This paper was presented at CUMREC '97, The College and University Information Services Conference. It is the intellectual property of the author(s). Permission to print out copies of this paper is granted provided that the copies are not made or distributed for commercial advantage and that the title and authors of the paper appear on the copies. To copy or disseminate otherwise, or to republish in any form, print or electronic, requires written permission from the authors.
While teaching and learning technologies are constantly changing, one thing remains constant: we still need to grade students’ performance. The methods of scoring students’ grades may vary and many instructors still prefer traditional methods using small grading books where scores for homework, projects, tests and exams are neatly recorded by hand. However, many more instructors are switching to computer based grading programs or spreadsheets, for they offer more convenience for computing intermediate or final grades, preparing reports, and recording comments. In certain course settings, even more compelling reasons suggest the use of modern technologies as a preferred method of grading. Consider, for example, a model, used in many universities, where one coordinating faculty supervises multi-section, multi-instructor courses offered to a big population of students. Critical for good organization of such courses is fast dissemination of the students’ grades to the coordinating faculty, students, academic advisors and administrators. With today's technologies, this can be achieved with the WWW which gives us the ability to exchange information across the Internet quickly and easily. This paper will propose two solutions, with different approaches, for grading programs in the Internet environment. The first method is traditional and uses Perl script; the second one uses Java programming language to create a server and applets.
UNIVERSITY OF KENTUCKY
WEB GRADER
AN INTERACTIVE GRADING SYSTEM

Functional requirements for a good grading system
Successful grading systems must be designed to meet the following requirements:

Ease of use
- The process must be simple to use for instructors, students, and school administrators. It should not impose added work on the instructor. Rather, it should be integrated into the work they must already do to keep rack of student scores. The method of entering scores, assignments, etc. must be simple. The system shouldn't require any training. A good solution would provide an intuitive graphical user interface. There shouldn't be a host of commands to remember.
- The system should not require complex installation. Anything more than a simple installation is not acceptable. Installation problems are the most common problems users experience with a system. If users cannot install the system, they will not use it.
- Software updates should be easy to distribute. As with all software, we are faced with new versions which either add features or fix existing defects. The system should be easy to upgrade when the need arises.

Functional Requirements
- Students should have real time access to the data. Students should have an easy method to access their scores for any course. They should be able to view the most recent data available. The system should provide methods for interpreting the data for the student. This includes any charts or graphs that can convey the information appropriately.
- Instructors should be able to maintain student scores. Instructors should be able to maintain scores for students in each of their courses. Each course should have a list of students, assignments and assignment categories associated with it. Instructors should be able to vary the weighting of each assignment or each category. The system will compute all student averages based on these weights. This allows the flexibility for each instructor to use their own method for computing overall averages.
- Instructors should be able to keep information on each student. The student record should include a running list of comments which can be accumulated for each student.
- School administrators should be able to browse student scores. Administrators given proper authority should be allowed to browse a student's scores. For example, the dean of students may wish to monitor a student's performance.

User Acceptance
- The system should not be operating system specific. Universities have a wide range of computer systems available to both faculty and students. If only one platform is
supported then you restrict which users can use the system. To gain widespread acceptance of a system such as this, you cannot afford to displease your users. Ideally all platforms would be supported. Realistically, the major operating systems would be supported. (ie. Windows NT, Windows '95, Windows 3.1, OS/2, Macintosh, and Solaris and other UNIX systems)
• The system must be flexible. The system should allow instructors to implement grading policies and procedures as they wish. Methods shouldn't be mandated to the instructors. This would be too restrictive and might discourage users.

**Security**
• The data must be very secure. Student scores represent personal data. Students should be able to access only their own scores. Instructors should be able to access only the scores of their own students.
• Security holes should not exist. The system should maintain security in such a way that the system security cannot be compromised.
• All data should be stored in a central, secure location. The importance of this data mandates that it be stored in a safe location. Ideally, this would be a database which is backed up regularly. Access to the database should be limited.

**Flexibility**
• The system should be designed to allow future enhancements. The system should be easy to maintain as well as improve. New functionality should affect the existing code minimally.
• Future enhancements should be taken into account during the design. When designing a large system such as this, thought must be given to areas of possible future development and it should be factored into the design.

**Possible solutions**

Natural solutions meeting most of the above requirements involve the Internet. Here is a brief description of some possible solutions for implementing this system. For each solution we summarize the important advantages and disadvantages.

**Solution with E-Mail**
One solution might be to have a C or C++ application for the instructor. This application would provide a GUI interface to the instructor. Data would be stored to the local machine. Students would receive their scores on a regular basis through electronic mail.

**Pros**
• No database to program to.
• Only one application to develop.

**Cons**
• The C or C++ application would have to be targeted for each platform used by instructors. (This requires a good deal of work.)
• Students just receive the data.
• No interaction is involved.
• No graphical representation of their scores.
• Not an extensible solution.
• Administrators would not be able to browse grades.
• Data is not stored securely.

**Solution with CGI scripts**
Another solution is to use CGI scripts on a server to implement both the instructor application for entering grades and to provide browsing capabilities to students and administrators.

Forms could be generated to allow instructors the ability to input grades, assignments, etc. The CGI scripts would record this data into the database and generate new pages with forms for each type of input.

**Pros**
• Data stored in database Access to database is centralized.
• Available on all platforms (via WWW).

**Cons**
• Code executes on server increasing load on server.
• Graphical user interface is quite limited.
• Not an extensible solution.
• Not easy to maintain.

**Solution with Java applets**
The third solution is to use a piece of new technology which allows for platform independent applications to be distributed over the network. The technology is the Java Programming Language. The applications will be written in Java to make use of its portability across platforms as well as the interactive interface it can provide to the user. Both instructors and students will be able to access the system through their Java enabled Web browser. The data will be stored in a database located on a remote machine. The applications will access the database through a server application. Thus, this system makes use of a client server architecture. The applications are clients to a data server. The data server provides them access to the data stored within the database.

This solution satisfies requirements of an easy to use and install application. It also provides for very good data security.

**Pros**
• Cross-platform support Java enabled browsers already exist on the following platforms: Windows NT, Windows '95, Windows 3.1, OS/2, Solaris, and IRIX.
• One line of source code will target each of these platforms.
• No installation.
• No installation problems since there is no installation. The users execute the application through their Web browser by going to the correct page. The code is automatically downloaded to their machine. This means that they always receive the latest code each time they use the application. Thus, there are no software update issues.
• Extensible. Java is an object-oriented language. This means that it is well suited to implement an object-oriented design. A good OOD will provide reusable objects which can be modified without affecting the objects they interact with. This type of design is the key behind an extensible application. Programming is easier; with a single line of source code, a developer does not have to worry about different class libraries or toolkits for each operating system. By using the Java class libraries, the developer is shielded from operating system specific interfaces.

Cons
• Requires third application: the data server to be written.
• Still bugs and quirks in the Java development environment.

Out of the above solutions we discard solution #1 which does not provide necessary interactions and interface. We will discuss details of solutions #2 and #3 in the following sections.

The grader system using Perl scripts
The grade recorder/browser is implemented in Perl scripts, which communicate with each other via the Common Gateway Interface (CGI) of the World Wide Web (WWW). The grade data is stored in a Sybase database, and accessed through calls to a C++ database client, written by the developers of the Stream-on-Line project, which was modified slightly for this project. Input is done through a variety of HTML screens. The program uses a Perl CGI library (cgi-lib.pl), written by Steven E. Brenner, with slight cosmetic changes. A primitive embedded SQL library, sql-lib.pl, was written for this project. Since Perl allows for the easy insertion of string variables within string literals, this library merely runs SQL queries sent to it, while the imbedding of the data is done entirely within the function calls. The C++ database client has full access over the entire database, placing the burden of ensuring security on the Perl scripts. The Perl scripts are designed to be used in a Netscape-secure environment, and to implement security by verifying the passwords provided by users against the passwords stored in the database. After the initial security check, the scripts merely pass the database key by CGI. Once a user is logged in, the Perl scripts check the HTTP_REFERER Unix environment variable to ensure that they are being called by the proper script. This is done to ensure that each script after the password prompt is being called by another script that is part of the grade browser. This way no one can write a script that injects the proper data into one of the internal scripts to bypass the password screens. Each Perl script generates one HTML screen, and passes its input through CGI to the next Perl script.
**Structure of the Perl and CGI based package**

**Student grade browser side:**

<table>
<thead>
<tr>
<th>Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stud.login.html</td>
<td>Generates the initial student login screen. It then calls studentinfo.cgi with the login information.</td>
</tr>
<tr>
<td>studentinfo.cgi</td>
<td>Checks the student's password. If the student fails the password check, a rejection screen is displayed. Otherwise, an information screen displaying the current student data and all classes in which the student is enrolled is shown. Two buttons call modify.stud.info.cgi and send.grades.cgi.</td>
</tr>
<tr>
<td>modify.stud.info.cgi</td>
<td>Displays a screen containing all user data in text input boxes. The user is then allowed to modify certain portions of their student info. Passes modified student info to commit.change.cgi.</td>
</tr>
<tr>
<td>commit.change.cgi</td>
<td>Takes the modified data from modify.stud.info.cgi and alters the information contained in the database. Reports any changes made, and has a button to call studentinfo.cgi once more.</td>
</tr>
<tr>
<td>send.grades.cgi</td>
<td>Displays all grades recorded in the database for the particular student. A button at the bottom calls studentinfo.cgi when the student is through viewing the information.</td>
</tr>
</tbody>
</table>

**Instructor grade browser/recorder side:**

<table>
<thead>
<tr>
<th>Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>login.cgi</td>
<td>Displays a login screen with all instructor names, then prompts the instructor to click the radio button beside their name and enter their password.</td>
</tr>
<tr>
<td>show.classes.cgi</td>
<td>Checks the user password obtained from login.cgi. If it is incorrect, a rejection screen is displayed. Otherwise, a list of all classes taught by the instructor is displayed, and the user is prompted to select one. Calls class.actions.cgi with the selected class.</td>
</tr>
<tr>
<td>class.actions.cgi</td>
<td>Displays a table, in grade book format, of all students, assignments, and grades. Radio buttons are beside all students and assignments. From this screen, the instructor may grade all students on one assignment, grade one student on all assignments, add, modify, or delete an assignment, or add or delete a student. Calls process.action.cgi with the selected student, assignment, and action.</td>
</tr>
<tr>
<td>process.action.cgi</td>
<td>Displays an intermediate screen if more data is required for the instructor's action (passed in from class.actions.cgi). For delete actions, asks for verification. Once all information is obtained, calls one of the following scripts, depending on the action desired: add.assign.cgi, add.student.cgi,</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>add.assign.cgi</td>
<td>Adds a new assignment to the database. Calls class.actions.cgi after giving a confirmation message.</td>
</tr>
<tr>
<td>add.student.cgi</td>
<td>Adds a student to the database. If the student's student ID number is not found in the database, the script creates a new entry for the student. Otherwise, it links the already existing entry to the current class. Calls class.actions.cgi after giving a confirmation message.</td>
</tr>
<tr>
<td>edit.assign.cgi</td>
<td>Alters the information in an already existing assignment in the database. Calls class.actions.cgi after giving a confirmation message.</td>
</tr>
<tr>
<td>grade.by.assign.cgi</td>
<td>Inserts/alters grades in the database for all assignments across a given assignment. Calls class.actions.cgi after giving a confirmation message. Deletes an assignment from the database. Calls class.actions.cgi after giving a confirmation message.</td>
</tr>
<tr>
<td>grade.by.student.cgi</td>
<td>Inserts/alters grades in the database for all assignments across a given assignment. Calls class.actions.cgi after giving a confirmation message.</td>
</tr>
<tr>
<td>remove.assignment.cgi</td>
<td>Deletes an assignment from the database. Calls class.actions.cgi after giving a confirmation message.</td>
</tr>
<tr>
<td>remove.student.cgi</td>
<td>Removes a student from the current course. Calls class.actions.cgi after giving a confirmation message.</td>
</tr>
</tbody>
</table>

**Grading system based on Java applets and Java server**

This solution involves two applications that support all the functions of the CGI script-based solution described above. One application is used by the instructor to maintain the course data. The other application is used by the student or administrator to browse the data.

The design of this system uses the Client/Server architecture. In this architecture the responsibilities are separated between the client and the server. The client makes requests of the server and the server will act on those requests and possibly reply to the client. This architecture can be used when the client wishes to off-load work to the server or when the server has resources that the client does not have access to.
In the case of this project, the server has access to the database while the client does not.

The Role of Web Browser

A web browser is a tool used to display HTML pages. The web browser uses the HTTP protocol to load pages over the network and display them to the user. Recently, browsers have added the functionality to execute Java applets. Here is a graphical representation of the system:

![Graphical representation of the system](image)

The components of the system are:

<table>
<thead>
<tr>
<th>Label</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>WWW Server</td>
<td>The WWW server is the machine serving the HTML pages. This machine is a Sun box running Solaris. The type of machine is only important in this case since one component of the system is an application running on this machine.</td>
</tr>
<tr>
<td>B</td>
<td>Java Data Server Application</td>
<td>The Java server application is used to handle communication with the Java applets. The Java applets communicate to this server. This server acts on the behalf of the applets to submit/retrieve data from the database. The server also provides a central point to maintain security for the system.</td>
</tr>
<tr>
<td>C</td>
<td>Sybase Database</td>
<td>The Sybase database is used to store all data used within this system. Tables are used to store instructor and student information as well as all scores. The database can exist on any machine on the network.</td>
</tr>
<tr>
<td>D</td>
<td>Instructor Applet</td>
<td>The instructor applet is a Java applet used by the instructor's to record students grades. The instructor can</td>
</tr>
</tbody>
</table>
execute this applet through any Java enabled browser.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Student Applet</td>
</tr>
<tr>
<td>F</td>
<td>Java Server to Sybase</td>
</tr>
<tr>
<td>G</td>
<td>Java Applets to Java Server Application</td>
</tr>
</tbody>
</table>

We will now describe each of the components of the system in detail and discuss the implementation as well as the justification for the design.

**Java Data Server**

The Java Data Server is a long running Java application. It executes on the WWW server. It has the responsibility of handling requests from the clients (the applets). It is not an applet but an application which can be executed from the command line. This application waits for the client to connect on a specified port. For each incoming connection, the application spawns a thread to handle the communication with the client. The client/server combination have a predefined protocol. The conversation between the two is initiated by a client request and followed by a server response. The set of requests is predefined.

For security, before the client can request/update data, a login request message must be sent. The server will validate the client at that time. Since all data flows through this application and the applets never communicate directly with the database, data security is maintained. This application is required to run on the WWW server. The reason for this is a Java security restriction. The current Java security model requires that the only machine that a client applet can connect to is the machine that served the applet. This restriction may be lifted in later versions of Java.

The data server has the responsibility of interacting with the database. Since the application is a Java application, we need a means to communicate with the Sybase database, which only provides a C interface. The solution to this problem is to use a JDBC driver. JDBC is an object-oriented interface to an arbitrary database. The JDBC driver is a driver for a specific database. Using the JDBC we can program to a database without knowing the type of database (so long as a JDBC driver exists for the database). This allows the Java programmer to remove himself from the details of programming to a specific C interface. The JDBC Sybase driver can be viewed as:
This represents the JDBC driver used for this project. The JDBC driver uses the Sybase OpenClient interface to communicate to the database. The OpenClient interface is for Solaris. However, should this application be targeted for another database server or platform, we could replace the JDBC driver with another. Since we are never using native operating system calls, or calls to a specific database interface, the application is platform independent. JDBC encapsulates the interaction with the native drivers allowing us to plug in new drivers with little effort should the need arise. The server uses JDBC to respond to client requests. Client requests either request data from the database or ask the server to make updates to the database. The server parses each message from the client and then takes the appropriate action. If the call to JDBC fails, the server will respond to the client with an error code. Data integrity is maintained since only one person can have any given course open at a time. This ensures that modifications are not lost if two people want to modify the same course simultaneously.

The data server also allows the client to be unaware of the SQL involved with each request. The information could be stored in a flat file and the interface would still be the same for the client. Placing all of the knowledge of the data source on the server rather than the client encapsulates that information and allows the client to be unchanged should the model change. This configuration also provides for a step towards the use of ORBs to implement the backend of this system. Moving the data model specific code off of the client and into the server is in line with an ORB implementation. An ORB implementation would mean that the server is replaced with an ORB, which returns back objects to our clients rather than the predefined packets my server returns. This appears to be the goal in an object-oriented environment since it completely separates the client from the source of the information.

**Instructor Applet**

The instructor applet is used by the instructor to maintain scores for their students. The applet provides a graphical user interface that allows instructors to easily maintain their grades.

The instructor applet communicates with the data server to retrieve and persist data. Since the instructor applet runs within the WWW browser, instructors can have access to the applet from any Java-enabled browser. This allows them access to the system from
anywhere on campus or even at home. The other reason they can have this type of access to the applet is because the data is stored in a database rather than a file on their local machine. Storing the data in a central location provides both security and flexibility to the instructors.

The communication between the applet and the server is done through TCP/IP. The applet opens a socket connection with the web server to connect to the server application. The applet sends request packets to the server through this socket connection. It listens to this socket for a response and then handles it appropriately.

The instructor applet offers a graphical user interface to the instructor. The applet has typical graphical widgets such as pulldown menus, dialogs, and a spreadsheet type interface for entering grades. The applet allows the instructor to add new assignments. The application allows instructors to have several different courses they are responsible for. For each course, the instructor can set the name of the course and create categories for the course. Each category represents a set of assignments that make up a percentage of the overall grade. The instructor can choose only one category if he/she does not require this feature. Each assignment can have an arbitrary number of points as well as a weight associated with it. For example, an instructor may create a course with the following categories: homework at 40% and exams at 60%. This configuration states that homework assignments represent 40% of the overall grade and that exams represent 60% of the overall grade for the course.

Instructors may add assignments and students to their courses. Each student added to the course requires a name, student id, e-mail, and some additional information. Although all of this data is not used presently, future enhancements may allow for automated e-mail generation to students or other features. Assignments may be added to the course at any time. Each assignment is represented by a name, a date, the total number of points, the weight, and finally the category it falls into. The number of possible points should be the maximum possible for the assignment. The weight defaults to 1.0. This weight represents the value of this assignment versus other assignments in the same category. The weight is used to weigh one assignment more than another without changing the total points possible. For example, an assignment worth 50 points has a weight of 2.0 associated with it. When the overall average is computed, that assignment will actually be worth 100 points and the points for each student will be doubled as well.

During the semester, students and assignments can be both added and removed. Also, assignments may be modified during the semester. This allows for the weights or point values for assignments to be modified throughout the semester. Student scores can be modified during the semester as well. By selecting a student and choosing the "Scores for student" menu choice, an instructor can view all grades for a student and make any necessary changes. The instructor can also view all scores for a given assignment and make modifications. These options are designed for easy entry of grades into the system. The final option provides a spreadsheet type interface to modify all scores. By allowing all
information for a given course to be modified throughout the semester, the application remains very flexible to the needs of each individual instructor.

Another piece of functionality in the system is the ability to generate a graph displaying the scores for a student's plotted against the average scores for the classroom. The idea here is to easily evaluate a student's performance compared to his/her peers. Later versions will support a variety of configurable charts to display graphically the information for the course.

Basically, the graphical user interface provides a simple and easy way for an instructor to manage the data associated with the class. There shouldn't be any additional effort over the instructor's current method of recording scores. The instructor application provides the instructor with the ability to easily maintain the scores of their students. The graphical user interface provided to the instructor makes keeping track of student scores both easy and intuitive. The instructor application stores all data in a Sybase database. By storing the data in a database, we insure data integrity for the system. All interaction with the database is hidden from the user. This makes the database integration seamless. It also allows for other data storage mechanisms to be used without affecting the clients.

The instructor applet is also implemented as a Java application. This is a standalone application which does not run through the browser. This application can be used to save the information to a file on the local machine. This implementation has the same look and feel as the applet but a slightly different purpose. This application does not use the database, so students cannot browse their grades using a browsing applet. However, this application does allow the instructor to export his/her grades in the form of an HTML file. This application is designed for settings which may not need the browsing capabilities or do not have the database available for the entire system.

**Student Applet**

The student applet is a grade browsing applet. It allows a student to browse his/her grades for various courses from within a web browser. Each student can login to the applet and see his/her up to date scores. This browser is also available to instructors or administrators. They can view students scores as well if they have the proper authority.

The browsing application provides the ability to browse course information. The browsing application is geared towards students, but it is available to instructors and even authorized administrators. The browsing applet uses a graphical user interface to provide very intuitive navigation. The central idea is that the application should be straightforward and easy to use. The browsing application allows a user to login and then select an available course. (Instructors and administrators will first choose a student.) After choosing a course, the user will be prompted with a menu of choices. The first option is to view the scores associated with the student. This option will generate a table listing the student's scores and his/her averages for each category and his/her overall average. Also displayed in this table will be the minimum, maximum, and average for the entire course for each score and average. This allows the user to compare the student's performance
statistically with the class as whole. The next option available to the user is to view the scores for all students in the course. However, names will not be displayed for other students if the user is a student. (They will be scrambled to protect each student's privacy.) This option allows users to get a more detailed picture of the performance of the class as a whole.

In the future, a more interactive graphing widget can be used to allow the student to determine what data should be graphed and how. This is where the interactivity of Java will shine.

The student applet uses the same means as the instructor applet to communicate with the server. Of course, not all of the messages are the same. Some messages are shared between the clients, though. This shows the reusability this architecture allows. At a later time, another client applet may be developed which could use the same server.

**Comparison of the Java applets with CGI scripts**

The features Java provides are perfectly suited for this type of application. For a cross platform application which needs to be easy to distribute and install, there is currently no better language. The development time is greatly reduced since only one API is ever needed. Implementing this in any other language would require different sets of code for each platform. Java has built-in networking, which makes the client/server architecture much easier to implement. Built-in threading also makes a professional application easier to develop. Java provides the JDBC interface. This allows the programmer to target most any database by using a single interface. This greatly increases the ability to reuse the code for the application. It also allows the system to be more flexible since the server is not tied down to a particular type of server.

On the other hand, CGI scripts are commonly used to implement application type behavior on the WWW. These scripts are used by the WWW server to generate HTML pages on the fly in response to the user's interaction with forms on the HTML pages. CGI scripts are programs that execute on the server in response to a user's interaction with an HTML page. They work by allowing the HTML page to specify a CGI script to execute in response to a user's interaction with a form. CGI scripts can take parameters as well as receive the user's input from the HTML form. CGI scripts are commonly used to process forms from users and then to generate a new HTML page in response, to display to the user. For example, a company might request those viewing their web pages to fill in a
form to register. A CGI script could be used to process this information into a database and then output an HTML page thanking the person for registering. The CGI script is a very useful tool for form processing; however, it is not an ideal solution for an application such as ours. The reason for this is that HTML pages are a static medium. Their interaction with the user is quite limited. Also, since the processing is done on the server, the response to the user may be slow. (Also, the load on the server will increase.)

Another reason CGI scripts aren't well suited for an interactive application is their lack of a state. That is, an HTML page doesn't have knowledge of the path the user took to get to that page and the information entered along the way. This lack of a state is what separates a CGI script generated page from a Java applet. The Java applet can have a state. It can store information and dynamically respond to the user.

CGI scripts have been able to work around this limitation by passing state information as the parameters to other CGI scripts. Obviously, this is not a very clean solution. Furthermore, it also is not a very secure solution. A primary piece of information that might be contained in the state of an applet is whether or not they have logged in and what their authentication id is. Using CGI scripts, this information would be seen by the user and thus it would be insecure.

It is neither possible or fair to condemn CGI scripts as a whole, however. While their scope is limited to more static interfaces and they probably aren't as good a solution as an applet, many powerful applications have been created with CGI scripts. In particular, we believe that our grading program based on scripts will be competitive with the Java solution, specifically among the users already accustomed to CGI script applications.

Closing Remarks
We will introduce our grading system to the College of Pharmacy for use in the Spring 1997 semester. Initially, this system will be applied by a selected group of faculty and instructors. Data for their classes will be preinserted by the SyBase database administrators from existing on-line class rosters. The CGI script version will be used for trials. This choice is dictated by a better familiarity of the users with this type of application.

At the same time we are planning to start phasing in the Java-based grading system. To facilitate this process and help train future users, a stand-alone Java application for the instructors has been developed. This application shares most of the same code as the applet version. It does, however, target a different audience. This application is primarily designed for instructors either not interested in providing the browsing features available in the system or those who currently do not have access to a database server to use for the grading system. This application does not use the database for storing information. Instead, it reads and writes the data on the local disk. This is the major difference which separates this version from its applet counterpart.
The stand-alone application provides some additional functionality to the instructor that does not appear in the applet version. The instructor has the ability to output the course information into an HTML file or a plain text file. This allows the instructor to view and/or distribute course information to the students enrolled in the class. It also supplies a convenient mechanism for students to receive grade-related information from the instructor.

We hope to report on the results of the trials at the conclusion of the Spring semester.