

University of Montana

ScholarWorks at University of Montana

Graduate Student Theses, Dissertations, &
Professional Papers

Graduate School

1996

Submitting service

Lin Ling

The University of Montana

Follow this and additional works at: <https://scholarworks.umt.edu/etd>

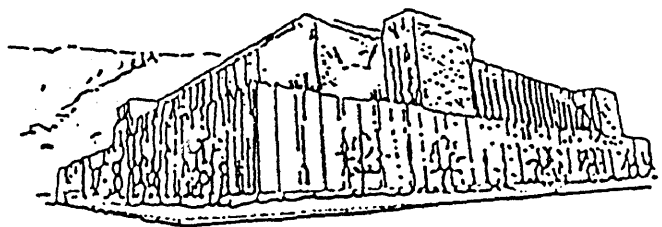
Let us know how access to this document benefits you.

Recommended Citation

Ling, Lin, "Submitting service" (1996). *Graduate Student Theses, Dissertations, & Professional Papers*. 5126.

<https://scholarworks.umt.edu/etd/5126>

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact scholarworks@mso.umt.edu.



Maureen and Mike
MANSFIELD LIBRARY

The University of **MONTANA**

Permission is granted by the author to reproduce this material in its entirety, provided that this material is used for scholarly purposes and is properly cited in published works and reports.

*** Please check "Yes" or "No" and provide signature ***

Yes, I grant permission

No, I do not grant permission

Author's Signature Jim Ling

Date Aug. 6th, 96

Any copying for commercial purposes or financial gain may be undertaken only with the author's explicit consent.

Submitting Service

by

Lin Ling

B.S. Harbin Institute of Technology, 1992

presented in partial fulfillment of the requirements

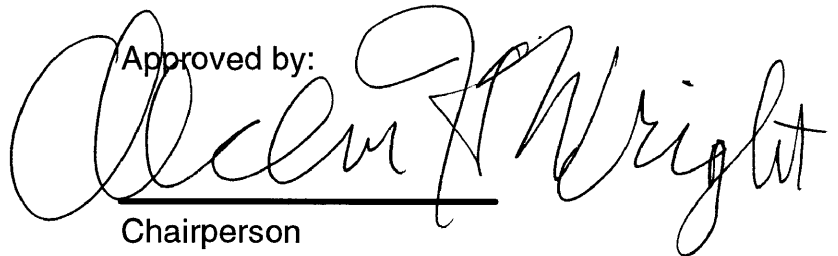
for the degree of

Master of Science

The University of Montana

1996

Approved by:



Dean J. Wright

Chairperson



Dean, Graduate School

8/7/96

Date

UMI Number: EP40590

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

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



UMI EP40590

Published by ProQuest LLC (2014). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against unauthorized copying under Title 17, United States Code



ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Ling, Lin, M.S., July 1996

Computer Science

Submitting Service (89 pp.)

Director: Dr. Alden Wright

A handwritten signature in black ink, appearing to read 'Alden Wright', written over the printed name.

The submitting service allows students to submit assignments to their instructors electronically. The entire project is a client-server application. It has four program components, the client program, the server program, the instructor program, and the administrator program. The client program is designed to be user friendly. It's easy for the students to use without instructions. The server program receives all requests from the client program and sends the feedback to the client program. The instructor and administrator program are for instructor and administrator use only. They are used to configure the server file system. The entire application is coded in Java. I did some prototypes in three different languages: Java, Visual Basic, and Visual C++. Since Java is system independent, it fits the requirements for this project. The documentation for the project concentrates on the user interface design chapter 4 and the detailed program design chapter 5. The interface design includes the client interface, the instructor interface, and the administrator interface design. The detailed program design is done using the Booch [1] method, and includes the client program, the server program, the instructor program, and the administrator program object-oriented design.

TABLE OF CONTENTS

| | |
|--|-----------|
| CHAPTER 1 INTRODUCTION | 1 |
| CHAPTER 2 REQUIREMENTS | 2 |
| CHAPTER 3 DESIGN ALTERNATIVES | 3 |
| 3.1 USE MAIL | 3 |
| 3.2 WHY NOT LET STUDENTS COPY THEIR ASSIGNMENT TO THE INSTRUCTOR'S UNIX DIRECTORY? | 3 |
| CHAPTER 4 THE USER INTERFACE | 5 |
| 4.1 <u>CLIENT INTERFACE</u> | 5 |
| 4.1.1 <i>Command Line</i> | 5 |
| 4.1.2 <i>Requirements</i> | 5 |
| 4.1.3 <i>The Client Dialog Interaction</i> | 6 |
| 4.1.4 <i>The Client Interface</i> | 8 |
| 4.2 <u>ADMINISTRATOR INTERFACE</u> | 16 |
| 4.2.1 <i>Command line</i> | 16 |
| 4.2.2 <i>Requirements</i> | 16 |
| 4.2.3 <i>Dialog interaction</i> | 17 |
| 4.2.4 <i>Interface</i> | 17 |
| 4.3 <u>INSTRUCTOR INTERFACE</u> | 20 |
| 4.3.1 <i>Command line:</i> | 20 |
| 4.3.2 <i>Requirement</i> | 20 |
| 4.3.3 <i>Instructor program dialog interaction</i> | 21 |
| 4.3.4 <i>Instructor program interface</i> | 22 |
| 4.3.5 <i>Storage For Server</i> | 30 |
| CHAPTER 5 THE DETAILED DESIGN | 32 |
| 5.1. <u>CLIENT PROGRAM</u> | 34 |
| 5.1.1 <i>Classes</i> | 34 |
| 5.1.2 <i>Module Diagram</i> | 45 |
| 5.1.3 <i>Object Diagram</i> | 47 |
| 5.1.4 <i>State Transition Diagram</i> | 51 |
| 5.1.5 <i>Interaction Diagram</i> | 57 |
| 5.2. <u>SERVER PROGRAM</u> | 58 |
| 5.2.1 <i>Classes</i> | 58 |
| 5.2.2 <i>Module Diagram</i> | 63 |
| 5.2.3 <i>Object Diagram</i> | 64 |
| 5.2.4 <i>State Transition Diagram</i> | 67 |
| 5.2.5 <i>Interaction Diagram</i> | 68 |
| 5.3. <u>ADMINISTRATOR PROGRAM</u> | 70 |
| 5.3.1 <i>Classes</i> | 70 |
| 5.3.2 <i>Module Diagram</i> | 73 |
| 5.3.3 <i>Object Diagram</i> | 74 |
| 5.3.4 <i>State Transition Diagram</i> | 75 |

| | |
|---|-----------|
| 5.4 <u>INSTRUCTOR PROGRAM</u> | 76 |
| 5.4.1 <i>Classes</i> | 76 |
| 5.4.2 <i>Module Diagram</i> | 84 |
| 5.4.3 <i>Object Diagram</i> | 85 |
| 5.4.4 <i>State Transition Diagram</i> | 87 |
| CHAPTER 6 IMPLEMENTATION | 88 |
| 6.1 CLIENT PROGRAM | 88 |
| 6.2 SERVER PROGRAMS | 88 |
| 6.3 PROTOTYPING EFFORT | 88 |
| 6.4 WHY NOT USE VISUAL C++ OR VISUAL BASIC..... | 89 |
| 6.5 PROJECT FEATURE | 89 |
| REFERENCE: | 89 |

LIST OF FIGURES

| | |
|---|----|
| FIG 4. 1 DIALOG ICONS | 6 |
| FIG 4. 2 LOGIN DIALOG INTERACTION..... | 7 |
| FIG 4. 3: SUBMITTING DIALOG BOX INTERACTION | 8 |
| FIG 4. 4: CLASS SELECTCOURSE INTERFACE..... | 9 |
| FIG 4. 5: CLASS MESSAGEIALOG INTERFACE..... | 9 |
| FIG 4. 6: CLASS SELECTLOGIN INTERFACE | 10 |
| FIG 4. 7 CLASS NEWACCOUNTLOGIN INTERFACE | 10 |
| FIG 4. 8 CLASS NEWLOGINIALOG INTERFACE | 11 |
| FIG 4. 9 CLASS NORMALLOGIN INTERFACE | 11 |
| FIG 4. 10 CLASS CLIENTINTERFACE INTERFACE..... | 11 |
| FIG 4. 11 CLASS SUBMIDIALOG INTERFACE..... | 12 |
| FIG 4. 12 FILE DIALOG | 12 |
| FIG 4. 13 CLASS VERIFYIALOG INTERFACE | 13 |
| FIG 4. 14 SUBMISSION FEEDBACK | 13 |
| FIG 4. 15 CLASS CHANGEPASSWORD INTERFACE | 14 |
| FIG 4. 16 CLASS LISTFILE INTERFACE | 14 |
| FIG 4. 17 CLASS VERIFYDELETE INTERFACE | 15 |
| FIG 4. 18 ADMINISTRATOR DIALOG INTERACTION | 17 |
| FIG 4. 19 CLASS ADMINISTRATORCONFIG INTERFACE | 18 |
| FIG 4. 20 CLASS ADDCOURSEIALOG INTERFACE | 18 |
| FIG 4. 21 CLASS DELETECOURSEIALOG INTERFACE | 19 |
| FIG 4. 22 CLASS SELECTCOURSEIALOG INTERFACE..... | 19 |
| FIG 4. 23 CLASS EDITCOURSEIALOG INTERFACE..... | 20 |
| FIG 4. 24 INSTRUCTOR DIALOG BOX INTERACTION | 21 |
| FIG 4. 25 INSTRUCTOR DIALOG BOX INTERACTION 2 | 22 |
| FIG 4. 26 CLASS INSTRUCTORCONFIG INTERFACE..... | 23 |
| FIG 4. 27 CLASS SELECTCOURSEIALOG INTERFACE..... | 23 |
| FIG 4. 28 CLASS OPERATIONS INTERFACE | 24 |
| FIG 4. 29 FILE DIALOG | 24 |
| FIG 4. 30 CLASS ADDUSERBYFILE INTERFACE | 25 |
| FIG 4. 31 CLASS ADDUSERIALOG INTERFACE | 26 |
| FIG 4. 32..... | 26 |
| FIG 4. 33 CLASS DELETEUSERIALOG INTERFACE..... | 27 |
| FIG 4. 34 CLASS EDITUSERIALOG INTERFACE | 27 |
| FIG 4. 35 CLASS USERINFOIALOG INTERFACE | 28 |
| FIG 4. 36 CLASS DELETEASSIGNMENTIALOG INTERFACE | 29 |
| FIG 4. 37 CLASS ADDASSIGNMENTIALOG INTERFACE..... | 29 |
| FIG 4. 38 SELECTASSIGNMENTIALOG INTERFACE..... | 30 |
| FIG 4. 39 EDITASSIGNMENTIALOG INTERFACE..... | 30 |
| FIG 4. 40 SERVER STORAGE..... | 31 |
| | |
| FIG 5. 1 PROGRAM INTERACTION | 33 |
| FIG 5. 2 CLIENT CLASS DIAGRAM | 34 |
| FIG 5. 3 CLASS ICON | 35 |
| FIG 5. 4 HAS RELATION | 35 |
| FIG 5. 5 USE RELATION | 35 |
| FIG 5. 6 MODULE DIAGRAM..... | 45 |
| FIG 5. 7 MODULE ICONS | 46 |

| | |
|--|----|
| FIG 5. 8 OBJECT DIAGRAM | 47 |
| FIG 5. 9 OBJECT ICON | 48 |
| FIG 5. 10 OBJECT DIAGRAM 2..... | 49 |
| FIG 5. 11 OBJECT DIAGRAM 3..... | 50 |
| FIG 5. 12 OBJECT DIAGRAM: SUBMIT..... | 51 |
| FIG 5. 13 STD ICONS | 52 |
| FIG 5. 14 STD: CHANGEPASSWORD..... | 53 |
| FIG 5. 15 STD: CLIENTINTERFACE | 54 |
| FIG 5. 16 STD: CLIENTTRANSFER | 55 |
| FIG 5. 17 STD: TRANSFERMESSAGE..... | 56 |
| FIG 5. 18 SUBMIT FILE INTERACTION DIAGRAM..... | 57 |
| FIG 5. 19 SERVER CLASS DIAGRAM..... | 58 |
| FIG 5. 20 SERVER MODULE DIAGRAM | 63 |
| FIG 5. 21 LOGIN OBJECT DIAGRAM..... | 64 |
| FIG 5. 22 OPERATION OBJECT DIAGRAM | 65 |
| FIG 5. 23 SUBMIT OBJECT DIAGRAM..... | 66 |
| FIG 5. 24 HANDLEINFO STATE TRANSITION DIAGRAM | 67 |
| FIG 5. 25 THREADEDSERVERHANDLER STATE TRANSITION DIAGRAM | 68 |
| FIG 5. 26 USER LOG IN INTERACTION DIAGRAM | 69 |
| FIG 5. 27 USER SUBMIT FILE INTERACTION DIAGRAM..... | 69 |
| FIG 5. 28 ADMINISTRATOR CLASS DIAGRAM | 70 |
| FIG 5. 29 ADMINISTRATOR MODULE DIAGRAM | 73 |
| FIG 5. 30 ADMINISTRATOR OBJECT DIAGRAM | 74 |
| FIG 5. 31 ADMINISTRATORCONFIG STATE TRANSITION DIAGRAM | 75 |
| FIG 5. 32 INSTRUCTOR CLASS DIAGRAM | 76 |
| FIG 5. 33 INSTRUCTOR MODULE DIAGRAM..... | 84 |
| FIG 5. 34 SELECT COURSE OBJECT DIAGRAM | 85 |
| FIG 5. 35 OPERATIONS OBJECT DIAGRAM..... | 86 |
| FIG 5. 36 OPERATION STATE TRANSITION DIAGRAM..... | 87 |

Chapter1 Introduction

My first question is why students need a submitting service. Student could submit assignments on paper, but for some course work, the paper doesn't capture all the information needed, especially for computer-oriented courses.

For example, students are given a word-processing assignment. They are assigned to submit a report by using word-processing techniques on the computer. It's very hard for the instructor to tell whether the students have used the correct word-processing techniques by just looking at the print out on paper. By using the submitting service, students must submit the original word-processing file. The instructor can grade the assignment easily.

Another example would be when students submit an executable file. There is no way for paper to represent an executable file; therefore, the executable file has to be submitted to the instructor electronically.

Also, submitting the assignment electronically will save a lot of paper and printing. Since the students can submit their assignments to the instructor directly without printing them on paper, vast amount of materials could be saved. Because the students submit the assignments electronically, printing is not required. The students do not have to worry about how to print, which always is a problem.

Electronic submitting is more time-efficient. The students can submit their assignments immediately after they finish them. The students do not have to wait for the next day in the class to submit the assignments, which also saves class time inevitably taken up during this process.

Chapter 2 Requirements

Since this application is going to be used by computer beginners, it must have a simple user interface.

Students submit assignments by using the client program. After the program is started, the program will ask the user to choose the file which is to be submitted. The user will be asked to verify this information.

The user will receive confirmation that the file has been successfully submitted. The specific requirements are as followed.

1. Students must be able to submit assignments electronically.
2. Students must be able to use the program package with little or no instruction.
3. Students must be able to receive feedback after the submission.
4. No student can access any other student's assignment.
5. No student can prevent or interfere with any other student's ability to submit assignments.
6. Submitted files must be grouped by course, section, and assignment.
7. Submitted files can be only accessed by the instructor.
8. Students have limited disk space for submission.
9. The students' disk space can be modified and limited by the instructor.
10. Students can delete their previous submissions.
11. The administrator is able to create the course root directory.
12. The client program must be able to run on a PC (MS Windows or Mac).
13. The server program must be able to run on a UNIX system.

Chapter 3 Design Alternatives

The project was designed as a client-server application. The entire project uses Java as the programming language. There are two design alternatives discussed below.

3.1 Use mail

The mail service is a possible solution for students submitting their assignments. The assumption is that the students know how to use mail with attachments. Since we are talking about computer beginners, this assumption may not be true. From the instructor's point of view, what happens if an instructor has 300 students in the class? The answer would be that the instructor will have at least 300 mail messages per assignment. It's easy to imagine that this would be a nightmare for the instructor to receive and save this much mail.

Another alternative for using mail would be to set up a separate program to receive the mail and to automatically reply to the sender. The students' files could be stored in a specified directory, but there is no way for students' to delete their previous submission. Therefore, I can not use mail for this project.

3.2 Why not let students copy their assignment to the instructor's UNIX directory?

First of all, the students might be able to use `Ws_FTP` to move their assignment from their PC to a UNIX work system. `Ws_FTP` is relatively more user-friendly compared to other implementations of FTP, but it still can not satisfy requirement 2. Students must be able to use the program package with little or no instruction. Since the students here are not computer-oriented, I can not expect them to be able to use either `Ws_FTP` or FTP properly.

Another problem for this option is a student's quota problem. The instructor can set up a subdirectory for students to submit their assignment, however if one student fills the block, the other students can not submit their assignments. This might prevent other students from submitting assignments; therefore, it could not meet requirement 5.

Furthermore, requirement 7 is unsatisfied. A student submits an assignment to a directory by using FTP. There are three types of access in the UNIX file system, (1) owner readable, (2) group readable, and (3) everyone readable. If the file is set to (1), it can only be accessed by the owner. If the file is set to (2), it can be accessed by a group of people. The group depends on the owner's group setting. If the file is set to (3), then everyone can read the file. If he/she wants the instructor to access it, it has to be (2) or (3) file access setting. If it's (2), all the students and the instructor are in one group, the file can be accessed by the students and the instructor. This makes it possible for a student to access others files. Certainly setting (3) will allow this to happen. Although the file's ownership can be changed, only super user can do that. Therefore, the submitted assignment can not be accessed by *only* the instructors.

Chapter 4 The User Interface

4.1 Client Interface

Purpose: Since the client program is going to be used by students with little computer knowledge, the interface of the client needs to be user-friendly.

4.1.1 Command Line

If the Java program code is stored in C:\clientClassDirectory\, the command line for the client program is as follows:

```
C:\clientClassDirectory\ java ClientTransfer serverAddress
```

This command line can be associated with an icon. For example, if the server is running on ninepipe.cs.umt.edu, the command line will be:

```
C:\clientClassDirectory\ java ClientTransfer ninepipe.cs.umt.edu
```

4.1.2 Requirements

The client interface and program are based on the following requirement.

1. Students are able to send files to server.
2. Students are able to select different courses.
3. Students are able to select assignments.
4. Students are able to have their individual accounts.
5. Students are able to change their passwords.
6. Students are able to delete previous files.

7. Students are able to know the limits of their quotas.

4.1.3 The Client Dialog Interaction

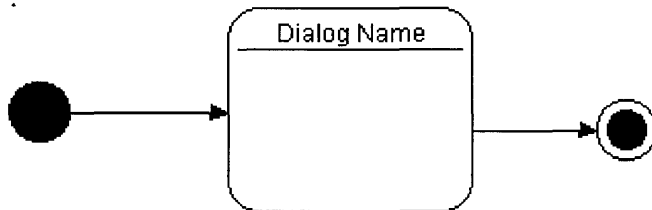


Fig 4. 1 Dialog Icons

Fig 4. 1 Dialog Icons shows the dialog interactions icons. The left most one is start icon, and the right most one is end icon. The center box represents the dialog box. The arrow is for message passing.

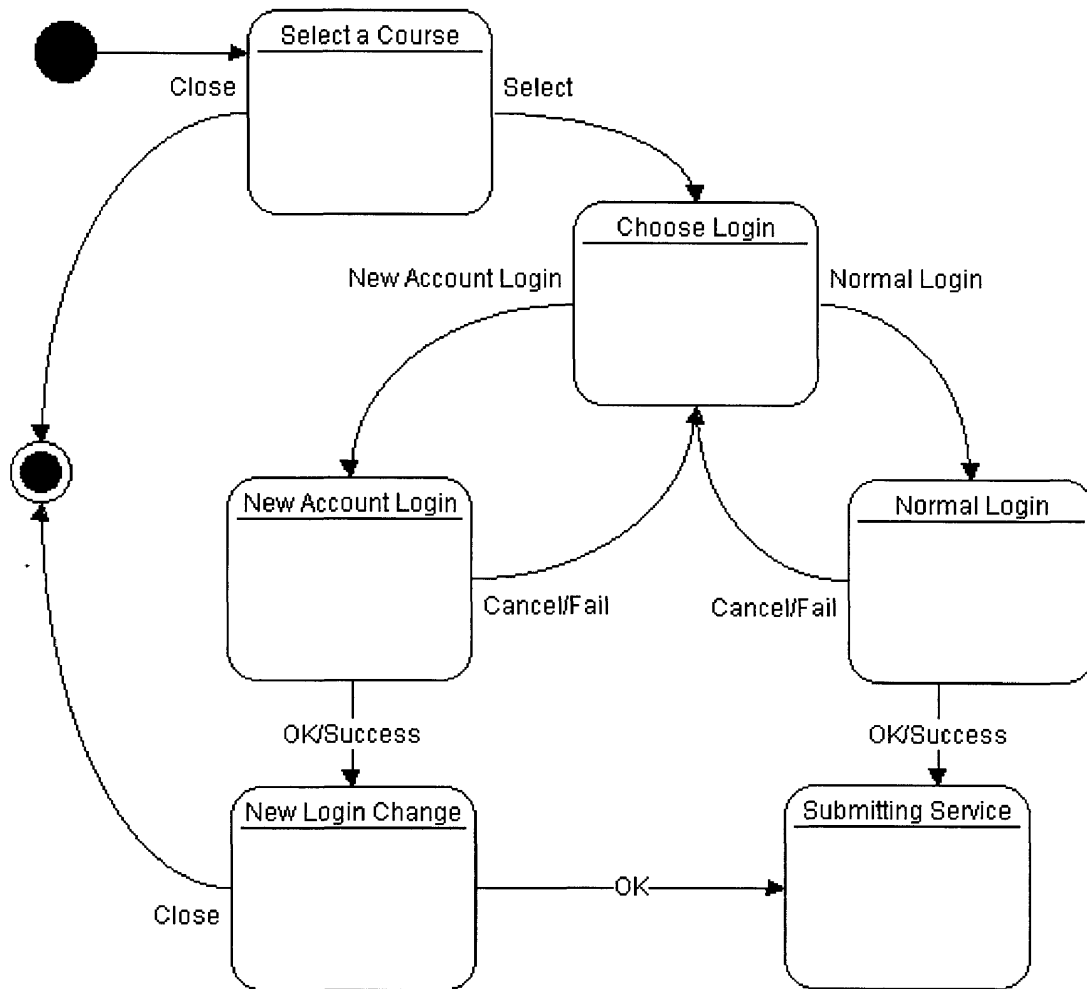


Fig 4. 2 Login Dialog Interaction

Fig 4. 2 Login Dialog Interaction shows the student log in process. It has normal login or new account login process.

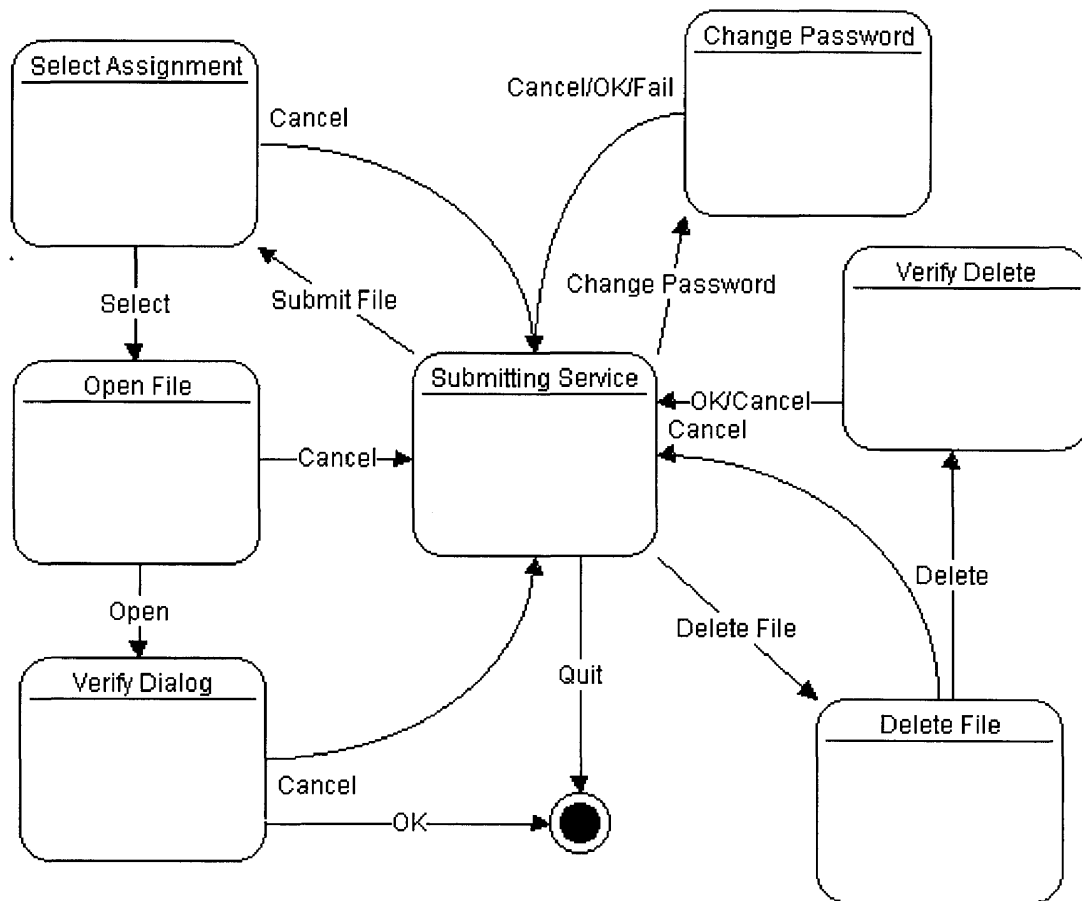


Fig 4. 3: Submitting Dialog Box Interaction

Fig 4. 3: Submitting Dialog Box Interaction shows the main services provided for the users.

Fig 4. 2 Login Dialog Interaction and Fig 4. 3: Submitting Dialog Box Interaction show the interactions between the client dialog boxes.

4.1.4 The Client Interface

4.1.4.1 Start the client program and select a course

First of all if a user enters the command line, the following window will show up on the screen.

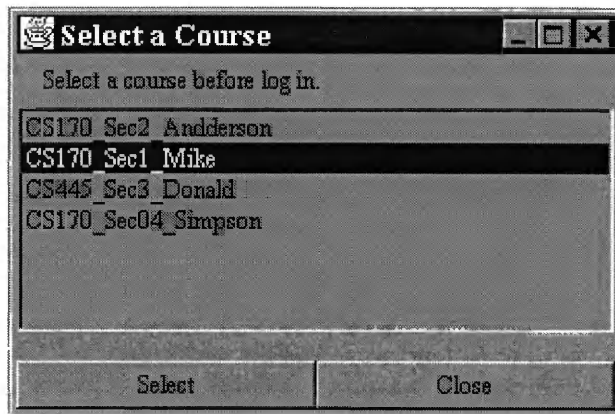


Fig 4. 4: Class SelectCourse Interface

In Fig 4. 4: Class SelectCourse Interface, the user is asked to select a course. The Select button allows the user to continue. If the Select button is pushed, and the user doesn't select any item, Fig 4. 5: Class MatDialog Interface will show up on the screen.

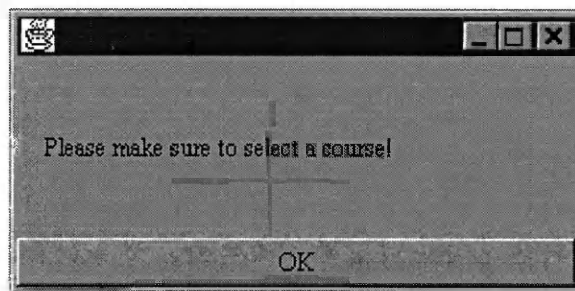


Fig 4. 5: Class MatDialog Interface

This is a message dialog box to tell the user what is wrong.

If the user makes a selection and clicks on the Select button, the Choose Login dialog box will show up.

4.1.4.2 Login process

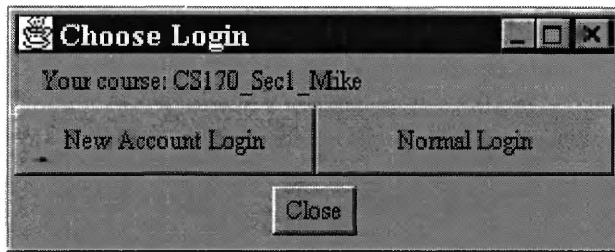


Fig 4. 6: Class SelectLogin Interface

To prevent students accessing each other's files, Only the one with the correct password can continue using the services. Fig 4. 6: Class SelectLogin Interface allows the user to select a login operation. Because the normal login and the first time login are different, this step is necessary and makes it easier to use.



Fig 4. 7 Class NewAccountLogin Interface

The first time user login is required to enter his/her student ID number as shown.

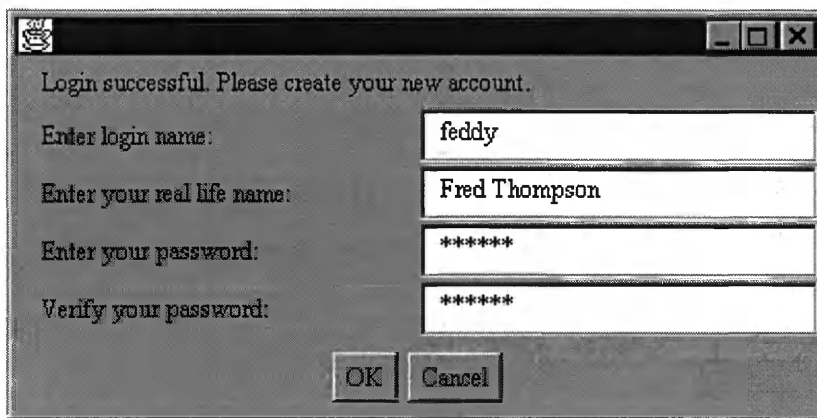
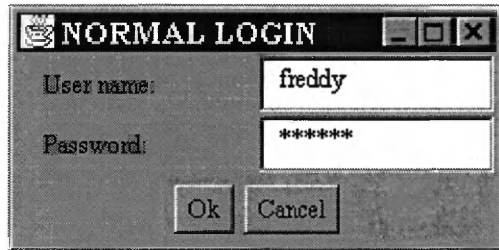


Fig 4. 8 Class NewLoginDialog Interface

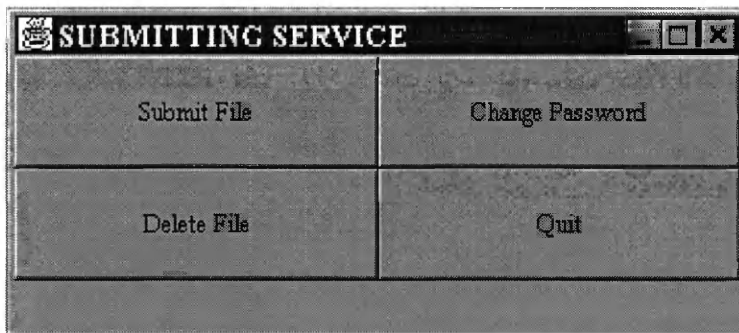
The following procedure allows the user to set up a real account which has his/her own login name, password and real name.

On Fig 4. 6: Class SelectLogin Interface if Normal Login button is pushed, another window will show up.

**Fig 4. 9 Class NormalLogin Interface**

This is the normal login for users who have previously logged in.

The login result will be displayed on a message box which is similar to Fig 4. 5: Class MessageDialog Interface. If login succeeds the following frame will show up.

**Fig 4. 10 Class ClientInterface Interface**

This operation box will let the user do all the things he/she wants to do.

First, to submit a file, the student pushes the Submit File button.

4.1.4.3 Select an Assignment

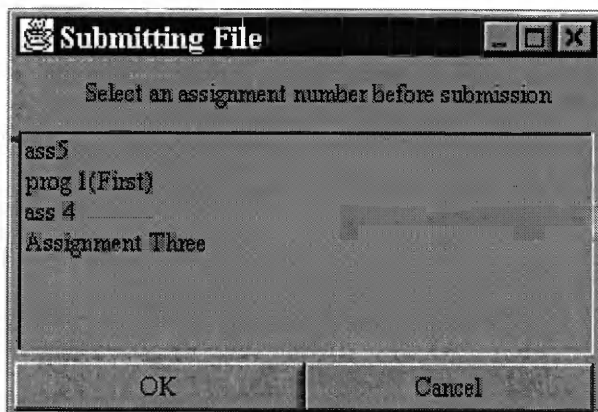


Fig 4. 11 Class SubmitDialog Interface

This box allows the user to select an assignment directory in which his/her submitting file will be stored.

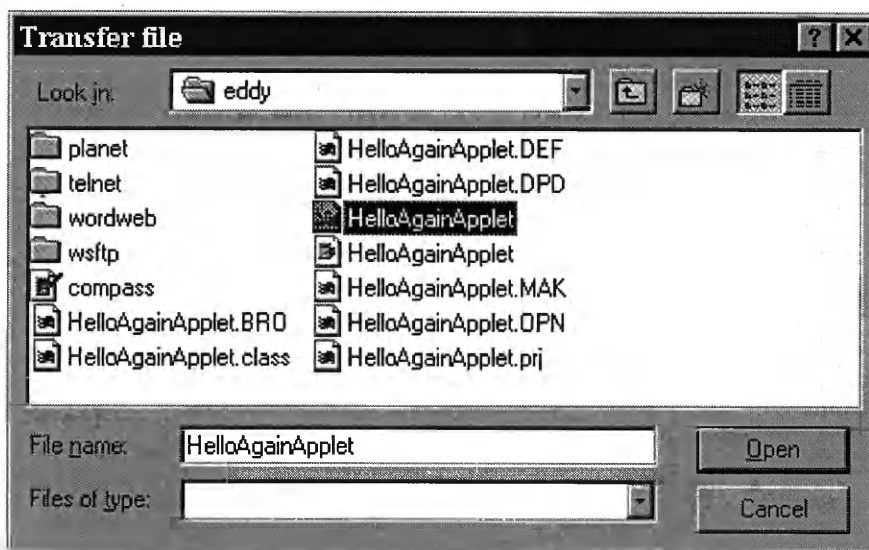


Fig 4. 12 File Dialog

This file dialog box allows the user to choose which file to send.

4.1.4.4 Verify the information and transfer the file to the server

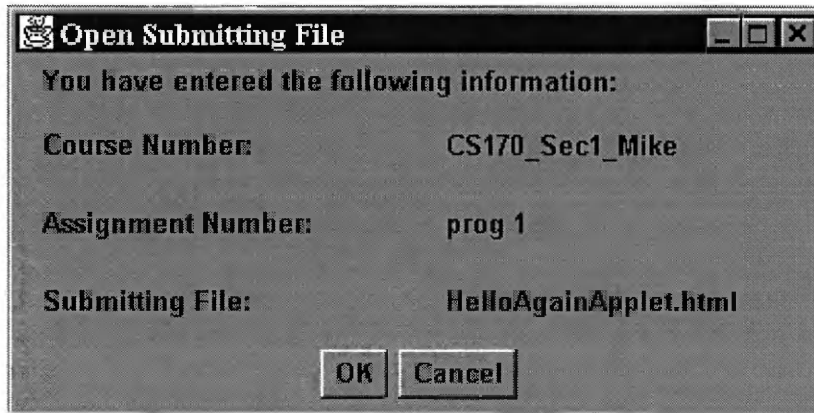


Fig 4. 13 Class VerifyDialog Interface

This is the confirmation box before the file is sent to the server. If it's correct, the student clicks on OK; otherwise, the student clicks on Cancel, and user can reenter it.

If OK is clicked:

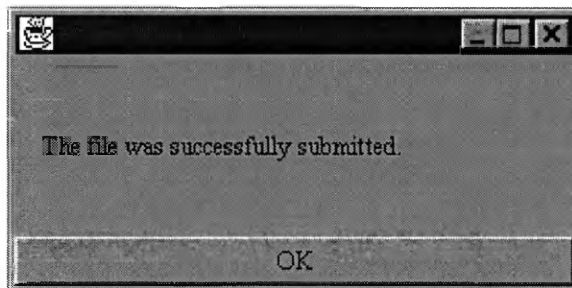


Fig 4. 14 Submission Feedback

4.1.4.5 Change user password

Fig 4. 15 Class ChangePassword Interface shows what happens if Change Password button is clicked:

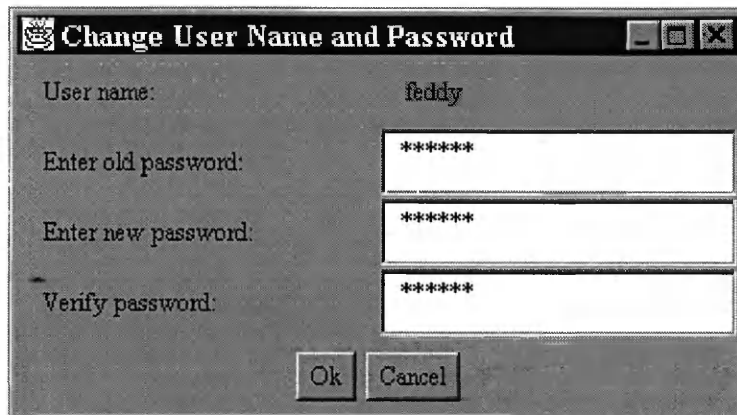


Fig 4. 15 Class ChangePassword Interface

This box will allow the user to change his/her password. A message box will verify that the password was successfully changed.

4.1.4.6 Delete user file and show quota

Fig 4. 16 Class ListFile Interface shows what happen if Delete File button is pushed:

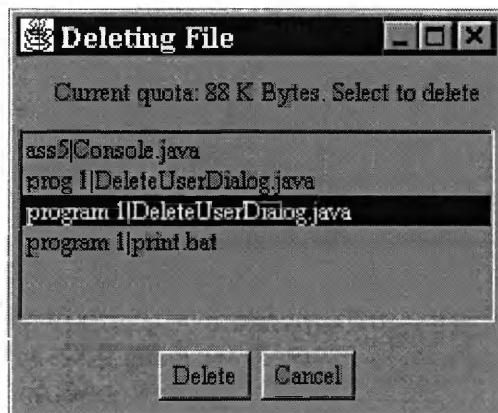


Fig 4. 16 Class ListFile Interface

This box will allow the user to delete his/her previous submissions, and it shows the user available quota. If the user selects file, DeleteUserDialog.java, and clicks on Delete button, the following verify dialog box will show.

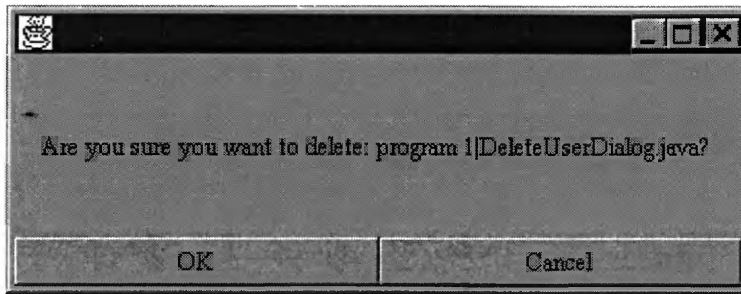


Fig 4. 17 Class VerifyDelete Interface

If Cancel button is pushed, the deletion is canceled. Otherwise, if OK is pushed, the deletion continues.

A message box will show the deletion result.

4.2 Administrator Interface

There is only one course root directory in the server file system. All the course directories are under course root directory. It is the administrator's responsibility to set up the working directories for staff. The administrator program is used only by the administrator to configure the course root directory.

4.2.1 Command line

A command line with the following format will start the administrator program.

UNIX: *java AdministratorConfig Directory1 Directory2*

The directories specify the absolute path of the course directory. There can be any number of directories as long as all the directories exist and in the order of parent to child. For instance, if the user wants to put the course directory under /work1/thompson, the command line would be:

UNIX: *java AdministratorConfig work1 thompson*

This command line can be simplified by putting it into a batch file, shell script, or making an icon for it.

4.2.2 Requirements

The requirements for the administrator interface and program are as follows.

1. Set up a course directory in a specific directory path.
2. Add a course.
3. Move a course.

4.2.3 Dialog interaction

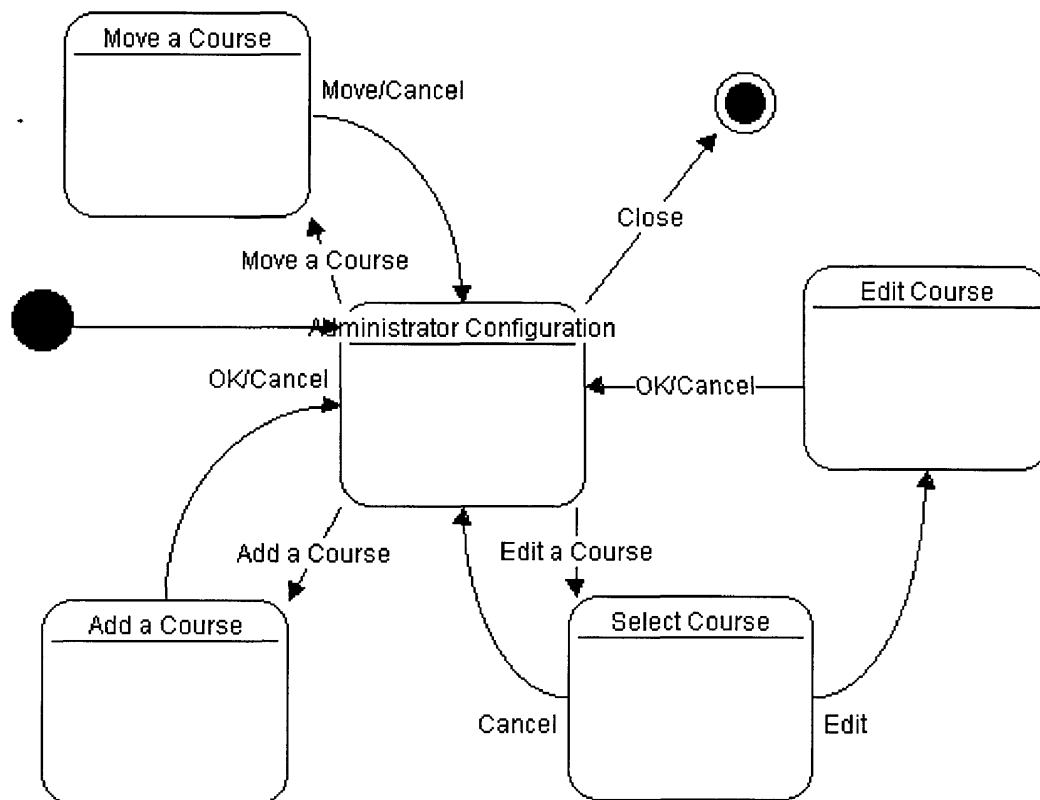


Fig 4. 18 Administrator Dialog Interaction

Fig 4. 18 Administrator Dialog Interaction shows the interactions between the dialog boxes.

4.2.4 Interface

4.2.4.1 Start the administrator program

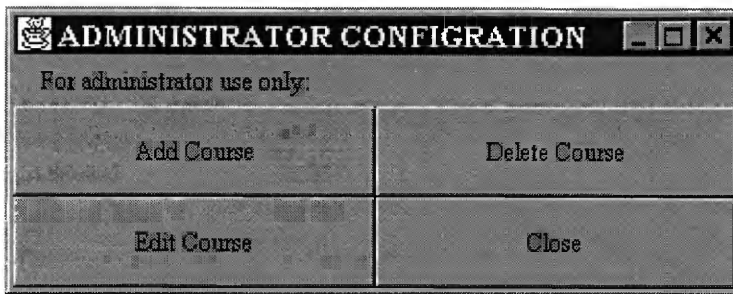


Fig 4. 19 Class AdministratorConfig Interface

This frame allows administrator to add a course, edit a course, or move a course.

If Add Course is clicked:

4.2.4.2 Add a course

 A dialog box titled "Add a Course" with standard window controls. It contains four text input fields stacked vertically. The first field is labeled "Course Number:" and contains the text "CS170". The second field is labeled "Section Number:" and contains "Sec04". The third field is labeled "Instructor Last Name:" and contains "Simpson". The fourth field is labeled "Quota (KB):" and contains "1000". At the bottom of the dialog are two buttons: "Ok" and "Cancel".

Fig 4. 20 Class AddCourseDialog Interface

There are four text fields for each course. The administrator completes these three fields and clicks on OK. A message box will display the add result.

Fig 4. 21 Class DeleteCourseDialog Interface shows what happens when Move Course button is clicked.

4.2.4.3 Delete a course

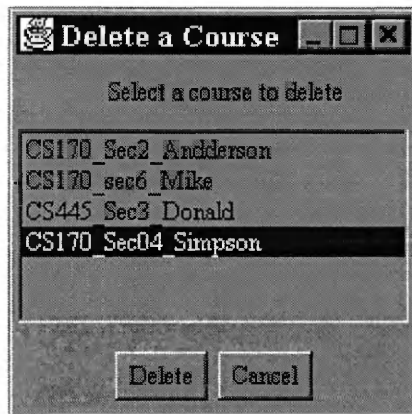


Fig 4. 21 Class DeleteCourseDialog Interface

This box allows administrator to select a course and move it to the Trash directory. Before the directory is moved to the trash directory, a verify delete dialog will allow the user to confirm the deletion. The administrator can easily delete the Trash directory by using system command, for example rm.

4.2.4.4 Edit a course

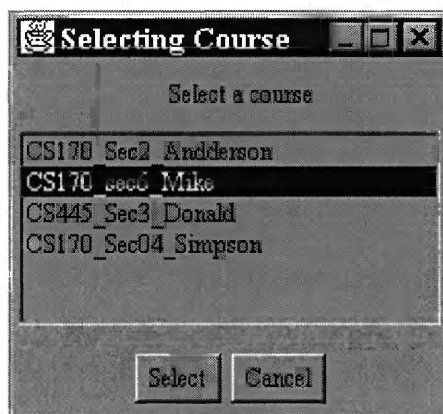


Fig 4. 22 Class SelectCourseDialog Interface

This box allows the administrator to select a course.

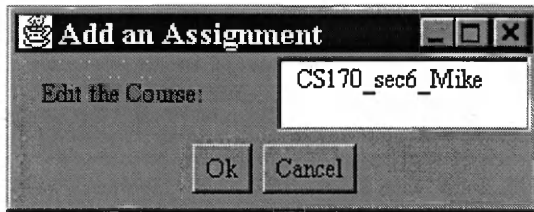


Fig 4. 23 Class EditCourseDialog Interface

This box allows the administrator to rename a course after he/she selects a course.

4.3 Instructor Interface

4.3.1 Command line:

The command line for the instructor program is similar to administrator program.

UNIX: java InstructorConfig Directory1 Directory2

4.3.2 Requirement

The requirements for the instructor program are as follows:

1. Select a course.
2. Add users where the users are listed in a file.
3. Add a single user.
4. Delete a user.
5. Add an assignment.
6. Move an assignment.
7. Edit the properties of a user.

4.3.3 Instructor program dialog interaction

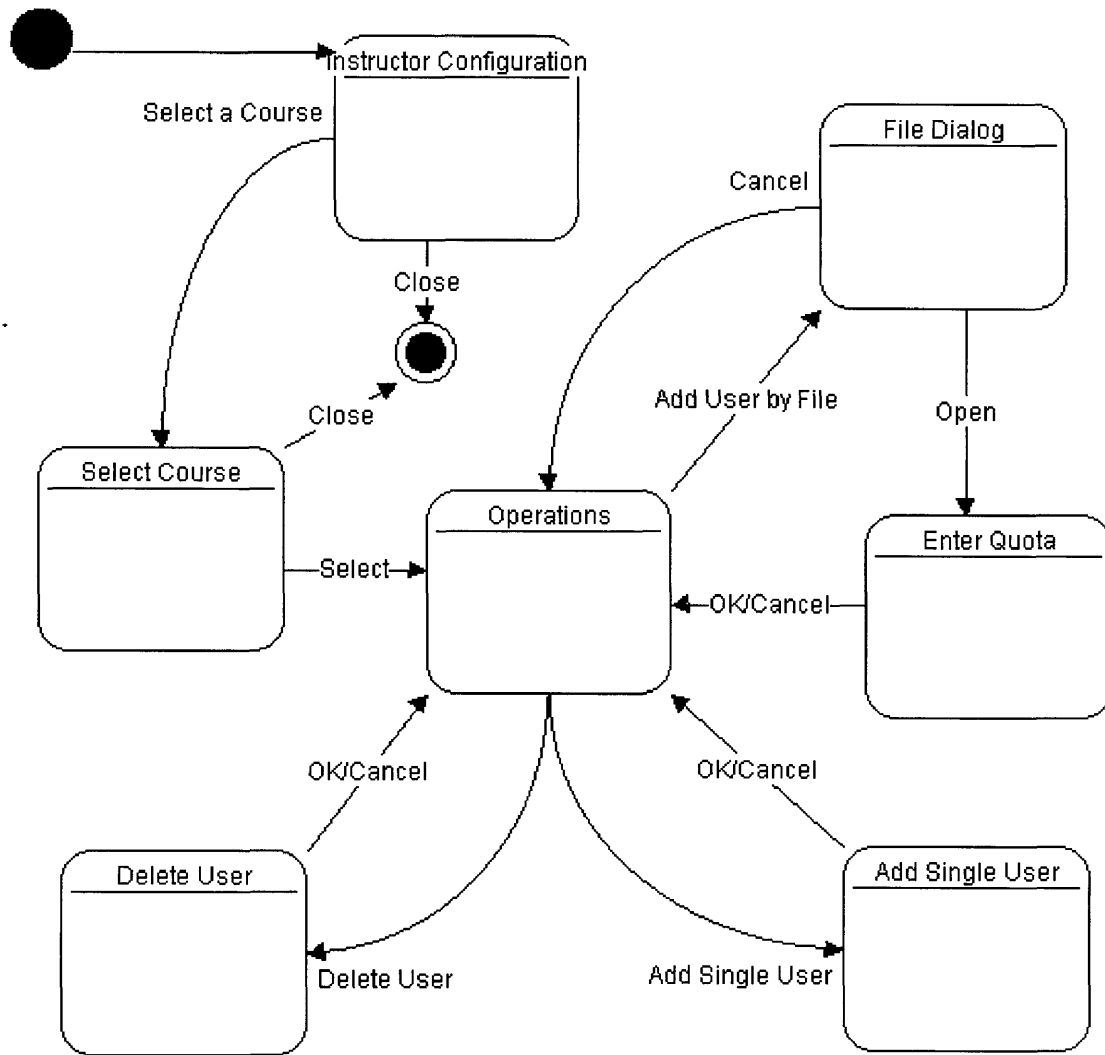


Fig 4. 24 Instructor Dialog Box Interaction

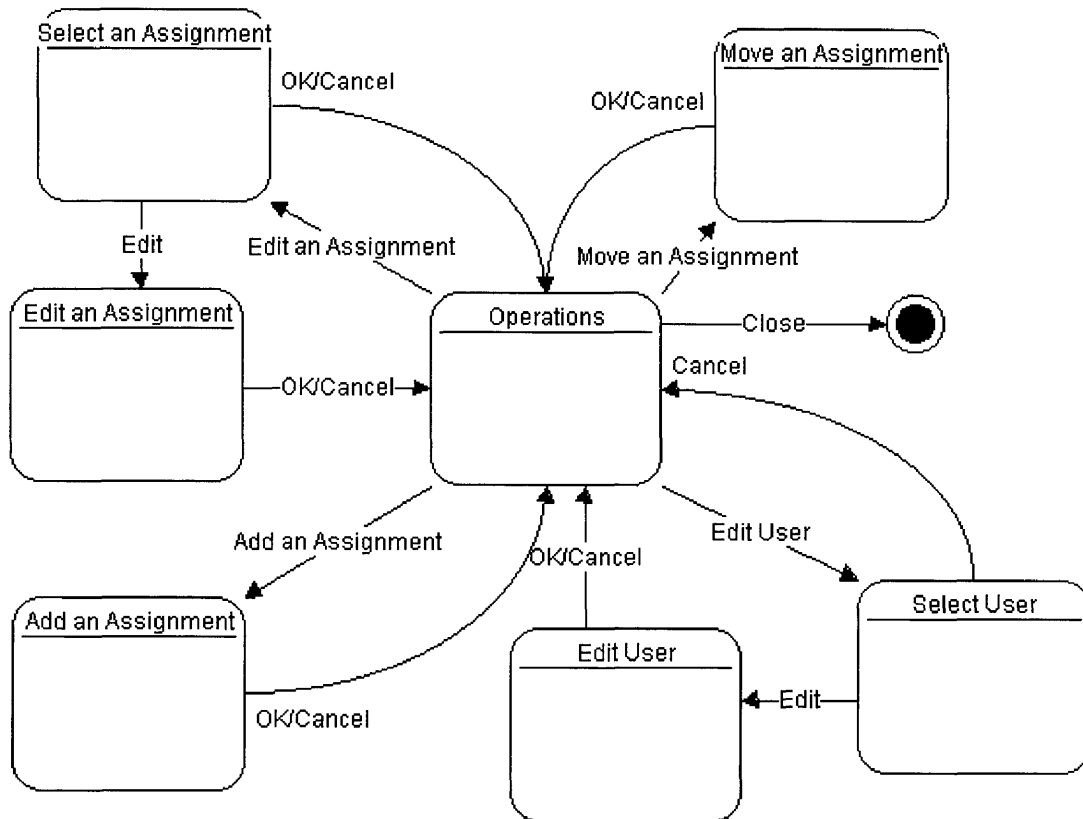


Fig 4. 25 Instructor Dialog Box Interaction 2

Fig 4. 24 Instructor Dialog Box Interaction and Fig 4. 25 Instructor Dialog Box Interaction 2 show the interactions between the instructor program dialog boxes. Since there is not enough room for edit assignment, it is not showing in the diagram above. Edit an assignment has the same interaction as edit a user.

4.3.4 Instructor program interface

4.3.4.1 Start the instructor program

When the Instructor Program starts, the following window shows up.

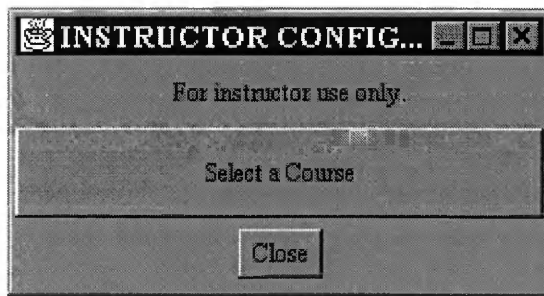


Fig 4. 26 Class InstructorConfig Interface

This box tells the users that this is the Instructor Program; the instructor is to click on the Select a Course button for the next step.

4.3.4.2 Select a course

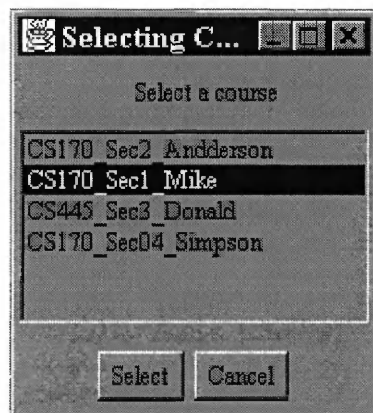


Fig 4. 27 Class SelectCourseDialog Interface

This box allows the instructor to select a course before moving.

If the Select button is pushed, the following box is shown.

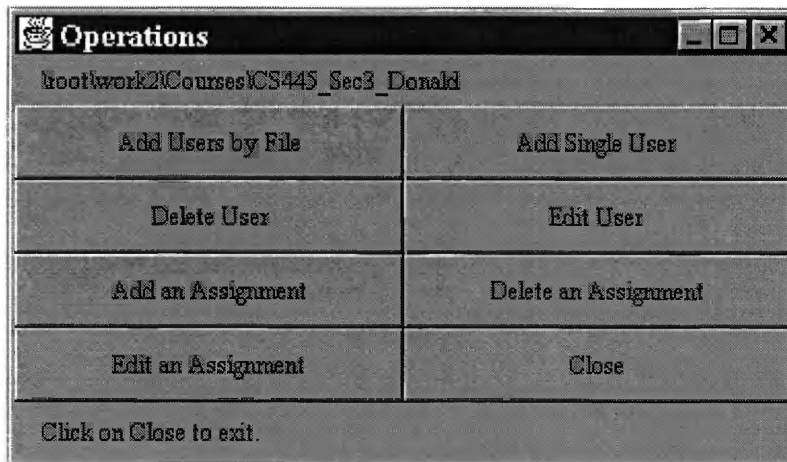


Fig 4. 28 Class Operations Interface

This box has all the operations the instructor needs to use.

4.2.4.3 Add user by file

When Add User by File is clicked, the following is shown

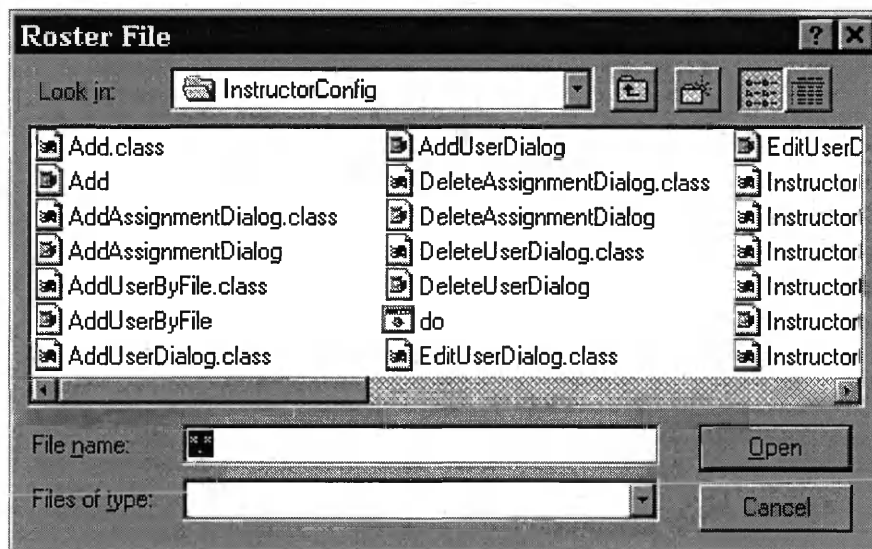


Fig 4. 29 File Dialog

This is the file box which allows the instructor to open a roster file to add a large number of students. The format of the roster file is as follows. The first part of each line up to the plus sign is ignored. The plus

sign is followed by a student 9 digit ID number, which is followed by the students real name. For example:

```
CS 495 30+517847676Harris, Fred L
CS 495 30+517135110Randall, Seth E
CS 495 30+540728657VanAlstein, Byron M
CS 495 30+517745010Ramsey, Benton S
CS 495 30+517828426Makich, Jane P
CS 495 30+517989209Berg, Ivan M
CS 495 30+517722085Feucht, Gordon C
CS 495 30+531603284Gilbertsen, Eric H
CS 495 30+322701507Zielinski, Michael V
```

This is a legal roster file. If the file does not have the format specified above, no user will be added.

If Open is pushed:

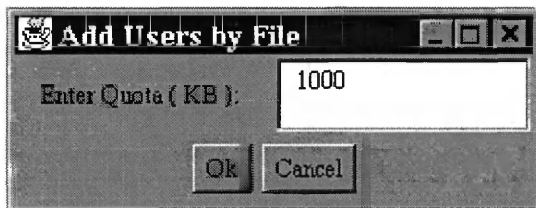
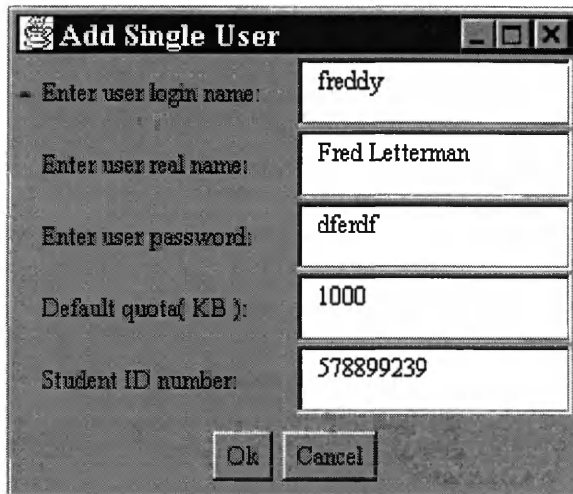


Fig 4. 30 Class AddUserByFile Interface

This box asks the instructor to enter a disk quota for those students just entered.

4.3.4.4 Add a single user.

Fig 4. 31 Class AddUserDialog Interface shows what happens when Add Single User button is pushed.



| | |
|------------------------|----------------|
| Enter user login name: | freddy |
| Enter user real name: | Fred Letterman |
| Enter user password: | dferdf |
| Default quota(KB): | 1000 |
| Student ID number: | 578899239 |

Ok Cancel

Fig 4. 31 Class AddUserDialog Interface

The instructor enters a single user.

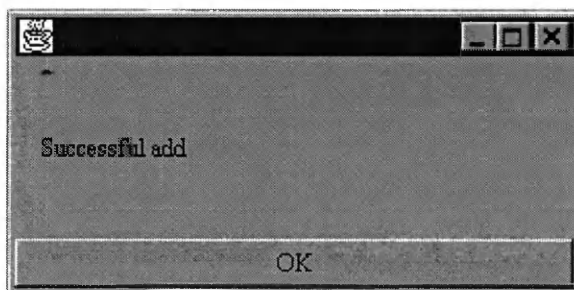


Fig 4. 32

This box confirms the add. If the user exists already, an add fail message will be displayed.

4.3.4.5 Delete a user

Fig 4. 33 Class DeleteUserDialog Interface show what happens when Delete User is pushed:



Fig 4. 33 Class DeleteUserDialog Interface

The instructor selects a user to delete. After the selection, the Delete button is pushed. Another message box similar to Fig 4. 32 will tell the result of deletion. The numbers in the list are the students ID number. After the instructor completes the Add User by File operation, these numbers are created. If the number from the student new account login matches the number listed, he/she can create his/her new account. This operation is discussed in 4.1.4.2 Log in process.

4.3.4.6 Edit an user

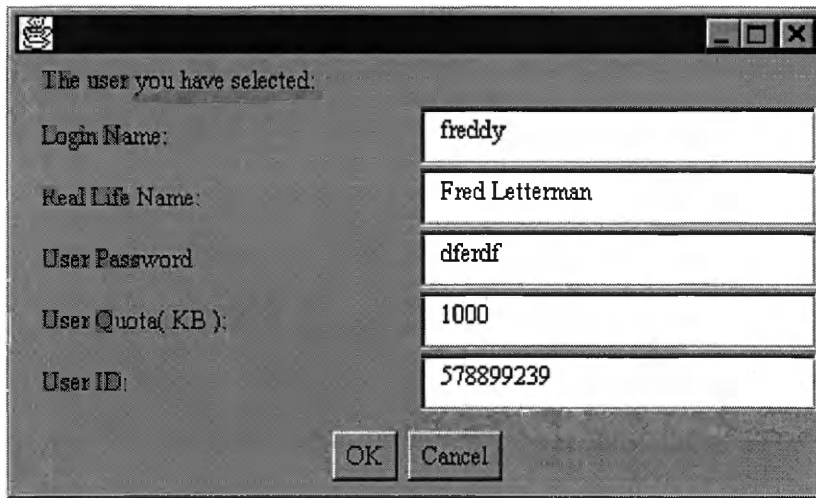
Fig 4. 34 Class EditUserDialog Interface shows what happens when Edit User is pushed:



Fig 4. 34 Class EditUserDialog Interface

The instructor selects a user to edit.

If Edit button is clicked:



The screenshot shows a dialog box titled "The user you have selected:". It contains five text input fields with the following values: Login Name: freddy, Real Life Name: Fred Letterman, User Password: dferdf, User Quota(KB): 1000, and User ID: 578899239. At the bottom, there are two buttons: OK and Cancel.

| Field | Value |
|-------------------|----------------|
| Login Name: | freddy |
| Real Life Name: | Fred Letterman |
| User Password | dferdf |
| User Quota(KB): | 1000 |
| User ID: | 578899239 |

Fig 4. 35 Class UserInfoDialog Interface

All the user information is displayed. The instructor clicks on a text field to edit the text. The editing result will be shown in a message box. The students can change their own password through the client, but they can not change their login name and quota. All the instructor interface section is used by instructors only.

4.3.4.7 Delete an assignment

Fig 4. 36 Class DeleteAssignmentDialog Interface shows what happens when Delete Assignment button is clicked:

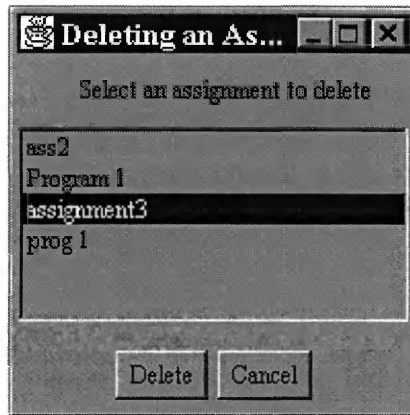


Fig 4. 36 Class DeleteAssignmentDialog Interface

The dialog box allows instructor to select a assignment directory to delete. If Delete button is pushed, a verify delete dialog will allow the user to confirm the deletion. If deletion is confirmed, the program moves the assignment directory to the Trash directory under the course root path. The instructor can delete the Trash directory by using a system command, such as rm in UNIX.

4.3.4.8 Add an assignment

Fig 4. 37 Class AddAssignmentDialog Interface shows what happen when Add Assignment button is pushed :

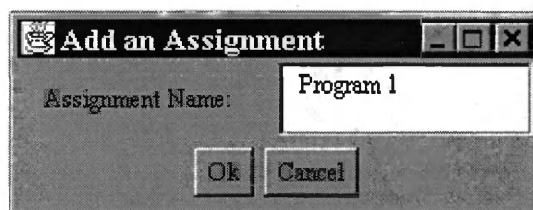


Fig 4. 37 Class AddAssignmentDialog Interface

This dialog box lets the instructor enter an assignment name, and create an assignment directory. If the assignment name is duplicated, a directory existing message will be displayed in a message box.

4.3.4.8 Edit an assignment

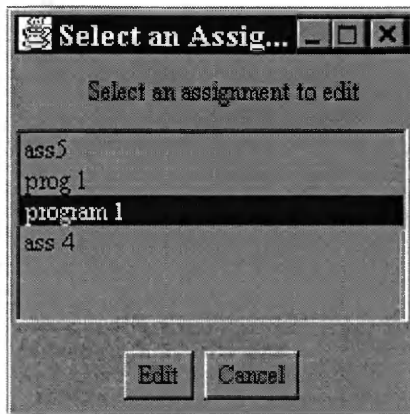


Fig 4. 38 SelectAssignmentDialog Interface

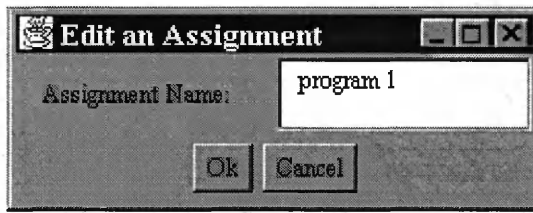


Fig 4. 39 EditAssignmentDialog Interface

The two boxes above show the two steps to edit an assignment.

4.3.5 Storage For Server

The hierarchy of the server storage is designed as shown by the following example.

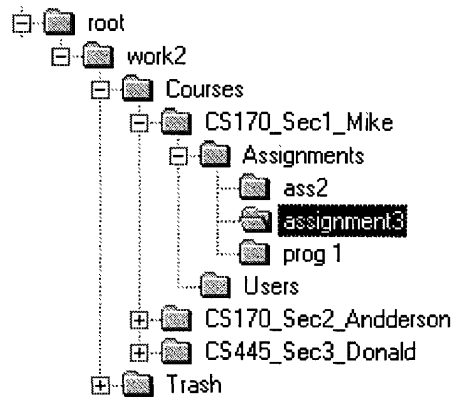


Fig 4. 40 Server Storage

The administrator program allows the administrator to create the course directory which has all the courses. The instructor program allows instructor to create the users directory which has all the user files and the assignment directory which has all the assignments. The trash directory allows the administrator or instructor to store the deleted directory. The instructor can use a system command to delete the trash directory.

Each student's assignment will be stored under a certain assignment directory. The User's directory under each course will store all the user login information as different files, and the file names are the students' login names. The user real name, password, and user quota are stored in these files. The user's submission file name is the user log in name concatenate "." and the submission original name. For example, if the user's login name is tifi and his/her submission is HelloAgain.html, the file stored in the server file system will be tifi.HelloAgain.html. In this way server program can collect the user files based on his/her login name prefix.

Chapter 5 The Detailed Design

The entire program design is based on the Booch [1] method. The purpose of the design is to construct a system that:

- “Satisfies a given (perhaps informal) functional specification
- Conforms to limitations of the target medium
- Meets implicit or explicit requirements on performance and resource usage
- Satisfies implicit or explicit design criteria on the form of the artifact
- Satisfies restrictions on the design process itself, such as its length or cost or the tools available for doing the design” [1]

My design is based on the general design principles described above. I used Object-Oriented design.

Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design.[1]

There are two important components to object-oriented design: (1) leads to an object-oriented decomposition and (2) uses different notations to express different models of the logical (class and object structure) and physical (module and process architecture) design of a system.

The entire project is decomposed into four programs as Fig 5. 1 Program Interaction shown. These programs are decomposed into different classes and objects. This design only uses the module architecture, and it's enough for the physical design.

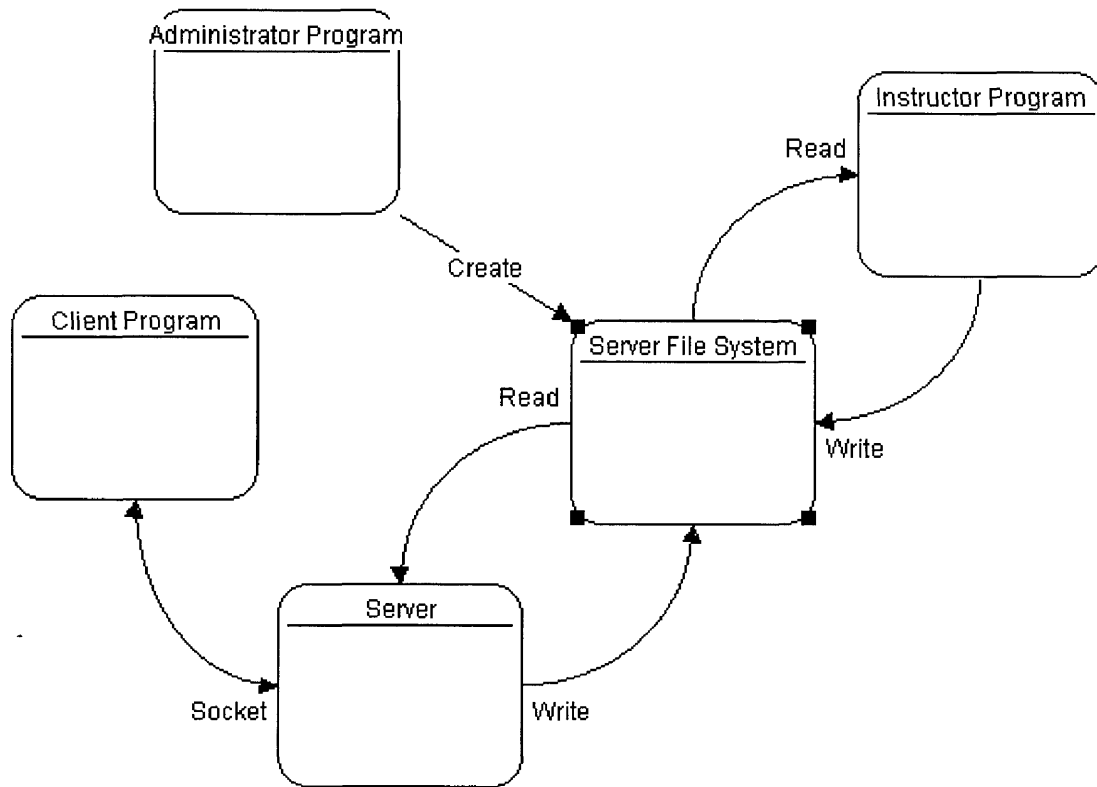


Fig 5. 1 Program Interaction

The figure above shows the interactions among the four programs and the server file system. The client program communicates with server program through sockets. The server, instructor, and administrator programs interact through common access to the server file system.

5.1. Client Program

5.1.1 Classes

Class is a set of objects that share a common structure and a common behavior.

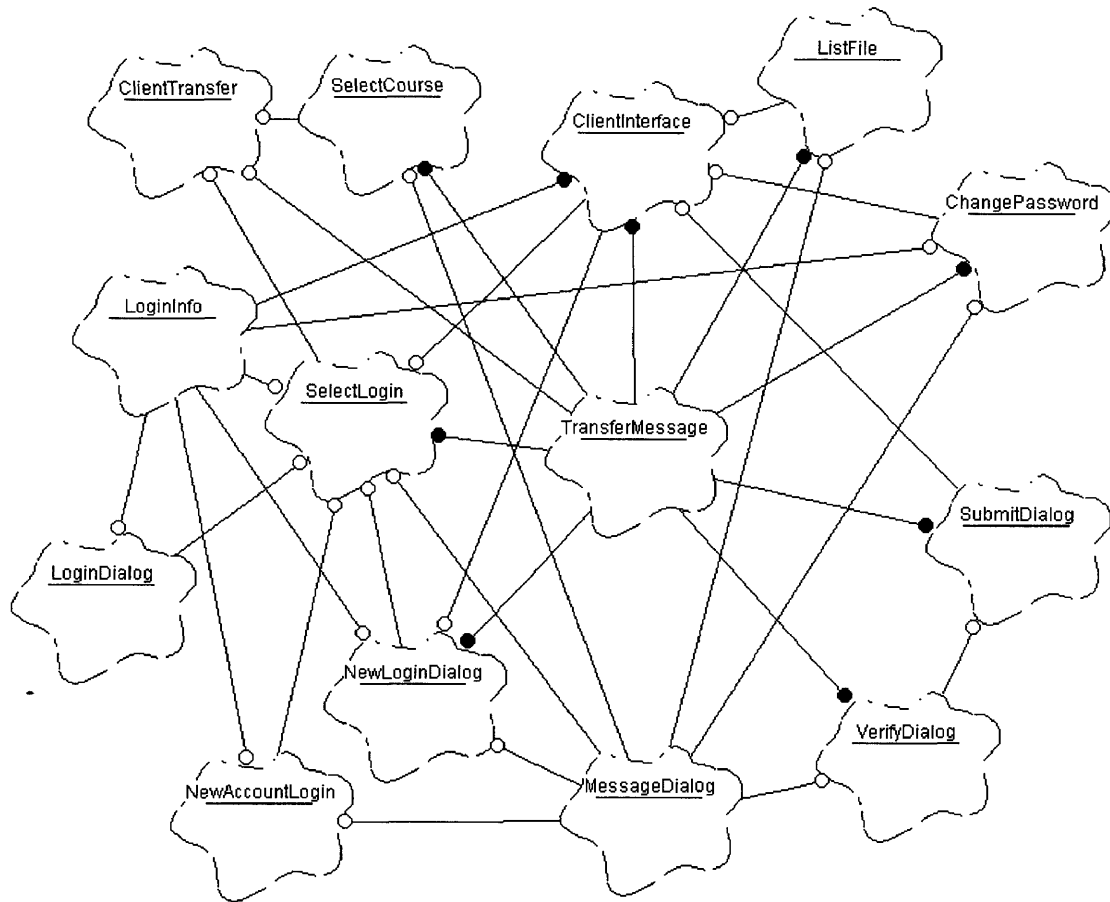


Fig 5. 2 Client Class Diagram

Fig 5. 2 Client Class Diagram shows the class relations among all the classes in client program.

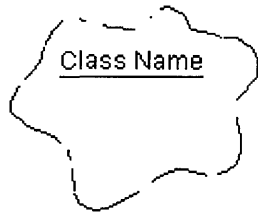


Fig 5. 3 Class Icon

Fig 5. 3 Class Icon shows the class icon used in Fig 5. 2 Client Class Diagram.

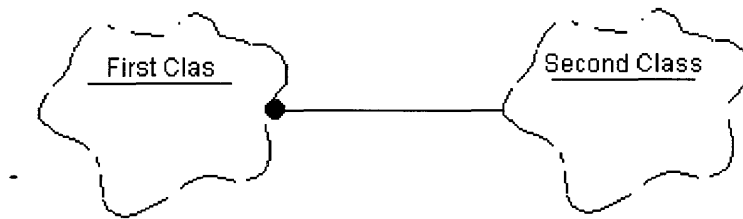


Fig 5. 4 Has Relation

Fig 5. 4 Has Relation shows that the second class is a component of the first class.

The “has” icon denotes a whole/part relationship (the “has a” relationship, also known as aggregation.), and appears as an association with a filled circle at the end denoting the aggregate. The class at the other end denotes the part whose instances are contained by the aggregate object. [1]

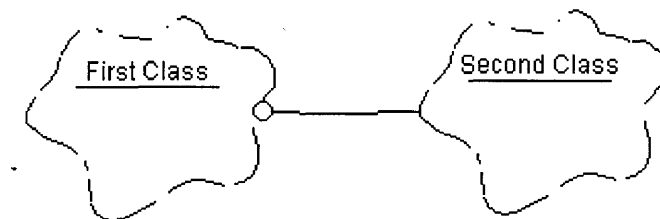


Fig 5. 5 Use Relation

Fig 5. 5 Use Relation shows that the first class uses the second class. In the other words, the first class uses the services of the second class.

The “using” icon denotes a client/supplier relationship, and appears as an association with an open circle at the end denoting the client.[1] This relationship indicates that the client in some manner depends upon the supplier to provide certain services.

Class Description:

Class ChangePassword

Super class: Dialog

ChangePassword(Frame, LoginInfo, TransferMessage)

boolean action(Event , Object)

boolean handleEvent(Event)

Semantics:

- ChangePassword: It takes Frame parent, LoginInfo u, and TransferMessage Tm as the arguments. It displays the old password, the new password, and the verify password text fields.
- It has OK and Cancel buttons.
- Action: If OK button is clicked, it takes the new password and sends it to server. If the password does not match the old password or the new password does not match verify password, the message will be displayed. It will receive and show the changed result from the server. If the Cancel button is pushed, it closes the window.
- HandleEvent: If the window destroy message is received, it closes the window.

Class ClientInterface

Super class: Frame

ClientInterface(TransferMessage, LoginInfo)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

- ClientInterface: It takes TransferMessage Tm and LoginInfo u as the arguments and displays Submit File, Change Password, Delete File, and Quit buttons.
- Action: It acts differently based on the button pushed.

| <u>Button</u> | <u>Invoked Object</u> |
|-----------------|-----------------------|
| Change Password | ChangePassword CP |
| Delete File | ListFile LF |
| Submit File | SubmitDialog SD |

If Quit is clicked, it exits the program.

- HandleEvent: If the window destroy message is received, it closes the window.

Class ClientTransfer

void main(String [])

Semantics:

- main: It is the main thread of control in the client program. It invokes SelectCourse object after the SelectLogin object is invoked.

Class ListFile

Super class: Dialog

ListFile(Frame, TransferMessage)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

- ListFile: It sends the file info request to the server, and receives the message. It decodes the message and displays it in a list. The Delete and Cancel buttons are displayed, too.
- Action: If Delete button is pushed, it collects the file names from the list and sends the deletion message to server. It receives the message from server, and displays the deletion result in a message dialog box. If there is an error, the error message will be displayed in a message box as well.
- HandleEvent: It stores the list selected or deselect result. If a window destroy message is received, it closes the window.

Class LoginDialog

Super class: Dialog

LoginDialog(Frame, LoginInfo)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

- LoginDialog: It takes Frame parent and LoginInfo u as the arguments, and displays login name and password, two text fields. The OK and Cancel buttons are displayed, too.
- Action: If OK button is pushed, it sends the login name and password to the parent. If the Cancel button is pushed, it closes the window.
- HandleEvent: If a window destroy message is received, it closes the window.

Class LoginInfo

LoginInfo(String, String)

It's the container class with user name and password, two fields.

Class MessageDialog

Super class: Dialog

MessageDialog(Frame, String)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

-
- MessageDialog: It takes Frame parent and String msg as the arguments. It displays the msg in the dialog box, and it displays the OK button as well.
- Action: If the OK is pushed, it closes the window.
- HandleEvent: If a window destroy message is received, it closes the window.

Class NewAccountLogin

Super class: Dialog

NewAccountLogin(Frame)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantic:

- NewAccountLogin: It displays a text field for the user to input login name which is the student ID number. The OK and Cancel buttons are displayed.

- Action: If the OK button is clicked, it gets the text from the text field and sends it back to the parent. If the Cancel button is clicked, it closes the window. The error message will be displayed in a message box if there is one.
- HandleEvent: If a window destroy message is received, it closes the window.

Class NewLoginDialog

Super class: Dialog

NewLoginDialog(Frame, TransferMessage)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

- NewLoginDialog: It takes Frame parent and TransferMessage Tm as the arguments, and it displays login name, real name, password, and verify password, four text fields. The OK and Cancel buttons are displayed, too.
- Action: If the OK button is pushed, it gets the text of these four text fields and packs them with header "NewLogin". After that, it sends the packed string to server and receives the result from server. It displays the message from server, or it displays the error message if there is one.
- HandleEvent: If a window destroy message is received, it closes the window.

Class SelectCourse

Super class: Frame

SelectCourse(String, TransferMessage)

boolean action(Event, Object)

boolean handleEvent(Event)

String GetCourseResult()

Semantics:

- **SelectCourse:** It takes String S and TransferMessage Tm as the arguments and decodes the course messages from S. Then it displays the course messages in a list, and it shows the Select and Close button, too.
- **Action:** If Select button is clicked, it stores the selection in courseResult. If the Close button is pushed, it exits the program.
- **HandleEvent:** It stores the selected or deselected item. If a window destroy message is received, it exits the program.
- **GetCourseResult:** It returns courseReslut object.

Class SelectLogin

Super class: Dialog

SelectLogin(String, TransferMessage)

boolean action(Event, Object)

boolean handleEvent(Event)

void processResult(Dialog, Object)

Semantics:

- **SelectLogin:** It takes String courseStr and TransferMessage Tm as the arguments and displays New Account Login, Normal Login, and Close buttons.
- **Action:** The action is based on the button clicked.

Button

Object

Normal Login

LoginDialog ld

| | |
|-------------------|---------------------|
| New Account Login | NewAccountLogin NAL |
| Close | System.exit(0) |

- **HandleEvent:** If a window destroy message is received, it exits the program.
- **processResult:** If it is a result of the normal login, and the login message from server is “ACCEPTED”, it invokes ClientInterface CI; otherwise the login fail message is displayed. If it is a result of the new account login, and the server returned message is “ACCEPTED”, it invokes NewLoginDialog NLD; otherwise, it displays the error message.

Class SubmitDialog

Super class: Dialog

SubmitDialog(Frame, TransferMessage)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

- **SubmitDialog:** It sends “AssInfo” to server and receives the assignment information from server. It decodes the information and displays it in a list. The OK and Cancel buttons are displayed, too.
- **Action:** If OK is clicked, it gets the selected item and opens a file dialog box. After the file is allocated, it invokes VerifyDialog VD. If Cancel is pushed, it closes the window.
- **HandleEvent:** It stores the selected or deselected item. If it receives window destroy message, it closes the window.

Class TransferMessage

TransferMessage()

```

void    Close()

boolean SendString( StringBuffer )

boolean SendFile( File )

String  ReceiveString()

```

Semantics:

- Close: It sends “Bye” message to server and closes the socket.
- SendString: It takes StringBuffer str as an argument and sends str to the server.
- SendFile: It takes File file as an argument and opens the file. After that, it sends the entire file to the server in Byte format.
- ReceiveString: It receives a string from server and returns the string.

Class VerifyDialog

Super class Dialog

```
VerifyDialog( Frame, String, String, String, File, TransferMessage )
```

```

boolean action( Event, Object )

boolean handleEvent( Event )

```

Semantics:

- VerifyDialog: It takes Frame parent, String course, String assNum, String fileName, File subfile, and TransferMessage Tm as the arguments. It displays the course number, assignment number, and file name which the user just selected. The OK and Cancel button are displayed, too.
- Action: If the OK button is pushed, it packs these three fields with header “Submit” and sends it to server. If it receives “Ready” message from server, it sends the entire file to the server. It displays the submitting result in a message box, as well.

- `HandleEvent`: If it receives window destroy message, it closes the window.

5.1.2 Module Diagram

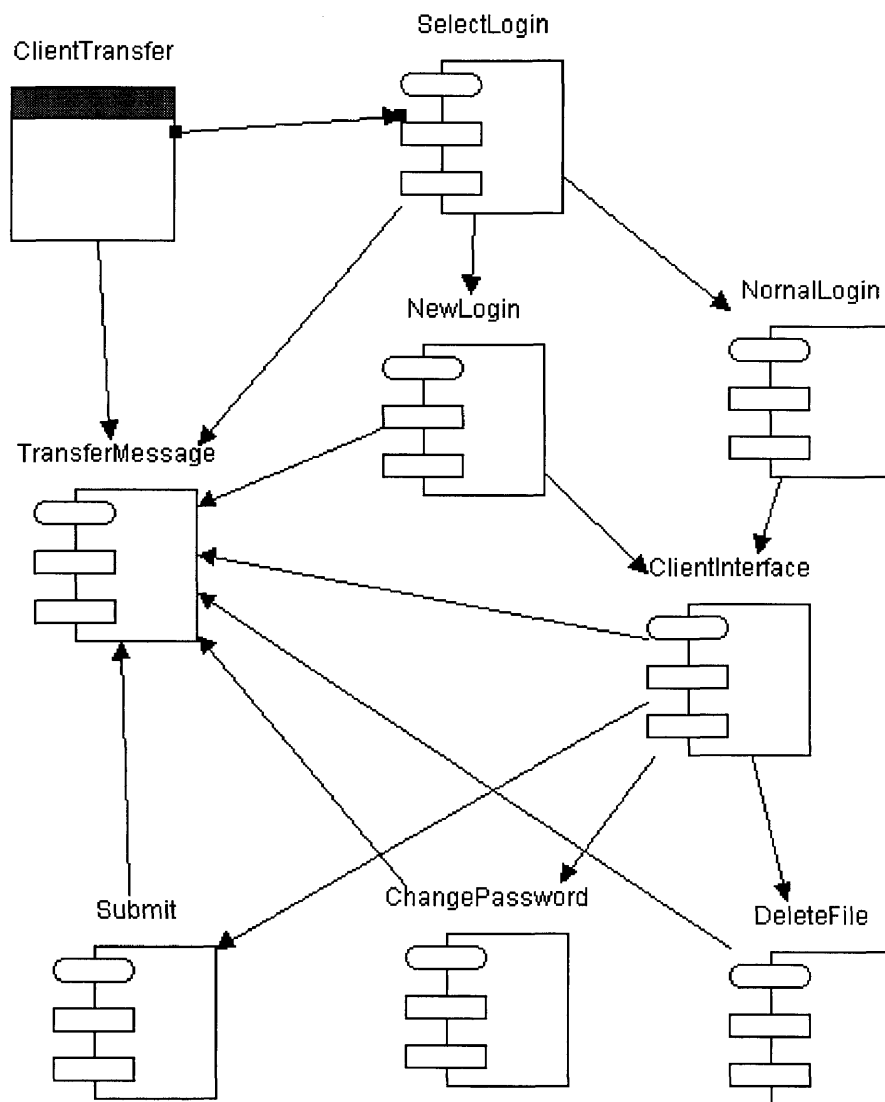


Fig 5. 6 Module Diagram

Fig 5. 6 Module Diagram shows the dependencies among the client modules.

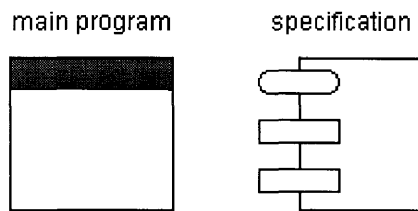
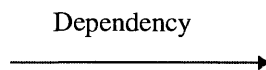


Fig 5. 7 Module Icons

Error! Reference source not found. shows the module icons used in Fig 5. 6 Module Diagram. The icon on the left is the main program module which has the main thread of control. The icon on the left is the specification icon for a module. Since the programming language for the project is Java, the specification and the body of the module are unified; therefore, there is no body icon showing in Fig 5. 6 Module Diagram, and the specification icon is enough.



Of course the arrow above is representing the dependency.

Each module may have one or more classes:

ClientTranfer module has ClientTransfer and SelectCourse classes, It has the main thread of control in the program.

SelectLogin module has SelectLogin class.

TransferMessage module has TransferMessage class.

NewLogin module has NowLoginDialog and NewAccountLogin class

NormalLogin module has NormalLogin class.

ClientInterface module has ClientInterface class.

Submit module has SubmitDialog and VerifyDialog class.

ChangePassword module has ChangePassword class.

DeleteFile module has ListFile class.

Note: The class MessageDialog and UserInfo could be the component of any module above.

5.1.3 Object Diagram

Object: An object has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable. [1]

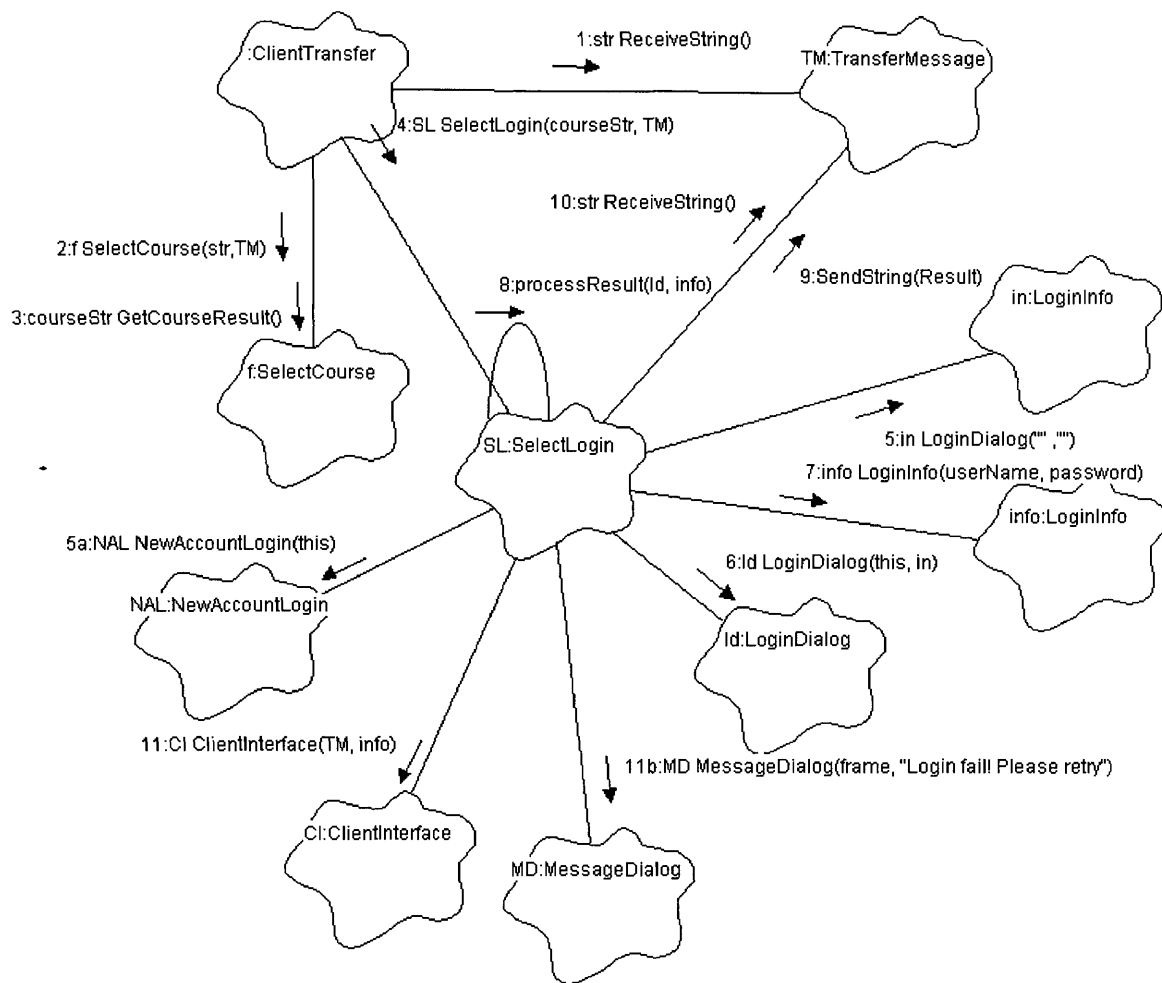


Fig 5. 8 Object Diagram

NOTE:

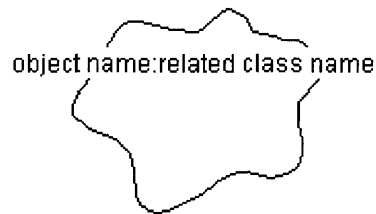


Fig 5. 9 Object Icon

Fig 5. 9 Object Icon shows the object icon. The object icon has its name followed by a colon and its class name.

Order : Message



The links between the objects are shown above. It has two parts. First part is the order which is an Arabic number. The second part is the message which is a function call. Of course, the arrow shows the direction of the calling.

Fig 5. 8 Object Diagram shows the partial object interactions among the client objects. The first object :ClientTransfer sends 1:str ReceiveString() message to object TM. Then the 2: f SelectCourse(str, TM) message is sent. Based on the order, the log in is processed properly.

There are three events in Fig 5. 8 Object Diagram based on the order:

1. Normal log in successfully: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.
2. Normal log in fail: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11b.
3. Partial new account log in: 1, 2, 3, 4, 5a.

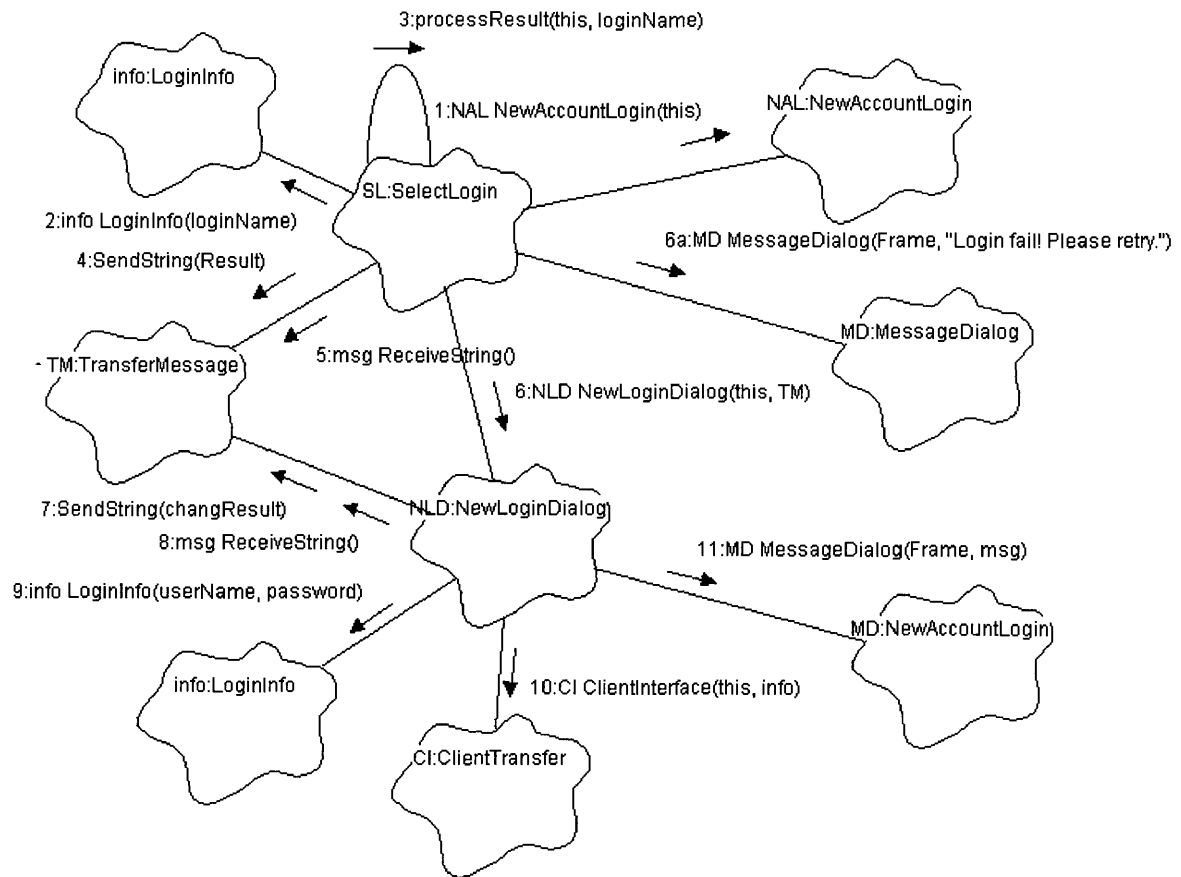


Fig 5. 10 Object Diagram 2

Based on the event 3, Partial new account log in, of Fig 5. 8 Object Diagram, Fig 5. 10 Object Diagram 2 shows two events:

1. Students successfully new account log in, and this process includes setting up a new account.
The order is: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.
2. Students log in fail. The order is: 1, 2, 3, 4, 5, 6a.

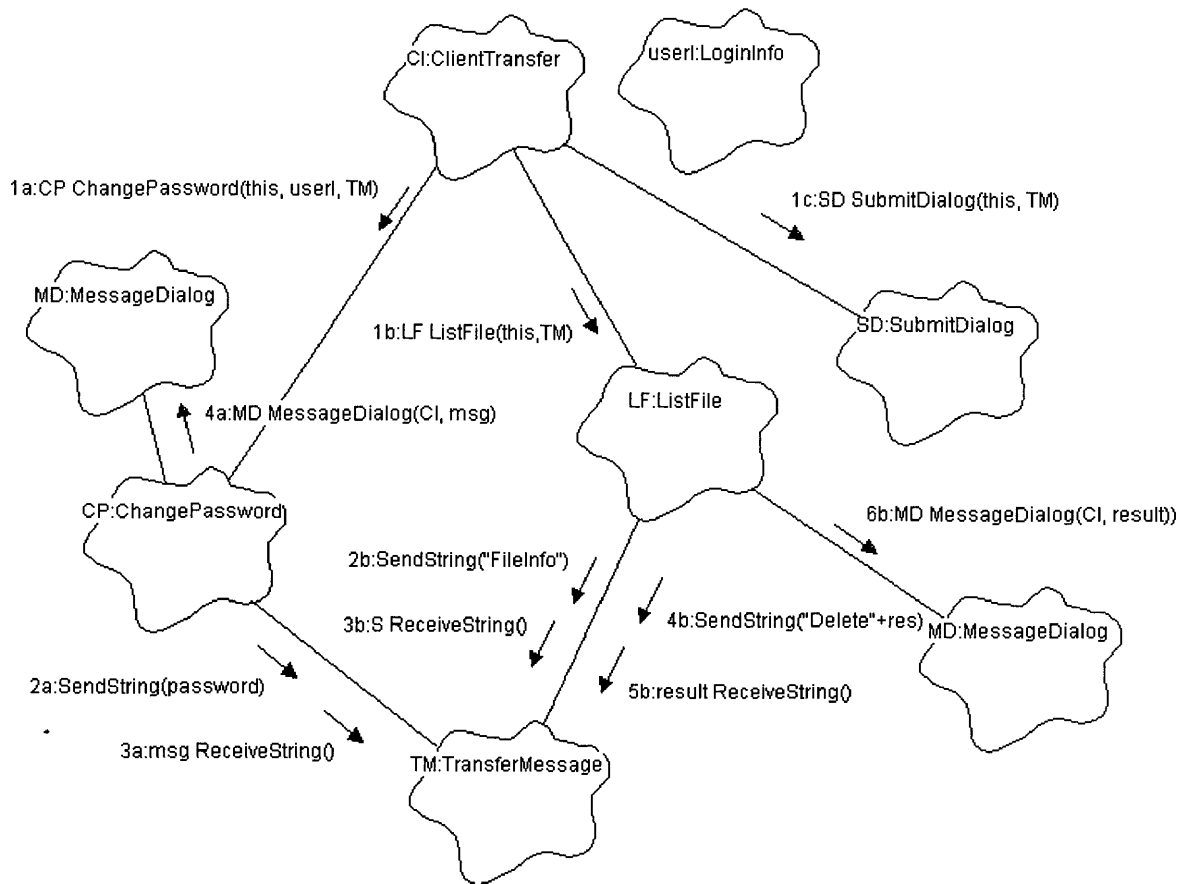


Fig 5. 11 Object Diagram 3

After students successfully log in, Fig 5. 11 Object Diagram 3 shows the three events they can do:

1. Student changes his/her password. The order is: 1a, 2a, 3a, 4a.
2. Student deletes his/her previous submission. The order is 1b, 2b, 3b, 5b, 6b.
3. Partial event for student submitting the file . The order is: 1c.

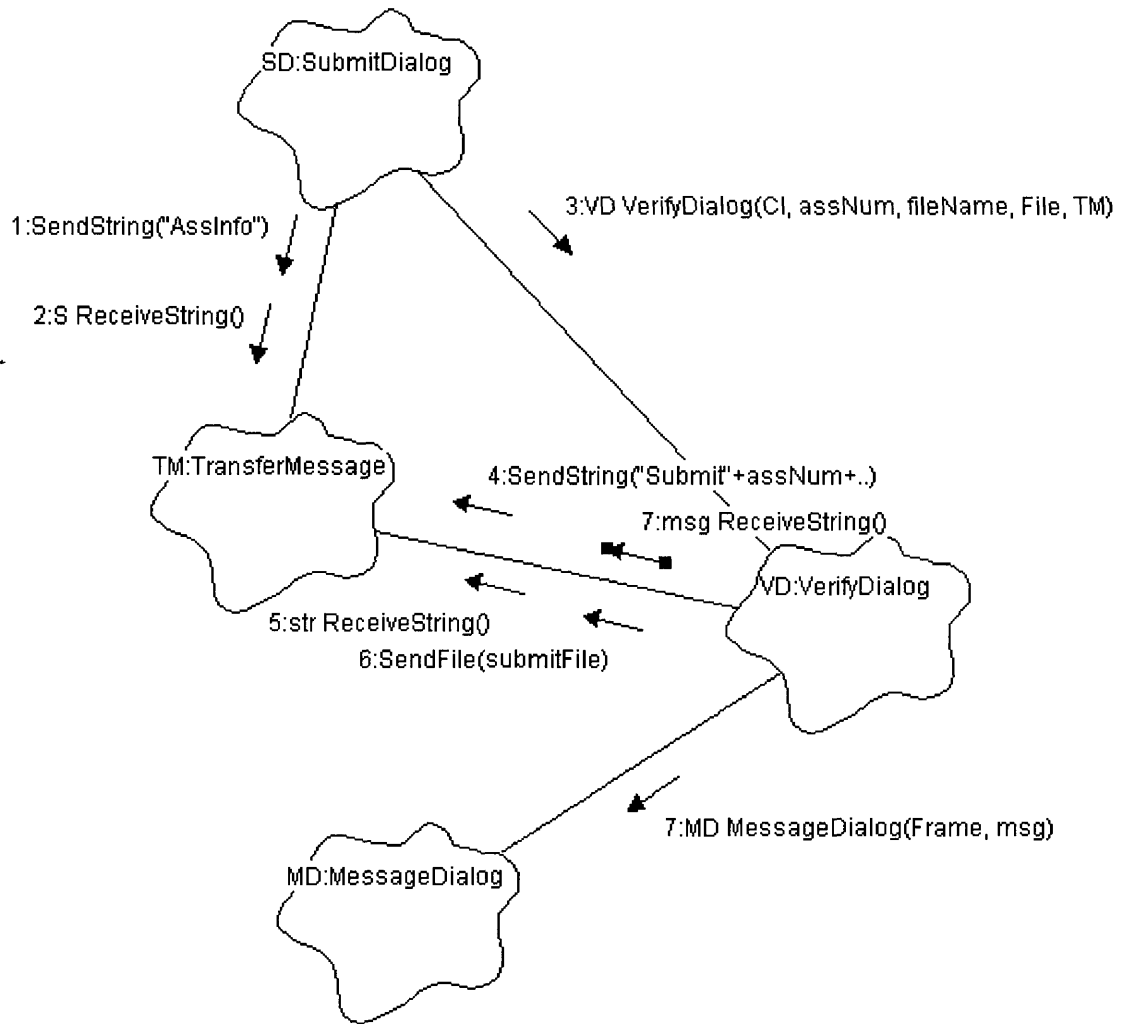


Fig 5. 12 Object Diagram: Submit

Based on the event 3 of Fig 5. 11 Object Diagram 3, Fig 5. 12 Object Diagram: Submit shows the most important event of this project: Submit.

5.1.4 State Transition Diagram

I only show the state transition diagram for the important classes here.

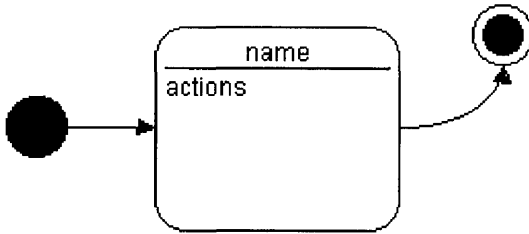
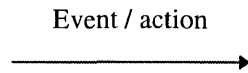


Fig 5. 13 STD icons

Fig 5. 13 STD icons shows the state transition diagram icons used in STD. The left most icon is the beginning. The middle box icon is the state icon which has name and actions. Of course the last one is the end icon.



The arrows used in the STD is showing the event/action as above.

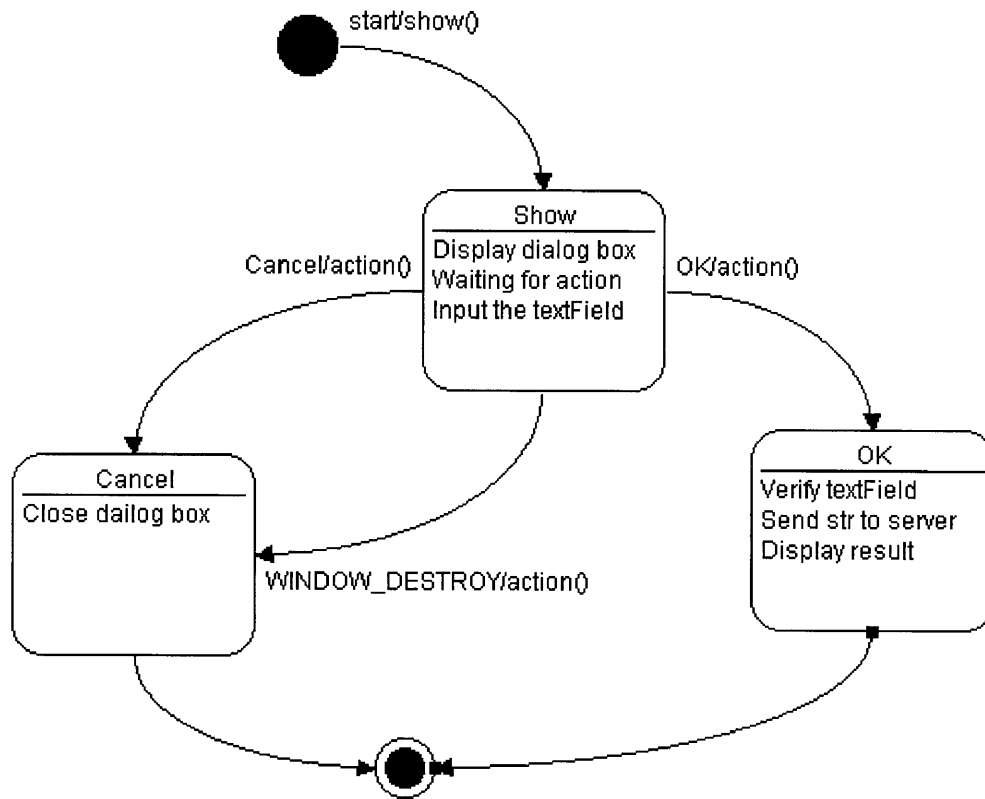


Fig 5. 14 STD: ChangePassword

Fig 5. 14 STD: ChangePassword is the state transition diagram for class ChangePassword.

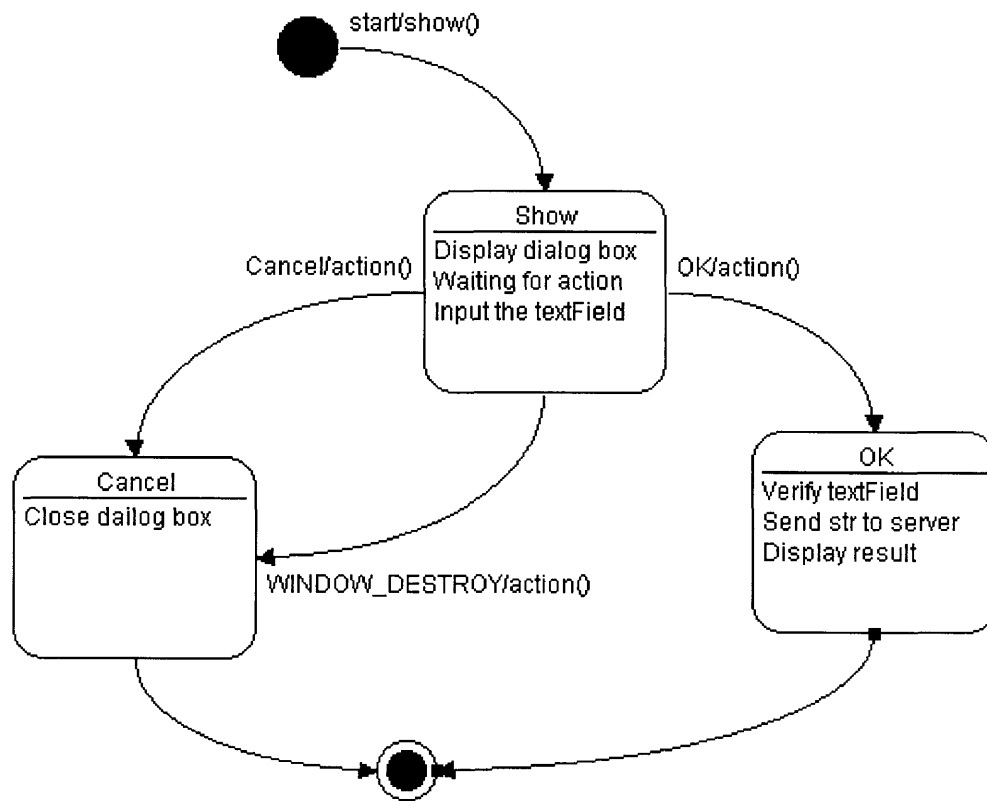


Fig 5. 15 STD: ClientInterface

Fig 5. 15 STD: ClientInterface is the state transitions diagram for class ClientInterface.

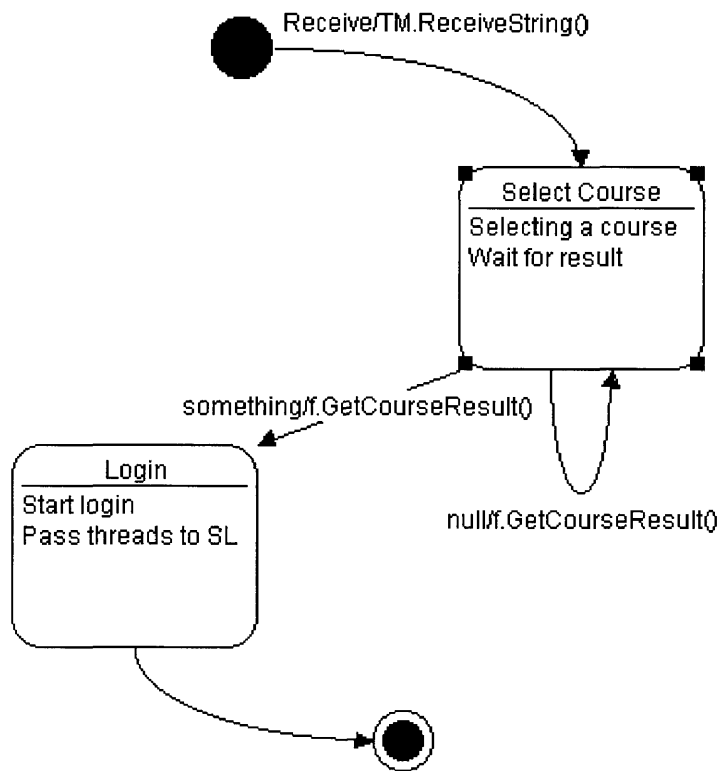


Fig 5. 16 STD: ClientTransfer

Fig 5. 16 STD: ClientTransfer is the state transition diagram for the class ClientTransfer.

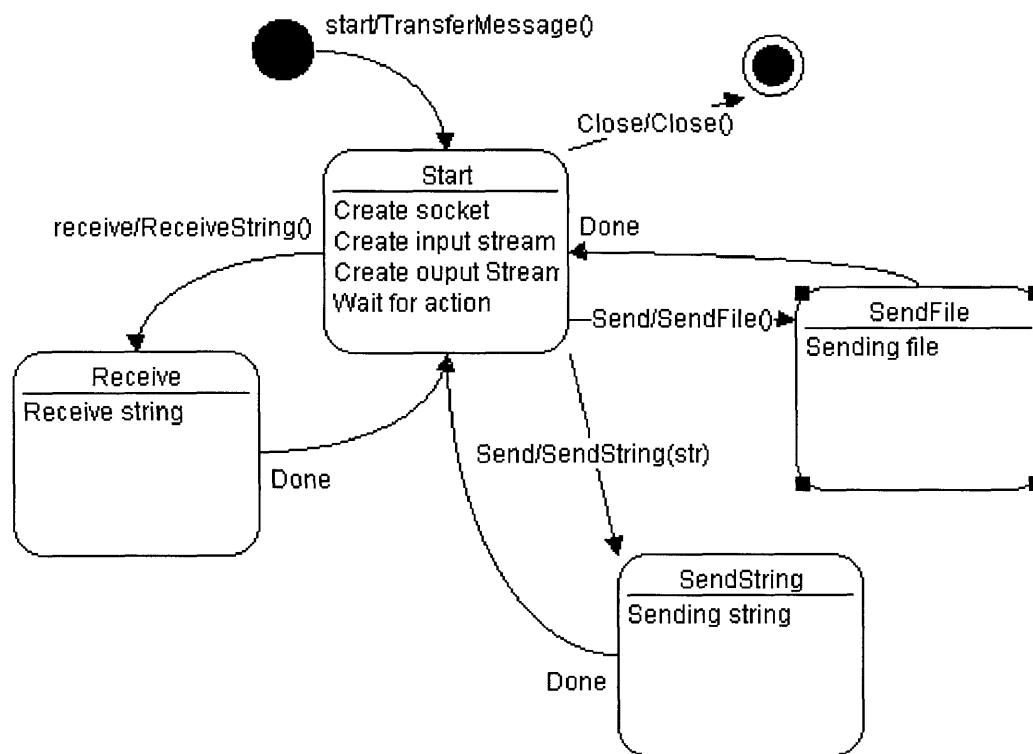


Fig 5. 17 STD: TransferMessage

Fig 5. 17 STD: TransferMessage is the state transition diagram for class TransferMessage.

5.1.5 Interaction Diagram

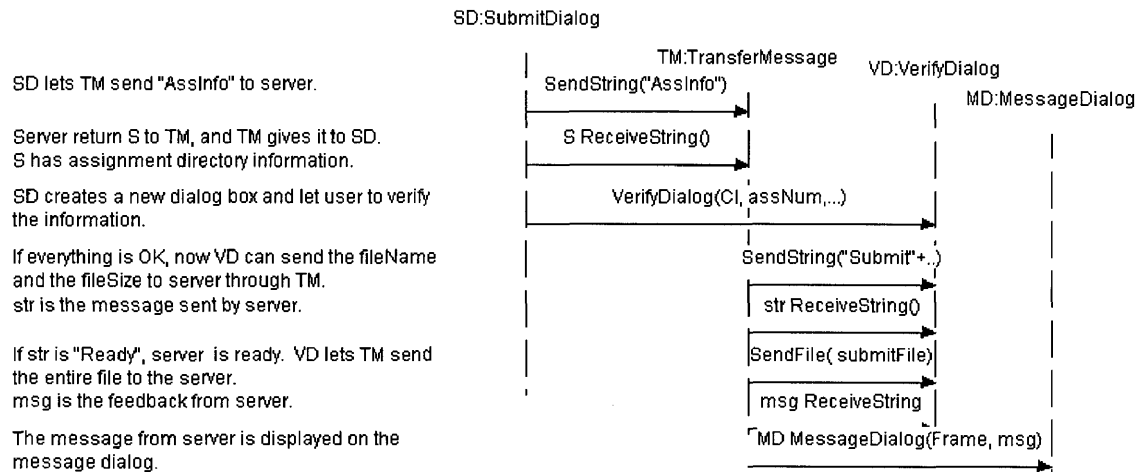


Fig 5. 18 Submit File Interaction Diagram

The interaction diagram traces the execution of a scenario. Fig 5. 18 Submit File Interaction

Diagram shows the interactions among four objects: SD, TM, VD, and MD. The scenario is a student submitting a file. This is the same as the event, submit, which I showed in the object diagram.

5.2 Server Program

The purpose of the server program is to respond to any request from the client and provide the specified resources for the client.

5.2.1 Classes

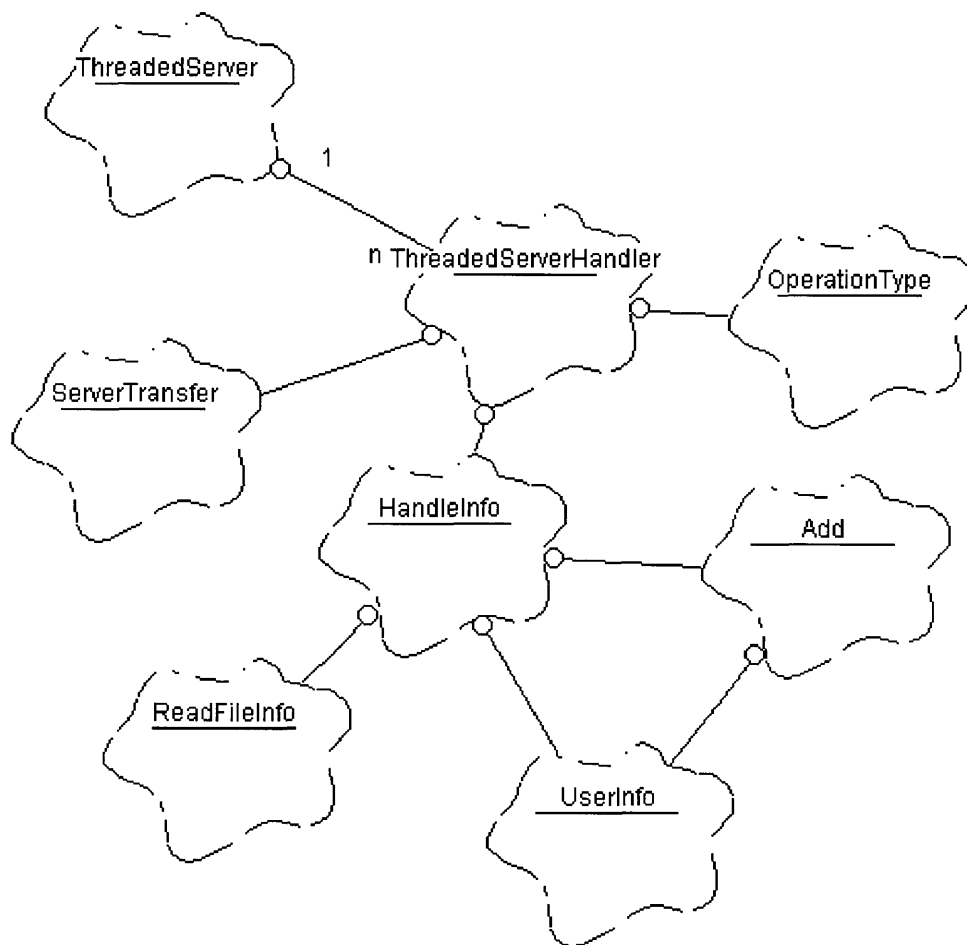


Fig 5. 19 Server Class Diagram

Fig 5. 19 Server Class Diagram shows the class relations among the server classed. Notice that there is only one instance for the class, ThreadedServer, and n instances for the others.

Class Description:

Class Add

Add(File)

boolean DoAdd(UserInfo)

.

Semantic:

- Add: It's the constructor of the Add class. It initializes a user file which is going to be created.
- DoAdd: It creates the user file and stores the user log in name, real name, password, and user quota into the file.

Class HandleInfo:

HandleInfo()

String GetCourseInfo(String)

String GetAssInfo(String)

String GetUserFileInfo(String)

.

int GetUserAvailableQuota()

boolean VerifyLoginInfo(String, String)

boolean DoChange(String , String)

boolean DoNewLoginChange(String, String)

boolean DoDelete(String, String)

boolean UpdateNameRealPassword(String, String, String)

boolean UpdatePassword(String)

File GetAssignmentPath()

File SetSubmitFileName(String, String)

int SetSubmitFileSize(String, String)

.

Semantics:

- **HandleInfo:** It's the constructor of the class **HandleInfo**. It locates the course root directory.
- **GetCourseInfo:** It takes **String** delimiter as an argument and gets all the course information in the course root path. It packs the course information into a string separated by the delimiter and returns the string.
- **GetAssInfo:** This operation is similar to **GetCourseInfo** except the directory is assignment directory.
- **GetUserFileInfo:** It takes **String** delimiter as an argument. It goes to all the assignment directories and gets all the user submitted files which are the files with the user login name prefix. It returns the information with delimiter packed in.
- **GetUserAvailableQuota:** It's the same operation as **GetUserFileInfo** except it gets the file size and adds them together. It returns the difference of the total size and user quota which is in the user file.
- **VerifyLoginInfo:** It takes **String** input and **String** delimiter as the arguments. The user course name, user login name, and user password are located in the input separated by delimiter. It decodes them and stores them in the current object memory for further use. Then it verifies the input password with password in the user login file. If they match return true, otherwise false.
- **DoChange:** It takes **String** str and **String** delimiter as the arguments. The new password is located in str. It decodes the new password from str, and passes it to **UpdatePassword**..
- **DoNewLoginChange:** It takes **String** str and **String** delimiter as the arguments. It decodes the user new login name, the user real name, and new password from str and passes them to **UpdateNameRealPassword**.
- **DoDelete:** It takes the **String** str and **String** delimiter as the arguments, and decodes the assignment path and the file name from the str with delimiter. It deletes the file and returns true; otherwise false.

- **UpdateNameRealPassword:** It does what the name of the function shows. It takes user name, user real name, and password as the arguments and updates them in the login file. Notice that the user file name will be changed to new login name. True is returned; otherwise false.
- **GetAssignmentPath:** It returns the assignment path which was set by the VerifyLininfo.
- **SetSubmitFileName:** It takes String str and String delimiter as the arguments, and decodes the file name from str. It sets up the file path based on the assignment path provided by the VerifyLoginInfo. It returns the file path if success; otherwise false.
- **SetSubmitFileSize:** It takes the String str and String delimiter as the arguments, and decodes the file size from str. It returns the file size if successful; otherwise false.

Class OperationType:

OperationType()

boolean IsDelete(String)

boolean IsChange(String)

boolean IsSubmit(String)

boolean IsNewLogin(String)

Semantic:

- **OperationType:** It's the constructor of class OperationType. It takes String str as an argument.
- **IsDelete:** It takes String str as an argument and decodes the deleting information from str. If it matches, returns true; otherwise false.
- **IsChange:** The same operation as IsDelete but matching change.
- **IsSubmit:** The same operation as IsDelete but matching submit.
- **IsNewLogin:** The same operation as IsDelete but matching new login.

Class ServerTransfer

```

        ServerTransfer( Socket, int )

void    Close()

boolean ReceiveStoreFile( File, int )

boolean SendString( String )

String  ReceiveString( )

```

Semantics:

- **ServerTransfer:** It takes Socket and int c as the arguments, and initializes the connection with the client through socket. C is the number of the clients who connect to the server.
- **ReceiveStoreFile:** It takes the File path and int fileSize as the arguments, and receives the file entirely and stores it in path with file size of fileSize initialized. True is returned if success; otherwise false.
- **SendString:** It takes String str as the argument, and sends str to the client. True is returned if successful; otherwise false.
- **ReceiveString:** It receives the string from the client, and it returns this string.

Class ThreadedServerHandler

```

ThreadedServerHandler( Socket, int )

run()

```

Semantic:

- **ThreadedServerHandler:** It passes Socket s and int count.
- **Run:** When the object of ThreadedServer is invoked, it executes run.

Class ThreadedServer

main(String [])

This is the main thread of control in the entire program.

Class UserInfo:

UserInfo(String, String, String, String, String)

This is the container class with LoginName, RealName, Quota, and Password, four fields.

5.2.2 Module Diagram

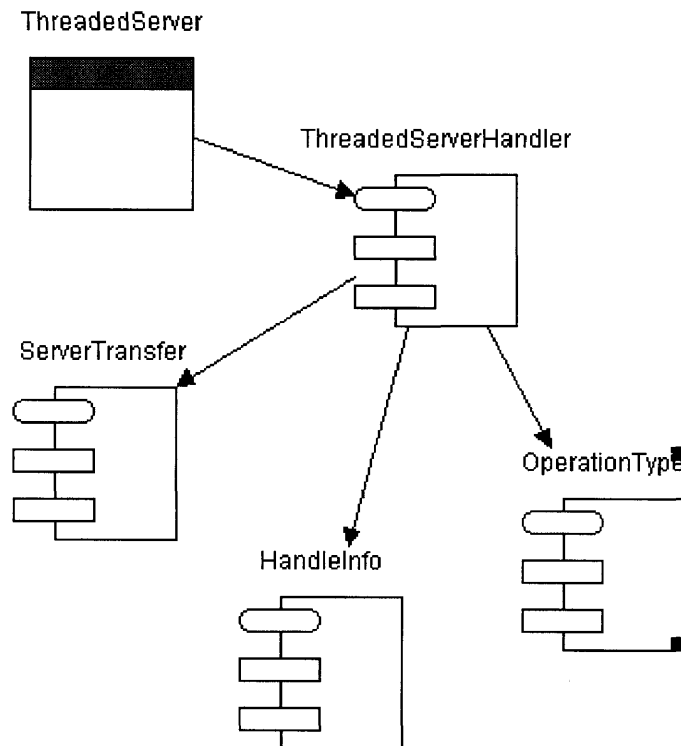


Fig 5. 20 Server Module Diagram

Fig 5. 20 Server Module Diagram shows the dependence of the modules.

ThreadedServer has ThreadedServer class which is a logical main module.

ThreadedServerHandler has ThreadedServerHandler class.

ServerTransfer module has ServerTransfer class.

OperationType module has OperationType class.

HandleInfo module has HandleInfo, Add, and Userinfo classes.

5.2.3 Object Diagram

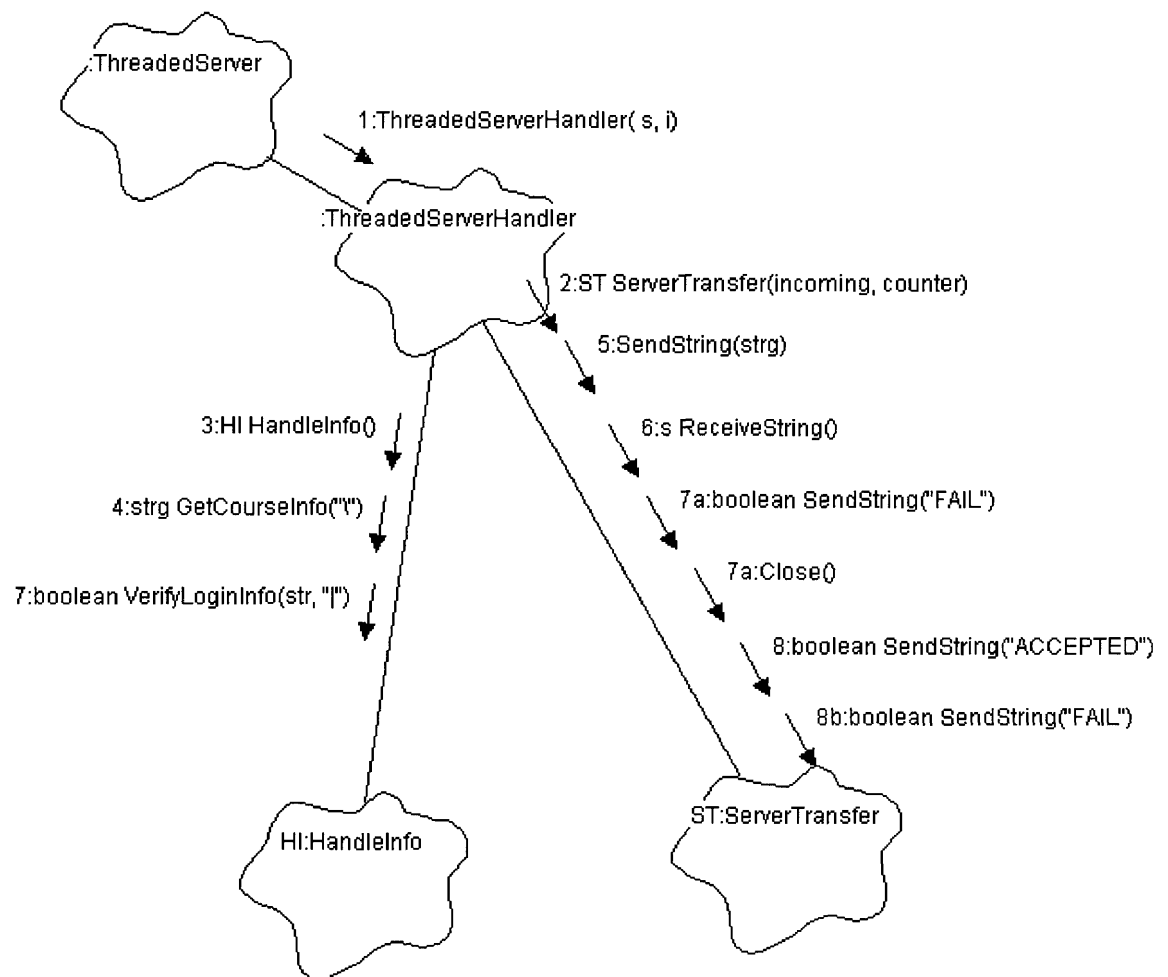


Fig 5. 21 Login Object Diagram

Fig 5. 21 Login Object Diagram shows the objects interaction of user log in. There are three events in it.

1. Process a successfully logged in user. The order is: 1, 2, 3, 4, 5, 6, 7, 8.
2. User quits log in. The order is: 1, 2, 3, 4, 5, 6, 7a, 8a.
3. User logs in fail. The order is 1, 2, 3, 4, 5, 6, 7, 8b.

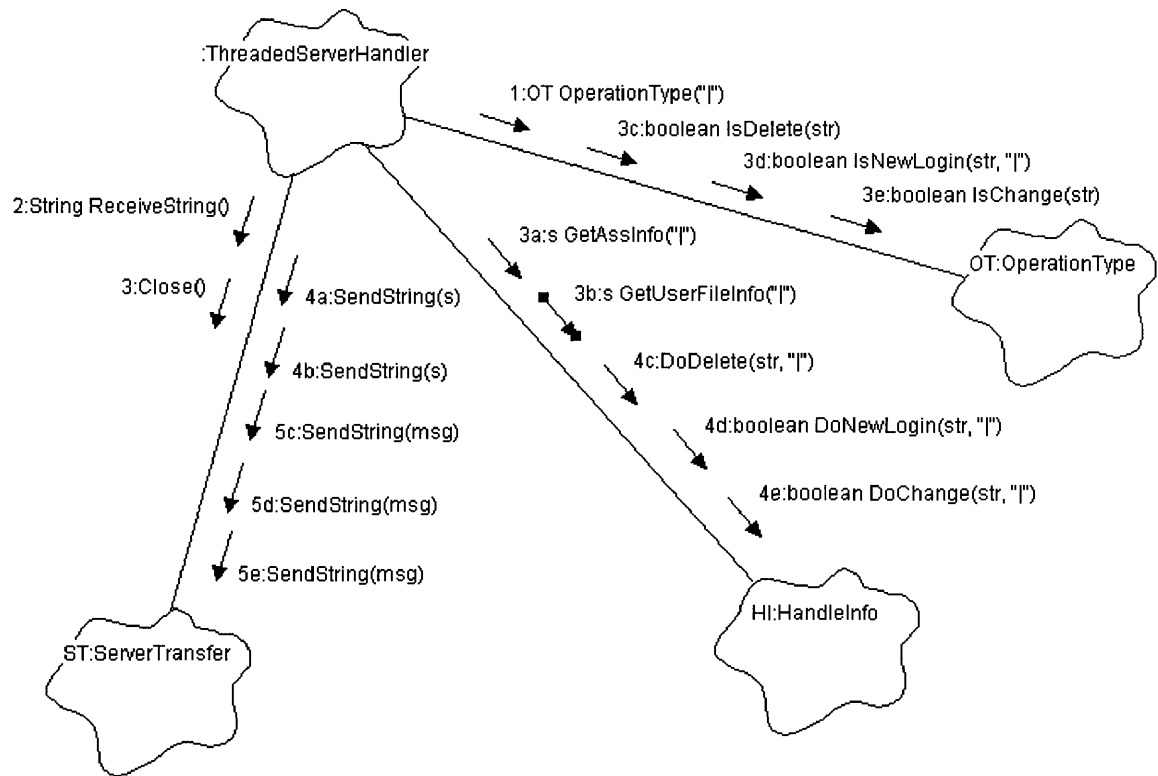


Fig 5. 22 Operation Object Diagram

Fig 5. 22 Operation Object Diagram shows the objects interactions after user successfully logged in. There are six events in it.

1. The client logs out. The order is: 1, 2, 3.
2. The client requests for the assignment information. The order is: 1, 2, 3a, 4a.
3. The client requests for the user file information. The order is: 1, 2, 3b, 4b.
4. Based on the client's request, the server deletes a user previous submission. The order is: 1, 2, 3c, 4c, 5c.

5. The server processes a new log in user. The order is: 1, 2, 3d, 4d, 5d,
6. Based on the client's request, the server changes a user password. The order is: 1, 2, 3e, 4e, 5e.

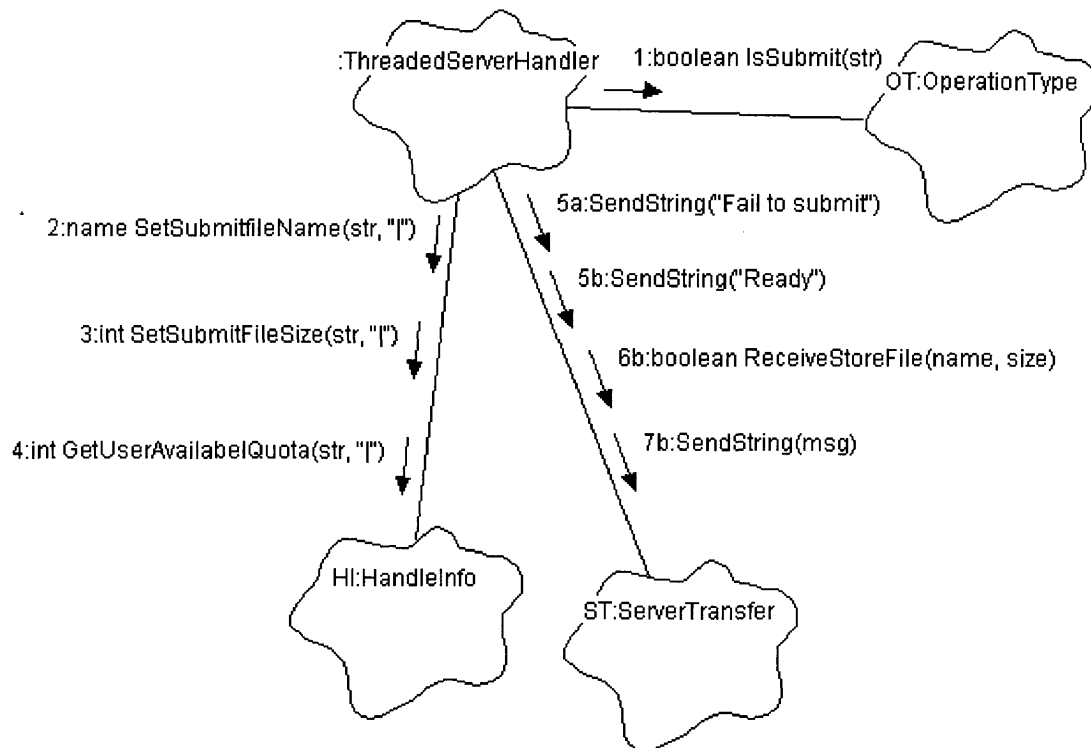


Fig 5. 23 Submit Object Diagram

Fig 5. 23 Submit Object Diagram shows the object interactions for receiving and storing the file. There are two events in this diagram.

1. The client fails to submit. The order is: 1, 2, 3, 4, 5a.
2. The client successfully submits a file. The order is: 1, 2, 3, 4, 5b, 6b, 7b.

5.2.4 State Transition Diagram

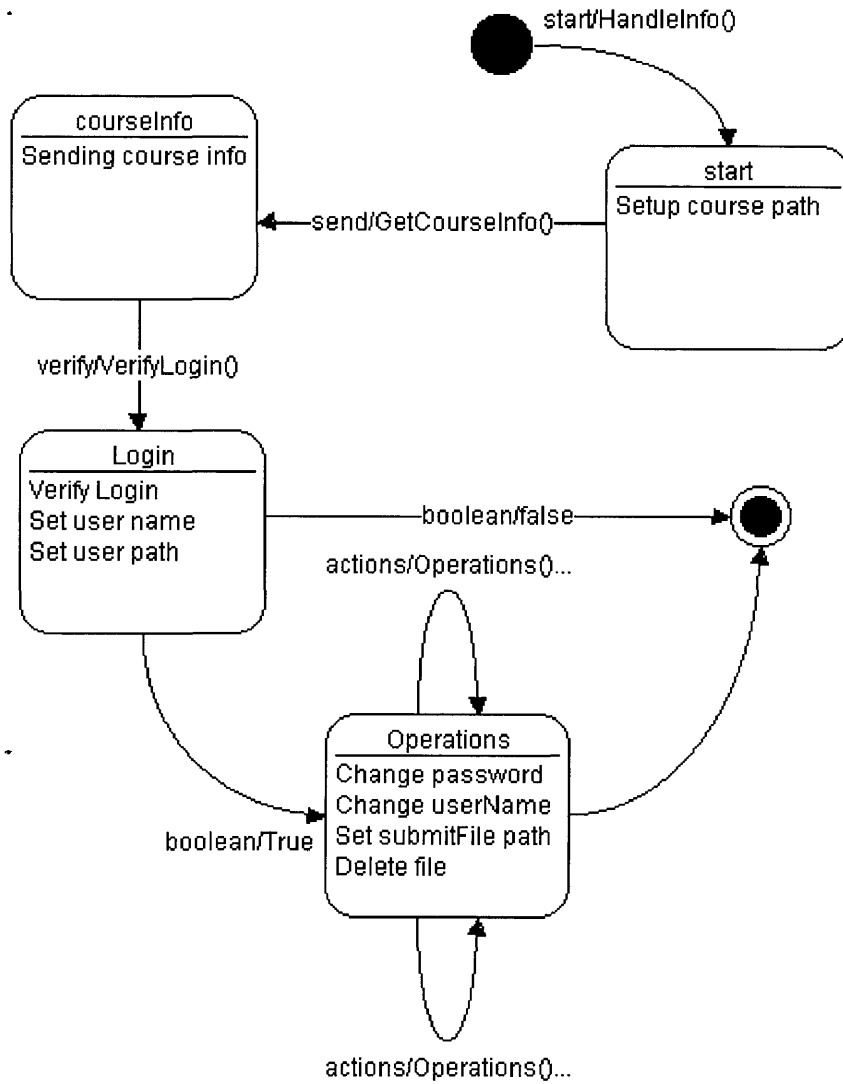


Fig 5. 24 HandleInfo State Transition Diagram

Fig 5. 24 HandleInfo State Transition Diagram is the state transition diagram for class `HandleInfo`.

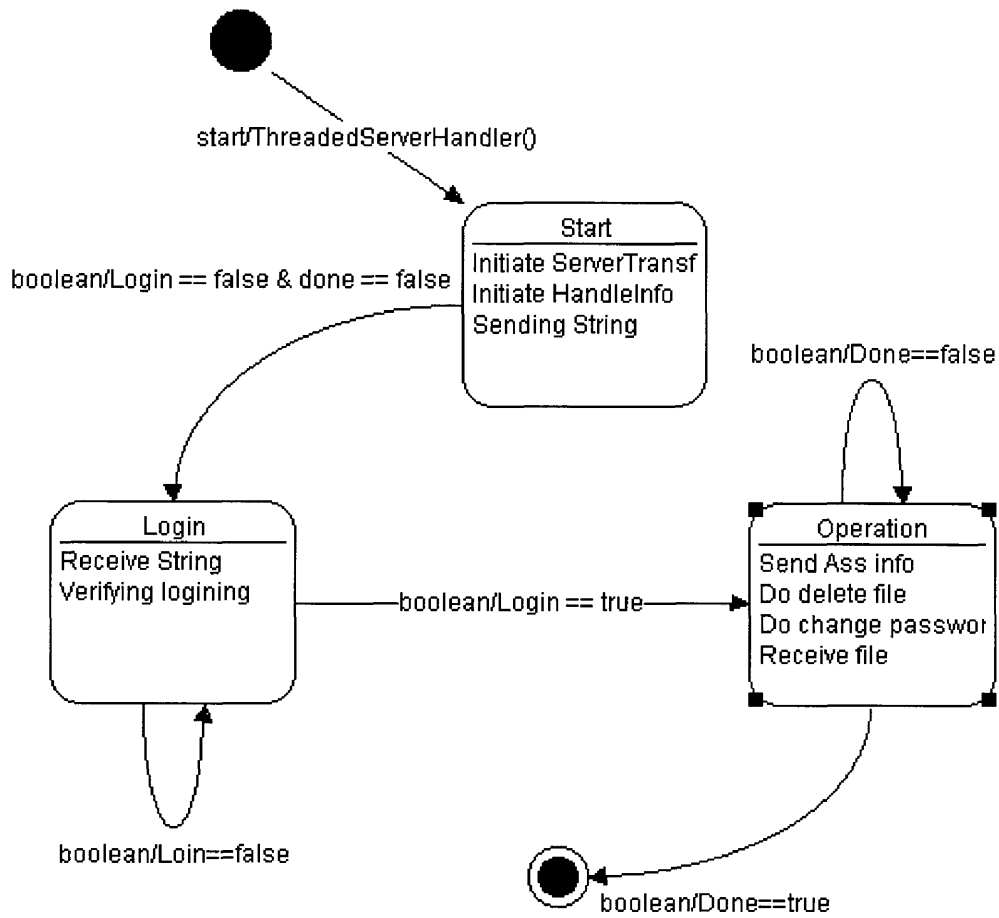


Fig 5. 25 ThreadedServerHandler State Transition Diagram

Fig 5. 25 ThreadedServerHandler State Transition Diagram is the state transition diagram for ThreadedServerHandler class.

5.2.5 Interaction Diagram

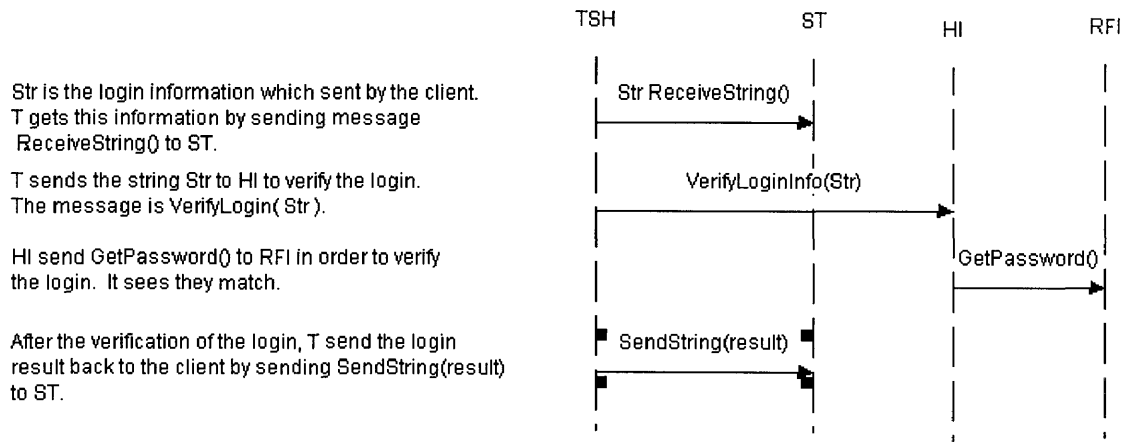


Fig 5. 26 User Log in Interaction Diagram

User successfully logs in process is the scenario for Fig 5. 26 User Log in Interaction Diagram.

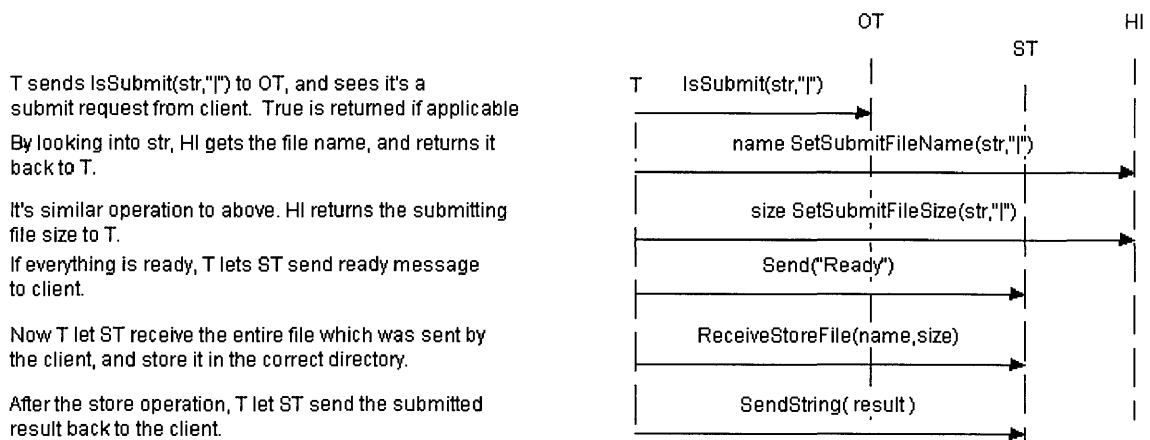


Fig 5. 27 User Submit File Interaction Diagram

User successfully submits a file is the scenario for the Fig 5. 27 User Submit File Interaction Diagram.

5.3 Administrator Program

Since the administrator program is very simple, there is no interaction diagram. The object diagram is enough.

5.3.1 Classes

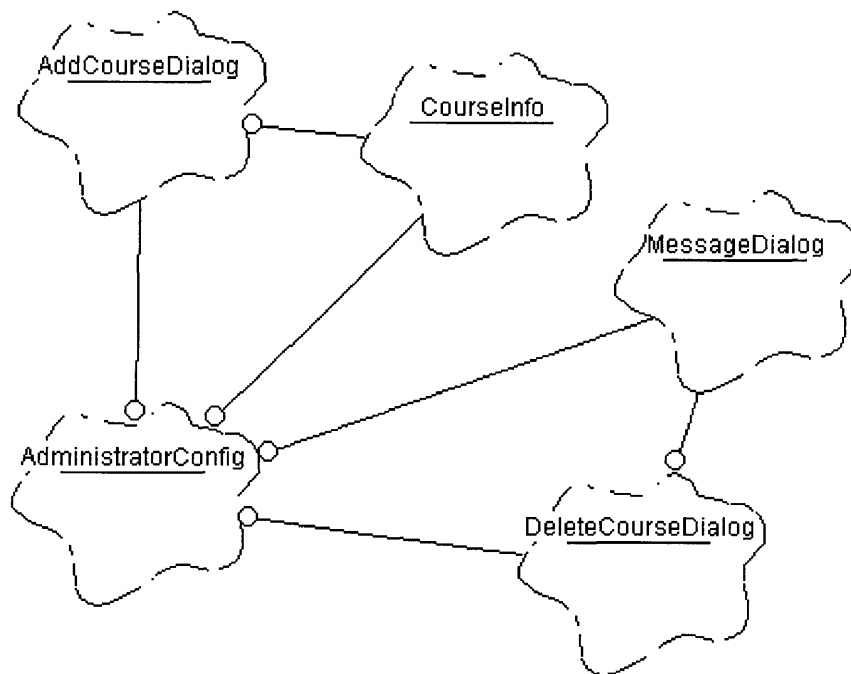


Fig 5. 28 Administrator Class Diagram

Class Description:

Class AddCourseDialog

Super class: Dialog

AddCourseDialog(Frame, CourseInfo)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantic:

- **AddCourseDialog:** It takes Frame parent and CourseInfo u as the arguments, and displays the frame. It lets the administrator enter the course name, the course section number, and instructor's last name. It displays OK and Close button as well.
- **Action:** If OK button is pushed, it takes the three text fields' results and returns them to parent. If Close button is clicked, it exits the program.
- **HandleEvent:** If an object of AddcourseDialog receives window destroy message, it closes the window.

Class CourseInfo

CourseInfo(String, String, String)

.This is container class with course number, section number, and the instructors last name (three fields).

Class DeleteCourseDialog

Super class Dialog

DeleteCourseDialog(Frame, String)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

- **DeleteCourseDialog:** It gets all the information in the course root directory and displays the information in a list. The Delete and Cancel buttons are displayed as well.

- Action: If OK is clicked, it gets the selected item. If the item is not null, it renames the course directory to the trash directory which is under course root path. It displays the message box, as well.
- HandleEvent: If the item is selected or deselected, it stores it. If a window destroy message is received, it closes the window.

Class AdministratorConfig:

• Super class: Frame

AdministratorConfig()

boolean action(Event, Object)

boolean handleEvent(Event)

processResult(Dialog, Object)

Semantics:

- AdministratorConfig: It displays the administrator configuration message and displays Add Course, Delete Course, and Close buttons.
- Action: If Add Course button is pushed, it invokes AddCourseDialog object. If Delete Course button is pushed, it invokes DeleteCourseDialog. If Close button is pushed, it exits the program.
- HandleEvent: If a window destroy message is received, it exits the program.
- ProcessResult: It takes Dialog source and Object result as the arguments. It receives the result of the Add Course operation. If this is the case, it adds the course and displays the result of adding.

Class MessageDialog

Super class: Dialog

MessageDialog(Frame, String)

```
boolean action( Event, Object )
```

```
boolean handleEvent( Event )
```

Semantics:

- **MessageDialog:** It takes Frame parent and String msg as the arguments. It displays the msg in the dialog box and OK button as well.
- **Action:** If OK is pushed, it closes the window.
- **HandleEvent:** If a window destroy message is received, it closes the window.

5.3.2 Module Diagram

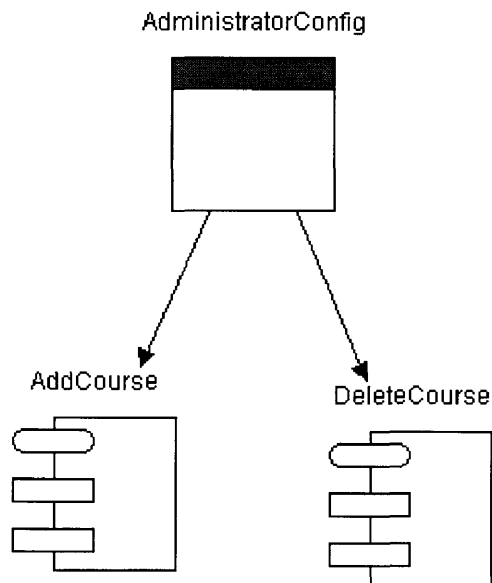


Fig 5. 29 Administrator Module Diagram

AdministratorConfig module has AdministratorConfig and MessageDialog class.

AddCourse module has AddCourseDialog and CourseInfo class.

DeleteCourse module has DeleteCourse and MessageDialog class.

5.3.3 Object Diagram

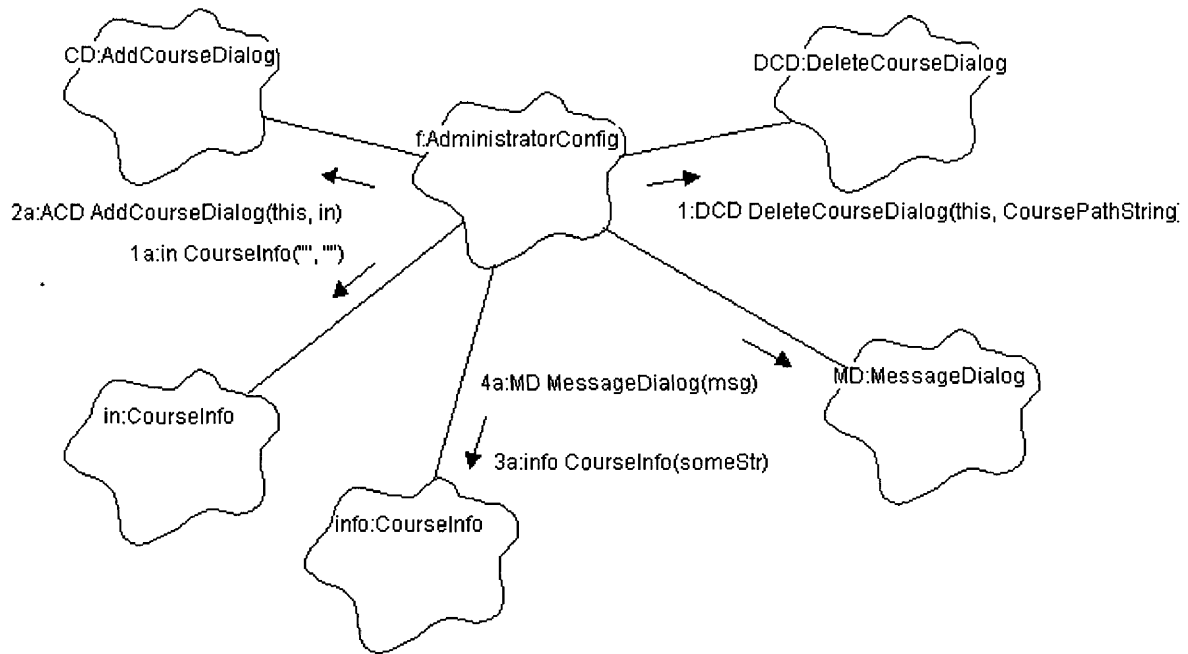


Fig 5. 30 Administrator Object Diagram

Fig 5. 30 Administrator Object Diagram shows the object diagram of the administrator program. There are two events here.

1. Add a course. The order is: 1a, 2a, 3a, 4a.
2. Delete a course. The order is 1.

Since the object interaction of the event 2, delete a course, is similar to the event 1, it only shows the first action of the event 2.

5.3.4 State Transition Diagram

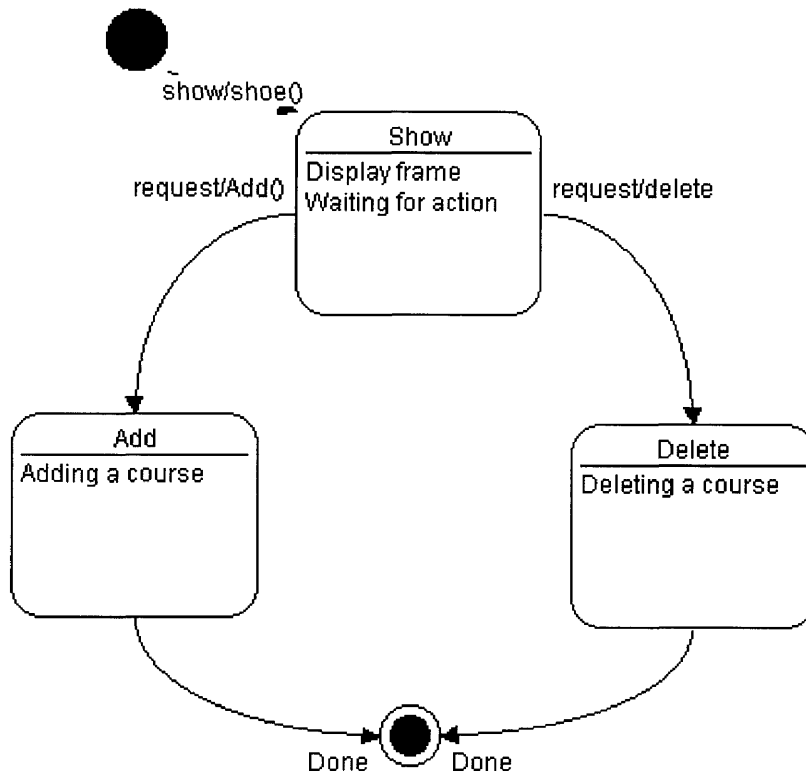


Fig 5. 31 AdministratorConfig State Transition Diagram

Fig 5. 31 AdministratorConfig State Transition Diagram is the state transition diagram of class AdministratorConfig.

5.4 Instructor Program

5.4.1 Classes

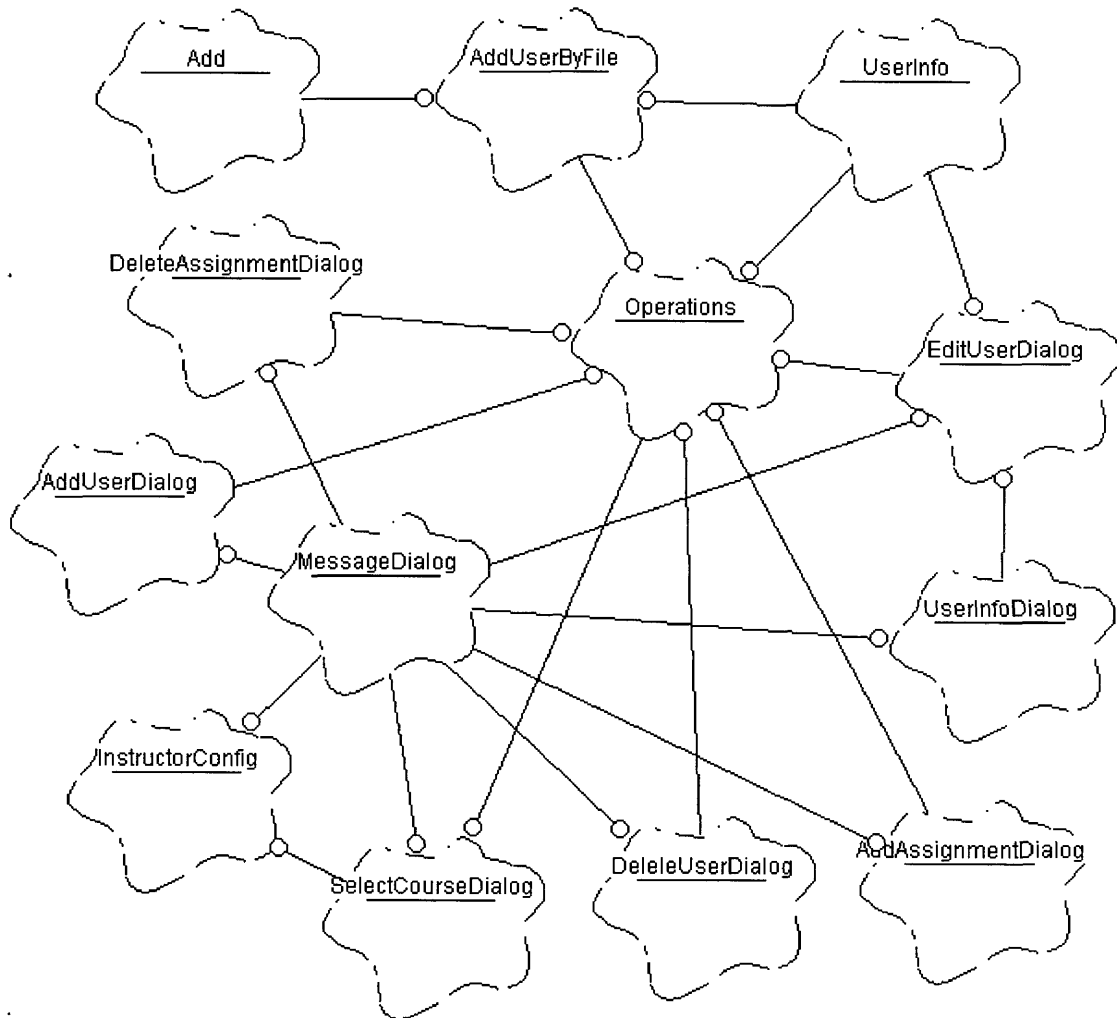


Fig 5. 32 Instructor Class Diagram

Class Description:

Class Add

Add(File)

boolean DoAdd(UserInfo)

Semantic:

- Add: It's the constructor of the Add class. It initializes the user file which is going to be created.
- DoAdd: It creates the user file and stores the user log in name, real name, password, and user quota into the file.

Class Operations

Super class: Frame

Operations(File)

boolean action(Event, Object)

boolean handleEvent(Event)

.

Semantics:

- Operations: It takes File course as an argument and displays Add User by File, Add Single User, Delete User, Edit User, Add an Assignment, and Delete an Assignment buttons on the frame. It shows a Close button as well.
- Action: It will invoke different object based on different button click.

| <u>Button</u> | <u>Object</u> |
|----------------------|------------------------|
| Add User by File | AddUserByFile |
| Add Single User | AddUserDialog |
| Delete User | DeleteUserDialog |
| Edit User | EditUserDialog |
| Add an Assignment | AddAssignmentDialog |
| Delete an Assignment | DeleteAssignmentDialog |

.

- `handleEvent`: If a window destroy message is received, it exits the program.

Class AddUserDialog

Super class: Dialog

`AddUserDialog(Frame, UserInfo, File)`

`boolean action(Event, Object)`

`boolean handleEvent(Event)`

-

Semantic:

- `AddUserDialog`: It takes Frame parent, UserInfo u, and File CourseDir as the arguments and displays the user login name, user real name, and user password, three text input fields. UserInfo u is the default value for these three text input fields. OK and Cancel button are displayed as well.
- `Action`: If OK button is clicked, it gets the text input result and invokes an object of Add class. If text field is null, the error message is displayed. If Cancel button is clicked, it closes the window.
- `HandleEvent`: If the window destroy message is received, it exits the program.

-

Class DeleteAssignmentDialog:

Super class: Dialog

`DeletAssignmentDialog(Frame, File)`

`boolean action(Event, Object)`

boolean handleEvent(Event)

Semantic:

- DeleteAssignmentDialog: It takes Frame parent and File path as the arguments and sets up the assignment directory. It gets all the information in the assignment directory and displays the information in a list. Delete and Cancel buttons are displayed as well.
- Action: If Delete button is pushed, it gets the selected item and moves the assignment directory to the trash directory. The success or error message will be displayed. If Cancel button is pushed, it closes the window.
- HandleEvent: If a window destroy message is received, it closes the window. It stores the selected or the deselected item as well.

Class DeleteUserDialog

Super class: Dialog

DeleteUserDialog(Frame, File)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantic:

- DeleteUserDialog: It takes Frame parent and File CourseDir as the arguments and sets up the user directory. It gets the all the user information from the user directory and displays the information in a list. Delete and Cancel button are displayed as well.
- Action: If Delete button is pushed, it gets the selected item and deletes the user file. The success or error message will be displayed. If Cancel button is pushed, it closes the window.
- HandleEvent: If a window destroy message is received, it closes the window. It stores the selected or the deselected item as well.

Class EditUserDialog

Super class: Dialog

 EditUserDialog(Frame, File)

 boolean action(Event, Object)

 boolean handleEvent(Event)

Semantic:

- **EditUserDialog:** It takes Frame parent and File CourseDir as the arguments, and sets up the user directory. It gets all the user information in the user directory and displays the information in a list. Edit and Cancel buttons are showed on the dialog box, too.
- **Action:** If OK button is pushed, it gets the selected user and opens the user login file. It packs the information read from the login file in an object of UserInfo, and sends it to UID which is an object of UserInfoDialog. If Cancel button is pushed, it closes the window.
- **HandleEvent:** If it receives the window destroy message, it closes the window. It stores the selected or the deselected item.

Class InstructorConfig

Super class: Frame

 InstructorConfig()

 boolean action(Event, Object)

 boolean handleEvent(Event)

Semantic:

- **InstructorConfig:** It displays Select a Course button and Close button.
- **Action:** If Select a Course button is pushed, it will invoke an instance of SelectCourseDialog. If the Close button is pushed, it exits the program.
- **HandleEvent:** If a window destroy message is received, it closes the window. It stores the selected or the deselected item as well.

Class MessageDialog

Super class: Dialog

MessageDialog(Frame, String)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

- **MessageDialog:** It takes Frame parent and String msg as the arguments. It displays the msg in the dialog box , and an OK button is displayed as well.
- **Action:** If OK is pushed, it closes the window.
- **HandleEvent:** If a window destroy message is received, it closes the window.

Class SelectCourseDialog

Super class: Dialog

SelectCourseDialog(Frame, File)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

- **SelectCourseDialog:** It takes Frame parent and File CourseD as the argument. It sets up the course Directory and puts all the information of the course directory in a list. It display the dialog box with the list, Select button and Cancel button.
- **Action:** If OK button is pushed, an object of operation is invoked; otherwise, the error message is displays in a message box. If Cancel button is pushed, it exits the program.
- **HandleEvent:** If it receives the window destroy message, it closes the window. It stores the selected or the deselected item as well.

Class UserInfo:

UserInfo(String, String, String, String)

This is the container class with LoginName, RealName, Quota, and Password (four fields).

Class UserInfoDialog

Super class Dialog

UserInfoDialog(Frame, UserInfo, File, File)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

- **UserInfoDialog:** It displays user log in name, user real name, user quota, and password text field. The default text for these four text fields is passed by an object of UserInfo. OK and Cancel buttons are displayed, too.
- **Action:** If OK button is pushed, it will delete the old user login file and creates a new user log in file with the new information provided. If Cancel is clicked, it closes the window. Error message is displayed in a message box if there is any error.

- **HandleEvent:** If a window destroy message is received, it closes the window. It stores the selected or the deselected item as well.

Class AddUserByFile

Super class Dialog

AddUserByFile(Frame, File, File)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

- **AddUserByFile:** It takes Frame parent, File courseD, and File rosterFile as the arguments, and displays a text field for inputting the user quota. OK and Cancel buttons are displayed, too.
- **Action:** If OK is pushed, it reads the roster file, gets the student's ID number as his/her login name and the real name, and puts them, with user quota, into a log in file. It continues this operation until the EOF of the roster file is reached. If Cancel is clicked, the window will be closed.
- **HandleEvent:** If it receives the window destroy message, it closes the window.

Class AddAssignmentDialog

Super class Dialog

AddAssignmentDialog(Frame, File)

boolean action(Event, Object)

boolean handleEvent(Event)

Semantics:

- **AddAssignmentDialog:** It takes Frame parent and file CourseDir as the arguments. If the assignment directory does not exist, it creates it. It display a text field for inputting the name of the assignment. OK and Cancel buttons are displayed, too.
- **Action:** If OK button is pushed, it takes the assignment name and creates this assignment directory. The error message is displayed if there is an error.
- **HandleEvent:** If a window destroy message is received, it closes the window.

5.4.2 Module Diagram

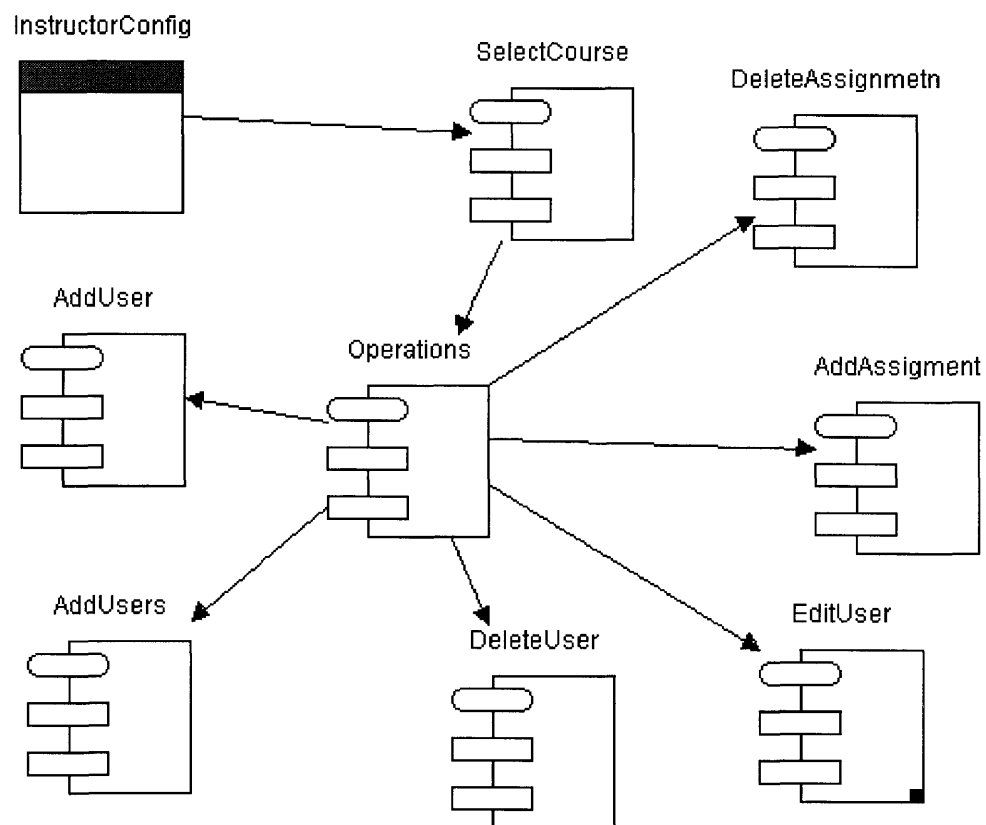


Fig 5. 33 Instructor Module Diagram

InstructorConfig module is the logical main thread of control, and it has InstructorConfig class.

SelectCourse module has SelectCourse class.

DeleteAssignment has DeleteAssignment class.

AddAssignment module has AddAssignment class.

AddUser module has AddUserDialog class.

AddUsers module has Add, AddUserByfile, UserInfo class.

DeleteUser module has DeleteUserDialog class.

EditUser module has EditUserDialog and UserInfoDialog class.

Note: Every module above could have MessageDialog class.

5.4.3 Object Diagram

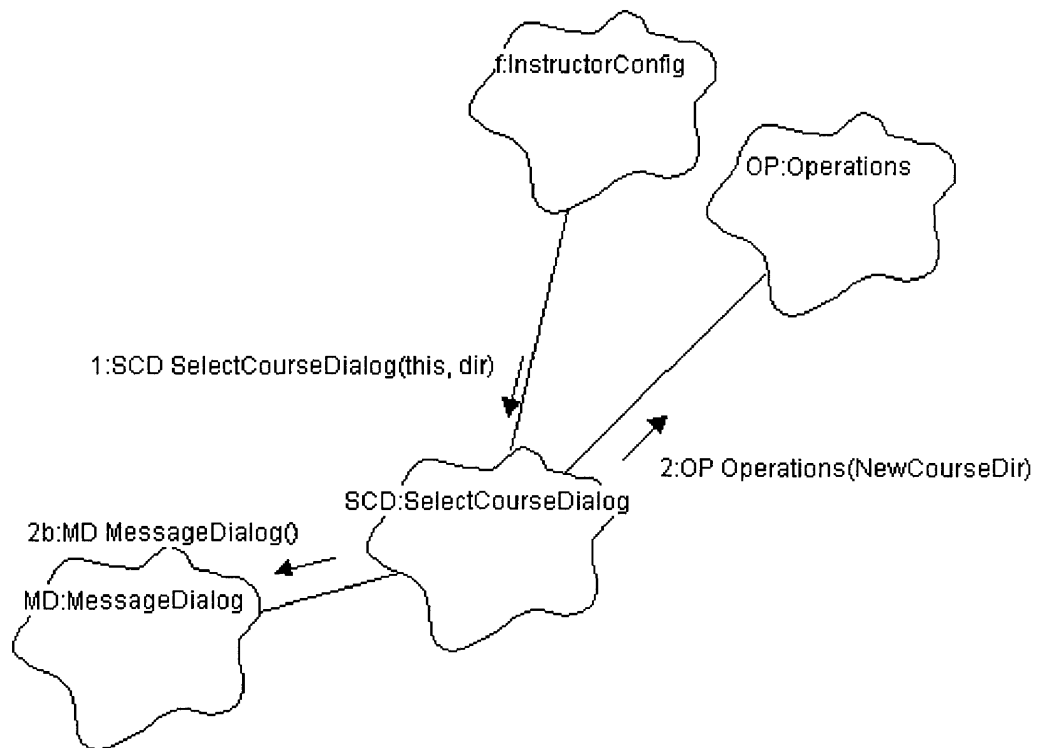


Fig 5. 34 Select Course Object Diagram

Fig 5. 34 Select Course Object Diagram shows the object interaction of selecting course. It contains two events as follow:

1. The instructor successfully selects a course. The order is 1, 2.
2. The instructor fails to selects a course. The order is 1. 2b.

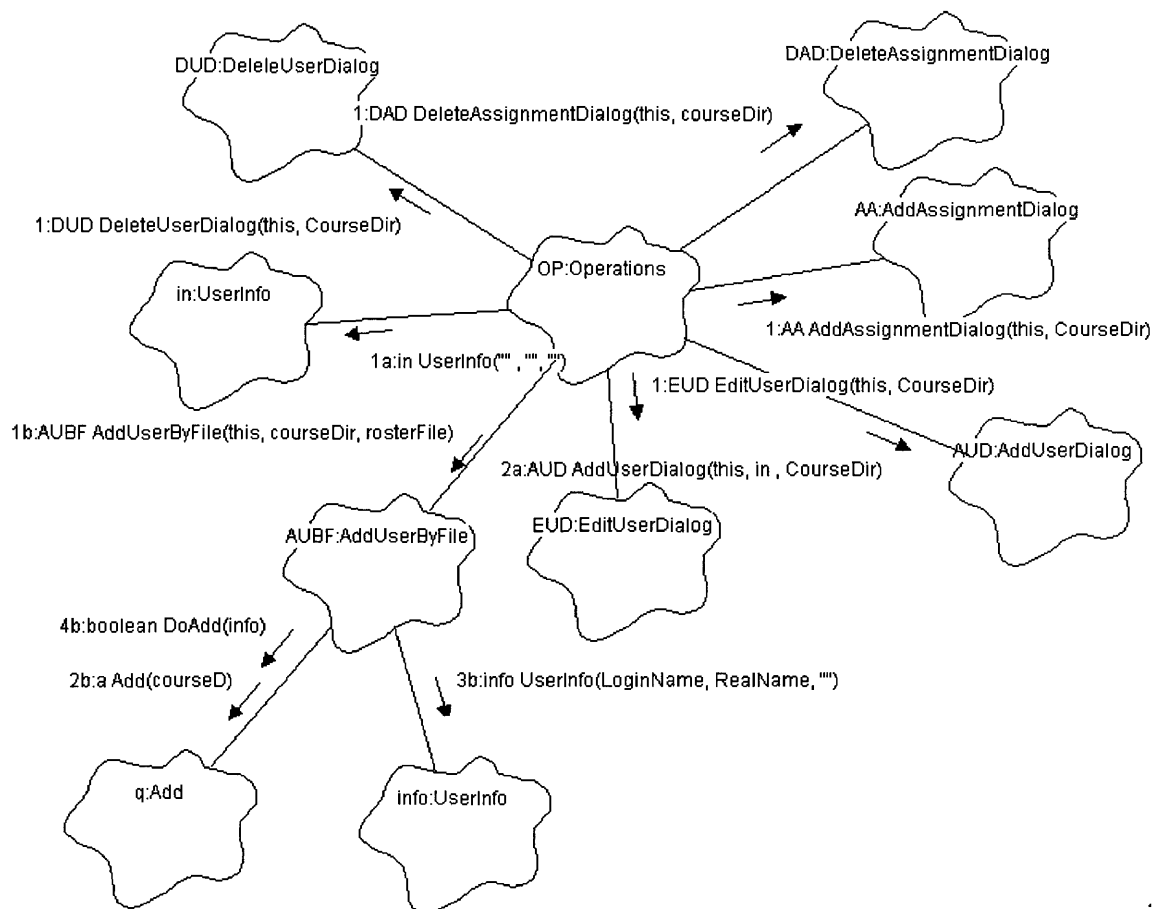


Fig 5. 35 Operations Object Diagram

After the instructor selects a course, Fig 5. 35 Operations Object Diagram shows all the operations the instructor can do.

1. Delete a user. The order is 1.
2. Edit a user. The order is 1.
3. Delete an assignment. The order is 1.

4. Add an assignment. The order is 1.
5. Add a single user. The order is 1a, 2a.
6. Add the users by a file. The order is 1b, 2b.

Notice that the first four events have the same order. Since the operation here is simple, the program executes one or the other.

5.4.4 State Transition Diagram

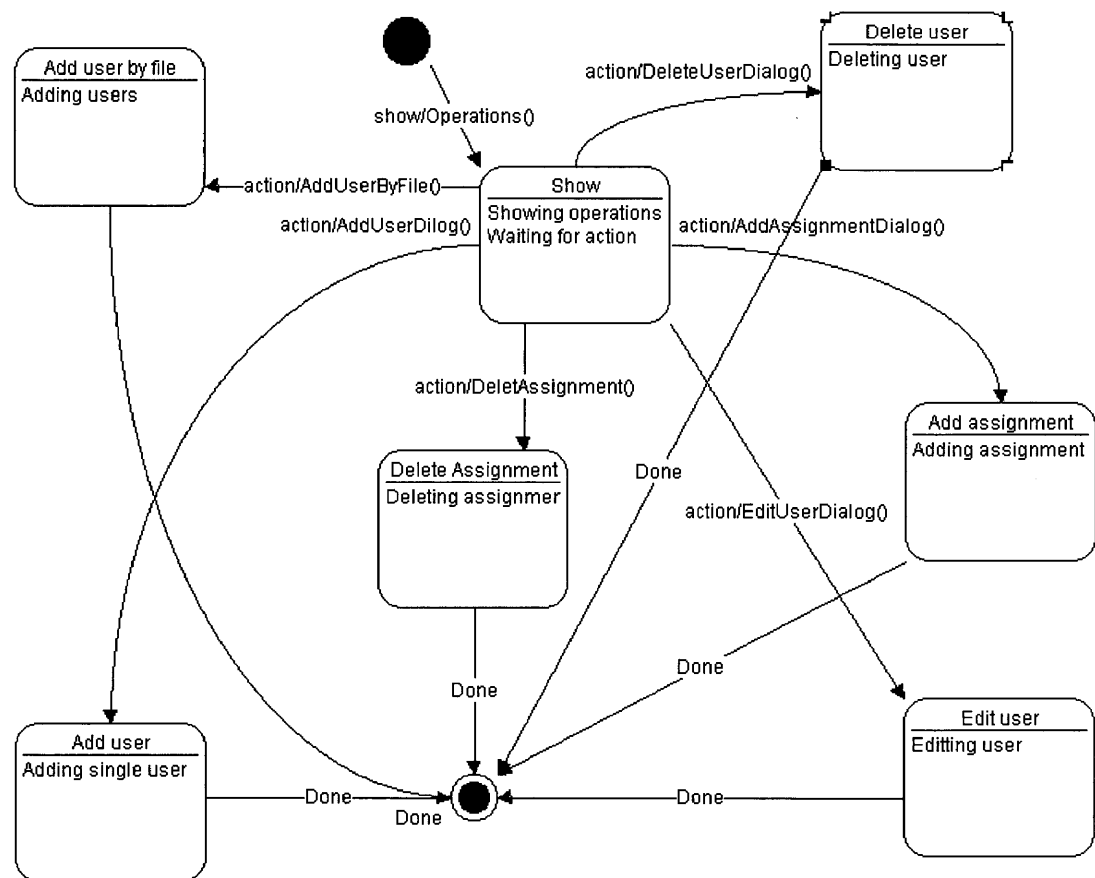


Fig 5. 36 Operation State Transition Diagram

Fig 5. 36 Operation State Transition Diagram shows the state transition diagram of class Operation.

Chapter 6 Implementation

Basically there is a client program and several server programs. The client program will run on any platform. The server program should run on most UNIX-based systems. The client and server programs will communicate using TCP *sockets*.

The actual implementation might be slightly different from the detailed design.

6.1 *Client program*

The client program establishes a TCP *socket* connection to the server program, and then this connection is used to transfer the file to the server and to receive an acknowledgment of receipt from the server. In order to be safe, client program interacts with the server program to authenticate users. The client interface and program chapters have fully described this.

6.2 *Server programs*

These programs will be run under the UNIX environment. If users want, these programs can be run on PC because Java is system independent. After the server receives the file from the client, the server will store the file in a directory specified by the instructor. Since the server program is owned by the instructor, the submitted assignment files will be owned by the instructor. In addition, the server program must run as a daemon process so that it is always available to accept connections from clients. The server programs include server program, instructor program, and administrator program.

6.3 *Prototyping effort*

- Multithreaded prototype completed.
- ASCII or binary file transfer prototype completed.

Before I started to write the program, I did these two prototypes in order to make sure the program task is possible. Because there is only one server and many users, multithread prototype is necessary. The file transfer is the most important part of the project, so the second prototype is necessary, too.

6.4 Why not use Visual C++ or Visual Basic

Because Java is system independent, I decided to use it. Visual C++ or Visual Basic application can be only used under the Microsoft windows operating systems. Therefore, Java is better choice here. As a matter of fact, I did some prototypes in both Visual C++ and Visual Basic, they both can accomplish the task. Since Java is Object-Oriented and multithreaded, it's easy to use for this particular project.

6.5 Project Feature

Whoever runs the server program, will have the ownership of all the submitted files. If there are more than one instructors, it will be a problem. The solution is that each instructor runs his/her own server program. This is the feature of the project. I call it Submitting Service 2. It will allow the instructors to run their own servers, and each instructor will have the ownership of his/her student submissions.

Submitting Service 2 will have a main server program which can allocate all the course information and the sub-server addresses. Each instructor runs a sub-server which handles all the user information and receiving student submissions. After the user starts the client program, it receives the course information from main server. The user makes a selection. Based on the user's selection, the client program finds the sub-server's address and connects it. The rest operations will be the same as I described before.

Reference:

1. Booch, Grady. 1994. *Object-Oriented Analysis and Design with Applications, Second Edition*. The Benjamin/Cummings Publishing Company, Inc..