databases like Oracle or DataStore. The mapping can be done with any relational database because all the entity beans in GRADER are container-managed persistence bean. You can deploy those beans in the deploy time to decide which relational database to use. The database tables that correspond to entity beans are:

CourseBeanTable, CourseGradeBeanTable, RuleBeanTable, StudentBeanTable, TeacherBeanTable, UserEJBTable.

4 OBJECT ORIENTED DESIGN AND IMPLEMETATION

4.1 *Class diagram*

GRADER class package dependency diagram and is presented in figure 8 and class diagrams are shown in figure 9,10,11.

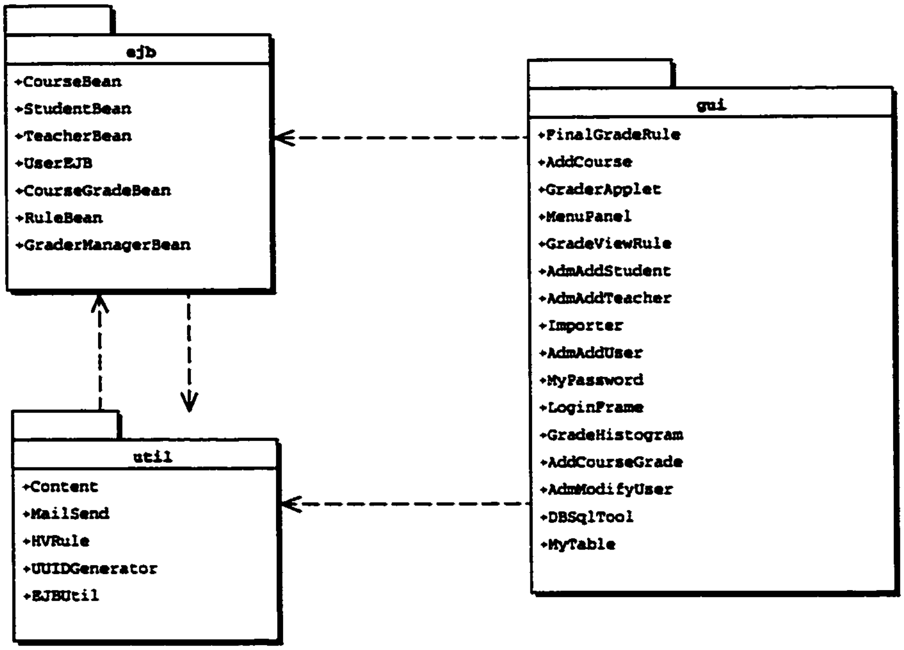


Figure 8: **GRADER class package diagram**

GRADER is designed to have three packages that are grader.ejb, grader.gui and grader.util. Package grader.ejb contains all EJB entity beans and session bean, and grader.gui contains all GUI class and help classes. Package grader.util only contains all packages needed for utility classes. The detail classes are described as follow:

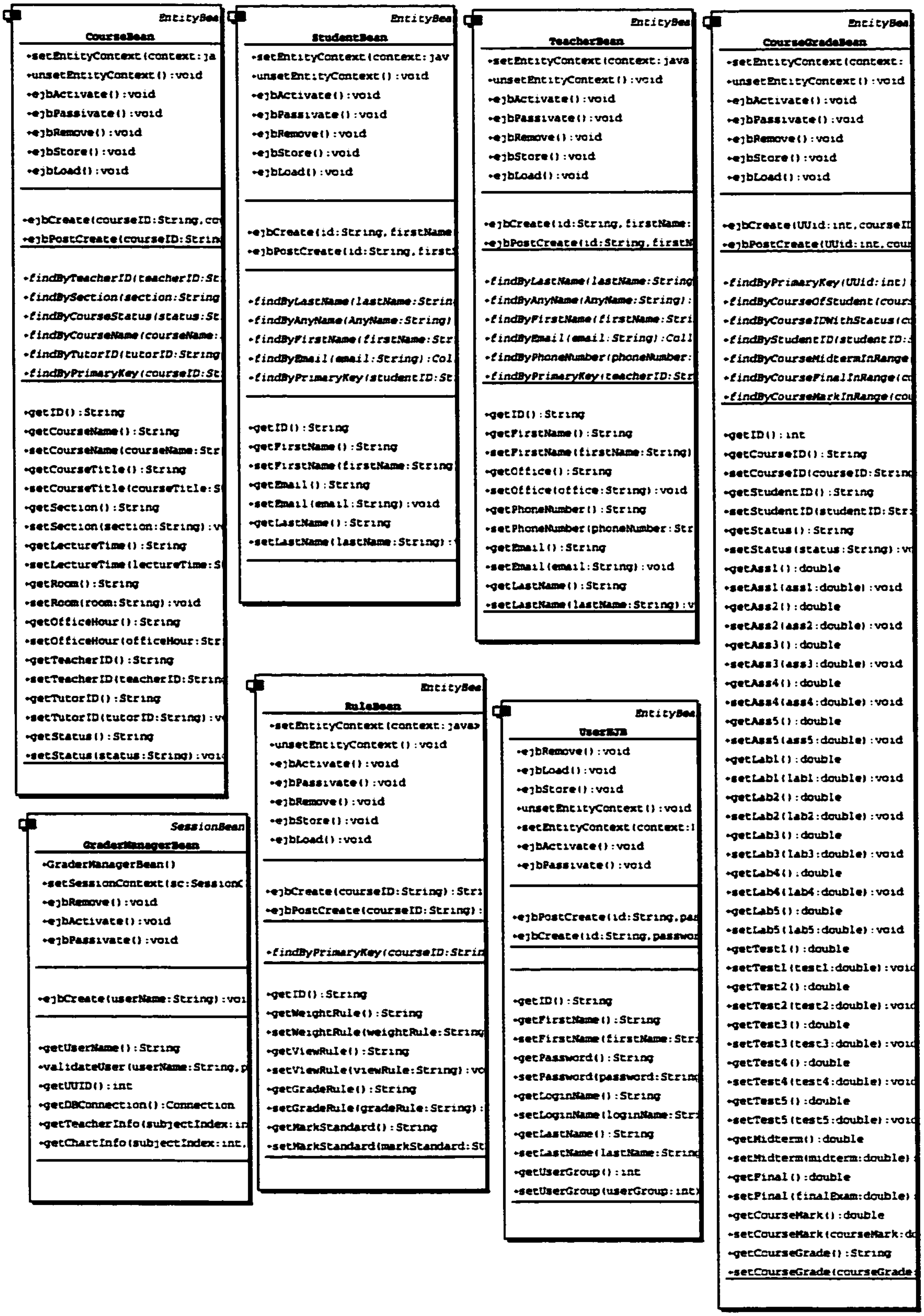
4.2 *GRADER class Packages*

4.2.1 PACAKGE GRADER.EJB

public interface User extends javax.ejb.EJBObject

-User EJB Entity bean remote interface

Figure 9: grader.ejb class diagram



EntityBea
CourseBean
+setEntityContext(context:ja
+unsetEntityContext():void
+ejbActivate():void
+ejbPassivate():void
+ejbRemove():void
+ejbStore():void
+ejbLoad():void
+ejbCreate(courseID:String,co
+ejbPostCreate(courseID:Strin
+findByTeacherID(teacherID:St
+findBySection(section:String
+findByCourseStatus(status:St
+findByCourseName(courseName:
+findByTutorID(tutorID:String
+findByPrimaryKey(courseID:St
+getID():String
+getCourseName():String
+setCourseName(courseName:Str
+getCourseTitle():String
+setCourseTitle(courseTitle:S
+getSection():String
+setSection(section:String):v
+getLectureTime():String
+setLectureTime(lectureTime:S
+getRoom():String
+setRoom(room:String):void
+getOfficeHour():String
+setOfficeHour(officeHour:Str
+getTeacherID():String
+setTeacherID(teacherID:Strin
+getTutorID():String
+setTutorID(tutorID:String):v
+getStatus():String
+setStatus(status:String):voi
EntityBea
StudentBean
+setEntityContext(context:jav
+unsetEntityContext():void
+ejbActivate():void
+ejbPassivate():void
+ejbRemove():void
+ejbStore():void
+ejbLoad():void
+ejbCreate(id:String,firstName
+ejbPostCreate(id:String,first
+findByLastName(lastName:Strin
+findByAnyName(AnyName:String)
+findByFirstName(firstName:Str
+findByEmail(email:String):Col
+findByPrimaryKey(studentID:St
+getID():String
+getFirstName():String
+setFirstName(firstName:String
+getEmail():String
+setEmail(email:String):void
+getLastName():String
+setLastName(lastName:String):
EntityBea
TeacherBean
+setEntityContext(context:java
+unsetEntityContext():void
+ejbActivate():void
+ejbPassivate():void
+ejbRemove():void
+ejbStore():void
+ejbLoad():void
+ejbCreate(id:String,firstName:
+ejbPostCreate(id:String,firstN
+findByLastName(lastName:String
+findByAnyName(AnyName:String):
+findByFirstName(firstName:Stri
+findByEmail(email:String):Coll
+findByPhoneNumber(phoneNumber:
+findByPrimaryKey(teacherID:Str
+getID():String
+getFirstName():String
+setFirstName(firstName:String)
+getOffice():String
+setOffice(office:String):void
+getPhoneNumber():String
+setPhoneNumber(phoneNumber:Str
+getEmail():String
+setEmail(email:String):void
+getLastName():String
+setLastName(lastName:String):v
EntityBea
CourseGradeBean
+setEntityContext(context:
+unsetEntityContext():void
+ejbActivate():void
+ejbPassivate():void
+ejbRemove():void
+ejbStore():void
+ejbLoad():void
+ejbCreate(UUid:int,courseID
+ejbPostCreate(UUid:int,cour
+findByPrimaryKey(UUid:int)
+findByCourseOfStudent(cours
+findByCourseIDWithStatus(co
+findByStudentID(studentID:
+findByCourseMidtermInRange
+findByCourseFinalInRange(co
+findByCourseMarkInRange(cou
+getID():int
+getCourseID():String
+setCourseID(courseID:String
+getStudentID():String
+setStudentID(studentID:Str
+getStatus():String
+setStatus(status:String):v
+getAss1():double
+setAss1(ass1:double):void
+getAss2():double
+setAss2(ass2:double):void
+getAss3():double
+setAss3(ass3:double):void
+getAss4():double
+setAss4(ass4:double):void
+getAss5():double
+setAss5(ass5:double):void
+getLab1():double
+setLab1(lab1:double):void
+getLab2():double
+setLab2(lab2:double):void
+getLab3():double
+setLab3(lab3:double):void
+getLab4():double
+setLab4(lab4:double):void
+getLab5():double
+setLab5(lab5:double):void
+getTest1():double
+setTest1(test1:double):voi
+getTest2():double
+setTest2(test2:double):voi
+getTest3():double
+setTest3(test3:double):voi
+getTest4():double
+setTest4(test4:double):voi
+getTest5():double
+setTest5(test5:double):voi
+getMidterm():double
+setMidterm(midterm:double)
+getFinal():double
+setFinal(finalExam:double)
+getCourseMark():double
+setCourseMark(courseMark:d
+getCourseGrade():String
+setCourseGrade(courseGrade
EntityBea
RuleBean
+setEntityContext(context:java
+unsetEntityContext():void
+ejbActivate():void
+ejbPassivate():void
+ejbRemove():void
+ejbStore():void
+ejbLoad():void
+ejbCreate(courseID:String):Stri
+ejbPostCreate(courseID:String):
+findByPrimaryKey(courseID:Strin
+getID():String
+getWeightRule():String
+setWeightRule(weightRule:String
+getViewRule():String
+setViewRule(viewRule:String):vo
+getGradeRule():String
+setGradeRule(gradeRule:String):
+getMarkStandard():String
+setMarkStandard(markStandard:St
EntityBea
UserEJB
+ejbRemove():void
+ejbLoad():void
+ejbStore():void
+unsetEntityContext():void
+setEntityContext(context:I
+ejbActivate():void
+ejbPassivate():void
+ejbPostCreate(id:String,pas
+ejbCreate(id:String,passwor
+getID():String
+getFirstName():String
+setFirstName(firstName:Str
+getPassword():String
+setPassword(password:String
+getLoginName():String
+setLoginName(loginName:Str
+getLastName():String
+setLastName(lastName:String
+getUserGroup():int
+setUserGroup(userGroup:int)
SessionBean
GraderManagerBean
+GraderManagerBean()
+setSessionContext(sc:SessionC
+ejbRemove():void
+ejbActivate():void
+ejbPassivate():void
+ejbCreate(userName:String):voi
+getUserName():String
+validateUser(userName:String,p
+getUUID():int
+getDBConnection():Connection
+getTeacherInfo(subjectIndex:in
+getChartInfo(subjectIndex:int,

public interface UserHome extends javax.ejb.EJBHome

-User EJB Entity home interface

public class UserEJB extends java.lang.Object implements javax.ejb.EntityBean

-User EJB Entity bean implementation

public interface Teacher extends javax.ejb.EJBObject

-Teacher EJB Entity bean remote interface

public interface TeacherHome extends javax.ejb.EJBHome

-Teacher EJB Entity bean home interface

public class TeacherBean extends java.lang.Object implements javax.ejb.EntityBean

-Teacher EJB Entity bean implementation

public interface Student extends javax.ejb.EJBObject

-Student EJB Entity bean remote interface

public interface StudentHome extends javax.ejb.EJBHome

-Student EJB Entity bean home interface

public class StudentBean extends java.lang.Object implements javax.ejb.EntityBean

-Student EJB Entity bean implementation

public interface Course extends javax.ejb.EJBObject

-Course EJB Entity bean remote interface

public interface CourseHome extends javax.ejb.EJBHome

-Course EJB entity bean home interface

public class CourseBean extends java.lang.Object implements

javax.ejb.EntityBean

-Course entity Bean implementation

public interface Rule extends javax.ejb.EJBObject

-Rule EJB Entity bean remote interface

public interface RuleHome extends javax.ejb.EJBHome

-Rule EJB Entity bean home interface

public class RuleBean extends java.lang.Object implements javax.ejb.EntityBean

-Rule EJB Entity bean implementation

public interface CourseGrade extends javax.ejb.EJBObject

-CourseGrade EJB Entity bean remote interface

public interface CourseGradeHome extends javax.ejb.EJBHome

-CourseGrade EJB entity bean home interface

public class CourseGradeBean extend java.lang.Object implements EntityBean

-CourseGrade EJB entity bean implementation

4.2.2 PACAKGE GRADER.GUI

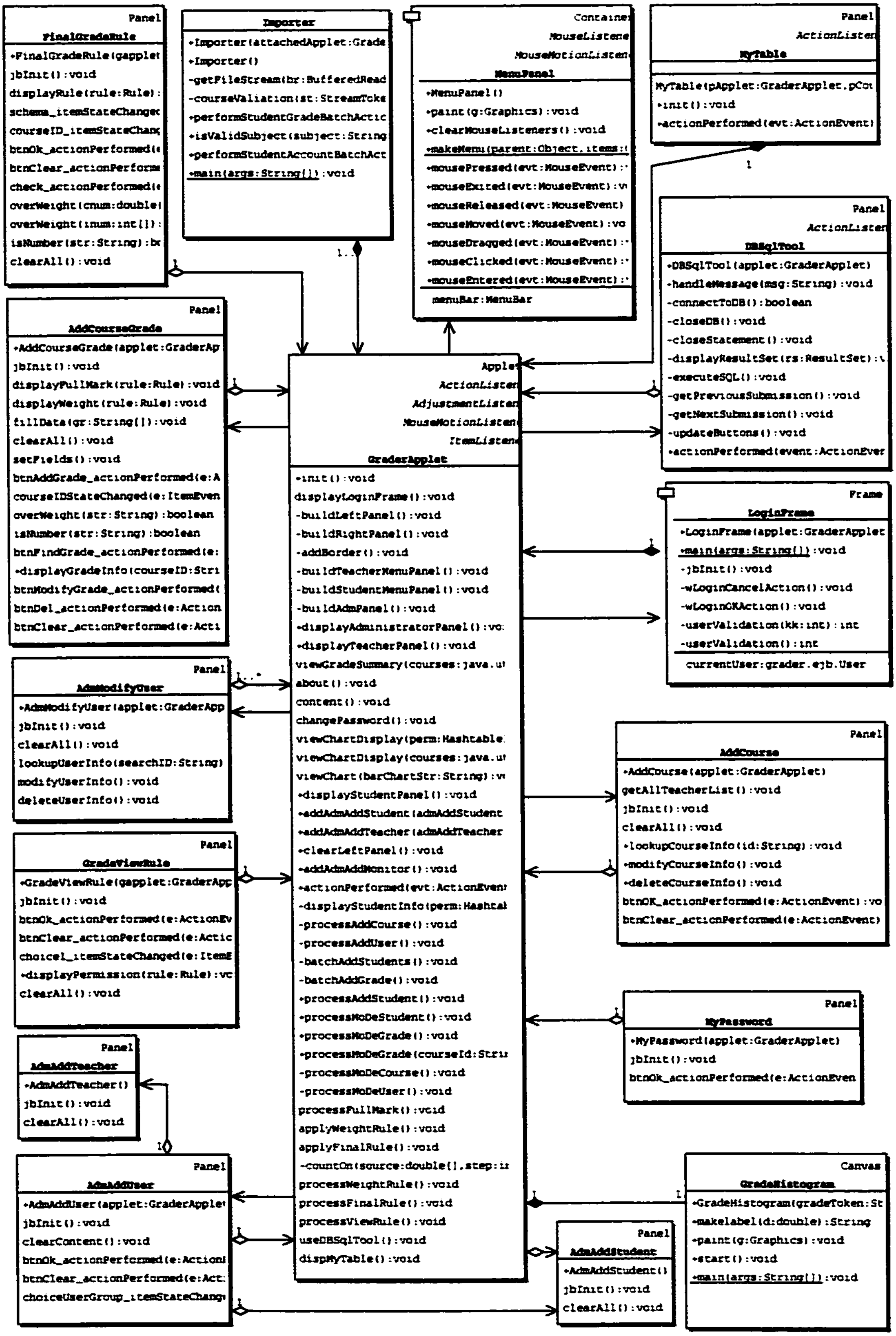
public class GraderApplet extends java.applet.Applet implements

java.awt.event.ActionListener, java.awt.event.AdjustmentListener,

java.awt.event.MouseMotionListener, java.awt.event.ItemListener

-Provides a full functionality to three kinds of users Monitor: Perform all users Add/Modify/Delete and SQL query course registration Teacher: Perform students registration and course grade management Student: Perform grade query and profile modification.

Figure 10: grader.gui class diagram


Panel
FinalGradeRule
+FinalGradeRule(gapplet
jbInit():void
displayRule(rule:Rule):
schema_itemStateChange(
courseID_itemStateChang
btnOk_actionPerformed(
btnClear_actionPerform
check_actionPerformed(
overWeight(cnum:double
overWeight(inum:int[]):
isNumber(str:String):bo
clearAll():void
Importer
+Importer(attachedApplet:Grade
+Importer()
-getFileStream(br:BufferedRead
-courseValiation(st:StreamToke
+performStudentGradeBatchActic
+isValidSubject(subject:String
+performStudentAccountBatchAct
+main(args:String[]):void
Container
MouseListene
MouseMotionListen
MenuPanel
+MenuPanel()
+paint(g:Graphics):void
+clearMouseListeners():void
+makeMenu(parent:Object,items:
+mousePressed(evt:MouseEvent):
+mouseExited(evt:MouseEvent):v
+mouseReleased(evt:MouseEvent)
+mouseMoved(evt:MouseEvent):vo
+mouseDragged(evt:MouseEvent):
+mouseClicked(evt:MouseEvent):
+mouseEntered(evt:MouseEvent):
menuBar:MenuBar
Panel
ActionListen
MyTable
MyTable(pApplet:GraderApplet,pCo
+init():void
+actionPerformed(evt:ActionEvent)
1
Panel
ActionListen
DBSqlTool
+DBSqlTool(applet:GraderApplet)
-handleMessage(msg:String):void
-connectToDB():boolean
-closeDB():void
-closeStatement():void
-displayResultSet(rs:ResultSet):v
-executeSQL():void
-getPreviousSubmission():void
-getNextSubmission():void
-updateButtons():void
+actionPerformed(event:ActionEver
1
1..
Panel
AddCourseGrade
+AddCourseGrade(applet:GraderAp
jbInit():void
displayFullMark(rule:Rule):void
displayWeight(rule:Rule):void
fillData(gr:String[]):void
clearAll():void
setFields():void
btnAddGrade_actionPerformed(e:A
courseIDStateChanged(e:ItemEven
overWeight(str:String):boolean
isNumber(str:String):boolean
btnFindGrade_actionPerformed(e:
+displayGradeInfo(courseID:Stri
btnModifyGrade_actionPerformed(
btnDel_actionPerformed(e:Action
btnClear_actionPerformed(e:Acti
Applet
ActionListen
AdjustmentListen
MouseMotionListen
ItemListen
GraderApplet
+init():void
displayLoginFrame():void
-buildLeftPanel():void
-buildRightPanel():void
-addBorder():void
-buildTeacherMenuPanel():void
-buildStudentMenuPanel():void
-buildAdmPanel():void
+displayAdministratorPanel():vo
+displayTeacherPanel():void
viewGradeSummary(courses:java.ut
about():void
content():void
changePassword():void
viewChartDisplay(perm:Hashtable
viewChartDisplay(courses:java.ut
viewChart(barChartStr:String):v
+displayStudentPanel():void
+addAdmAddStudent(admAddStudent
+addAdmAddTeacher(admAddTeacher
+clearLeftPanel():void
+addAdmAddMonitor():void
+actionPerformed(evt:ActionEvent
-displayStudentInfo(perm:Hashtal
-processAddCourse():void
-processAddUser():void
-batchAddStudents():void
-batchAddGrade():void
+processAddStudent():void
+processMoDeStudent():void
+processMoDeGrade():void
+processMoDeGrade(courseId:Stri
-processMoDeCourse():void
-processMoDeUser():void
processFullMark():void
applyWeightRule():void
applyFinalRule():void
-countOn(source:double[],step:i
processWeightRule():void
processFinalRule():void
processViewRule():void
useDBSqlTool():void
dispMyTable():void
Frame
LoginFrame
+LoginFrame(applet:GraderApplet
+main(args:String[]):void
-jbInit():void
-wLoginCancelAction():void
-wLoginOKAction():void
-userValidation(kk:int):int
-userValidation():int
currentUser:grader.ejb.User
1..*
Panel
AdmModifyUser
+AdmModifyUser(applet:GraderApp
jbInit():void
clearAll():void
lookupUserInfo(searchID:String)
modifyUserInfo():void
deleteUserInfo():void
Panel
AddCourse
+AddCourse(applet:GraderApplet)
getAllTeacherList():void
jbInit():void
clearAll():void
+lookupCourseInfo(id:String):void
+modifyCourseInfo():void
+deleteCourseInfo():void
btnOK_actionPerformed(e:ActionEvent):vo
btnClear_actionPerformed(e:ActionEvent)
Panel
GradeViewRule
+GradeViewRule(gapplet:GraderApp
jbInit():void
btnOk_actionPerformed(e:ActionEv
btnClear_actionPerformed(e:Actic
choice1_itemStateChanged(e:ItemE
+displayPermission(rule:Rule):vc
clearAll():void
Panel
MyPassword
+MyPassword(applet:GraderApplet)
jbInit():void
btnOk_actionPerformed(e:ActionEven
Panel
AdmAddTeacher
+AdmAddTeacher()
jbInit():void
clearAll():void
1
Panel
AdmAddUser
+AdmAddUser(applet:GraderApple
jbInit():void
clearContent():void
btnOk_actionPerformed(e:Action
btnClear_actionPerformed(e:Act
choiceUserGroup_itemStateChang
Canvas
GradeHistogram
+GradeHistogram(gradeToken:St
+makelabel(d:double):String
+paint(g:Graphics):void
+start():void
+main(args:String[]):void
1
Panel
AdmAddStudent
+AdmAddStudent()
jbInit():void
clearAll():void

public class AddCourse extends java.awt.Panel

-AddCourse is used to add a new course entry to Grader distributed system.

public class AddCourseGrade extends java.awt.Panel

-AddCourseGrade class is for a teacher to add course grade entry to system.

public class AdmAddStudent extends java.awt.Panel

-AdmAddStudent is used to register a new student entry in the database.

public class AdmAddUser extends java.awt.Panel

-AdmAddUser is used to register a new user in DB

public class AdmAddTeacher extends java.awt.Panel

-AdmAddTeacher is used to register a new teacher in DB

public class AdmModifyUser extends java.awt.Panel

-AdmModifyUser is a GUI object to let a user modify an exsited user profile in Database.

public class DBSqlTool extends java.awt.Panel implements java.awt.event.ActionListener

-DBSqlTool is used to let a Monitor manipulate the database directly with out other GUI.

public class FinalGradeRule extends java.awt.Panel

-FinalGradeRule class is a GUI component to let a teacher assign final course grade schema.

public class MenuPanel extends java.awt.Container

implements java.awt.event.MouseListener, java.awt.event.MouseMotionListener

-MenuPanel: The Menu Container class. Uses text Labels and PopupMenus to simulate a Menu Bar. Standard MenuBars can only be used in Frame windows, but this can also be used in Applets, as well as within any other Container class.

USAGE: Set up your MenuBar object as usual, then call setMenuBar(menuBar). You can use MenuPanel.makeMenu(...) to help you create MenuBar items.

public class GradeViewRule extends java.awt.Panel

-GradeViewRule class is used to provide a GUI interface to let the teacher manipulate the viewing rule of the courses.

public class Importer extends java.lang.Object

-Importer class is used to deal with the student account and course grade batch mode processing.

public class MyPassword extends java.awt.Panel

- The student profile modification GUI

public class MyTable extends java.awt.Panel

- This is class used to display grade detailed grid table view GUI

public class LoginFrame extends java.awt.Frame

-This is entry class for Grader system. It is for first login window

public class GradeHistogram extends java.awt.Canvas

-GradeHistogram is for drawing student Grade Histogram graphics

4.2.3 PACAKGE GRADER.UTIL

public class Content extends java.lang.Object

-Provides a full help content to three kind of users group Monitor/Teacher/Student

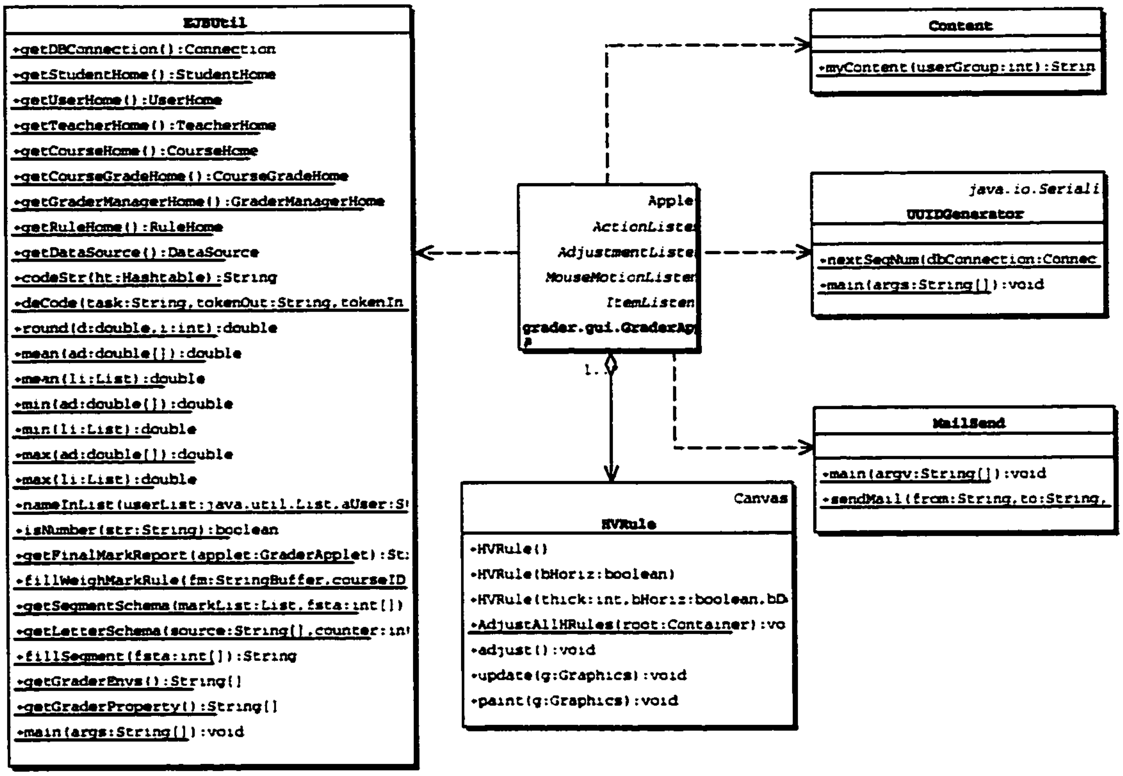
public class MailSend extends java.lang.Object

-MailSend is used to construct and send an RFC822 (plain text) email content.

public class HVRule extends java.awt.Canvas

-Component class to draw small Horiz or Vert Rule Separator lines.

Figure 11: grader.util class diagram


EJBUtil
+getDBConnection():Connection
+getStudentHome():StudentHome
+getUserHome():UserHome
+getTeacherHome():TeacherHome
+getCourseHome():CourseHome
+getCourseGradeHome():CourseGradeHome
+getGraderManagerHome():GraderManagerHome
+getRuleHome():RuleHome
+getDataSource():DataSource
+codeStr(ht:Hashtable):String
+deCode(task:String,tokenOut:String,tokenIn
+round(d:double,i:int):double
+mean(ad:double[]):double
+mean(li:List):double
+min(ad:double[]):double
+min(li:List):double
+max(ad:double[]):double
+max(li:List):double
+nameInList(userList:java.util.List,aUser:S
+isNumber(str:String):boolean
+getFinalMarkReport(applet:GraderApplet):St
+fillWeighMarkRule(fm:StringBuffer,courseID
+getSegmentSchema(markList:List,fsta:int[])
+getLetterSchema(source:String[],counter:in
+fillSegment(fsta:int[]):String
+getGraderEnvs():String[]
+getGraderProperty():String[]
+main(args:String[]):void
Apple
ActionListe
AdjustmentListe
MouseMotionListen
ItemListen
grader.gui.GraderAp
p
1..
Content
+myContent(userGroup:int):Strin
java.io.Seriali
UUIDGenerator
+nextSeqNum(dbConnection:Connec
+main(args:String[]):void
MailSend
+main(argv:String[]):void
+sendMail(from:String,to:String,
Canvas
HVRule
+HVRule()
+HVRule(bHoriz:boolean)
+HVRule(thick:int,bHoriz:boolean,bD
+AdjustAllHRules(root:Container):vo
+adjust():void
+update(g:Graphics):void
+paint(g:Graphics):void

public class UUIDGenerator extends java.lang.Object implements

java.io.Serializable

-This class is used to generate unique primary keys for CourseGradeBean.

public final class EJBUtil extends java.lang.Object

-This is a utility class for obtaining EJB components references and other application lever methods collection.

4.3 Use Cases of the System

Use Case diagrams provide a way of describing the external view of the system and its interactions with the outside world. Use case diagram shows the relationship among actors and use cases within a system. We here present use case of the GRADER system.

4.3.1 USE CASE OF THE MONITOR

A monitor actually can interact with GRADER for new user registration, modification and course information maintenance. Figure 12 shows the use case.

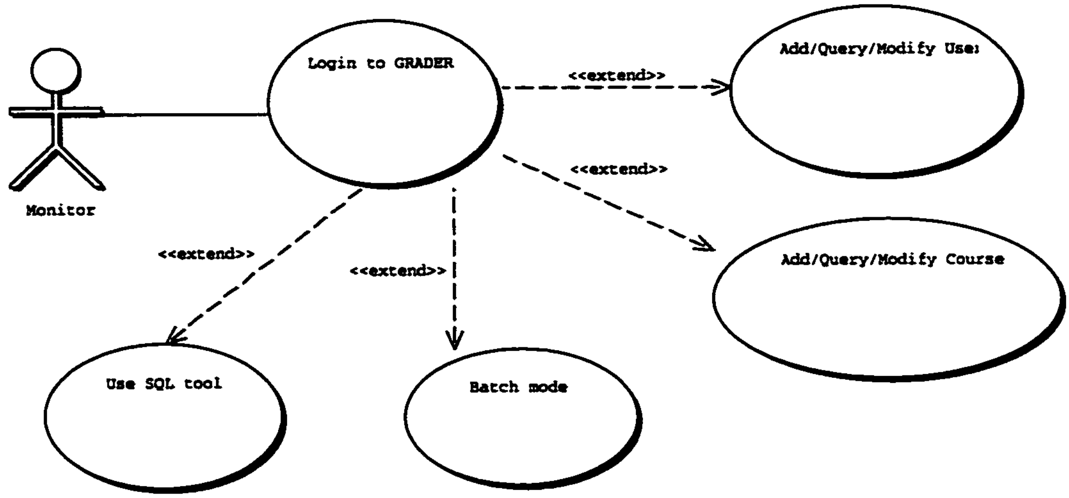


Figure 12: The Monitor use case diagram

4.3.2 USE CASE OF STUDENT

A student can query with the GRADER system to view their course information and modify their profile. Figure 13 shows this use case.

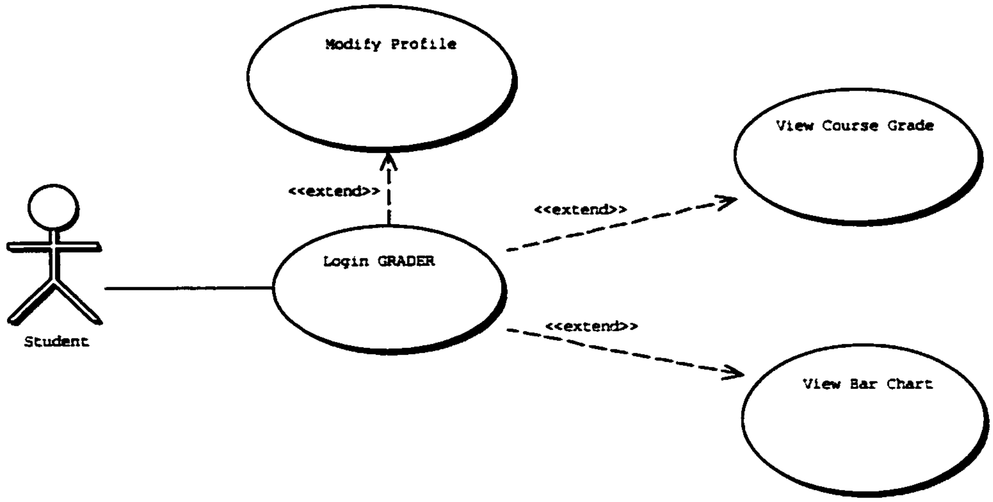


Figure 13: The student use case diagram

4.3.3 USE CASE OF TEACHER

A teacher can interact with the GRADER system to manage student information and course grade. This is main functionality of system. Figure 13 shows the use case.

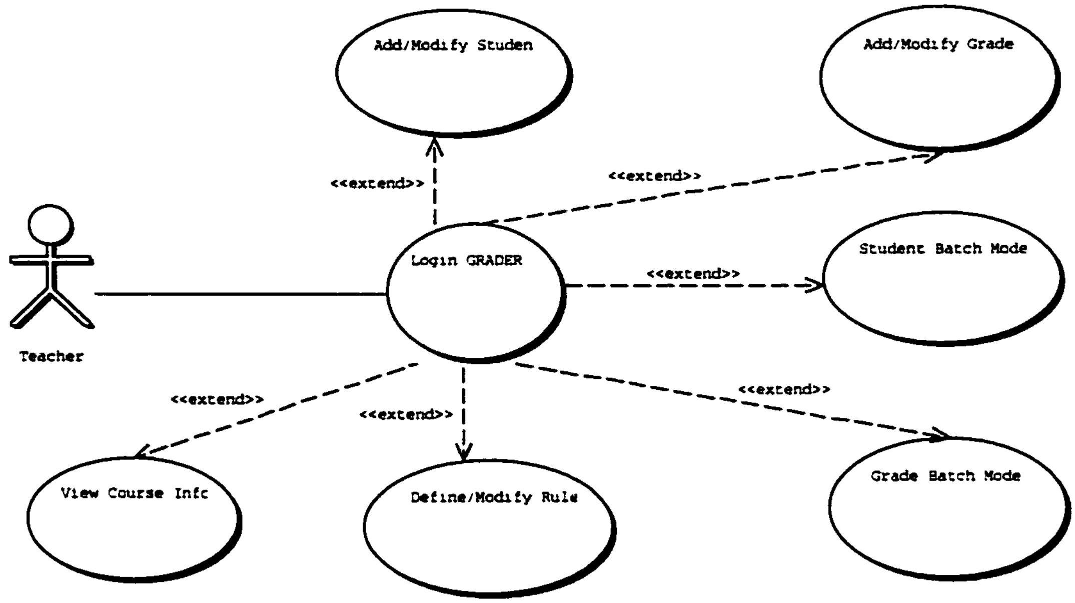


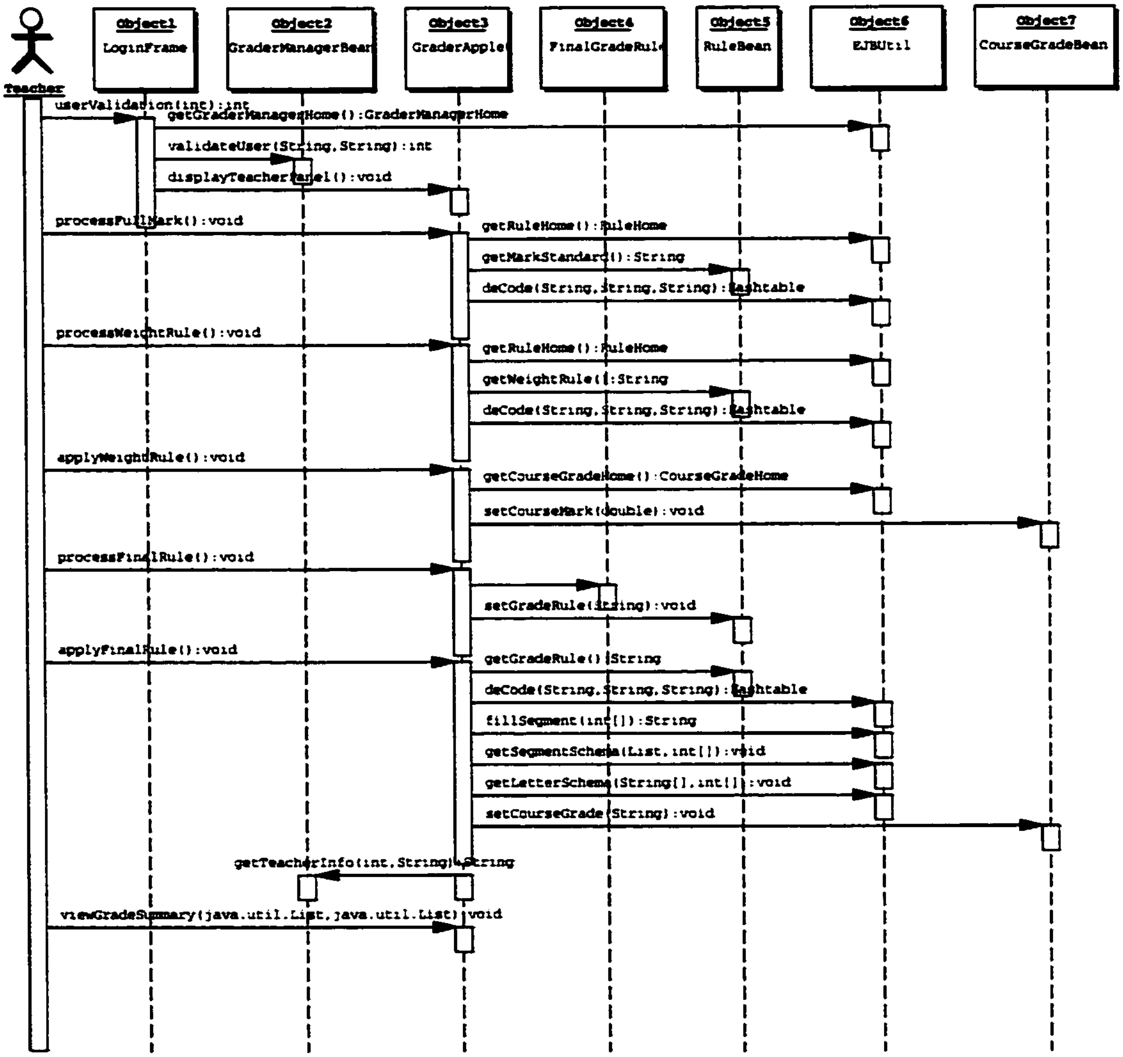
Figure 14: **The teacher use case diagram**

4.4 Final grade Sequence diagram

To further describe the GRADER system dynamic interaction, we present a detail sequence diagram in Figure 15 to show corresponding interactions between classes to achieve the goal of the teacher's "get final grade". The teacher first logs in to the GRADER system, *LoginFrame* is the actually class to validate the user and assign next the GUI to the coming user. *LoginFrame* will try first to get a *GraderManagerbean* bean home and then invoke its method to validate the teacher's status. Teacher then needs to verify all needed full course standards are already there and weight also ready for calculating. In this process classes like *GraderApplet*, *CourseGradeBean*, *RuleBean* and

EJBUtil cooperate to finish the required job. After that teacher can issue a command to ask the system calculate the course final grade. In this calculating process, class *GraderApplet* will first retrieve the final grade rule if it already exists or let the user define a new rule and then compare the course final mark to get the final grade letter for each student in the course and display this result to the teacher.

Figure 15: Final Grade sequence diagram



Teacher
Object1
LoginFrame
Object2
GraderManagerBean
Object3
GraderApplet
Object4
FinalGradeRule
Object5
RuleBean
Object6
EJBUtil
Object7
CourseGradeBean
userValidation(int):int
getGraderManagerHome():GraderManagerHome
validateUser(String,String):int
displayTeacherPanel():void
processFullMark():void
getRuleHome():RuleHome
getMarkStandard():String
deCode(String,String,String):Hashtable
processWeightRule():void
getRuleHome():RuleHome
getWeightRule():String
deCode(String,String,String):Hashtable
applyWeightRule():void
getCourseGradeHome():CourseGradeHome
setCourseMark(double):void
processFinalRule():void
setGradeRule(String):void
applyFinalRule():void
getGradeRule():String
deCode(String,String,String):Hashtable
fillSegment(int[]):String
getSegmentSchema(List,int[]):void
getLetterSchema(String[],int[]):void
setCourseGrade(String):void
getTeacherInfo(int,String):String
viewGradeSummary(java.util.List,java.util.List):void

4.5 *Detailed implementation*

4.5.1 GRADER DATABASE SCHEMA

To meet the EJB entity beans data structure, we need to have same database schema in the back-end. Each container-managed persistence bean will mapping to its corresponding table when being deployed, and all beans data fields have one-to-one mapping relationship to table fields including primary key. The following figure shows the GRADER database schema.

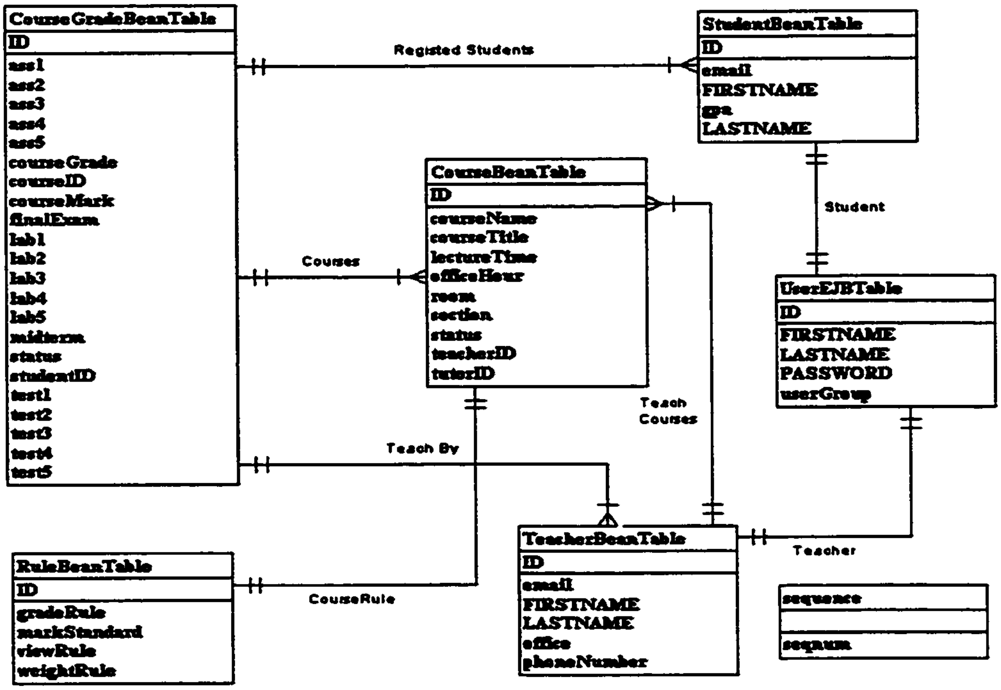
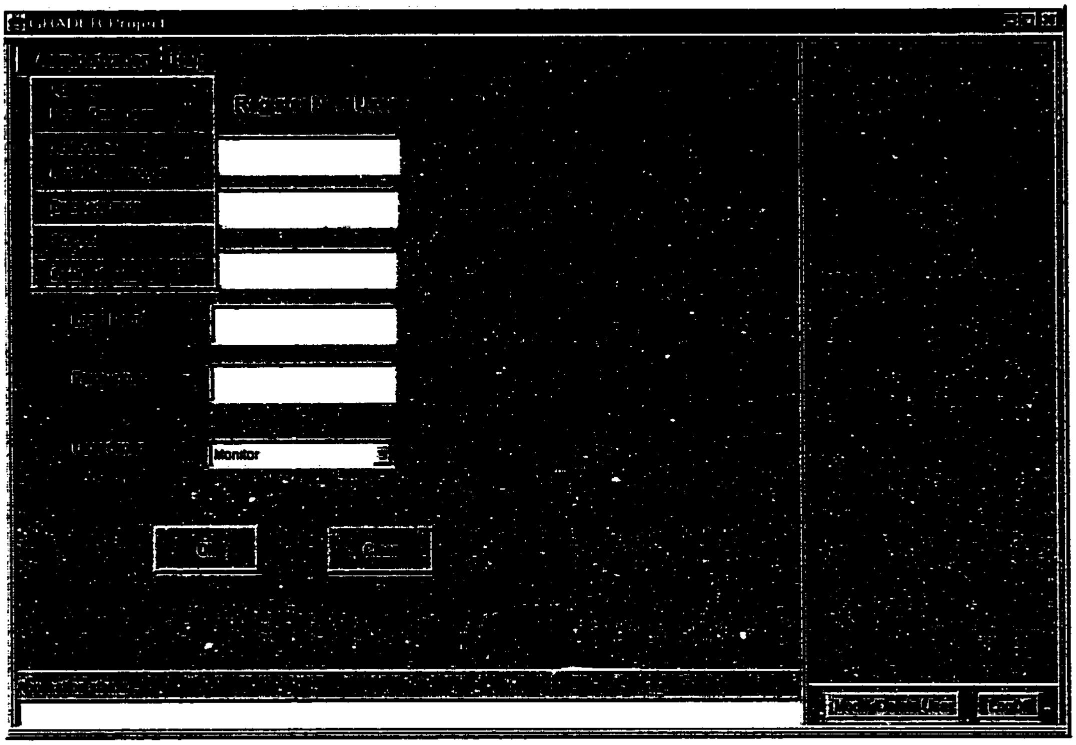


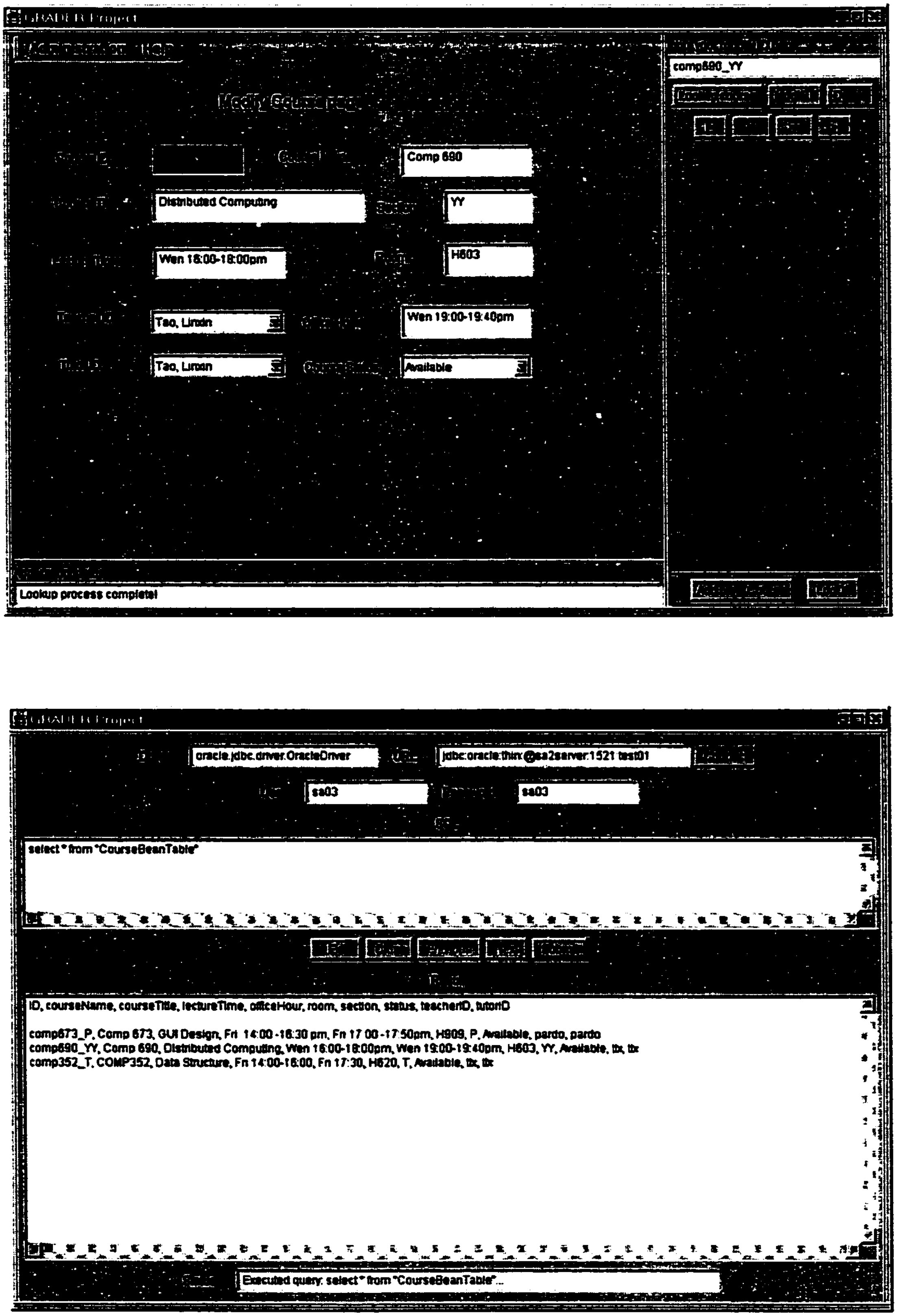
Figure 16: **GRADER Database schema diagram**

4.5.2 GRADER GUIs

GRADER front tier presents rich interfaces to the end-user, and lets each user group have different access GUIs to interact with the system. The login GUI is provided to be a entry GUI to all three user groups for validating the user's identity and then forward to the predefined assigned user main GUI.

In monitor GUIs as shown below, the monitor first gets the page of "Add New User" where the monitor can register a system new user of any group. If choosing "Modify/Delete user" menu item, the system will let monitor either browser user one by one or find specific user information quickly by the given user ID. The monitor can also use the "Batch add mode" to add whole class students name at one shot. This is convenient to deal with bunch data import to the GRADER system. The monitor GUI was designed to be responsible for course information manipulation to centralize the course resources management instead of letting the teacher manage this trial job. A more powerful GUI is the "SQL tool", which can let the monitor connect a remote database server to perform some database tuning and direct table management. This GUI accepts standard SQL standard statements and uses JDBC to connect to the database. It can be also used to establish initial GRADER database tables if the J2EE EJBs deploy tool doesn't create these tables.





comp690_YY

Comp 690

Distributed Computing

YY

Wen 16:00-18:00pm

H603

Tao, Limin

Wen 19:00-19:40pm

Tao, Limin

Available

Lookup process complete!

oracle.jdbc.driver.OracleDriver

jdbc:oracle:thin:@sa2server:1521:test01

sa03

sa03

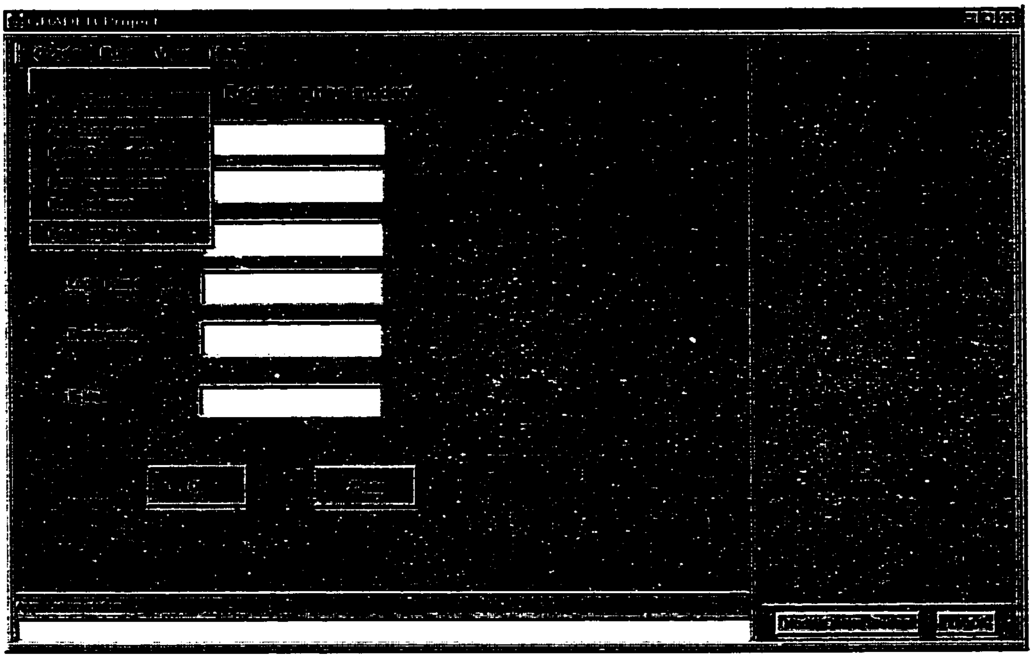
```
select * from "CourseBeanTable"
```

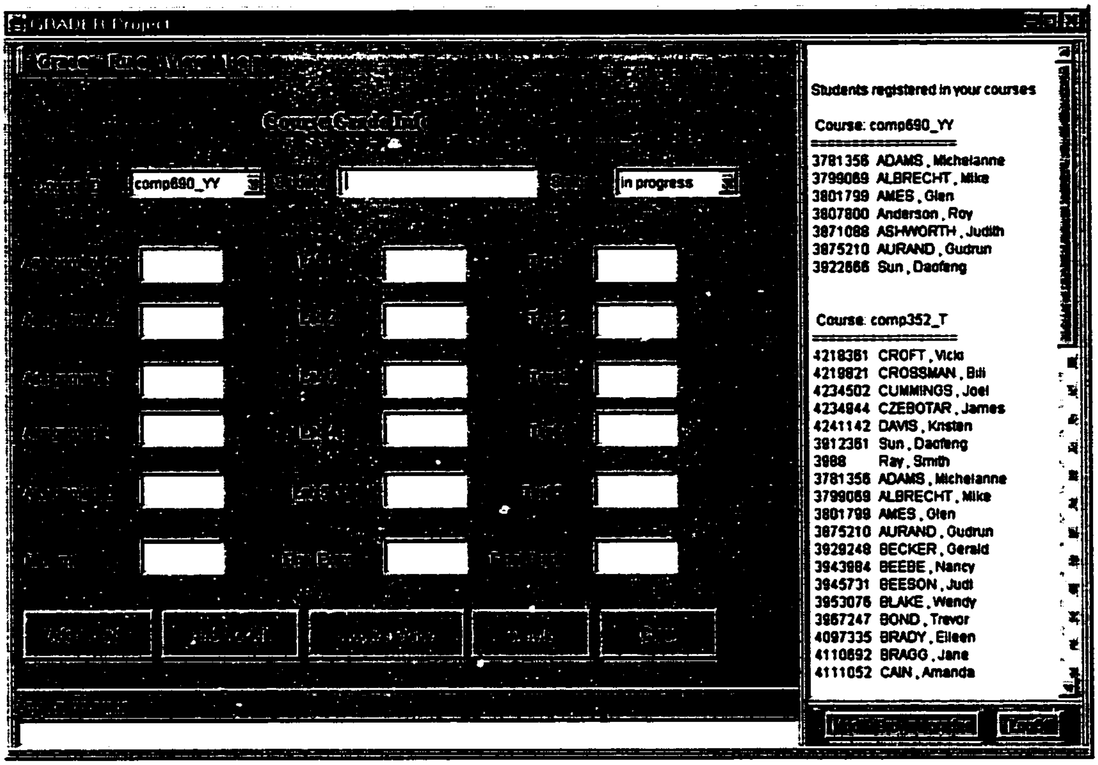
```
ID, courseName, courseTitle, lectureTime, officeHour, room, section, status, teacherID, tutorID

comp673_P, Comp 673, GUI Design, Fri  14:00 -16:30 pm, Fri 17:00 -17:50pm, H909, P, Available, pardo, pardo
comp690_YY, Comp 690, Distributed Computing, Wen 16:00-18:00pm, Wen 19:00-19:40pm, H603, YY, Available, tlx, tlx
comp352_T, COMP352, Data Structure, Fri 14:00-16:00, Fri 17:30, H620, T, Available, tlx, tlx
```

Executed query: select * from "CourseBeanTable"...

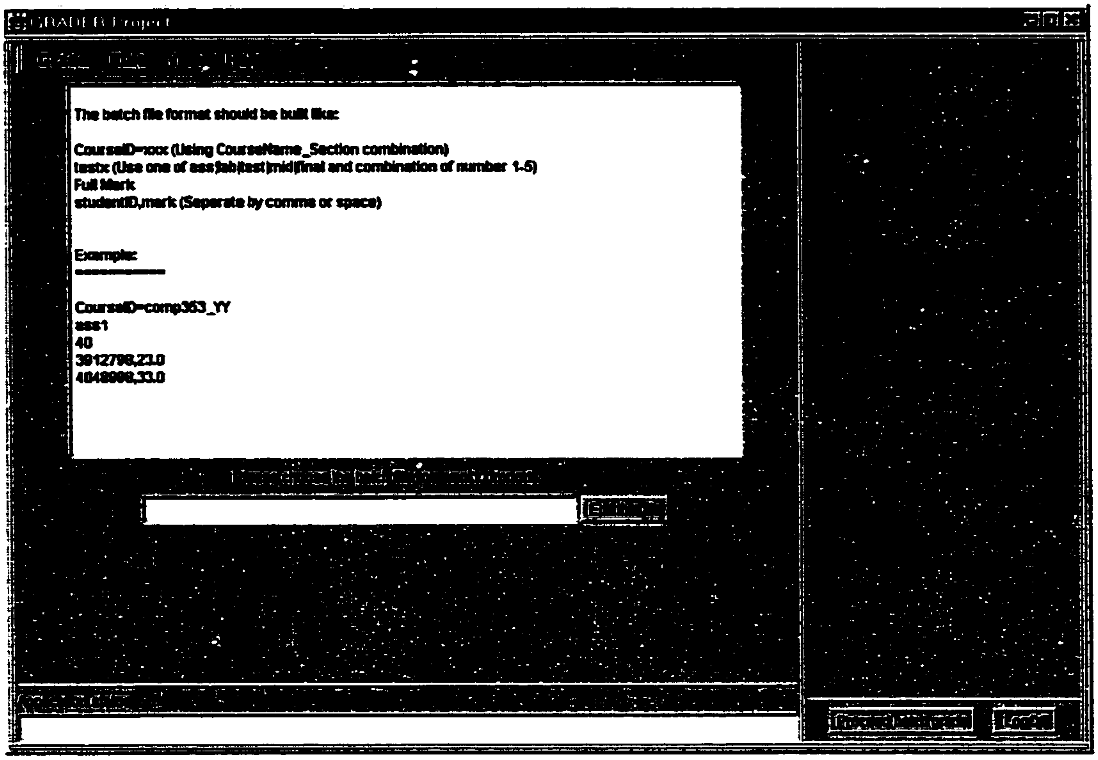
Figure 17: The monitor group GUIs

The GUIs provided for the teacher group user has all the functionalities to let the teacher finish all jobs. The entry page is the "Course Grade Info" page, which can let the teacher add a new course grade item into the system as well as providing a detailed students registration list to current teacher's courses. The teacher can then choose another menu item from the menu and perform related operation. The "Modify/Delete student " menu item can let the teacher either browse student profiles or find a specific student by given student ID, and modify the entry of that student. By choosing the "Batch register student", the teacher can register whole class student at one shot just like the monitor, but only for current teachers own courses. If "Email notification" is activated, all the registration students can get an email notification containing his/her login name and password, and the teacher can get a registration summary report containing the rest of all students. The teacher can choose "Add Course grade" to add an individual course grade entry, or use "Batch add grade" to add many entries at the same time like adding student accounts. "Modify/Delete grade" is used to let the teacher to browse course grade information or to find a specific student record from the database, and perform either the modification or delete operation. Figure 18 shows the GUIs of this group.

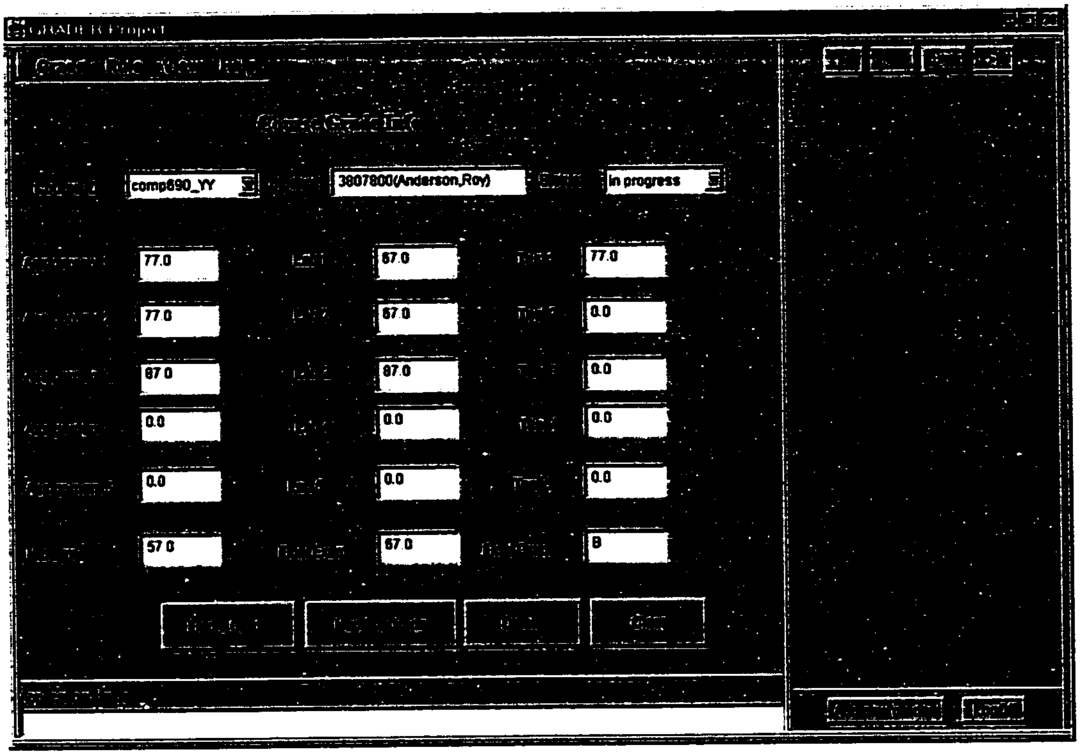




```
Students registered in your courses

 Course: comp690_YY
 ==================
3781356  ADAMS , Michelanne
3799069  ALBRECHT , Mike
3801799  AMES , Glen
3807800  Anderson , Roy
3871088  ASHWORTH , Judith
3875210  AURAND , Gudrun
3922666  Sun , Daofeng


 Course: comp352_T
 ==================
4218361  CROFT , Vicki
4219821  CROSSMAN , Bill
4234502  CUMMINGS , Joel
4234844  CZEBOTAR , James
4241142  DAVIS , Kristen
3912361  Sun , Daofeng
3988     Ray , Smith
3781356  ADAMS , Michelanne
3799069  ALBRECHT , Mike
3801799  AMES , Glen
3875210  AURAND , Gudrun
3929248  BECKER , Gerald
3943984  BEEBE , Nancy
3945731  BEESON , Judi
3953076  BLAKE , Wendy
3967247  BOND , Trevor
4097335  BRADY , Eileen
4110692  BRAGG , Jane
4111052  CAIN , Amanda
```



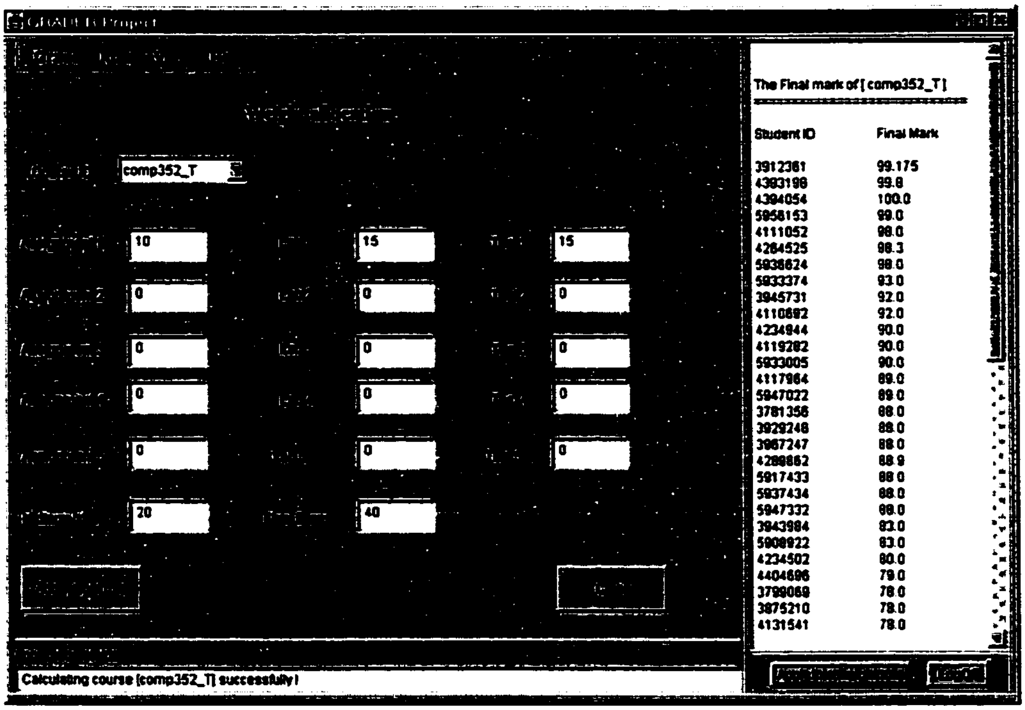
```
The batch file format should be built like:

CourseID=xxx (Using CourseName_Section combination)
testx (Use one of ass|lab|test|mid|final and combination of number 1-5)
Full Mark
studentID,mark (Separate by comma or space)


Example:
==========

CourseID=comp363_YY
ass1
40
3912798,23.0
4048998,33.0
```



comp690_YY	3807800(Anderson,Roy)	in progress
77.0	67.0	77.0
77.0	67.0	0.0
87.0	87.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0
57.0	67.0	B

Figure 18: The teacher group GUIs (1)

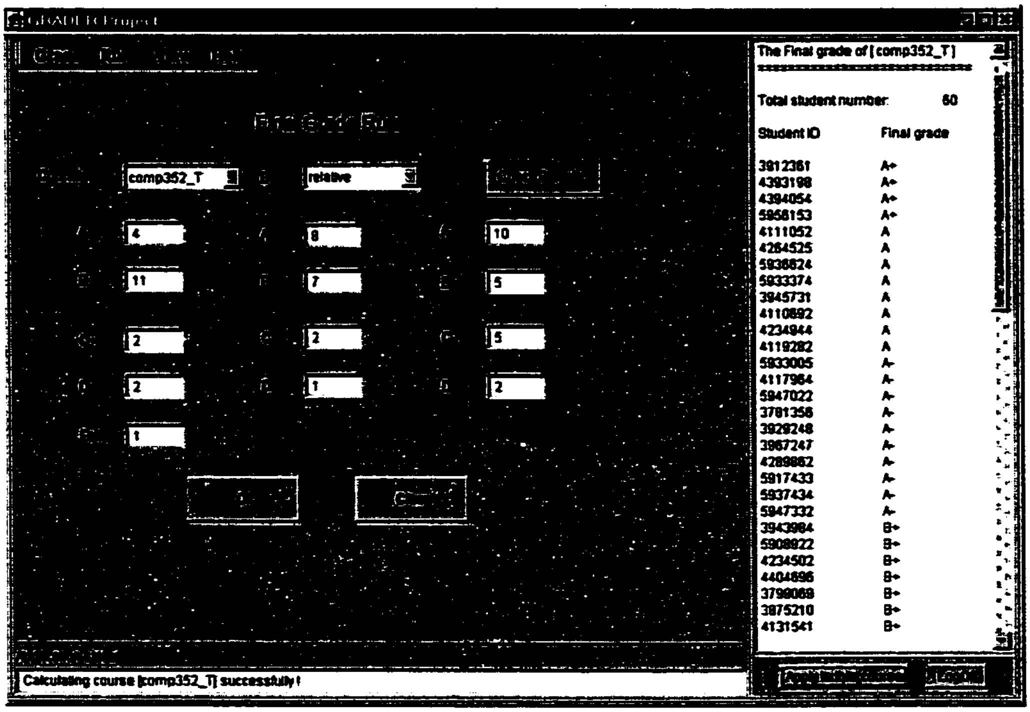
The teacher group GUIs also provides course grades automatically calculation function to easy grade management by using "Define full mark", "Define weight rule" and "Define final rule" menu items. The teacher needs to define each course full mark standard and weight rule for course items like assignments, labs, midterm exam and final exam. In the module "Define final rule", two rules based methods are provided to let the teacher calculate course final grade letter either by absolute or relative. This gives flexibility for the teacher to choose an appropriate way to calculate the course grade. The teacher can still manually modify the final grade result after system calculation by finding the specific entry and modify it using the "Modify/Delete grade" module. Command buttons "Apply this weight rule" and "Apply to this course" are used to activate the calculation process for course final mark and final grade. Figure 19 shows this group GUIs. GRADER imposes a strict data validation in these GUI to decrease the input error as less as possible. For example, the teacher cannot give a weigh rule which has total value greater than 100, or Final rule must has incremental number from low to high not intermingled letter in "absolute" case and can't allocate more than total students number in the total grade number allocation. GRADER can also detect if the user inputs a digital number as well to make a sure the GUI has right number to calculate.



comp352_T

```
The Final mark of [ comp352_T ]
==============================

Student ID          Final Mark

3912361             99.175
4393198             99.8
4394054             100.0
5958153             99.0
4111052             98.0
4264525             98.3
5938624             98.0
5933374             93.0
3945731             92.0
4110892             92.0
4234944             90.0
4119282             90.0
5933005             90.0
4117964             89.0
5947022             89.0
3781358             88.0
3929248             88.0
3967247             88.0
4289862             88.9
5917433             88.0
5937434             88.0
5947332             88.0
3943984             83.0
5908922             83.0
4234502             80.0
4404896             79.0
3798069             78.0
3875210             78.0
4131541             78.0
```

Calculating course [comp352_T] successfully!



comp352_T

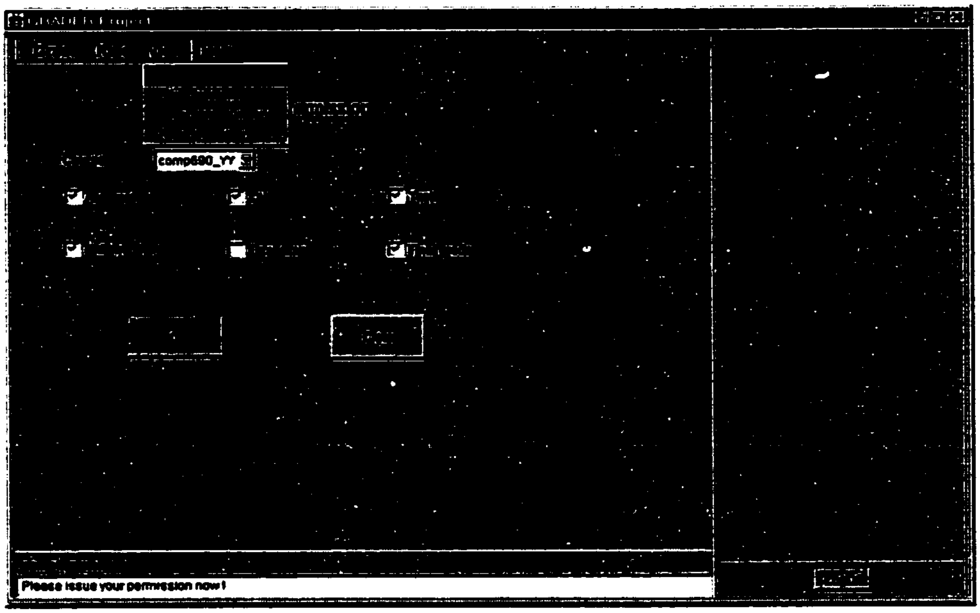
relative

```
The Final grade of [ comp352_T ]
================================

Total student number:      60

Student ID          Final grade

3912361             A+
4393198             A+
4394054             A+
5958153             A+
4111052             A
4264525             A
5936824             A
5933374             A
3945731             A
4110892             A
4234944             A
4119282             A
5933005             A-
4117964             A-
5947022             A-
3781358             A-
3929248             A-
3967247             A-
4289862             A-
5917433             A-
5937434             A-
5947332             A-
3943984             B+
5908922             B+
4234502             B+
4404896             B+
3798069             B+
3875210             B+
4131541             B+
```

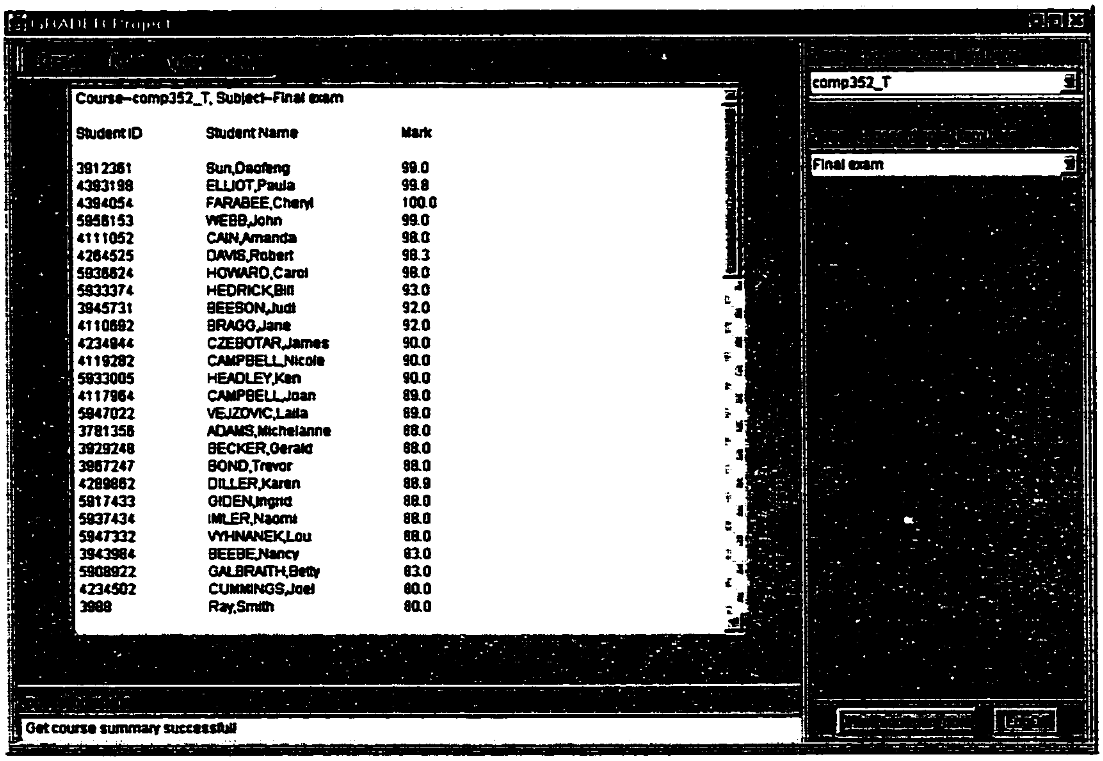
Calculating course [comp352_T] successfully!

Figure 19: The teacher group GUIs (2)

GRADER also provides course information viewing GUIs to let the teacher view detailed information for each course and bar chart graphics of grade statistics. The teacher can even define the student viewing items permission to selectively publish the course information to students. Teachers have the full control to hide some course item from students at their will. Following Figure20 shows the GUIs.



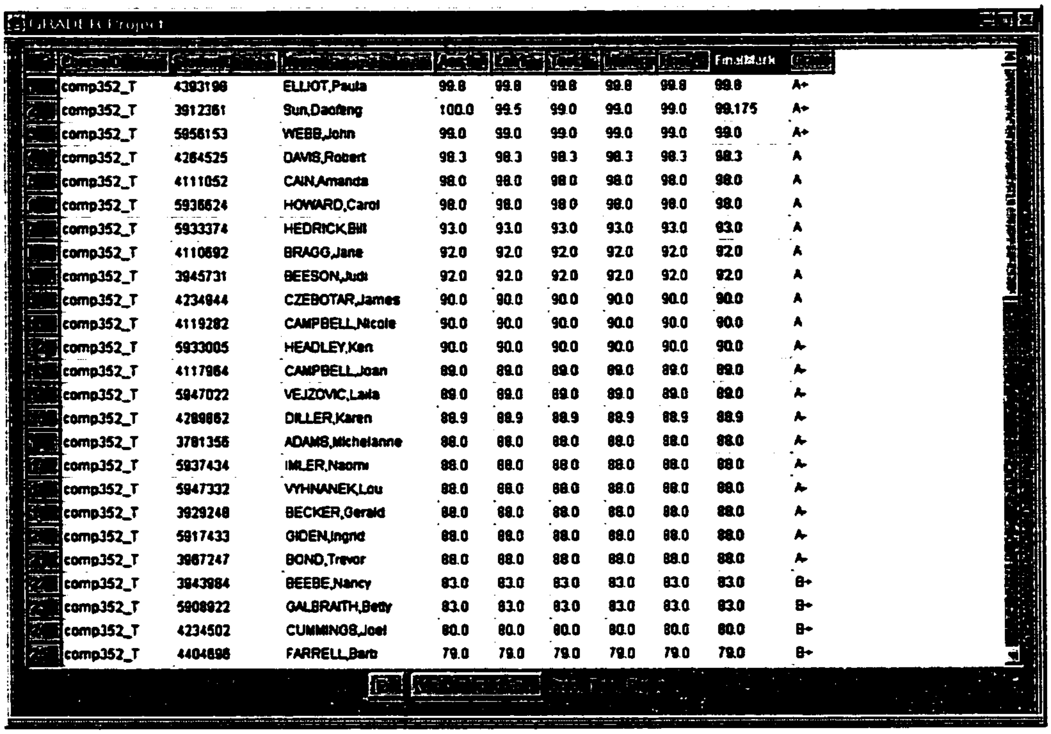
Please issue your permission now!



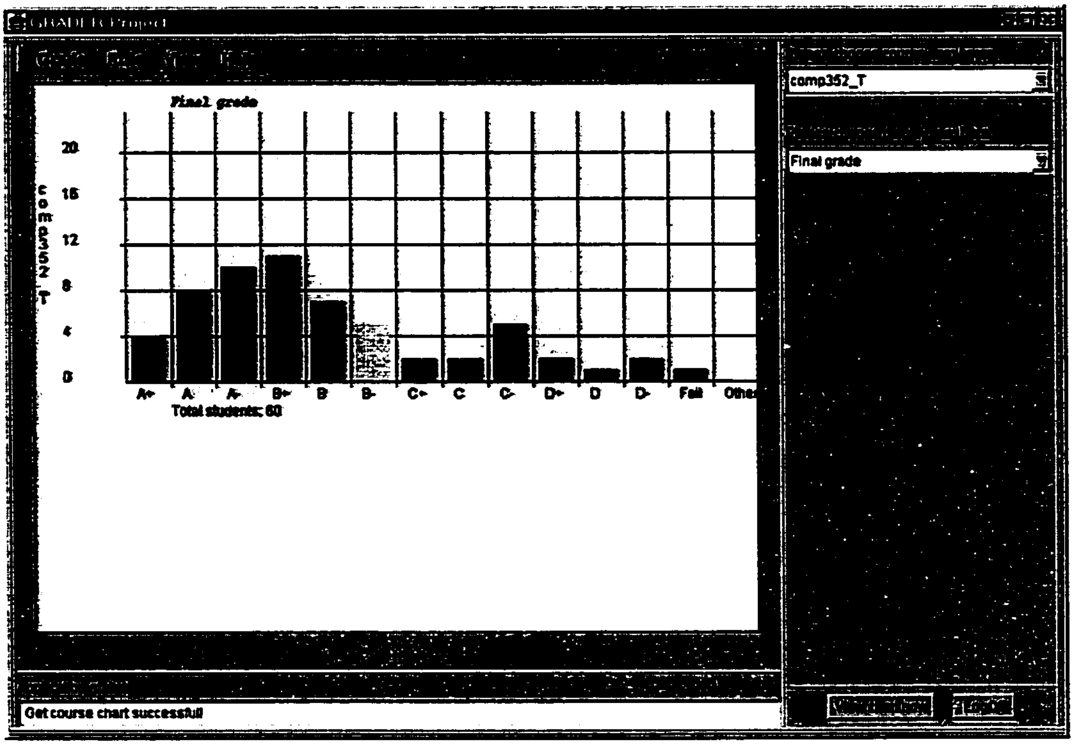
Course--comp352_T, Subject--Final exam

Student ID	Student Name	Mark
3812361	Sun,Daofeng	99.0
4383198	ELLIOT,Paula	99.8
4384054	FARABEE,Cheryl	100.0
5956153	WEBB,John	99.0
4111052	CAIN,Amanda	98.0
4264525	DAVIS,Robert	98.3
5936624	HOWARD,Carol	98.0
5933374	HEDRICK,Bill	93.0
3945731	BEESON,Judi	92.0
4110682	BRAGG,Jane	92.0
4234844	CZEBOTAR,James	90.0
4119282	CAMPBELL,Nicole	90.0
5833005	HEADLEY,Ken	90.0
4117864	CAMPBELL,Joan	89.0
5947022	VEJZOVIC,Laila	89.0
3781356	ADAMS,Michelanne	88.0
3929248	BECKER,Gerald	88.0
3867247	BOND,Trevor	88.0
4289862	DILLER,Karen	88.9
5817433	GIDEN,Ingrid	88.0
5837434	IMLER,Naomi	88.0
5947332	VYHNANEK,Lou	88.0
3943984	BEEBE,Nancy	83.0
5908922	GALBRAITH,Betty	83.0
4234502	CUMMINGS,Joel	80.0
3988	Ray,Smith	80.0

Get course summary successful!


GRADER Project

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	FinalMark	[illegible]
comp352_T	4393198	ELLIOT,Paula	99.8	99.8	99.8	99.8	99.8	99.8	A+
comp352_T	3912361	Sun,Daofeng	100.0	99.5	99.0	99.0	99.0	99.175	A+
comp352_T	5958153	WEBB,John	99.0	99.0	99.0	99.0	99.0	99.0	A+
comp352_T	4264525	DAVIS,Robert	98.3	98.3	98.3	98.3	98.3	98.3	A
comp352_T	4111052	CAIN,Amanda	98.0	98.0	98.0	98.0	98.0	98.0	A
comp352_T	5936624	HOWARD,Carol	98.0	98.0	98.0	98.0	98.0	98.0	A
comp352_T	5933374	HEDRICK,Bill	93.0	93.0	93.0	93.0	93.0	93.0	A
comp352_T	4110692	BRAGG,Jane	92.0	92.0	92.0	92.0	92.0	92.0	A
comp352_T	3945731	BEESON,Judi	92.0	92.0	92.0	92.0	92.0	92.0	A
comp352_T	4234844	CZEBOTAR,James	90.0	90.0	90.0	90.0	90.0	90.0	A
comp352_T	4119282	CAMPBELL,Nicole	90.0	90.0	90.0	90.0	90.0	90.0	A
comp352_T	5933005	HEADLEY,Ken	90.0	90.0	90.0	90.0	90.0	90.0	A-
comp352_T	4117964	CAMPBELL,Joan	89.0	89.0	89.0	89.0	89.0	89.0	A-
comp352_T	5947022	VEJZOVIC,Laila	89.0	89.0	89.0	89.0	89.0	89.0	A-
comp352_T	4289862	DILLER,Karen	88.9	88.9	88.9	88.9	88.9	88.9	A-
comp352_T	3781356	ADAMS,Michelanne	88.0	88.0	88.0	88.0	88.0	88.0	A-
comp352_T	5937434	IMLER,Naomi	88.0	88.0	88.0	88.0	88.0	88.0	A-
comp352_T	5947332	VYHNANEK,Lou	88.0	88.0	88.0	88.0	88.0	88.0	A-
comp352_T	3929248	BECKER,Gerald	88.0	88.0	88.0	88.0	88.0	88.0	A-
comp352_T	5917433	GIDEN,Ingrid	88.0	88.0	88.0	88.0	88.0	88.0	A-
comp352_T	3967247	BOND,Trevor	88.0	88.0	88.0	88.0	88.0	88.0	A-
comp352_T	3943984	BEEBE,Nancy	83.0	83.0	83.0	83.0	83.0	83.0	B+
comp352_T	5908922	GALBRAITH,Betty	83.0	83.0	83.0	83.0	83.0	83.0	B+
comp352_T	4234502	CUMMINGS,Joel	80.0	80.0	80.0	80.0	80.0	80.0	B+
comp352_T	4404698	FARRELL,Barb	79.0	79.0	79.0	79.0	79.0	79.0	B+


GRADER Project

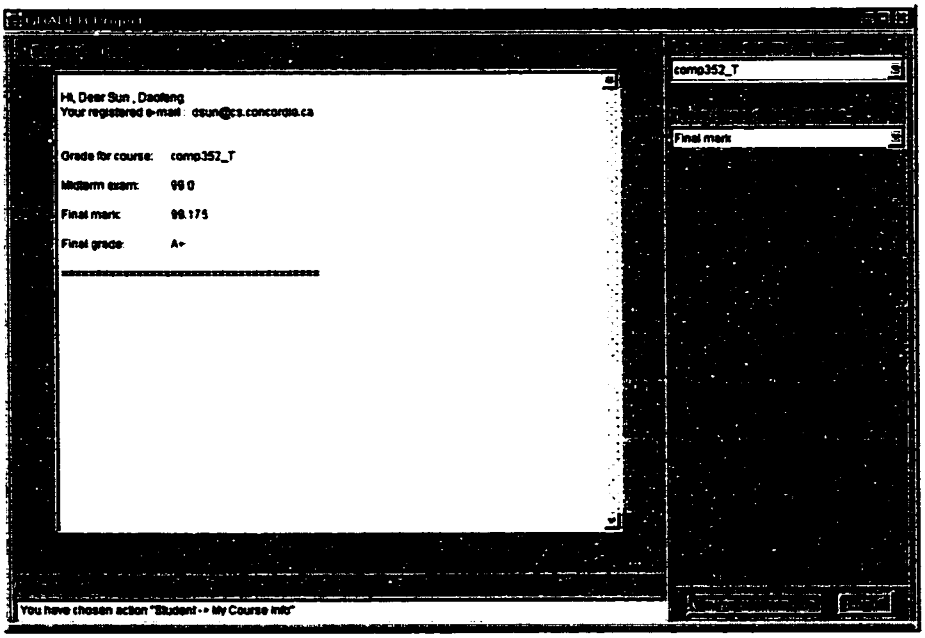
comp352_T

Final grade

Get course chart successfull

Figure 20: The teacher group GUIs (3)

Student group GUIs are pretty straightforward for proving viewing information of the courses student registered. Students can view their courses grade detailed information as well as the class grade distribution bar chart. Students can also modify their personal information like their email address or password as the password produced by system as soon as they get the GRADER registration email notification. The system validation can guide the user to use the correct length password and refuse to take not consistent password. Figure 21 shows this group GUIs.



Hi, Dear Sun , Daofeng
Your registered e-mail : dsun@cs.concordia.ca

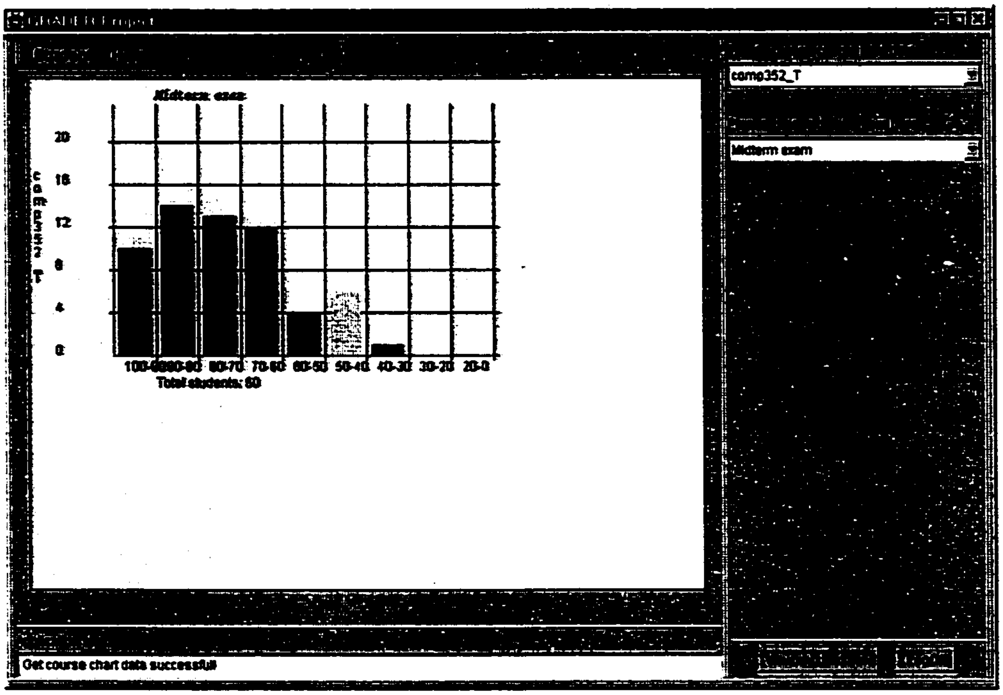
Grade for course: comp352_T

Midterm exam: 99.0

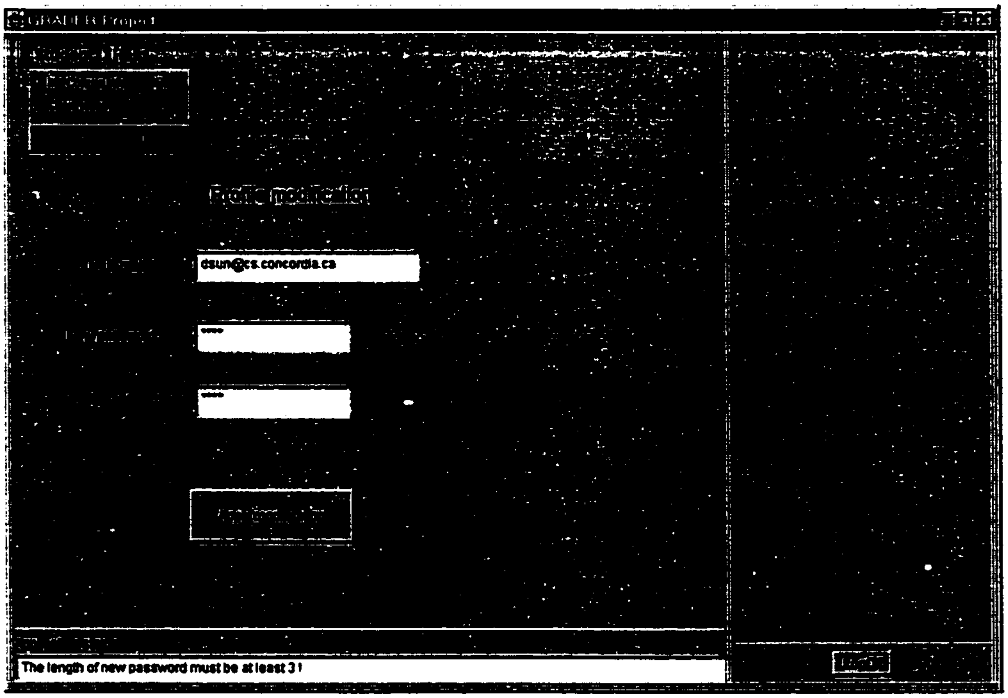
Final mark: 98.175

Final grade: A+

You have chosen action "Student -> My Course Info"



Get course chart data successfull



dsun@cs.concordia.ca

****

****

The length of new password must be at least 3 !

Figure 21: The student group GUIs

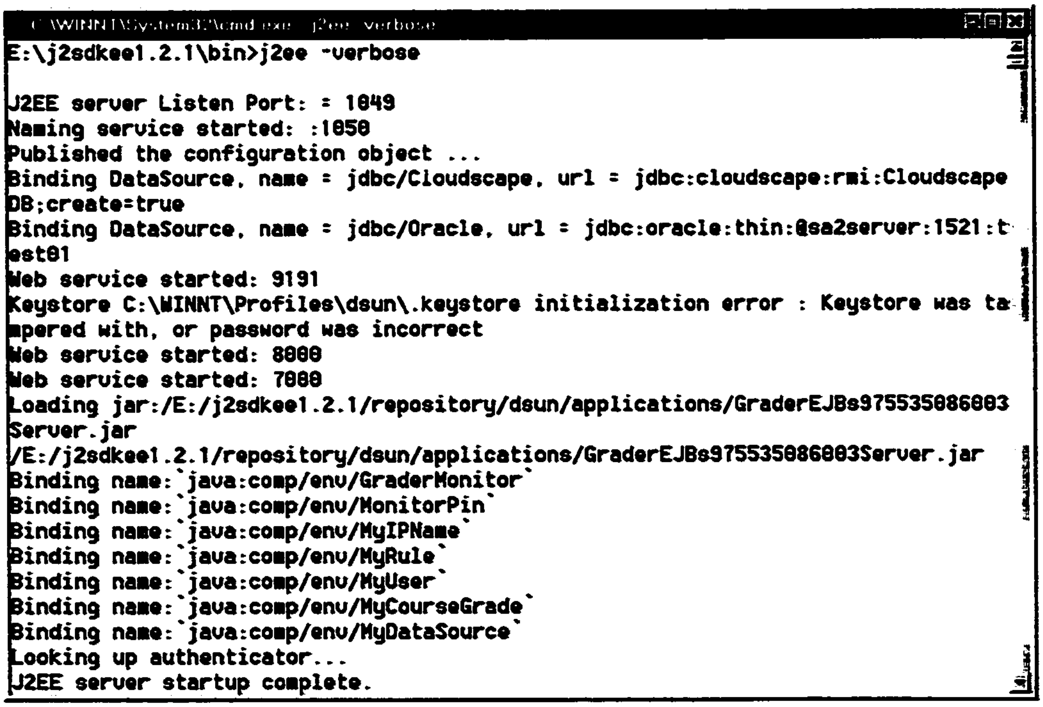
In any case, each group member can use the "Help" menu to get that group operation on line help by choosing the "Content" menu item. This help context will give the user detailed instruction about the usage of the GRADER system.

4.5.3 EJB DEPLOYMENT PROCESS

In order to deploy the GRADER distributed system, the following software is needed installed in the deployed computer.

- JDK 2 standard edition [4] version 1.2.1 or later
- J2EE application server [2] version 1.2.1 or later
- Oracle database or other relational database if not use default CloudScape database

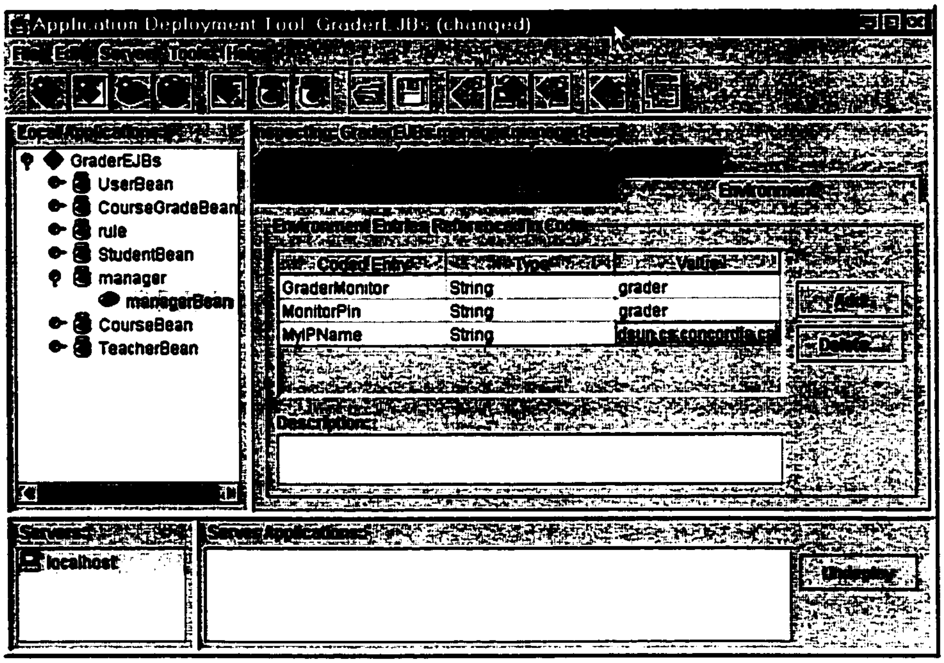
Define your JAVA_HOME system variable to point the JDK2 installation folder and another system variable J2EE_HOME to point the J2EE installation folder. Start the j2ee server by execute batch "j2ee.bat -verbose" file under %J2EE_HOME%\bin. After you start j2ee server, you need to start the CloudScape database server using command "cloudscape.bat -start" or start a chosen other database server like Oracle8i by its related instructions. Now the servers are ready to store data or let you deploy the GRADER system. The j2ee start process is like figure 22.



```
E:\j2sdkee1.2.1\bin>j2ee -verbose

J2EE server Listen Port: = 1049
Naming service started: :1050
Published the configuration object ...
Binding DataSource, name = jdbc/Cloudscape, url = jdbc:cloudscape:rmi:Cloudscape
DB;create=true
Binding DataSource, name = jdbc/Oracle, url = jdbc:oracle:thin:@sa2server:1521:t
est01
Web service started: 9191
Keystore C:\WINNT\Profiles\dsun\.keystore initialization error : Keystore was ta
mpered with, or password was incorrect
Web service started: 8000
Web service started: 7000
Loading jar:/E:/j2sdkee1.2.1/repository/dsun/applications/GraderEJBs975535086003
Server.jar
/E:/j2sdkee1.2.1/repository/dsun/applications/GraderEJBs975535086003Server.jar
Binding name:`java:comp/env/GraderMonitor`
Binding name:`java:comp/env/MonitorPin`
Binding name:`java:comp/env/MyIPName`
Binding name:`java:comp/env/MyRule`
Binding name:`java:comp/env/MyUser`
Binding name:`java:comp/env/MyCourseGrade`
Binding name:`java:comp/env/MyDataSource`
Looking up authenticator...
J2EE server startup complete.
```

Figure 22: J2EE server start process

Open j2ee deploy tool by execute batch file “deploytool.bat” in the same folder as “j2ee.bat”. You can then choose menu item “Open Application” under “File” menu to open GRDAER system EJB ear file “GraderEJBs.ear” and highlight bean “manager” -> “managerBean”. By choosing the right panel “Environment” tab, you can define the GRADER system default monitor name and password and also j2ee server machine IP name. After this modification, you can deploy GRADER EJB beans by using “Deploy Application” menu item under “Tools” menu to deploy all the EJB beans to the machine which you just start the j2ee server. The deploy tool figure is as follows:



Coded Entry	Type	Value
GraderMonitor	String	grader
MonitorPin	String	grader
MyIPName	String	dsun.cs.concordia.ca

Figure 23: **J2EE deployment tool snapshot**

4.5.4 HOW TO RUN THE GRADER SYSTEM

We just described the Grader Distributed EJB Beans deployment steps above, now let us put all things together to see how to run the whole system both on the server side or client side.

Server side:

- Install J2EE server and copy "ioserl2.dll" file under J2EE_HOME\nativelib folder to any search path folder (like c:\winnt\system32).
- Make sure you have set up JAVA_HOME and J2EE_HOME environment variables with their installation folders name. Modify the "orb.properties" file under J2EE_HOME/ config to set the host name as the current deployed machine name.
- Sign the file "grader.jar" inside the GRADER deployment zip file "grader_project.zip" and exports signer's certification file to clients who will access GRADER system. Or just simply unzip "grader.cer" file from "grader_project.zip" and let client import this certification file.
- Extract the GRADER deployment zip file "grader_project.zip" to folder under J2EE_HOME\ public_html. All classes archive signed jar files "sGraderAll.jar" and the index.html and grader.html files will go their right folder. Now the deployment is ready. Use any text editor to open "grader.html" file and modify applet parameter host value to be your deployed server IP.
- Run the j2ee server and cloudscape database server (or other database server) as described 3.5.3.

Server now is ready for client request!

Client side:

- The client side needs to have the JRE run time environment (Java 2 plug-in) installed. Copy the j2ee.jar file and two cloudscape jar files or oracle driver jar files into JRE_Home\JRE\1.2\lib\ext folder

- Copy J2EE native lib “ioser12.dll” file to any user system search path or modify system environment variable to let PATH point to the folder containing this “ioser12.dll” file
- Import signer’s certification file “grader.cer” to generate a user keystore and modify client side policy file as described in 4.3
- You can optionally enable client side Java 1.2 Console to let user see running results.

The client browser can now point URL to http://serverIP:8000 to enter The GRADER system now.

5 GRADER Security Control

5.1 Applet sandbox restriction

An applet being a Java program that is run from inside a Web browser, is subjected to some security restrictions. The original Java applet security model implements a sandbox that imposes strict controls on what certain kinds of Java programs can and cannot do. In general, applets loaded over the net are prevented from reading and writing files on the client file system, and only from making network to the applet-originating host [24]. By default all downloaded applets are considered untrusted, which means that all these applets are only permitted to run in client side binding applet sandbox security model and can't access local resources. Hence, they are of limited functionality if the application is expected to have complex ability.

All Netscape Navigator versions subsequent to 3.0 are Java enabled. All Microsoft Internet Explorer versions subsequent to 3.0 also include Java. The two browsers' security [16] policies are, at the present time, very similar. Both are somewhat strict. The following rules apply to all untrusted applets running under Netscape Navigator and Internet Explorer [24]:

- Applets can't read or write files at client local disk.
- Applets can't open a client-side network connection to any machine other than the applet's origin host.
- Applets after JDK 1.1 can open a server socket as long as the port number is greater than the privileged port number on the machine (usually 1024).
- Applets can read only nine system properties. Applets are not permitted to read any other system properties.

- If an applet is loaded using the file: URL, and it does not reside in a directory in CLASSPATH, it will be loaded by an Applet Class Loader.

Both browsers offer complex security models based on digital signatures and partial trust. Although the models are quite similar to the model defined by Java 2, there are many detailed differences that annoy developers and users. To solve this problem we have to make a trusted applet to break the applet sandbox restriction. There are two ways for an applet to be considered trusted, the applet is installed on the local hard disk, or the applet is signed by an identity marked as trusted in your identity database. GRADER is expected to present its complex front-end functionality like local disk file access and read some system properties, so we need to make it a trusted applet to take full power of Java ability.

5.2 Make trusted GRADER Applet

GRADER is a distributed system and has to use J2EE platform to get the EJB capability. We need to use The Java1.2 plug-in to deploy the system and we can naturally use the Java1.2 security model to leverage the GRADER full applet ability. The Java 1.2 security model is policy based and it will force the Java application abide the policy file definition. We think GRADER is trusted Java application, and give GRADER all the needed permission to perform its tasks.

Permission represents access to a system resource. In order for a resource access to be allowed for an applet (or an application running with a security manager), the corresponding permission must be explicitly granted to the code attempting the access. A Policy file represents the policy for Java application resource accessibility. The default

Policy implementation obtains its information from static ASCII policy configuration files. A policy file can be composed via a simple text editor, or via the graphical Policy Tool utility ***policytool*** in JDK. The user policy file ".java.policy" is by default located at:

C:\Winnt\Profiles\login_name on multi-user Windows NT systems
C:\Windows\Profiles\login_name on multi-user Windows 95 systems
C:\Windows on single-user Windows 95 systems
user.home/.java.policy (Solaris)

When the Policy is initialized, the system policy is loaded in first, and then the user policy is added to it. If neither policy is present, a built-in policy is used. This built-in policy is the same as the original sandbox policy. Policy file locations are specified in the security properties file, which is located at

java.home/lib/security/java.security (Solaris)
java.home\lib\security\java.security (Windows)

Using the Policy Tool saves typing and eliminates the need for you to know the required syntax of policy files, thus reducing errors. There is a tool named ***keytool*** from JDK that can be used to create public/private key pairs and self-signed X.509 v1 certificates [24], and to manage keystores. A keystore is a protected database that holds keys and certificates for an enterprise. Access to a keystore is guarded by a password (defined at the time the keystore is created, by the person who creates the keystore, and changeable only when providing the current password). In addition, each private key in a keystore can be guarded by its own password. Another tool ***jarsigner*** are used to digitally sign Java applications or applets using produced keys and certificates from keystores. Using keytool, it is possible to display, import, and export X.509 v1, v2, and v3 certificates stored as files, and to generate new self-signed v1 certificates. By using these tools JDK 1.2 provides the basic technology for loading and authenticating signed classes. This enables browsers to run trusted applets in a trusted environment. The GRADER is using these tools for finer-grained policy control to get flexible security policies. We will show the detail steps to demonstrate how to apply this.

5.3 How to sign GRAGER

In order to deploy GRADER and let the client browser executed as an applet client, we need to sign the class archive file and export the signed certification file to the client side. The Client then imports this certification and adds a new Java security policy entry to the client policy file. The steps are as follows:

Server side

- Step 1: ***Produce owner public key***

```
JDK_HOME/bin>keytool -genkey -keystore C:\WINNT\dsun\.keystore -alias dsun
Enter keystore password:  grader
What is your first and last name?
  [Unknown]:  daofeng sun
What is the name of your organizational unit?
  [Unknown]:  concordia
What is the name of your organization?
  [Unknown]:  concordia
What is the name of your City or Locality?
  [Unknown]:  montreal
What is the name of your State or Province?
  [Unknown]:  quebec
What is the two-letter country code for this unit?
```

```
 [Unknown]: ca
Is <CN=daofeng sun, OU=MRI, O=MRI, L=montreal, ST=quebec, C=ca> correct?
 [no]: y
Enter key password for <dsun>    (RETURN if same as keystore password):[return]
```

- Step2: ***Use above public key to sign a jar file***

  ```
  JDK_HOME/bin >jarsigner -keystore C:\WINNT\Profiles\dsun\.keystore -signedjar sGraderAll.jar GraderAll.jar dsun
  Enter Passphrase for keystore: grader
  ```

- Step 3: ***Export the Public Key Certificate file and signed jar file***

  ```
  JDK_HOME/bin >keytool -export -keystore C:\WINNT\Profiles\dsun\.keystore -alias dsun -file grader.cer
  Enter keystore password: grader
  Certificate stored in file < grader.cer>
  ```

<u>Client side</u>

- Step 4: Import signer's certification file to produce client's keystore file

  ```
  JDK_HOME/bin >keytool -import -alias dsun -file grader.cer -keystore
  c:\userStore
  ```

Step5: Modify java policy file

Use java policy tool ***policytool*** to create or modify existing ".java.policy" file in the folder of client profile folder(i.e C:\WINNT\Profiles\dsun\). Eventually the following two lines should be in ".java.policy" file.

```
keystore "C:/ userStore"
grant signedBy "dsun" {permission java.security.AllPermission ;};
```

Now the signed GRADER applet can break the applet restriction and access EJBs and local machine resources like the read/write file.

6 Conclusion

GRADER is implemented as a multi-tier component-based distributed system and uses EJB technology to demonstrate the significant benefits of server-side software components: business logic focus, portable deployment, and easy maintenance; reusable building blocks for rapidly developing and extending Web-based applications; widely robust portable applications across heterogeneous operating system, platform independency, middleware, and protection of user access, as well as standard JDBC database connection to embrace legacy EIS systems.

The J2EE architecture and EJB specification make EJB components and component-based distributed applications available and superior to other distributed systems like DNA or CORBA. This wide support of the EJB technology offers flexibility and a clear Model/View/ Control design paradigm and object oriented implementation. Each tier has specific responsibility to provide service or request service to other tiers. The Java J2EE platform architecture also offers two other choices for building the middle-tier servers for Web-based applications: JavaServer Pages, Servlets, which we have not yet explored yet in GRADER.

By using applet to be a client of GRADER system, we are taking the benefit of rich Java language power, which provides GRADER complex user interfaces and more functionalities than using JSP or JavaScript technologies dynamic HTML pages.

The restrictions of the EJB component distributed-based system are:

- All the entity beans properties have to be defined to be public which violate the object encapsulation rule.

- EJBs use network to communicate each other and increase the overhead of the system and cost some system performance.
- Container-managed persistence is only single-table based, and has to use bean- managed persistence if the application involves complex objects.

The applet client solution raises the complexity of the application deployment and the browser has to download big jar file from server before GRADER can run on the client side. A signed applet has to be used to release the applet security restriction. It also may have a firewall tunnel problem if the network proxy is not well defined.

Despite of the above limitations, GRADER demonstrates a well-defined Object-oriented design and implementation process. GRADER provides the end user a friendly and powerful GUIs layout and interactivity. It also shows a practical realization of component-based distributed system using EJB technology, which could become the principal component models of the application.

GRADER performance can be enhanced in the middle-tier by using cached beans objects and using bulk access objects model. These improvements should be addressed in the further work [27].

References

1. Clemens Szyperski, “Component Software”, ISBN 0201178885 , Addison-Wesley

2. Designing Enterprise Applications with Java 2 Platform, Enterprise Edition and Java 2 Platform, Enterprise Edition specification, v1.2 http://java.sun.com/j2ee/

3. "Enterprise JavaBeans™: Server Component Model for Java "http://www.javasoft.com/products/ejb/white_paper.html

4. Enterprise JavaBeans Specifications v1.1 Sun Microsystems: http://java.sun.com/products/ejb/

5. Sun J2EE Bulletins http://java.sun.com/j2ee/bulletin.html

6. Server side portal: http://theserverside.com/home

7. Ed Roman, "Mastering Enterprise JavaBeans, and the Java 2 Platform, Enterprise Edition" published by John Wiley & Sons, 1999

8. Ash Rofail ;Tony Martin, “Building N-Tier Applications with COM and Visual Basic 6.0” ,Wiley Computer Publishing, John Wiley & Sons, Inc. ISBN: 0471295493

9. CORBA Specification, Version 2.3.1, OMG http://www.omg.org/

10. The Simple Object Access Protocol (SOAP), Working Draft, W3C http://www.w3.org/TR/SOAP

11. The XML RPC Specification, UserLand Software. http://www.xml-rpc.com/spec

12. The Battle For The Middle Tier http://www.objectwatch.com/issue12.htm

13. For a complete listing of JavaWorld articles covering EJB: http://www.javaworld.com/javaworld/topicalindex/jw-ti-ejb.html

14. A set of related patterns about scripted components and scripting languages is online at http://www-dse.doc.ic.ac.uk/~np2/patterns/scripting

15. Ralph Johnson and Brian Foote, "Designing Reusable Classes", Journal of Object-Oriented Programming, http://www.laputan.org/drc/drc.htm

16. For a complete listing of JavaWorld articles covering EJB:
http://www.javaworld.com/javaworld/topicalindex/jw-ti-ejb.html

17. Avalon --an up-and-coming open source server-side Java framework:
http://java.apache.org/framework/

18. BEA WebLogic --Popular J2EE Server http://www.weblogic.com

19. Design patterns: http://hillside.net/patterns/patterns.html

20. Boss -- an open source application server http://www.jboss.org

21. Al Williams, Kim Barber, and Paul Newkirk Coriolis, "Active Server Pages Black Book" The Coriolis Group ISBN: 1576102475

22. Frank E. Redmond , "DCOM: Microsoft Distributed Component Object Model" IDG Books Worldwide, Inc. ISBN: 0764580442

23. Microsoft DNA architecture http://www.microsoft.com/dna

24. Gary McGraw, Edward W. Felten "Securing Java" 2nd Edition by Wiley, ISBN: 047131952X

25. Michael Hicks, Suresh Jagannathan, Richard Kelsey, Jonathan T. Moore and Cristian Ungureanu. "Transparent communication for distributed objects in Java" pp. 160-170 Proceedings of the ACM 1999 conference on Java Grande June 12 - 14, 1999,

26. Matthew Izatt, Patrick Chan and Tim Brecht."Ajents: towards an environment for parallel, distributed and mobile Java applications", Proceedings of the ACM 1999 conference on Java Grande, June 12 - 14, 1999, pp. 15 - 24

27. Vlada Matena. "Advanced Programming with EJB Technology",JavaOne website http://industry.java.sun.com/javaone/99/tracks/

28. Nick Kassem. "The Application Programming Model for the Java 2 Platform, Enterprise Edition",,JavaOne website http://industry.java.sun.com/javaone/99/tracks/

29. Vlada Matena,Rahul Sharma."The Enterprise JavaBeans Component Architecture Specification - Overview of Architecture and Roadmap".

30. Mark Hapner,Bill Shannon. "The Java 2 Platform, Enterprise Edition"

Appendix A: GRADER User Manual

Log in to the system:

After you enter the URL to point to Grader (Java Applet client/EJB server), you can put either your ID or your login name in "Login ID" field and password in "Password" field , then press "Ok" button, If you want type again due to an error, just press the "Cancel" button and then type again. The system will display waiting information at the bottom of the window, and lead you to the next main menu shortly, given the correct pair of login ID/password.

Monitor user group:

If the user login as a monitor group member, the system will lead to the monitor GUI after the user successfully logs in to the system. The first page is the "Register New User" page, which can be used to create a new user in either group from Monitor/Teacher/Student.

Menu instruction:

[ADMINISTRATION MENU]

Add user-- same as the first entry page of monitor login. This GUI can let the current user add a new user of any type of Monitor/Teacher/Student. You can change to "Modify/Delete User" page by clicking the right bottom button "Modify/Delete User" or button "Log Off" to exit.

Modify/Delete User-- This leads to the "Modify Existing User" page let current user to modify an existing user profile as well as delete a user from database. Four navigation buttons are provided for easy browsing through the all user list from database or by type a specific user ID in the upper field, and then click "Look-up user" button to quickly retrieve user's information if this user exist. By clicking "Add New User" button in right bottom, that will lead you back to the "Register New user" page.

Add course-- This leads to page "Register New Course" which lets the current user add a new course to the system. By typing course information in corresponding fields, then press "Ok" button to save it to the database. CourseID and Teacher ID fields must be filled to proceed. By clicking the "Modify/Delete Course" right-bottom button that leads to another "Modify/Delete course" page.

Modify/Delete Course-- This leads to the "Modify Course page", which is used to modify or delete or just browse course entries in the database. Groups of navigation buttons are provided to easily browse through the course information. By typing a specific course ID in the upper field, one can quickly get that course profile from database given it already exists in the database.

Batch add mode-- This is a GUI to let the monitor add many students to the system at once, A file needs to exist in the user's local file system or use "Edit this file" button to create a new file which follow the format the system requires. After you edit the file put the right bottom button named as "Process batch add" to register all students in your file. This GUI has a function to notify the monitor and the student with the registration information via e-mail from user's profile. To activate this function, the user needs to check the option "Email notification" on. If you do not want this function, just uncheck this option.

SQL tool-- This is a powerful GUI which lets the monitor connect to a remote or local database by a given driver and connection URL in the "Driver:" and "URL:" fields. The system provides a default string for these fields, which are the Oracle driver and a type 4 JDBC connections URL string. Just check the database vendor to know the strings and put them here to connect with the database server or the local database. After getting the connection, the monitor can use this GUI to perform any SQL query over the database and see the result in the text fields below. These query statements issued by the monitor are stored in the system and can be recalled by pressing the navigation group buttons “previous/next”. The status field displays the application status, and by clicking the button "Exit" to quit the SQL tool page.

Email Notification-- By checking this option the "Batch add user" page can activate the email notification function and send a summary report of finished registered users list to login the monitor and send to all students a separate email notification about their login name and password to let students have the access right to log to GRADER system as a student group member.

Teacher user group:

If the user logs in as a teacher group member, the system will lead to the teacher GUI after user successfully logs to the system. The first page is the "Course Grade Info" page, which can be used to create a new grade item in courses taught by the login teacher. The right panel will display all course registration information (student ID and name under each course).

Menu instruction:

[GRADE MENU]

Add student--This GUI can be used to let the teacher add a new student to Grader system. The teacher can change to the "Modify/Delete Student" page by clicking the right-bottom button "Modify/Delete Student" or button "Log Off" to exit. This GUI only establishes user profile and does not bind to any specific course. You still need to use the "Add Course grade" GUI to register student to a specific course.

Modify/Delete student-- This leads to the "Modify Existed Student" page, which lets the teacher modify an existing student profile as well as delete a student from the database. Four navigation buttons are provided for easy browsing of the entire student list from the database or by typing a specific student ID in the upper field and then click "Lookup student" button to quickly retrieve this student's information if the student exist. Clicking "Add New Student" button at the bottom will lead the teacher get back to the GUI "Register New Student" page.

Add Course grade-- This leads to same page as the teacher first logged into Grader, and lets the teacher add a new grade to system as well as register the student to this course. By choosing course ID from the list and typing the student ID in corresponding fields, then type grade fields and press "Add a Grade" button to save it to the database. Course ID and Student ID fields must be filled to proceed. By clicking "Modify/Delete grades" in right bottom button can lead to another "Modify/Delete grades" page. A monitor defines the courses of the course ID list and the teacher himself cannot change this item.

Modify/Delete grade-- This leads to the "Course Grade Info" page, but having the functionality to modify/delete/browse any grade records in the database. Groups of navigation buttons are provided for easy browsing of the grades information. By choosing a different course ID and typing specific student ID in the corresponding fields and then press button "Find a grade", you can quickly get that student's course grade information from the database given this student already registered to the chosen course.

Change any grade and then press button "Modify a grade" to modify this grade item in the database, pressing the "Delete" button will delete this grade item from the database, and you will need to rebuild this again by going to "Add Course Grade" GUI.

Batch register student-- This is a GUI to let the teacher register many students to the course at one time, as well as add these students to GRADER user profile system, A file needs to be exist in the teacher's login client side file system or use "Edit this file" button to create a new file locally, which follows the format specified by the system. After you edit the file, put the right-bottom button named as "Process batch add" to register all students in your file list. This GUI has a function to report to the teacher all students registration summary and notify the student with the registration information via e-mail from the user's profile. To activate this function the teacher needs to check the option "Email notification" on. If don't want this function, just uncheck this option (default status).

Batch add grade -- This is a GUI to let the teacher add whole class grade to the given course at one time including its full mark standard, A file needs to exist in the teacher's login client side file system, or use "Edit this file" button to create a new file locally, which follows the format specified by the system. After you edit the file put the right bottom button named as "Process batch grade" to add partial or whole class students grade into GRADER.

Email Notification-- By checking this option, the "Batch register student" page can be activated the email notification function and send a summary report of the finished registered students list to the logged on teacher, and send to all student a separate email notification about their login name and password to let students have the access right to log into GRADER as a student group member.

[RULE MENU]

Define full mark -- A GUI to be used for assign all assignment/lab/test/midterm/final exam full mark. These values will be used to calculate the course final mark; each section student grade is just relative to this full grade mark. Choose a course ID from the list, and then assign full marks to the missing fields and then press button "Add full mark" to save to the database. This GUI is the complementation to the "Batch add grade" GUI, which also add a full mark by each grade section. Here, you can add all full mark standards to all courses taught by the teacher.

Define weight rule-- This GUI can let a teacher define each grade percentage in the final mark. The total must be exactly 100 in order to pass the system's input validation. After weight rule assignment, press button "Add weight rule" to save it to the database and then you can press "Apply this weight rule" to calculate whole class final mark to the given course. If you miss some student grades, you just need to add these grades and then come to this GUI to press "Apply this weight rule" again to recalculate to refresh the final mark of the course.

Define final rule-- This is the GUI to allocate final the letter to the chosen course. You can first press button "Check Statistics" to get the whole course final mark and its distribution, and then by assigning the number to the course. Two methods are provided to let the teacher have the flexibility to get the course final letter. Relative is the way you just assign a letter number in each frame; the system will try to calculate the final grade relative to the schema. Using absolute way, you have to type all frame grade margins and follow the number order in order to pass the input validation. After you input type finished, first click button "Ok" to save this rule, and press button "Apply to this course" will calculate the course final grade and display the result in the right panel. You can immediately see the result and adjust again.

[VIEW MENU]

Define views rule--This GUI can be used to assign student permission to the course grade. The teacher can hide the grade he/she does not want the student to see.

Views grade table--This GUI can be used to present a grade detail grid view as a table and let the teacher sort the different field and choose a special entry to edit.

View grade summary--This GUI give full course grade summary of the chosen course and either assignment, lab, test, midterm exam, final exam, final mark and final grade. The summary includes max/min/mean grade and grade distribution in the class. This gives the teacher detailed information about his/her course. Just choose the course ID and item subject from the display list, and then press the "View summary now" button to get the result in the left panel.

View grade chart-- This is graphic bar chart display to the midterm/final exam/final mark/final grade. This GUI gives a more straightforward way to the teacher about the class grade. Just choose courseID from list and item subject from display list, and then press the "View chart now" button to get the bar chart in the left panel.

Student user group:

If a user logs in a student group member, the system will lead to the student GUI after the user successfully logs in to the system. The first page is all course grade summaries, which includes all courses registered by the current student.

Menu instruction:

[STUDENT MENU]

My course info-- This GUI is the same page as the student login to the Grader system. In the right panel, the student can choose a different courseID and course item, and then press button "View my summary now" to see chosen course detailed information including grade statistics. The course items are just items defined by the teacher permission GUI.

Chart view-- This is graphic bar chart display to the midterm/final exam/final mean/final grade. This GUI gives more straightforward way to the student about the class grade. Just choose the courseID from the list and item subject from display list, then press button "View chart now" to get the bar chart in the left panel. The course items are just items defined by the teacher permission GUI.

Change password-- This GUI is used to let the student change his/her personal profile like password and e-mail address.

[HELP MENU]

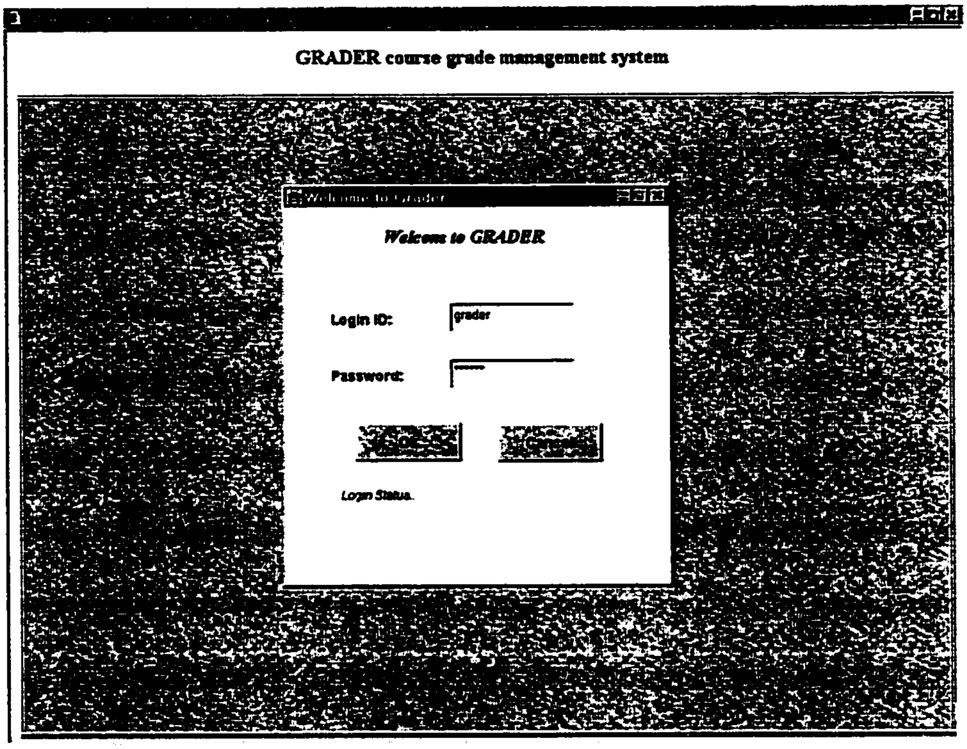
Content -- display correspond user group content text for help usage

About-- display Grader version information

Appendix B: GRADER walkthrough

Monitor:

- Login in as a monitor using the same entry GUI as other groups



GRADER course grade management system

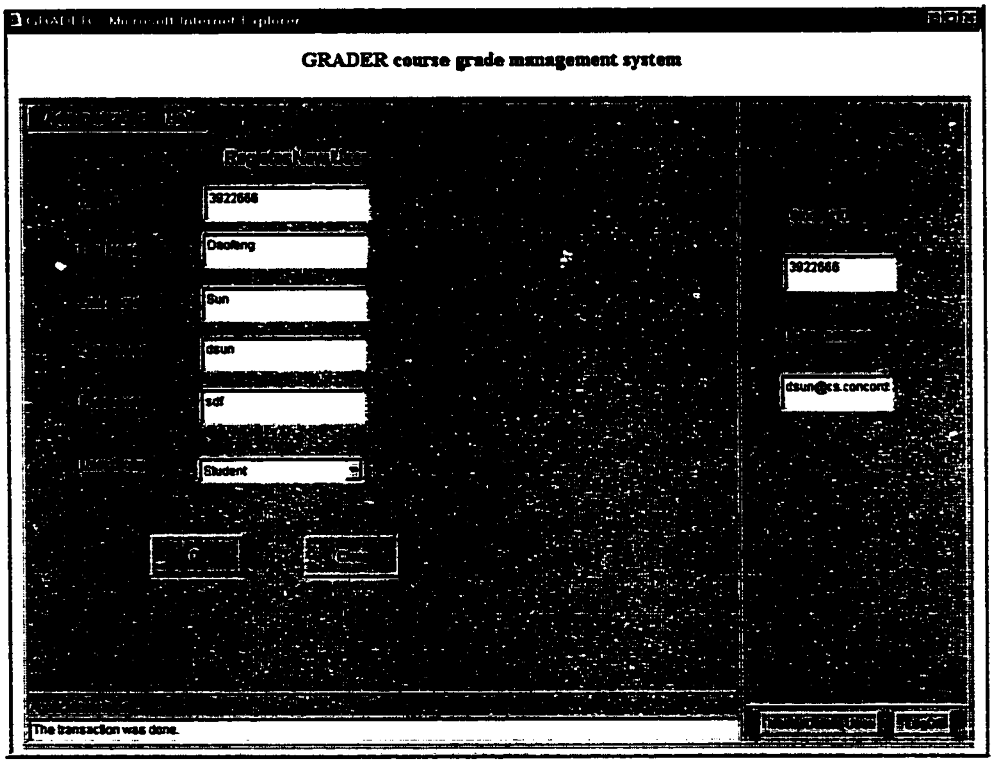
Welcome to GRADER

Login ID: grader

Password: ******

Login Status.

- Add new user (student group)



GRADER course grade management system

Register New User

3922666

Daofeng

Sun

dsun

sdf

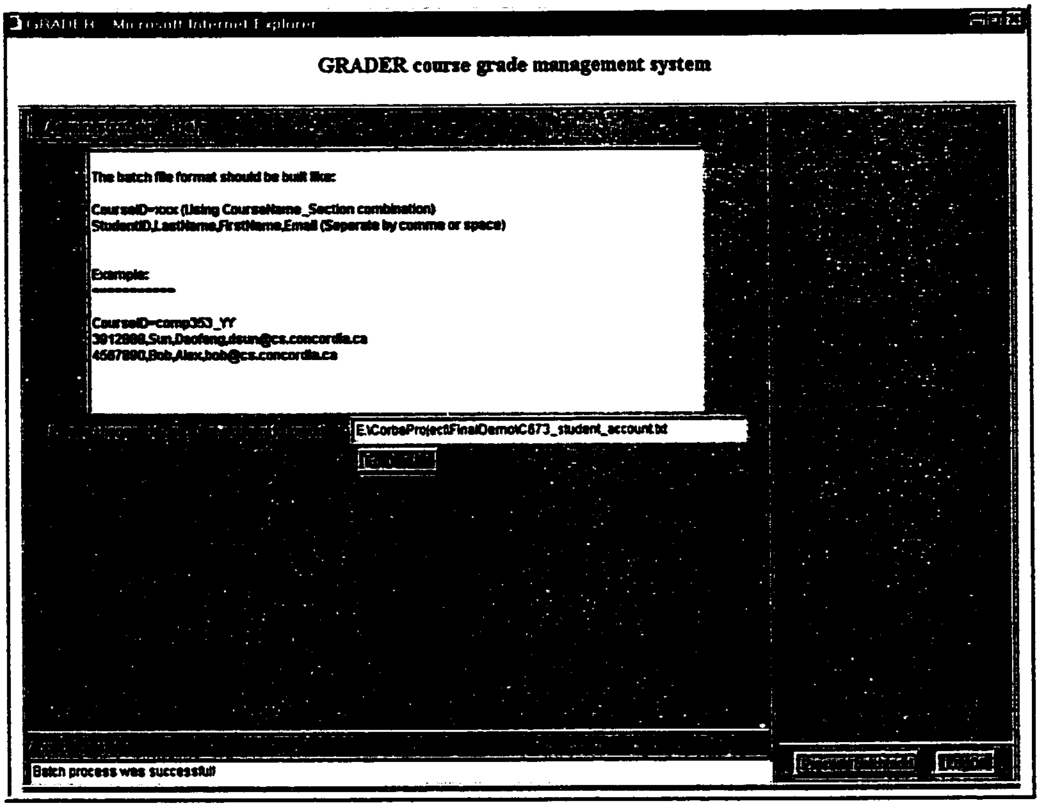
Student

3922666

dsun@cs.concord

The transaction was done.

- Add new students account in batch mode



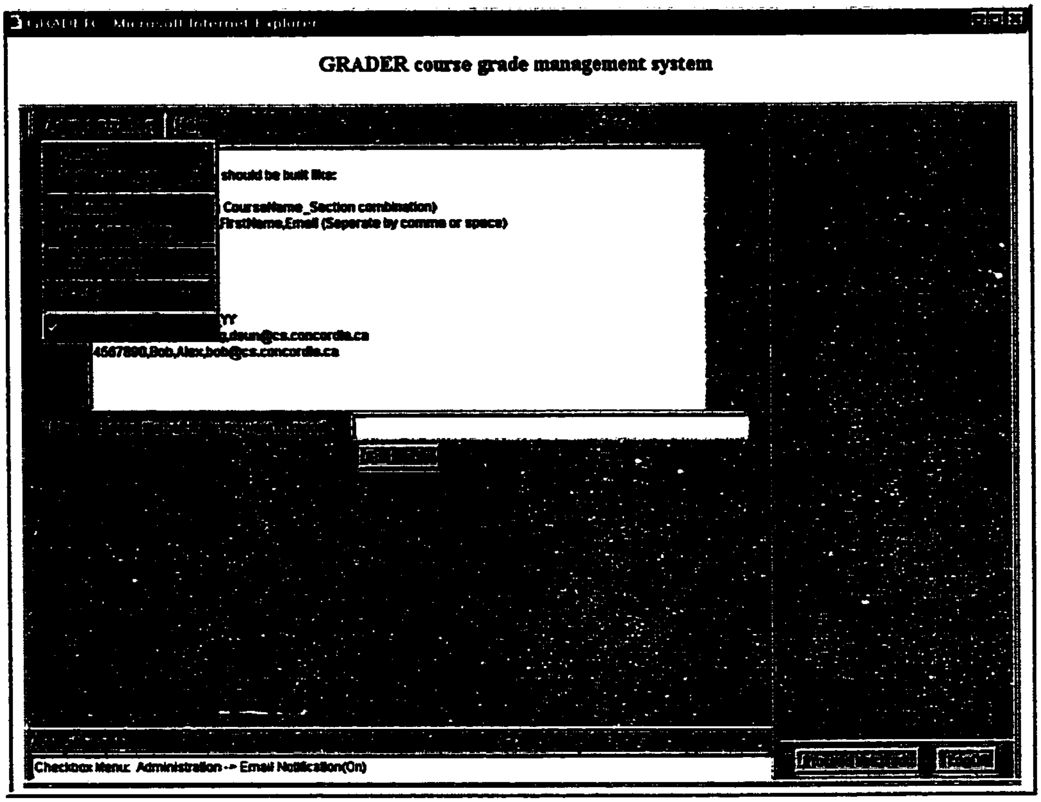
GRADER course grade management system

```
The batch file format should be built like:

CourseID=xxx (Using CourseName_Section combination)
StudentID,LastName,FirstName,Email (Seperate by comma or space)


Example:
-----------

CourseID=comp353_YY
3912888,Sun,Daofeng,dsun@cs.concordia.ca
4567890,Bob,Alex,bob@cs.concordia.ca
```

E:\CorbaProject\FinalDemo\C673_student_account.txt

Batch process was successful!

- Enable e-mail notification if you want to notify students of the registration status

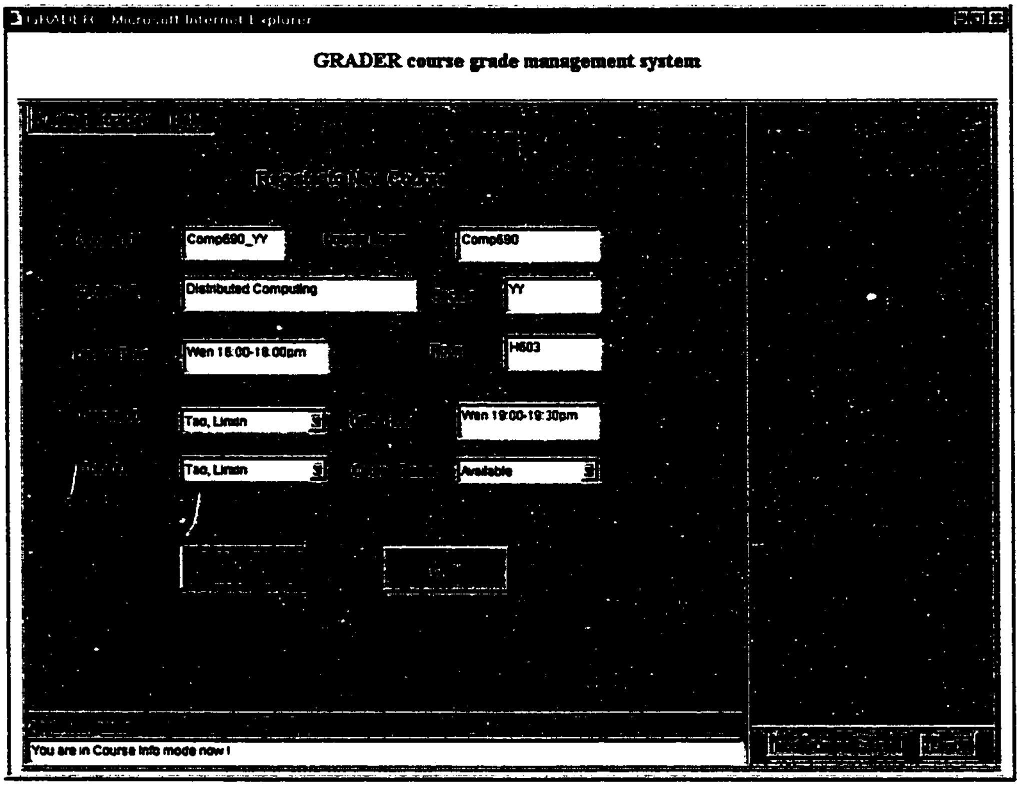


GRADER course grade management system

```
should be built like:

CourseName_Section combination)
FirstName,Email (Seperate by comma or space)


YY
g,dsun@cs.concordia.ca
4567890,Bob,Alex,bob@cs.concordia.ca
```

Checkbox Menu: Administration--> Email Notification(On)

- Register to a new course



GRADER course grade management system

Comp690_YY

Comp690

Distributed Computing

YY

Wen 16:00-18:00pm

H603

Tao, Linxin

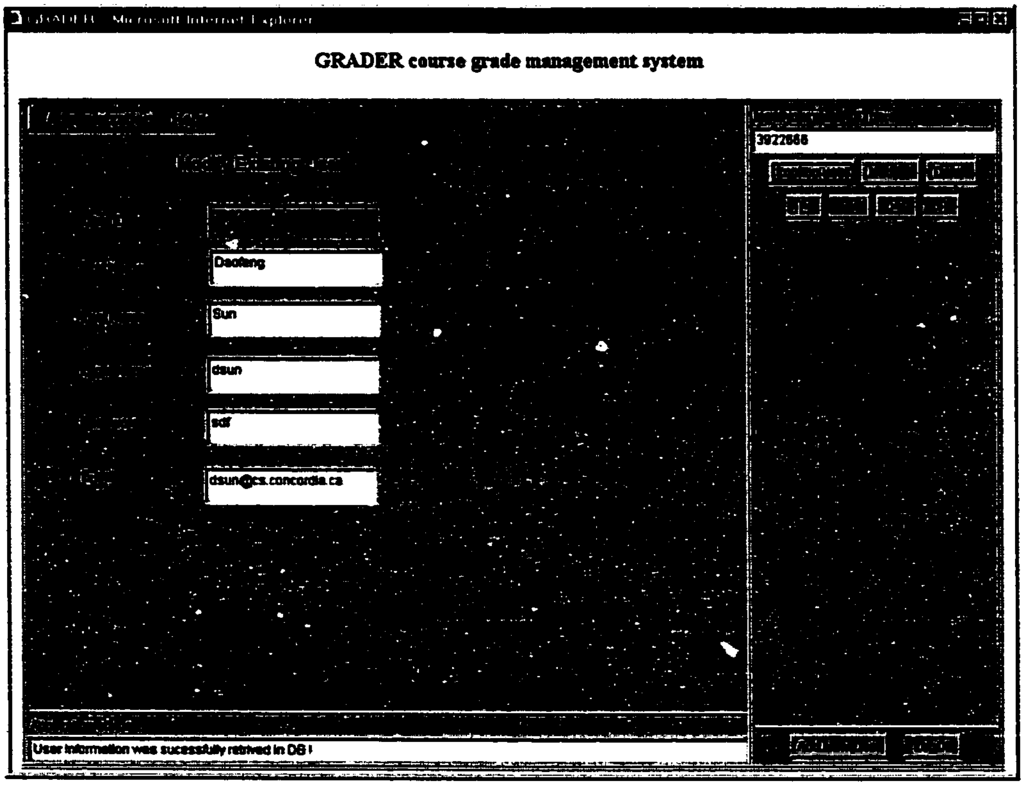
Wen 19:00-19:30pm

Tao, Linxin

Available

You are in Course Info mode now !

- Modify or look up an existing user



GRADER course grade management system

3922666

Daofang

Sun

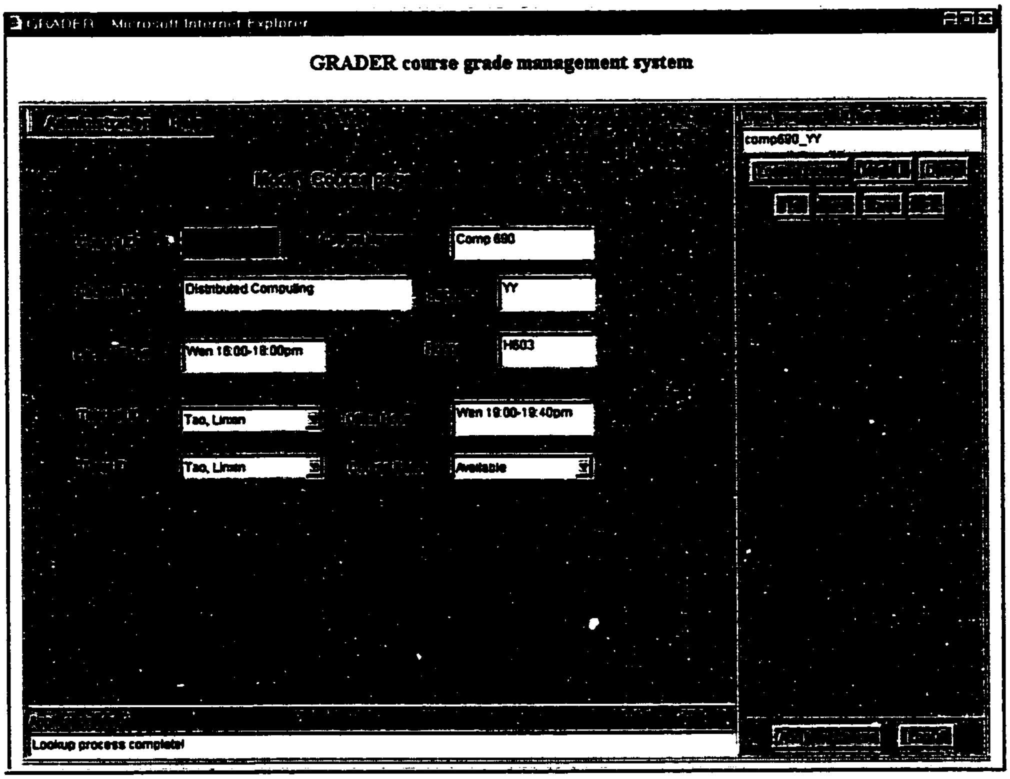
dsun

sdf

dsun@cs.concordia.ca

User information was sucessfully retrived in DB !

- Modify or look up an existing course



GRADER course grade management system

Comp 690

Distributed Computing

YY

Wen 18:00-18:00pm

H603

Tao, Liman

Wen 19:00-19:40pm

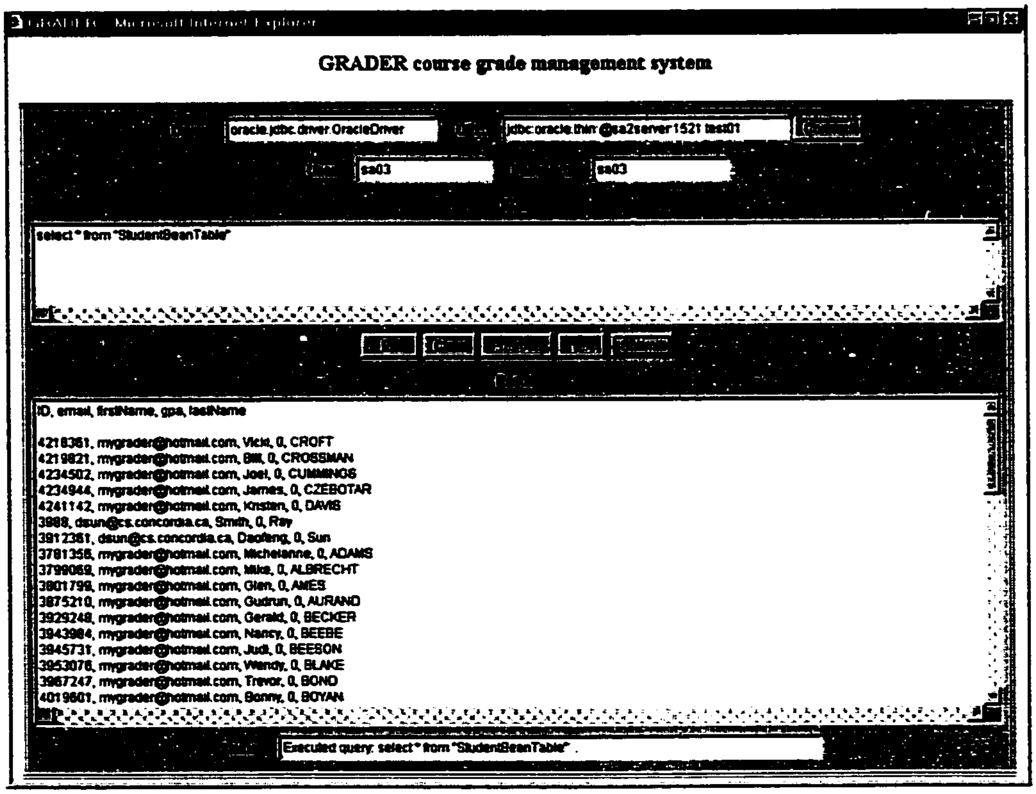
Tao, Liman

Available

comp690_YY

Lookup process complete!

- SQL tool usage



GRADER course grade management system

oracle.jdbc.driver.OracleDriver

jdbc:oracle:thin:@sa2server:1521:test01

sa03

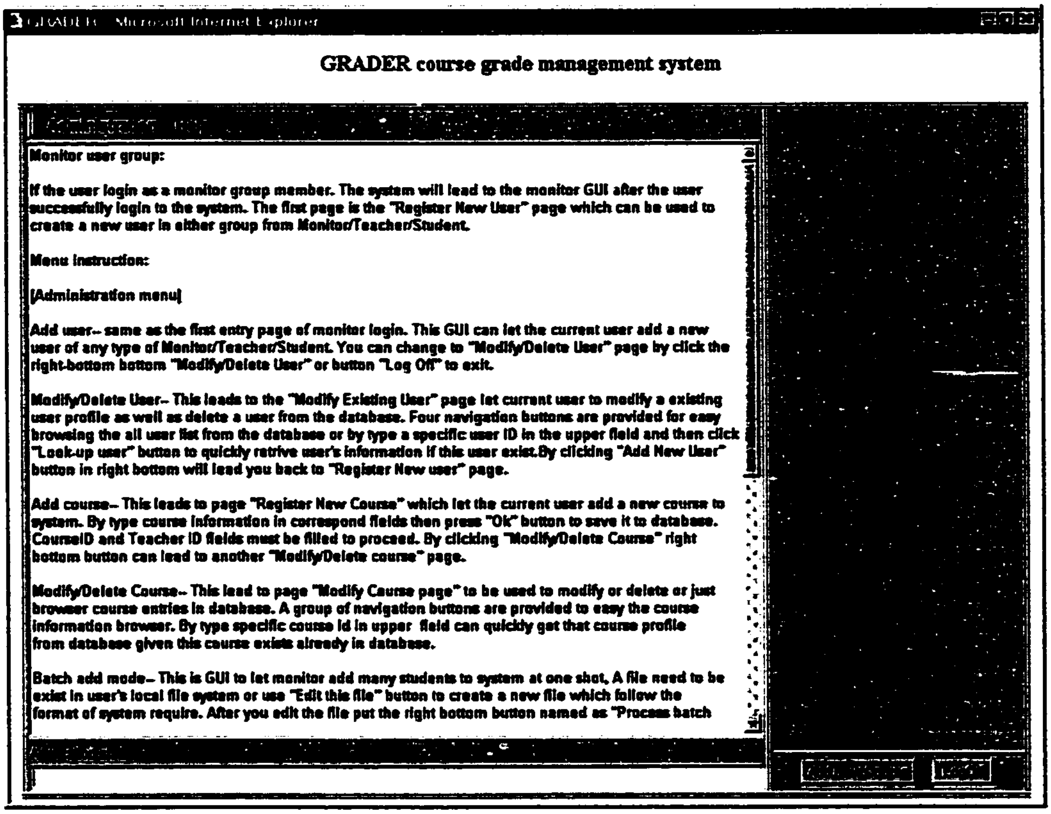
sa03

```
select * from "StudentBeanTable"
```

```
ID, email, firstName, gpa, lastName

4218361, mygrader@hotmail.com, Vicki, 0, CROFT
4219821, mygrader@hotmail.com, Bill, 0, CROSSMAN
4234502, mygrader@hotmail.com, Joel, 0, CUMMINGS
4234944, mygrader@hotmail.com, James, 0, CZEBOTAR
4241142, mygrader@hotmail.com, Kristen, 0, DAVIS
3988, dsun@cs.concordia.ca, Smith, 0, Ray
3912361, dsun@cs.concordia.ca, Daofeng, 0, Sun
3781358, mygrader@hotmail.com, Michelanne, 0, ADAMS
3799069, mygrader@hotmail.com, Mike, 0, ALBRECHT
3801799, mygrader@hotmail.com, Glen, 0, AMES
3875210, mygrader@hotmail.com, Gudrun, 0, AURAND
3929248, mygrader@hotmail.com, Gerald, 0, BECKER
3943984, mygrader@hotmail.com, Nancy, 0, BEEBE
3945731, mygrader@hotmail.com, Judi, 0, BEESON
3953076, mygrader@hotmail.com, Wendy, 0, BLAKE
3967247, mygrader@hotmail.com, Trevor, 0, BONO
4019601, mygrader@hotmail.com, Bonny, 0, BOYAN
```

Executed query: select * from "StudentBeanTable" .

- Monitor online Help like other two groups



GRADER course grade management system

Monitor user group:

If the user login as a monitor group member. The system will lead to the monitor GUI after the user successfully login to the system. The first page is the "Register New User" page which can be used to create a new user in either group from Monitor/Teacher/Student.

Menu instruction:

[Administration menu]

Add user-- same as the first entry page of monitor login. This GUI can let the current user add a new user of any type of Monitor/Teacher/Student. You can change to "Modify/Delete User" page by click the right-bottom bottom "Modify/Delete User" or button "Log Off" to exit.

Modify/Delete User-- This leads to the "Modify Existing User" page let current user to modify a existing user profile as well as delete a user from the database. Four navigation buttons are provided for easy browsing the all user list from the database or by type a specific user ID in the upper field and then click "Look-up user" button to quickly retrive user's information if this user exist.By clicking "Add New User" button in right bottom will lead you back to "Register New user" page.

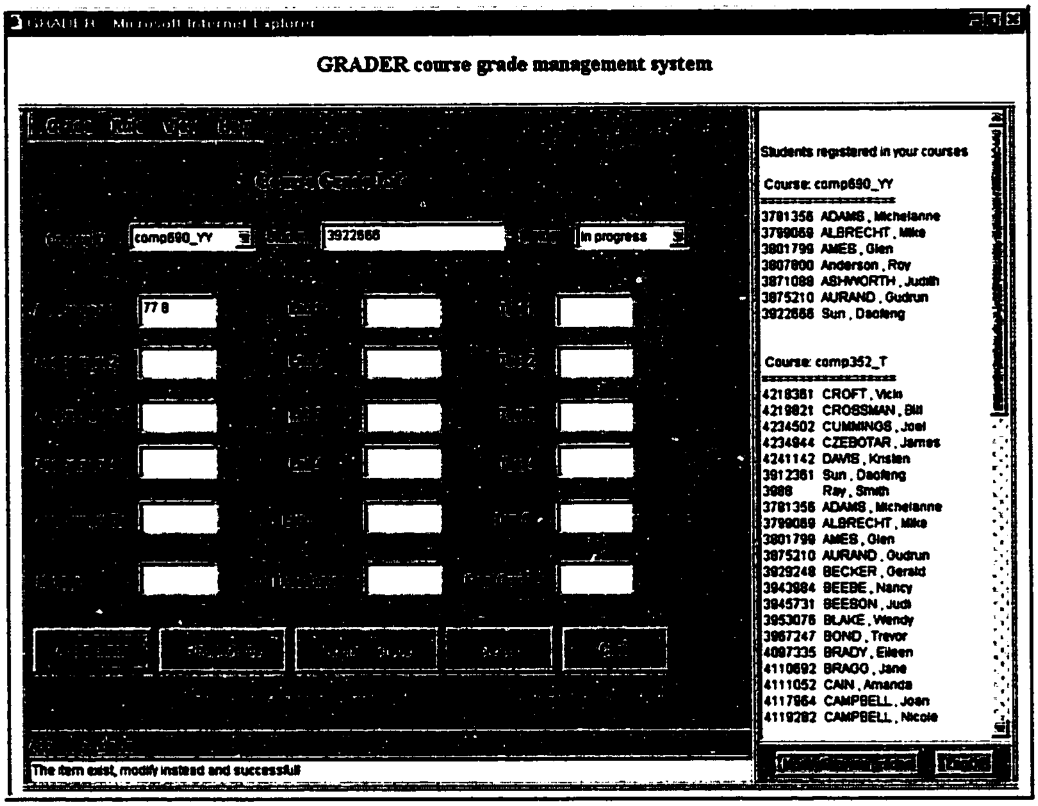
Add course-- This leads to page "Register New Course" which let the current user add a new course to system. By type course information in correspond fields then press "Ok" button to save it to database. CourseID and Teacher ID fields must be filled to proceed. By clicking "Modify/Delete Course" right bottom button can lead to another "Modify/Delete course" page.

Modify/Delete Course-- This lead to page "Modify Course page" to be used to modify or delete or just browser course entries in database. A group of navigation buttons are provided to easy the course information browser. By type specific course id in upper field can quickly get that course profile from database given this course exists already in database.

Batch add mode-- This is GUI to let monitor add many students to system at one shot, A file need to be exist in user's local file system or use "Edit this file" button to create a new file which follow the format of system require. After you edit the file put the right bottom button named as "Process batch

Teacher:

- Login in as a teacher (as a monitor login)
- Add course grade entry



GRADER course grade management system

comp690_YY

3922688

In progress

77 8

The item exist, modify instead and successful

Students registered in your courses

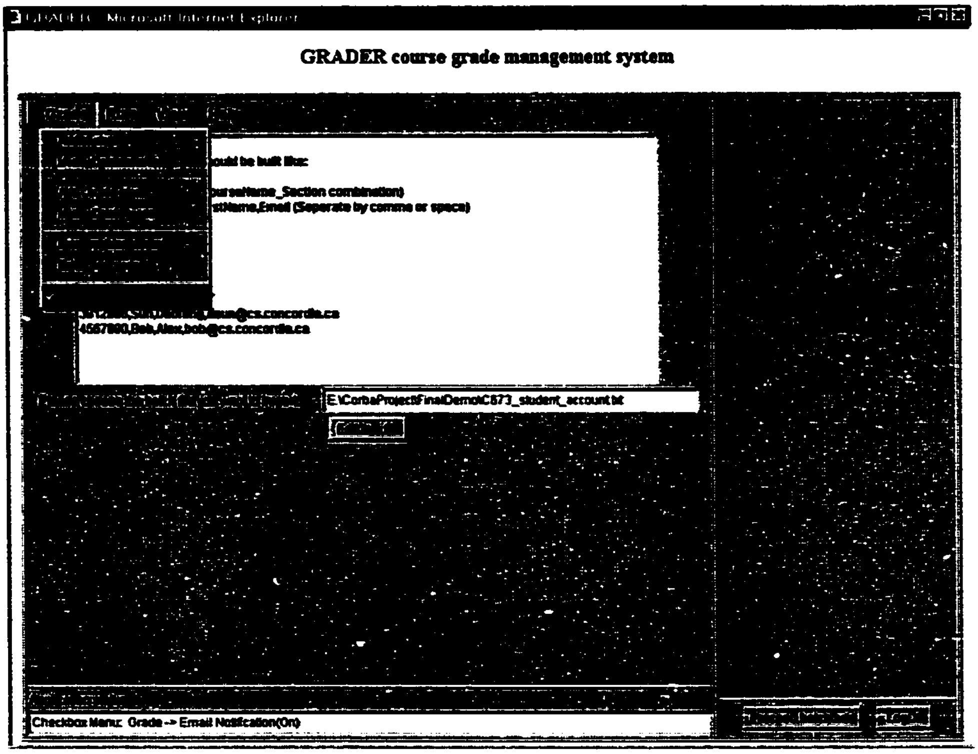
Course: comp690_YY

3781358 ADAMS , Michelanne
3799069 ALBRECHT , Mike
3801799 AMES , Glen
3807800 Anderson , Roy
3871088 ASHWORTH , Judith
3875210 AURAND , Gudrun
3922688 Sun , Daofeng

Course: comp352_T

4218381 CROFT , Vicki
4219821 CROSSMAN , Bill
4234502 CUMMINGS , Joel
4234944 CZEBOTAR , James
4241142 DAVIS , Kristen
3912361 Sun , Daofeng
3988 Ray , Smith
3781358 ADAMS , Michelanne
3799069 ALBRECHT , Mike
3801799 AMES , Glen
3875210 AURAND , Gudrun
3929248 BECKER , Gerald
3943984 BEEBE , Nancy
3945731 BEESON , Judi
3953076 BLAKE , Wendy
3967247 BOND , Trevor
4097335 BRADY , Eileen
4110692 BRAGG , Jane
4111052 CAIN , Amanda
4117964 CAMPBELL , Joan
4119282 CAMPBELL , Nicole

- Add students, account in batch mode



GRADER course grade management system

ould be built like:

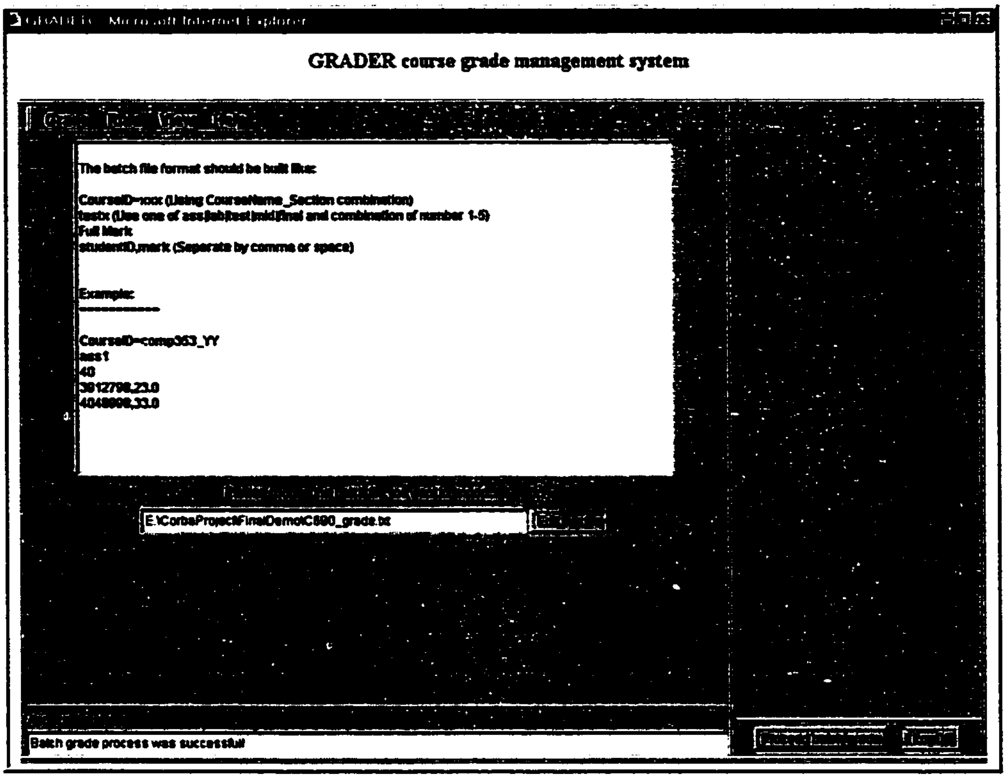
ourseName_Section combination)
stName,Email (Seperate by comma or space)

3912361,Sun,Daofeng,d_sun@cs.concordia.ca
4567890,Bob,Alex,bob@cs.concordia.ca

E:\CorbaProject\FinalDemo\C673_student_account.txt

Checkbox Menu: Grade -> Email Notification(On)

- The teacher can also add a course grade in batch mode



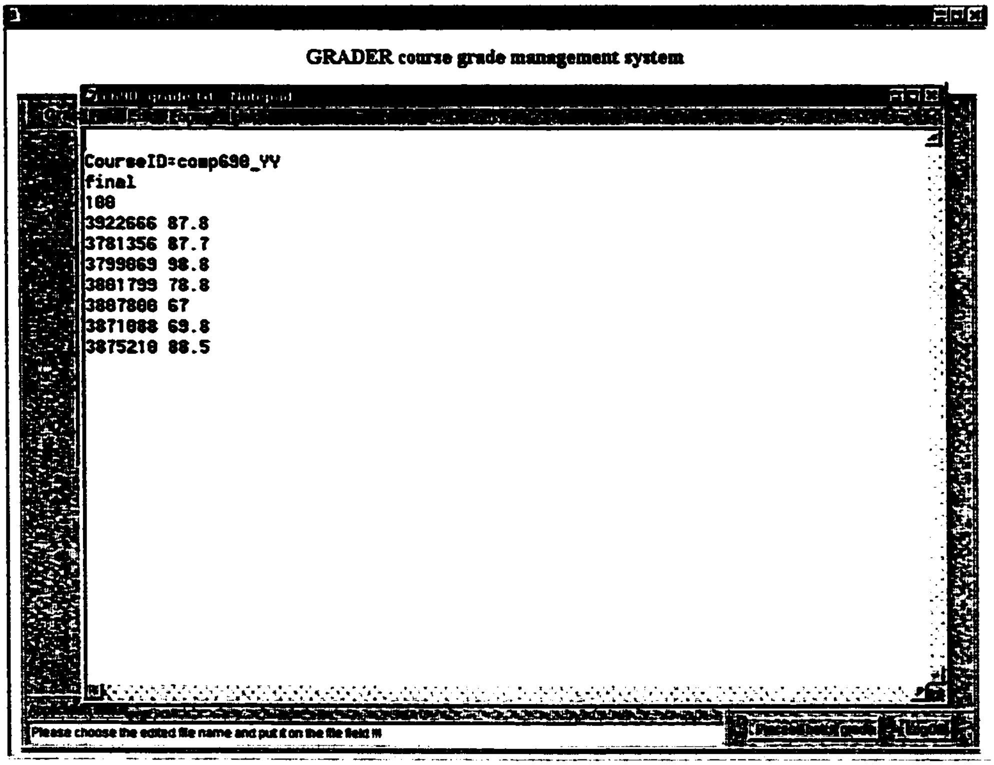
GRADER course grade management system

```
The batch file format should be built like:

CourseID=xxx (Using CourseName_Section combination)
testx (Use one of ass|lab|test|mid|final and combination of number 1-5)
Full Mark
studentID,mark (Separate by comma or space)


Example:
-----------

CourseID=comp363_YY
ass1
40
3012798,23.0
4048808,33.0
```

E:\CorbaProject\FinalDemo\C690_grade.txt

Batch grade process was successful!

- The teacher can also edit the batch file by clicking "Edit this file"

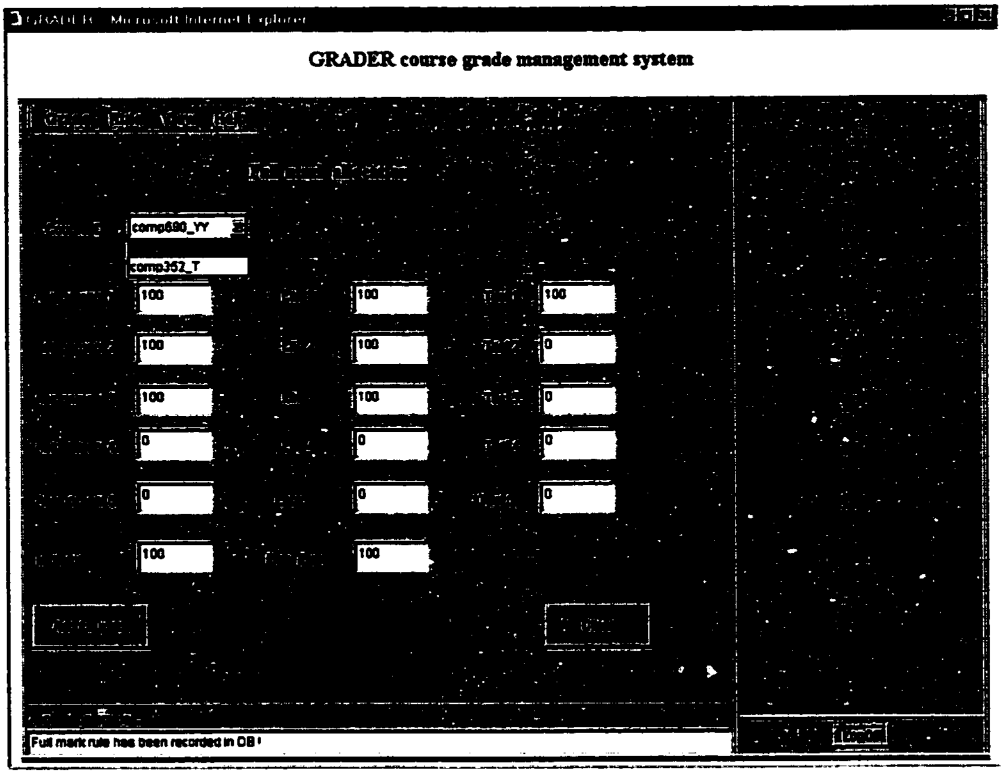


GRADER course grade management system

```
CourseID=comp690_YY
final
100
3922666 87.8
3781356 87.7
3799869 98.8
3881799 78.8
3887808 67
3871088 69.8
3875210 88.5
```

Please choose the edited file name and put it on the file field !!!

- Define course full mark standard



GRADER - Microsoft Internet Explorer

GRADER course grade management system

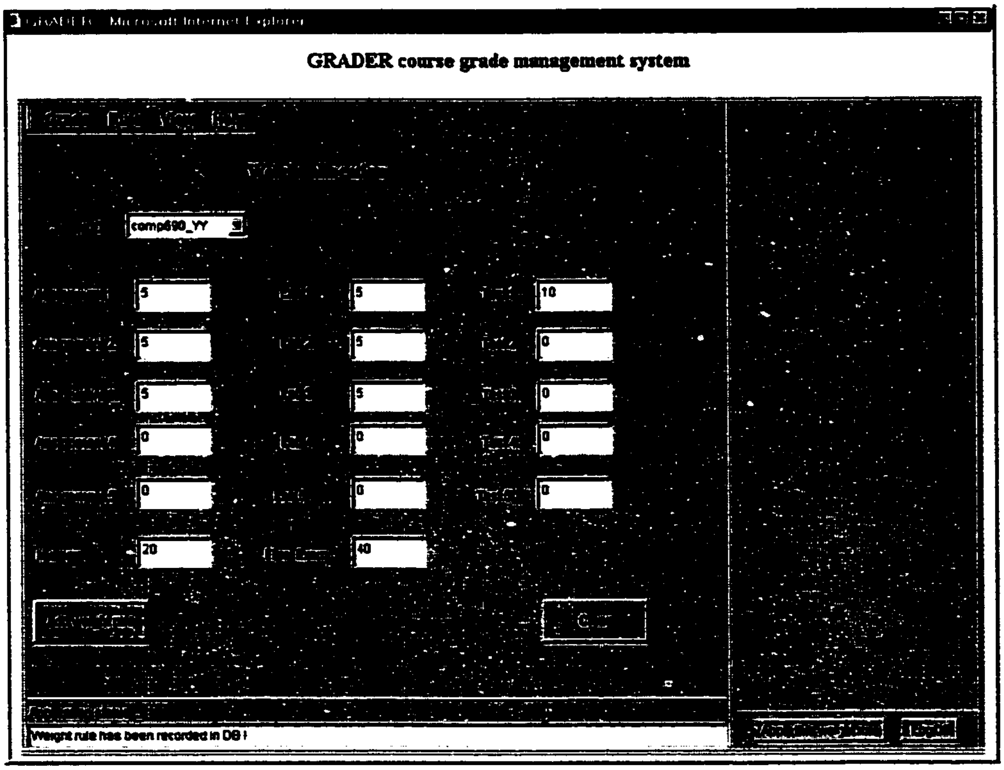
comp690_YY

comp352_T

100	100	100
100	100	0
100	100	0
0	0	0
0	0	0
100	100	

Full mark rule has been recorded in DB !

- Define course weight rule



GRADER - Microsoft Internet Explorer

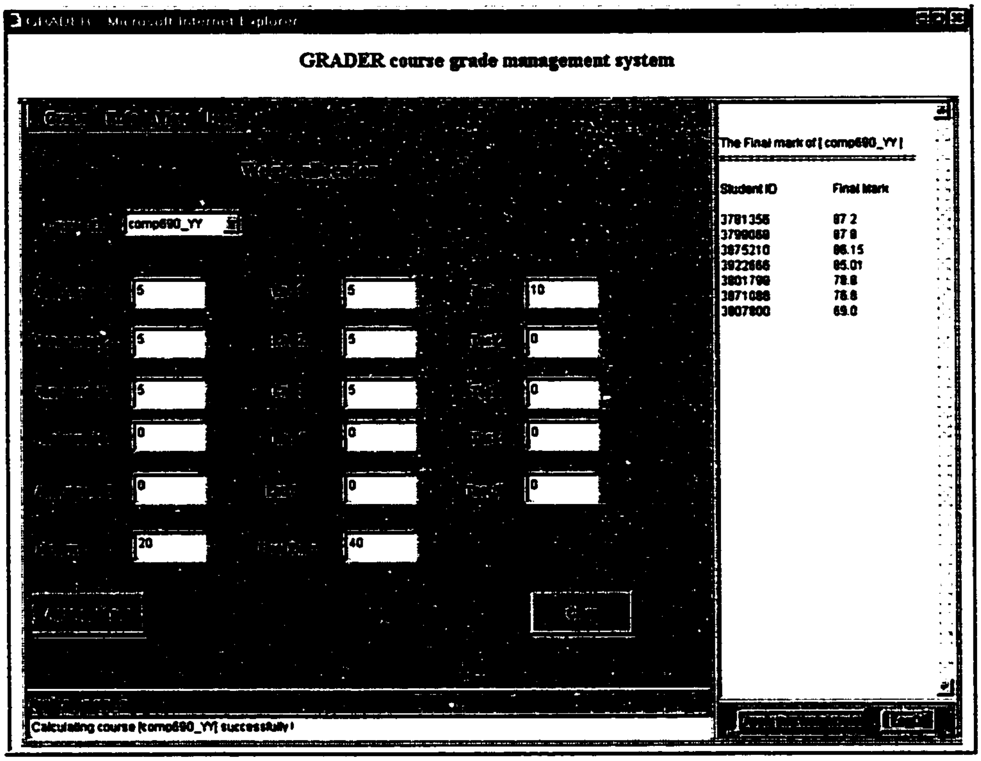
GRADER course grade management system

comp690_YY

5	5	10
5	5	0
5	5	0
0	0	0
0	0	0
20	40	

Weight rule has been recorded in DB !

- Click "Apply this weight rule" to calculate the course final mark



GRADER - Microsoft Internet Explorer

GRADER course grade management system

comp690_YY

5 5 10

5 5 0

5 5 0

0 0 0

0 0 0

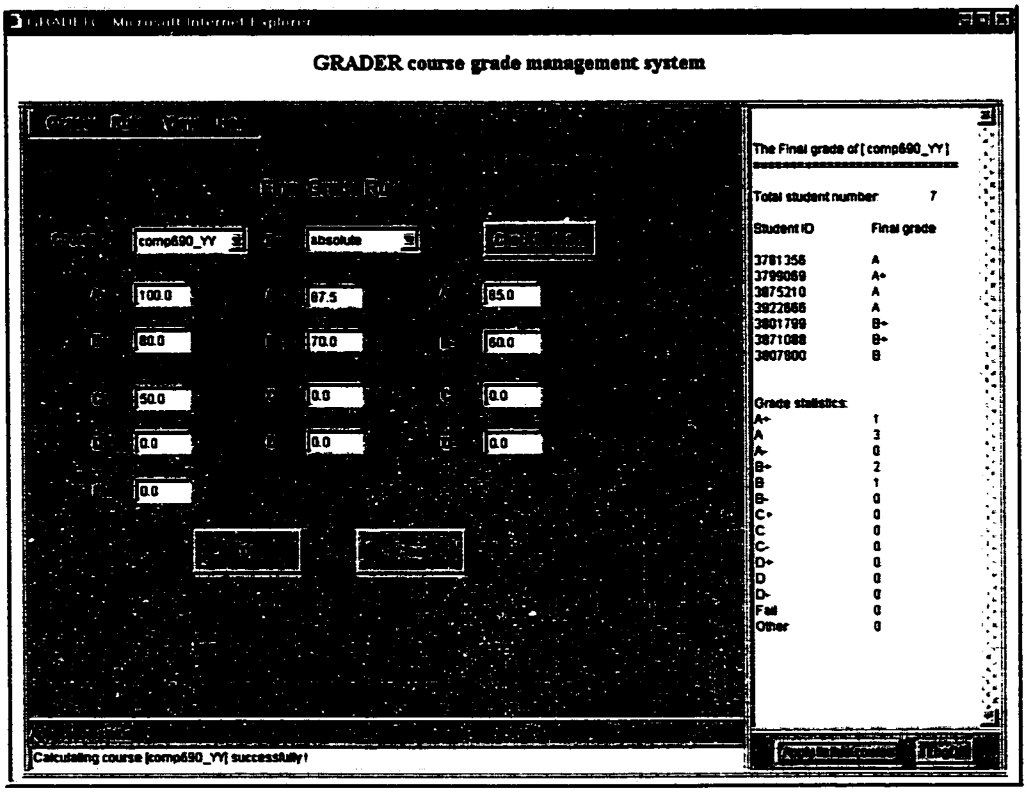
20 40

Calculating course [comp690_YY] successfully!

The Final mark of [comp690_YY]

Student ID	Final Mark
3781356	87.2
3799069	87.8
3875210	86.15
3922666	85.01
3801799	78.8
3871088	78.8
3807800	69.0

- Define the final grade rule in absolute mode, and then apply this rule



GRADER - Microsoft Internet Explorer

GRADER course grade management system

comp690_YY absolute

100.0 87.5 85.0

80.0 70.0 60.0

50.0 0.0 0.0

0.0 0.0 0.0

0.0

Calculating course [comp690_YY] successfully!

The Final grade of [comp690_YY]

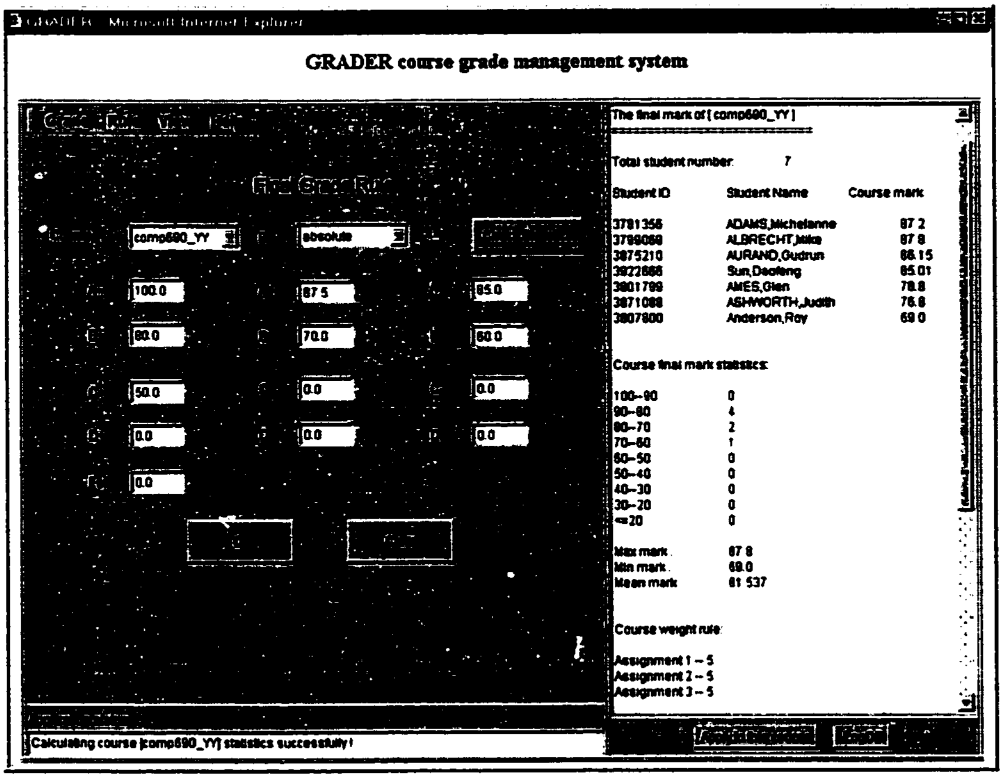
Total student number 7

Student ID	Final grade
3781356	A
3799069	A+
3875210	A
3922666	A
3801799	B+
3871088	B+
3807800	B

Grade statistics:

A+	1
A	3
A-	0
B+	2
B	1
B-	0
C+	0
C	0
C-	0
D+	0
D	0
D-	0
Fail	0
Other	0

- Check course statistics, and define the final grade rule in relative mode



GRADER course grade management system

Final Grade Rule

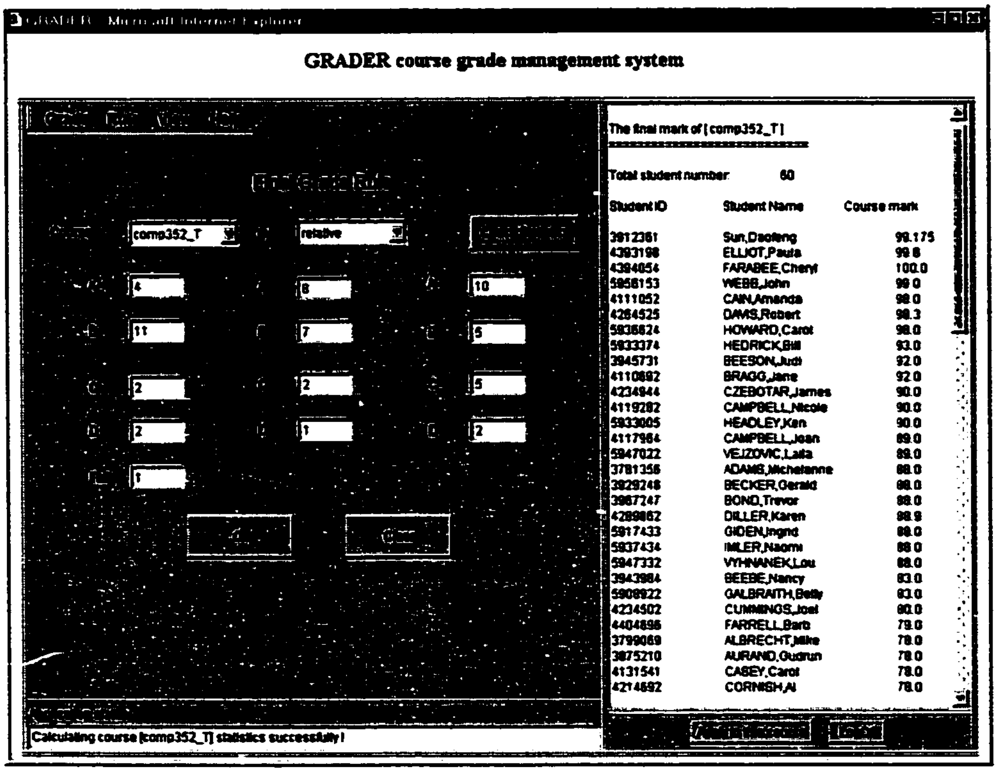
comp690_YY   absolute

100.0   87.5   85.0
80.0   70.0   60.0
50.0   0.0   0.0
0.0   0.0   0.0
0.0

```
The final mark of [ comp690_YY ]
================================

Total student number:        7

Student ID       Student Name          Course mark

3781356          ADAMS,Michelanne        87 2
3799069          ALBRECHT,Mike           87 8
3875210          AURAND,Gudrun           86.15
3922866          Sun,Daofeng             85.01
3801799          AMES,Glen               78.8
3871088          ASHWORTH,Judith         76.8
3807800          Anderson,Roy            69 0

Course final mark statistics:

100--90          0
90--80           4
80--70           2
70--60           1
60--50           0
50--40           0
40--30           0
30--20           0
<=20             0

Max mark .       87 8
Min mark .       69.0
Mean mark        81 537

Course weight rule:

Assignment 1 -- 5
Assignment 2 -- 5
Assignment 3 -- 5
```

Calculating course [comp690_YY] statistics successfully!

- Check statistics about another course



GRADER course grade management system

Final Grade Rule

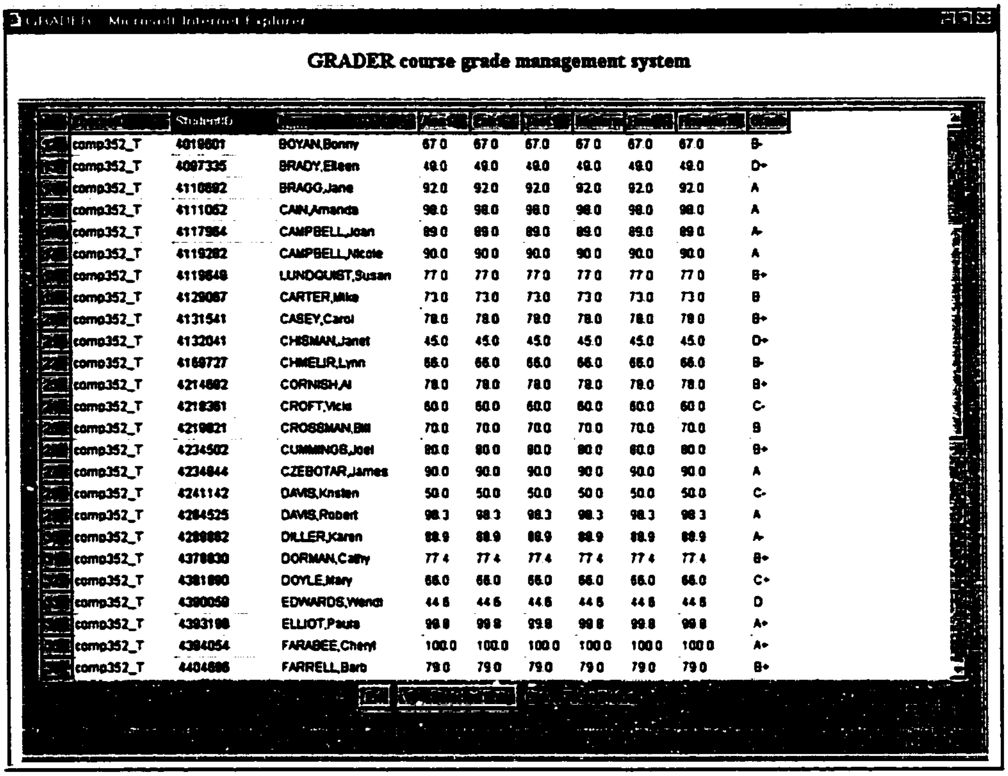
comp352_T   relative

4   8   10
11   7   5
2   2   5
2   1   2
1

```
The final mark of [ comp352_T ]
================================

Total student number:        60

Student ID       Student Name          Course mark

3912361          Sun,Daofeng             98.175
4393198          ELLIOT,Paula            99 8
4394054          FARABEE,Cheryl          100.0
5958153          WEBB,John               99 0
4111052          CAIN,Amanda             98.0
4264525          DAVIS,Robert            98.3
5936624          HOWARD,Carol            98.0
5933374          HEDRICK,Bill            93.0
3945731          BEESON,Judi             92.0
4110892          BRAGG,Jane              92.0
4234944          CZEBOTAR,James          90.0
4119282          CAMPBELL,Nicole         90.0
5933005          HEADLEY,Ken             90.0
4117964          CAMPBELL,Joan           89.0
5947022          VEJZOVIC,Laila          89.0
3781356          ADAMS,Michelanne        88.0
3929248          BECKER,Gerald           88.0
3967247          BOND,Trevor             88.0
4289862          DILLER,Karen            88.8
5917433          GIDEN,Ingrid            88.0
5937434          IMLER,Naomi             88 0
5947332          VYHNANEK,Lou            88.0
3943984          BEEBE,Nancy             83.0
5908922          GALBRAITH,Betty         83.0
4234502          CUMMINGS,Joel           80.0
4404896          FARRELL,Barb            79.0
3799069          ALBRECHT,Mike           78.0
3875210          AURAND,Gudrun           78.0
4131541          CASEY,Carol             78.0
4214692          CORNISH,Al              78.0
```

Calculating course [comp352_T] statistics successfully!

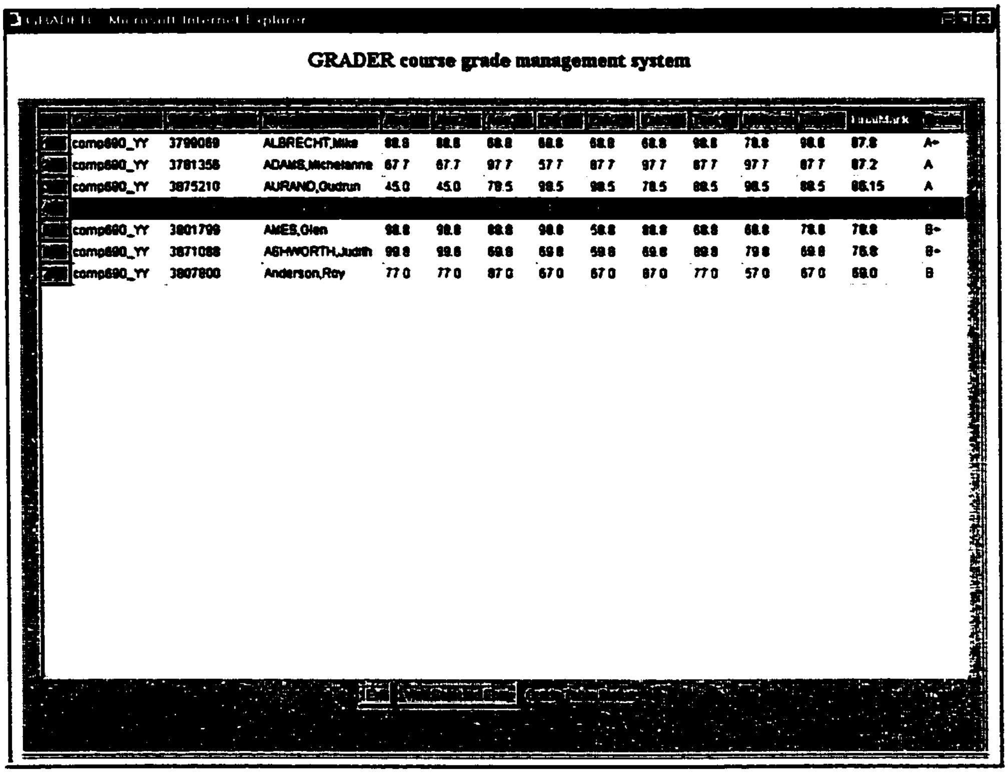
- View grade in a table, and sort the student ID field



GRADER course grade management system

[illegible]	StudentID	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
comp352_T	4019801	BOYAN,Bonny	67.0	67.0	67.0	67.0	67.0	67.0	B-
comp352_T	4097335	BRADY,Eileen	49.0	49.0	49.0	49.0	49.0	49.0	D+
comp352_T	4110892	BRAGG,Jane	92.0	92.0	92.0	92.0	92.0	92.0	A
comp352_T	4111062	CAIN,Amanda	98.0	98.0	98.0	98.0	98.0	98.0	A
comp352_T	4117964	CAMPBELL,Joan	89.0	89.0	89.0	89.0	89.0	89.0	A-
comp352_T	4119282	CAMPBELL,Nicole	90.0	90.0	90.0	90.0	90.0	90.0	A
comp352_T	4119848	LUNDQUIST,Susan	77.0	77.0	77.0	77.0	77.0	77.0	B+
comp352_T	4129087	CARTER,Mike	73.0	73.0	73.0	73.0	73.0	73.0	B
comp352_T	4131541	CASEY,Carol	78.0	78.0	78.0	78.0	78.0	78.0	B+
comp352_T	4132041	CHISMAN,Janet	45.0	45.0	45.0	45.0	45.0	45.0	D+
comp352_T	4169727	CHMELIR,Lynn	66.0	66.0	66.0	66.0	66.0	66.0	B-
comp352_T	4214882	CORNISH,Al	78.0	78.0	78.0	78.0	78.0	78.0	B+
comp352_T	4218361	CROFT,Vicki	60.0	60.0	60.0	60.0	60.0	60.0	C-
comp352_T	4219821	CROSSMAN,Bill	70.0	70.0	70.0	70.0	70.0	70.0	B
comp352_T	4234502	CUMMINGS,Joel	80.0	80.0	80.0	80.0	80.0	80.0	B+
comp352_T	4234844	CZEBOTAR,James	90.0	90.0	90.0	90.0	90.0	90.0	A
comp352_T	4241142	DAVIS,Kristen	50.0	50.0	50.0	50.0	50.0	50.0	C-
comp352_T	4284525	DAVIS,Robert	98.3	98.3	98.3	98.3	98.3	98.3	A
comp352_T	4288882	DILLER,Karen	88.9	88.9	88.9	88.9	88.9	88.9	A-
comp352_T	4378830	DORMAN,Cathy	77.4	77.4	77.4	77.4	77.4	77.4	B+
comp352_T	4381880	DOYLE,Mary	66.0	66.0	66.0	66.0	66.0	66.0	C+
comp352_T	4390058	EDWARDS,Wendi	44.6	44.6	44.6	44.6	44.6	44.6	D
comp352_T	4393198	ELLIOT,Paula	99.8	99.8	99.8	99.8	99.8	99.8	A+
comp352_T	4394054	FARABEE,Cheryl	100.0	100.0	100.0	100.0	100.0	100.0	A+
comp352_T	4404886	FARRELL,Barb	79.0	79.0	79.0	79.0	79.0	79.0	B+

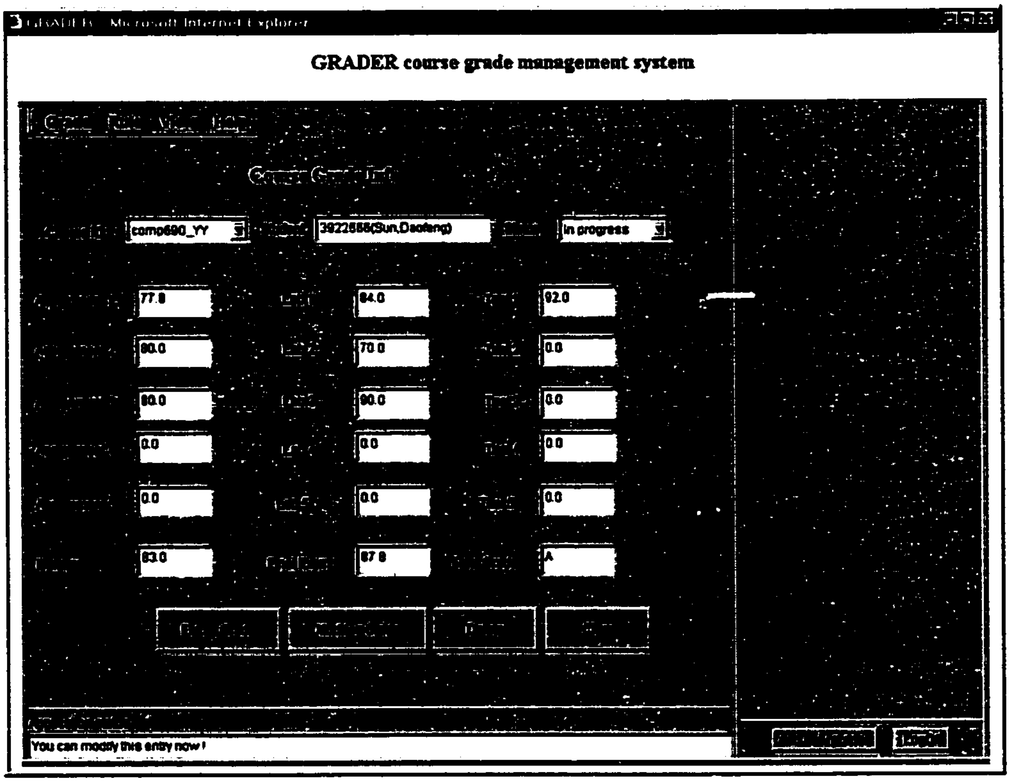
- View an added course grade in a table, and highlight a record to edit



GRADER course grade management system

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	FinalMark	[illegible]
comp690_YY	3799089	ALBRECHT,Mike	88.8	88.8	68.8	68.8	68.8	68.8	98.8	78.8	98.8	87.8	A-
comp690_YY	3781356	ADAMS,Michelanne	67.7	87.7	97.7	57.7	87.7	97.7	87.7	97.7	87.7	87.2	A
comp690_YY	3875210	AURANO,Gudrun	45.0	45.0	78.5	98.5	98.5	78.5	88.5	98.5	88.5	86.15	A
comp690_YY	3801799	AMES,Glen	98.8	98.8	88.8	98.8	58.8	88.8	68.8	68.8	78.8	78.8	B-
comp690_YY	3871088	ASHWORTH,Judith	99.8	99.8	69.8	69.8	59.8	69.8	89.8	79.8	69.8	76.8	B-
comp690_YY	3807800	Anderson,Roy	77.0	77.0	87.0	67.0	67.0	87.0	77.0	57.0	67.0	69.0	B

- Edit a grade entry in another GUI



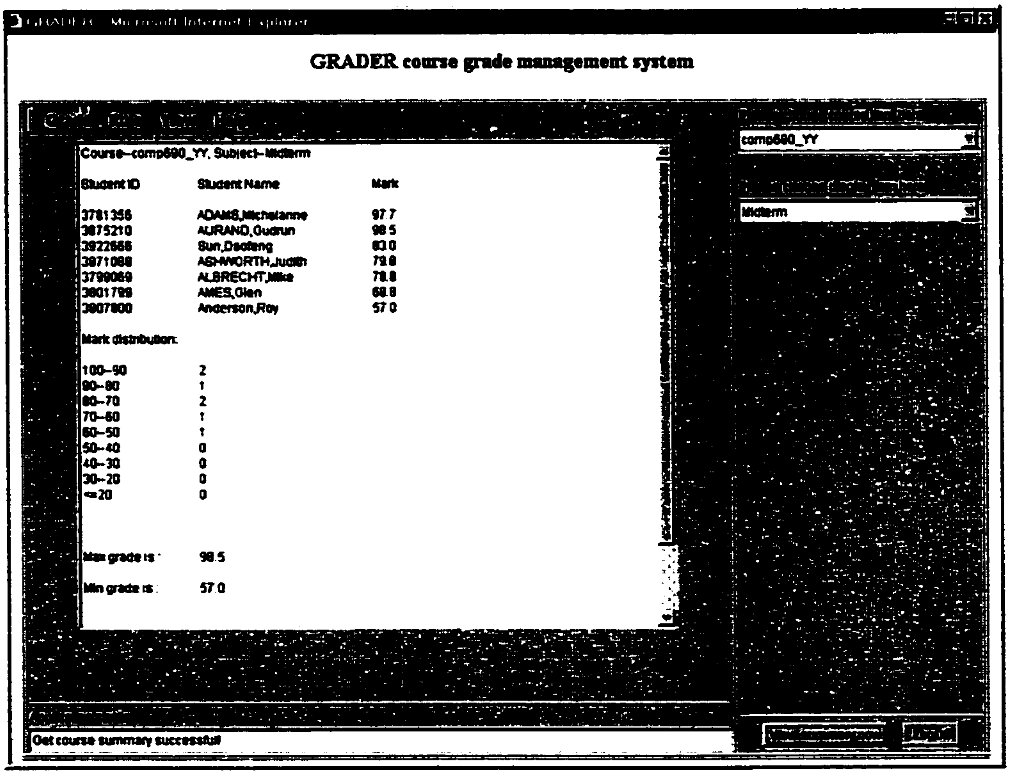
GRADER course grade management system

comp690_YY | 3922688(Sun,Daofeng) | In progress

[illegible]	77.8	[illegible]	84.0	[illegible]	92.0
[illegible]	80.0	[illegible]	70.0	[illegible]	0.0
[illegible]	80.0	[illegible]	90.0	[illegible]	0.0
[illegible]	0.0	[illegible]	0.0	[illegible]	0.0
[illegible]	0.0	[illegible]	0.0	[illegible]	0.0
[illegible]	83.0	[illegible]	87.8	[illegible]	A

You can modify this entry now !

- View a course one item (midterm) detail information



GRADER course grade management system

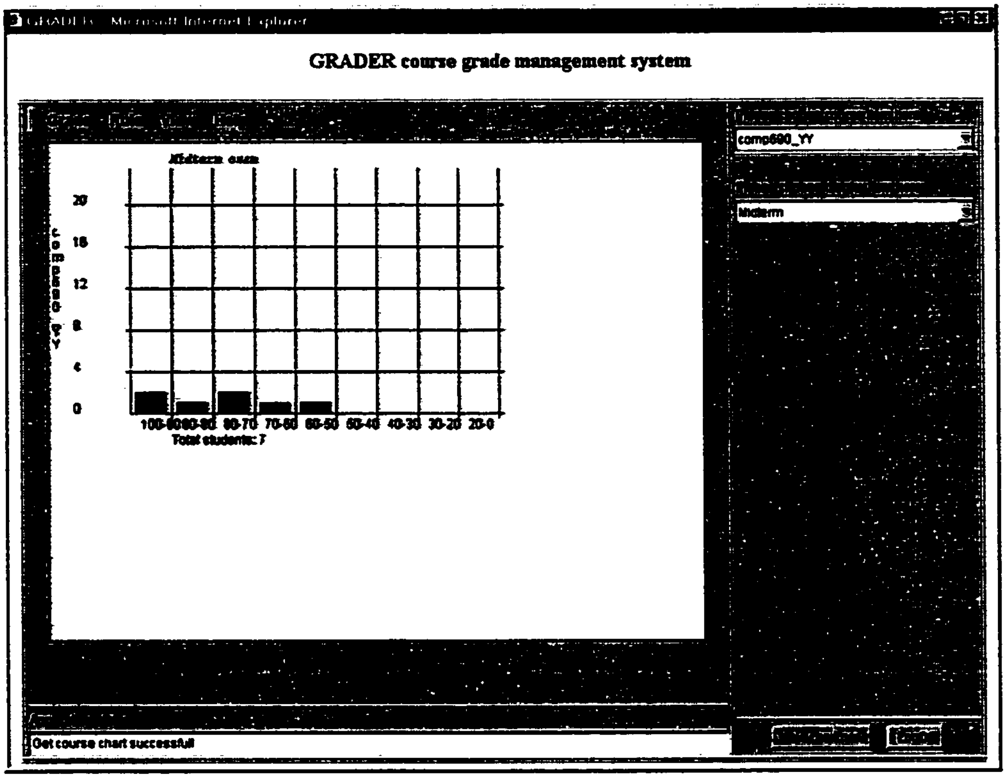
```
Course--comp690_YY, Subject--Midterm

Student ID      Student Name          Mark

3781356         ADAMS,Michelanne      97.7
3875210         AURAND,Gudrun         98.5
3922688         Sun,Daofeng           83.0
3871088         ASHWORTH,Judith       79.8
3799069         ALBRECHT,Mike         78.8
3801799         AMES,Glen             68.8
3807800         Anderson,Roy          57.0

Mark distribution:

100--90         2
90--80          1
80--70          2
70--60          1
60--50          1
50--40          0
40--30          0
30--20          0
<=20            0


Max grade is :  98.5

Min grade is :  57.0
```

comp690_YY

Midterm

Get course summary successful!

- View course grade in chart manner

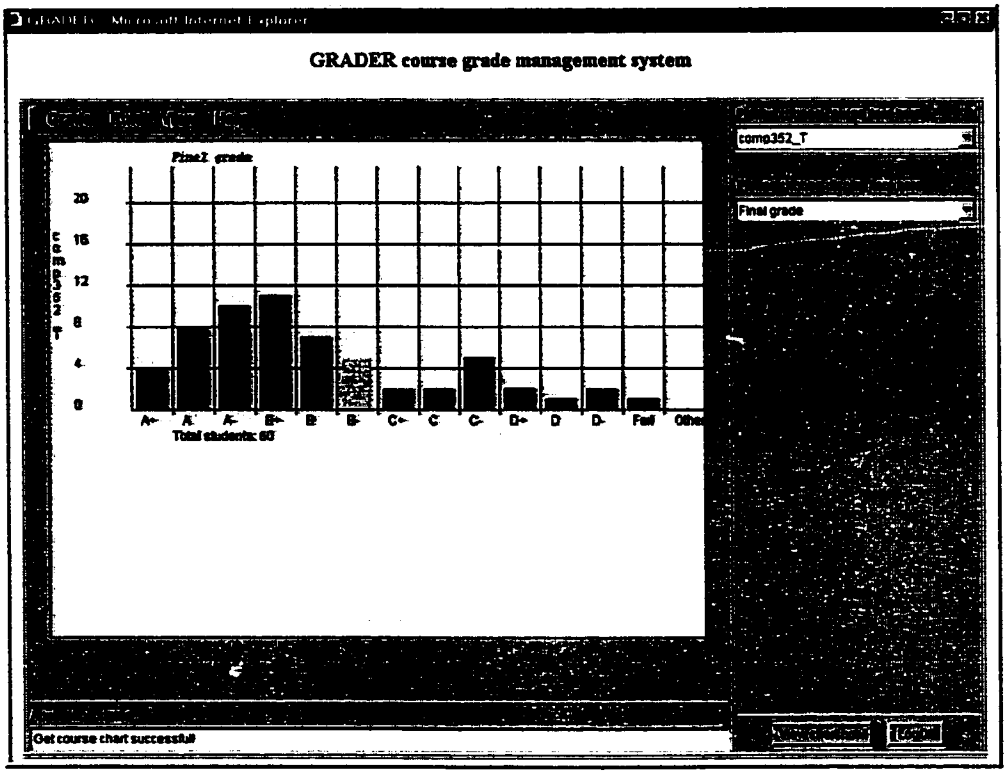


GRADER - Microsoft Internet Explorer

GRADER course grade management system

Get course chart successfull

- View another course grade in chart manner

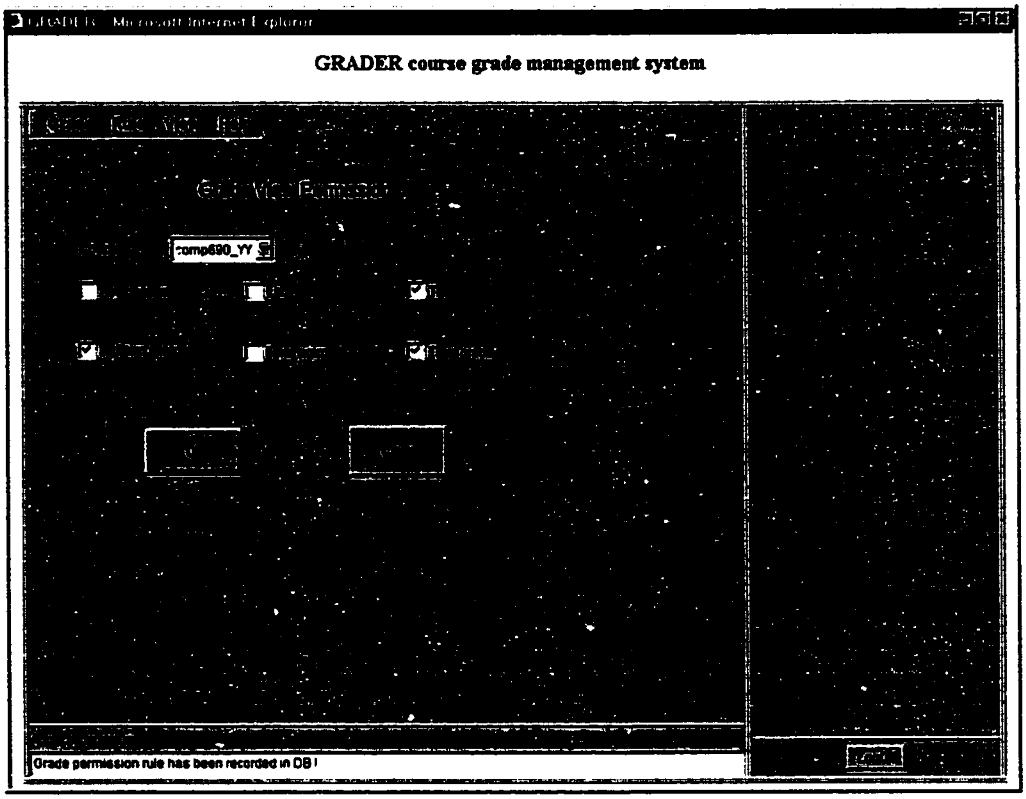


GRADER - Microsoft Internet Explorer

GRADER course grade management system

Get course chart successfull

- Set up a student viewing permission



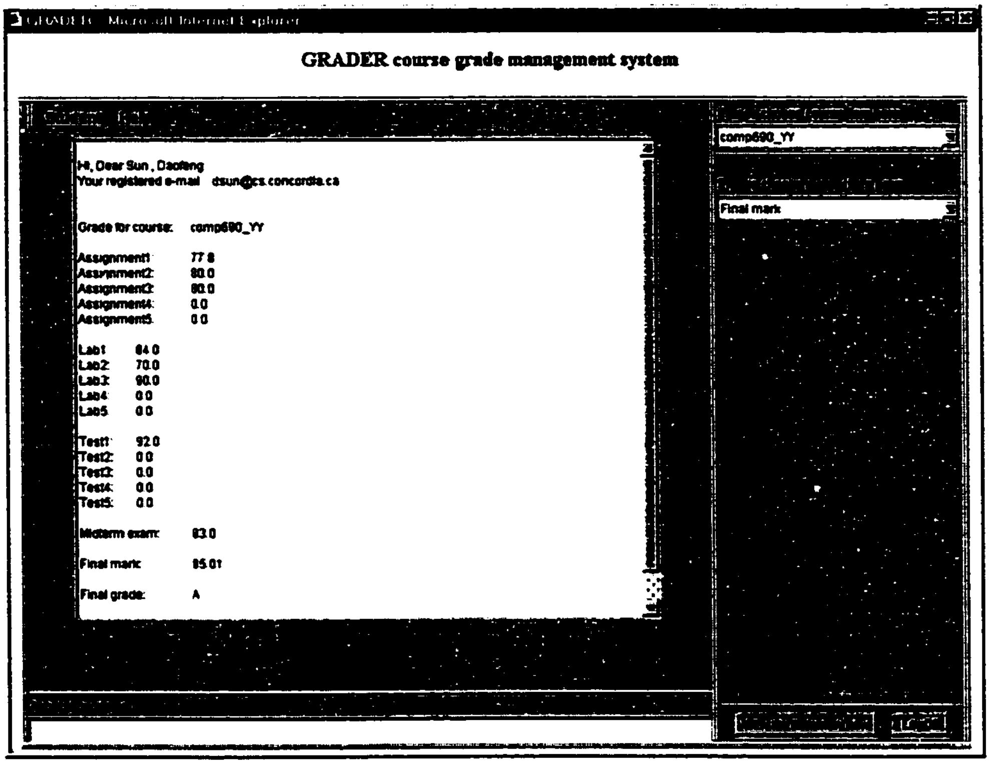
GRADER course grade management system

comp690_YY

Grade permission rule has been recorded in DB !

Student:

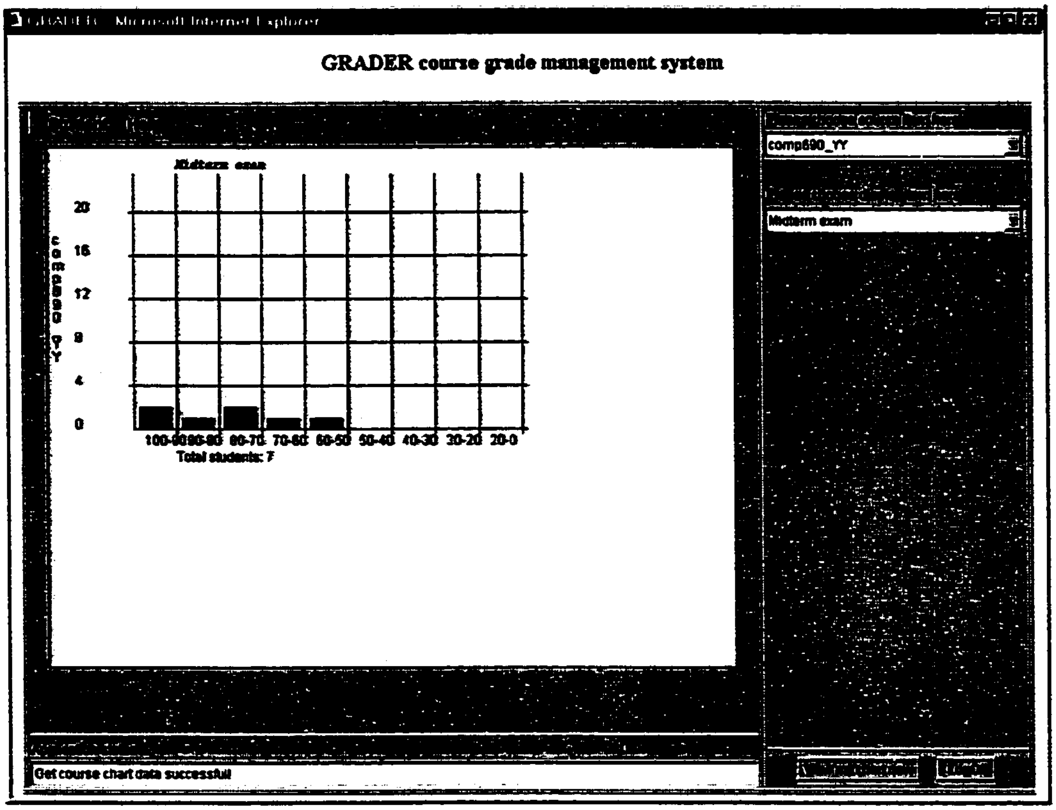
- Login as a student group (as monitor login manner)
- View course grade information



GRADER - Microsoft Internet Explorer
GRADER course grade management system
comp690_YY
Final mark

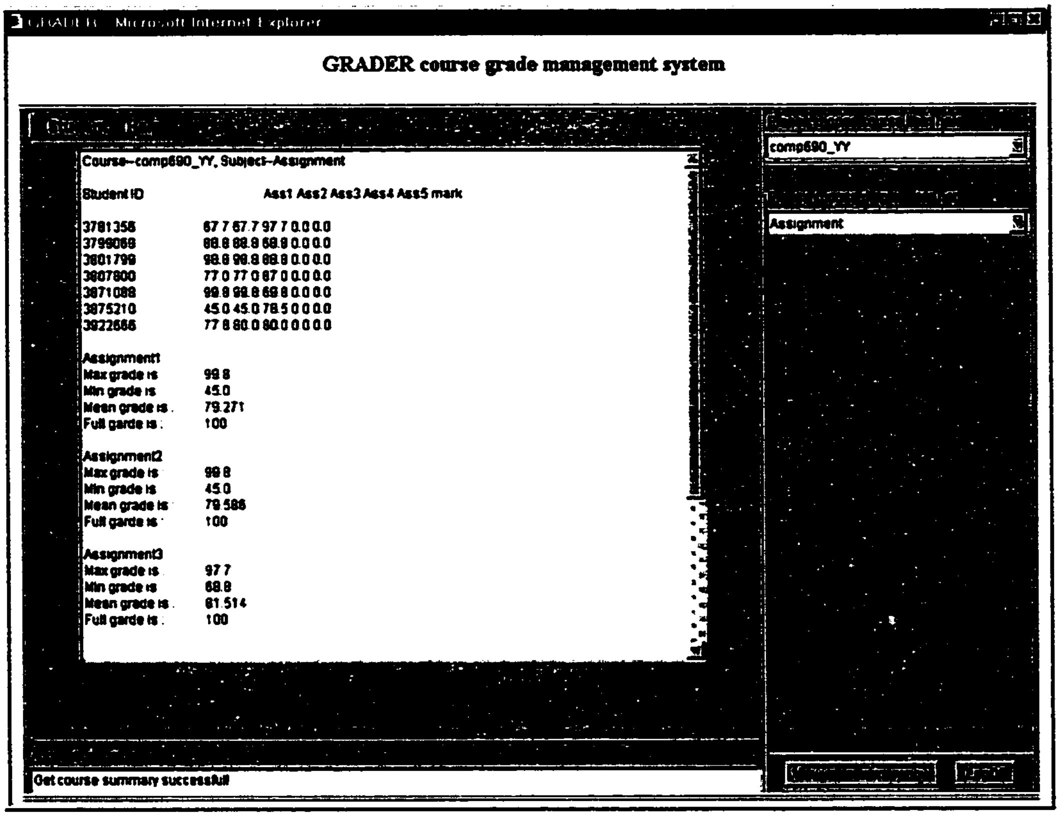
```
Hi, Dear Sun , Daofeng
Your registered e-mail   dsun@cs.concordia.ca


Grade for course:     comp690_YY

Assignment1:          77.8
Assignment2:          80.0
Assignment3:          80.0
Assignment4:          0.0
Assignment5:          0.0

Lab1     84.0
Lab2:    70.0
Lab3:    90.0
Lab4:    0.0
Lab5:    0.0

Test1:   92.0
Test2:   0.0
Test3:   0.0
Test4:   0.0
Test5:   0.0

Midterm exam:         83.0

Final mark:           85.01

Final grade:          A
```

- View grade chart



Get course chart data successfull

- View course item statistics information



GRADER course grade management system

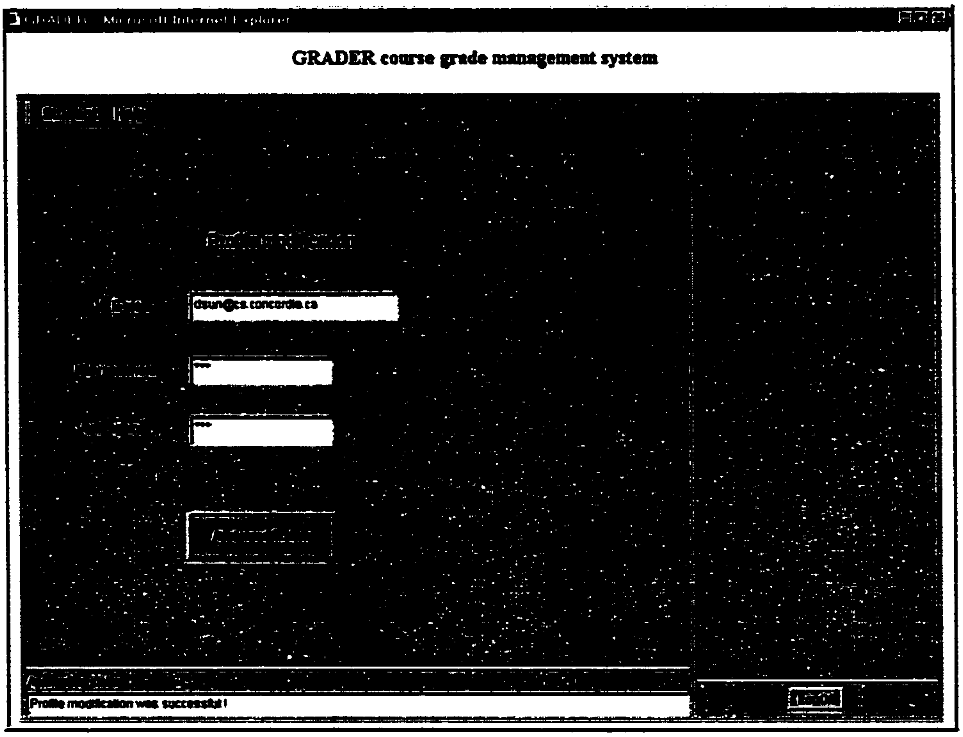
comp690_YY

Assignment

```
Course--comp690_YY, Subject-Assignment

Student ID                  Ass1 Ass2 Ass3 Ass4 Ass5 mark

3781358          67 7 67.7 97 7 0.0 0.0
3799069          88.8 88.8 68.8 0.0 0.0
3801799          98.8 98.8 88.8 0.0 0.0
3807800          77.0 77 0 87 0 0.0 0.0
3871088          99.8 99.8 69.8 0.0 0.0
3875210          45.0 45.0 78.5 0 0 0.0
3922666          77 8 80.0 80.0 0 0 0.0

Assignment1
Max grade is     99 8
Min grade is     45.0
Mean grade is .  79.271
Full garde is :  100

Assignment2
Max grade is     99 8
Min grade is     45.0
Mean grade is    79 586
Full garde is    100

Assignment3
Max grade is     97 7
Min grade is     68.8
Mean grade is .  81.514
Full garde is :  100
```

Get course summary successfull

- View or modify a student profile



GRADER course grade management system

dsun@cs.concordia.ca

***

***

Profile modification was successful !