

CENG 491

SENIOR PROJECT

Detailed Design Report

PDC

(PROJECT DEVELOPMENT CENTER)

PROJECT TITLE: VIRTUAL CLASSROOM

COMPANY STAFF:	Harun Alpak	1249903
	Ömer Faruk Aygün	1297506
	Ufuk Biçen	1297555

INTRODUCTION.....	3
A.1 Purpose of This Document.....	3
A.2 Purpose and Scope of This Project.....	4
A.3 Overall Description of The Project.....	5
.....	7
B. PROJECT MANAGEMENT PLAN.....	7
B.1 Approach and Methodology.....	7
B.2 Major Milestones and Project Schedule.....	8
B.3 Time and Effort Estimation.....	9
B.4 Technical Requirements	9
C. DIAGRAMS.....	12
C.1. DFD & CFD DIAGRAMS.....	12
C.1.1. DFD & CFD LEVEL 0.....	12
C.1.2. DFD & CFD LEVEL 1.....	13
C.1.3.1. WHITEBOARD DFD&CFD LEVEL 2.....	15
C.1.3.2. CHAT TOOL DFD&CFD LEVEL 2.....	15
C.1.3.3 Q/A BOX DFD&CFD LEVEL 2.....	16
C.1.3.4 STREAMING DFD&CFD LEVEL 2.....	17
C.2. USE CASE DIAGRAMS.....	19
C.2.1 BEFORE VIRTUAL CLASSROOM.....	19
C.2.2. COMMUNICATION & STREAMING.....	20
C.2.3. WHITEBOARD	21
C.3. CLASS DIAGRAMS.....	22
C.4 SEQUENCE DIAGRAMS.....	24
C.4.1 LOGIN MODULE.....	24
C.4.2 INSTRUCTOR ACTIONS BEFORE VIRTUAL CLASSROOM.....	25
C.4.3. STUDENT ACTIONS BEFORE VIRTUAL CLASSROOM.....	26
C.4.4. CHAT.....	27
C.4.5. INSTRUCTOR-STUDENT INTERACTION	28
C.4.6. WHITEBOARD OPERATIONS.....	29
C.4.7. SLIDES ON WHITEBOARD.....	30
C.5. STATE CHART DIAGRAM.....	32
D. CLASS DESCRIPTIONS.....	33
E. GRAPHICAL USER INTERFACES.....	57
E.1. LOGIN INTERFACE.....	57
E.2 Admin Interface.....	58
E.3 Instructor Interface.....	58
E.4 Virtual Classroom.....	59
F. CONCLUSION.....	59
G. APPENDIX.....	61

INTRODUCTION

A.1 Purpose of This Document

This document is written as a result of the design steps carried out whole semester by

PDC group. This document covers all the initial design approaches, design steps, analysis and milestones of the system and establishes a basis for the implementation of the total project in the future.

The first part of final design report is project management. In this part complete and detailed information about approach, methodology, technical requirements, estimations, schedule, and major constraints exist. Project Management step is the first major part of our project design.

Next part, diagrams part covers revised versions of the previous data flow and control flow diagrams, use case diagrams, class diagram, sequence diagrams and state chart diagrams that are designed with UML.

The following part can be counted as Architectural design part in which important points of architectural design exist. In this part we gave all modules together and showed interactions of each other. Also we explained all classes and their properties in this part.

The next part is the GUI design where some sample designed GUI 's are explained and displayed.

This document aims to give a clear and detailed explanation about all design steps of “Virtual Classroom project “carried out by PDC.

A.2 Purpose and Scope of This Project

The growing popularity of Internet and e-learning introduced new terms to education, such as “virtual classroom”. May be in the near future students will not go to anywhere in order to take diploma. The concept “virtual classroom” defines a simulation of the real classroom that enables users to attend a class from anywhere through Internet and provides a learning experience. This project is intended to give both teacher and student the ability to do their jobs effectively and efficiently without even leaving their places. The project is simply described as modeling a real world object (i.e. classroom) into computer world.

By this solution we aimed to meet the expectations of both students and instructors. Our system generally has the following features and capabilities:

Student Side:

- If there exists an online course the student will log in the virtual classroom
- In the virtual classroom the student will be able to communicate in real time with other students by our chat tool
- Also the student will be able to ask questions to instructor by question/answer tool
- During lectures the students will watch the instructor in real time
- The students will not be able to use whiteboard unless the instructor gives permission

Instructor Side:

- The instructor will start any course after logging in to the system
- The instructor will be able to upload some documents like presentation slides to a folder that will be used during lectures
- The system will provide an offline tool to create some material that will be used during lectures also the instructor will be able to edit the existing course material (uploaded before)
- In the virtual classroom the instructor will use whiteboard to explain the lectures
- The instructor will be able to open some slides on the whiteboard area and will be able to edit them, at this time all the clients (students) will observe the changes
- The instructor will have chance to kick off some students from virtual classroom due to their bad behaviours

A.3 Overall Description of The Project

Design Constraints and Assumptions

- While developing our solution we made some assumptions and put some constraints. First of all we assumed that our system would not have a very complicated database. We will keep a very limited database that will store only users and courses. In our analysis report and initial design report we have designed our system with a full course and user management system. In this final design we assumed that all this parts will be handled with other solutions.
- Another constraint is that there will be just only one online course in the virtual classroom and at most thirty students can be in the class.

- And also the system will not work in the browser, it will be an apart application.
- During lectures only instructor will use whiteboard.
- Our limited database will reside on server machine
- As mentioned before our system will have a very limited database usage but for database issues there will admin user type that he/she will create user or course.

Overall Description

There will be three kind of user of this system that they will interact with. The first is the “admin”, the second kind is the “instructor” and the last actor is the “student”. Admin will have very limited function in our final design. Admin will log in the system in the log in menu and the system will direct this user to admin interface. In this module there will be options to create user and course into the database. These are the full responsibilities of admin.

The second user type is the student. The students will be recorded in the database and they will login to the system like any other user. When students write username and password to the appropriate fields and clicks login button the system will check the specified user name and password with the database data. If there exists an online course in the virtual classroom the student will direct to the virtual classroom. If there exists no online course at that moment the system will notify the student and return.

The last user type is the instructor. The instructors will enter the system from the same login menu by specifying username, password and user type like all other users. If the instructor login is successful the system will display all the courses recorded to the database. By this way any instructor will be lecturer of any course recorded in the database. In this intermediate interface the instructor will have option of creating material in order to use during course or directly starting of a class. If instructor selects a course from the list and clicks “create material” button a whiteboard-like tool will be displayed. By using this menu the instructor can edit existing course material that is prepared before or create new material by using the functions of this tool. This tool is designed in a user-friendly approach by providing very basic but enough primitives to create a PowerPoint-like material. After

creating the material instructor will save this document in a specific folder in the server machine. And if instructor wants he/she can start a class from this interface.

In the virtual classroom instructor is the only user who has permission to use the whiteboard. Students will not be able to use the whiteboard unless instructor gives permission. Whiteboard is an MSPaint-like tool. By using this tool instructor draw something on the board like line, ellipse, rectangle and random lines by specifying color and size of the pen. Also he/she can copy, paste, delete drawn objects by selecting. In addition to these capabilities instructor may open an existing slide on the board and edit it by whiteboard drawing tool. When instructor opens a slide on the board every other client will observe in real time.

In the virtual classroom a chat tool will provide a real time communication between users. The users will chat in a general chat room, so students will not talk to each other privately. If a new user joins or leaves the classroom the server will inform each client by this tool. A version of chat tool is question/answer tool that will enable students to ask questions to instructor. When student writes a question and send it to the instructor, the system will inform the instructor. By this way the student may participate to the course.

Another option for instructor is to give permission to a specific student to use whiteboard. He/she will do this action by editing the user list box. By changing the permission of students the instructor may kick off some students from the course due to their bad behaviours.

Another function of this system is streaming. During lectures the instructor's voice and view will be captured and broadcasted to all clients.

B. PROJECT MANAGEMENT PLAN

B.1 Approach and Methodology

We are using the oldest and most widely used software engineering paradigm that is called “Linear Sequential Model”. Since our team is a small team we cannot use RAD and incremental methods. It is very important for our model to state all the requirements explicitly; therefore we spent very long time to find out all the requirements at the beginning. We made customer contacts and searched through Internet in order to remove the uncertainties at the beginning. At each step (Analysis – Design – Code - Test) of this model we are making or will make duty distribution among team members. Although it has some weaknesses, it is the most suitable model for our team.

As a team organization system our model is “Controlled Decentralized” (CD) team model. The project is not so huge so we did not choose “Controlled Centralized” (CC) model. And also we thought that the “Democratic Decentralized” (DD) model could be time consuming for our team. Having a permanent team leader is most suitable for our team spirit, therefore we chose CD model.

B.2 Major Milestones and Project Schedule

In the development phase of the project we will have nine major milestones. During the development of project we will try to obey these milestones. We are in the prototype phase currently. Project schedule is added in the Appendix part.

- **Determination of Project Scope:** The scope of the project was determined. The boundaries of the project were specified. The method and methodology that will be followed was decided.

- **Gathering Requirements:** Examining of the existing virtual classroom systems, interviews with the students and the instructors were done. In this case the “Smart Class” program of METU was examined and a meeting was conducted with supervisor of this program.

- **Analysis of Requirements:** The advantages and disadvantages of the existing systems were listed and, new features and the most efficient and useful components were analyzed that can function in an ideal system.

- **Architectural Design:** Modules of the system were constructed and combined

- **User Interface Design:** An easy to understand, functional and user friendly GUI were developed.

- **Developing Prototype:** A prototype will be developed in order to show the system with its main features and functionalities.

- **Implementation:** The implementation phase will take place in the second semester since all the requirements of the system will be clear and architectural design will be successfully finished.

- **Testing:** In the testing part white box testing and black box testing methods will be used. Since we want to develop a system with the least possible bugs we will spend some more time on this phase.

- **Documentation:** The user manuals and the help specifications of the system will be developed.

B.3 Time and Effort Estimation

Estimations are essential to have a general idea about the schedule, cost, effort. These are required in the early phases of the project. Then we restore the estimates with the help of metrics. We will use these metrics to determine progress and to estimate future projects. If we can make our plans according to these estimations then it will be easier to manage risks and increase efficiency.

We used Empirical Model to have an idea about estimation. This estimation model formulized as $E = L^3 / P^3 * t^4$ where L is lines of code, P is the productivity factor, E is the effort in year-person and t is the time of the project in years. We made the following assumptions:

$$L = 15000 \text{ LOC}$$

$$P = 16000$$

$$t = 7 / 12 \text{ then}$$

$$E = 15000^3 / 16000^3 * 0.58^4 = 7.12 \text{ person/year}$$
$$= 0.59 \text{ person/month}$$

B.4 Technical Requirements

Software Requirements:

- **Server:** Since we will use Microsoft .NET products, we are planning to use Windows Server 2003 or any other Microsoft's solution
- **Database:** A Microsoft compatible database management system like MSSQL Server2000 will be enough.
- **Multimedia:** For video and audio streaming issues we need the following components
 - Windows Media Server
 - Microsoft Media SDK packages
- **Development:** For development phase of our system we decided again on Microsoft's products. Our development platform will be .NET and we are planning to use the following tools and languages.
 - Visual C# .NET programming language for main development
 - Microsoft Visual Studio .NET 2003 as development tool.
- **Other Development Software:**
 - Windows XP operating system
 - MS Office Packet for reports or any other documents
 - Microsoft Visio for diagrams
 - SmartDraw for diagrams or time charts
- **Client Side:** A proper working operating system that .NET Framework is installed on.

Hardware Requirements:

- **Server:** We need a reliable server for our system. This machine must be fast and must show high performance in all situations. At least 512MB RAM and Pentium 4 2000MHz processor seems to be the minimum requirements for this machine. Any IBM, HP machine can be selected for this purpose
- **Database:** We do not need any extra hardware for this sense since we do not need much database applications.
- **Developers:** To develop the system each group member needs a good PC with at least Pentium 4 2000MHz, 256MB Ram and 20Gb Hard disk having machines. And also the other requirements like mouse, keyboard etc.
- **Web cam:** for video transmitting a web cam will be needed also

Development Platform Analysis

We will use .NET as for our development platform that it will ease our work. Microsoft announced .NET in June 2000. It is a very young technology that has a broad new vision for integrating the Internet and the Web in the development, engineering and use of software. One key aspect of the .NET strategy is its independence from a specific language or platform. Rather than forcing developers to use a single programming language, developers can create a .NET application in any .NET-compatible language. Programmers can contribute to the same software project, writing code in the .NET languages (such as C#, Visual C++ .NET, Visual Basic .NET and many others) in which they are most competent. The .NET architecture can exist on multiple platforms, further extending the portability of .NET programs. In addition, the .NET strategy involves a new program-development process that could change the way programs are written and executed, leading to increased productivity. The strategies of .NET can be categorized as follows:

- The .NET strategy extends the concept of software reuse to the Internet, allowing programmers to concentrate on their specialties without having to implement every component of every application. Instead, companies can buy Web services and devote their time and energy to developing their products. The .NET strategy further extends the concept of software reuse to the Internet by allowing programmers to concentrate on their specialties without having to implement every component. Visual programming has become popular, because it enables programmers to create applications easily, using such prepackaged components as buttons, text boxes and scrollbars. Similarly, programmers may create an application using Web services for databases, security, authentication, data storage and language translation without having to know the internal details of those components.
- The .NET strategy incorporates the idea of software reuse. When companies link their products in this way, a new user experience emerges. For example, a single application could manage bill payments, tax refunds, loans and investments, using Web services from various companies. An online merchant could buy Web services for online credit-card payments, user authentication, network security and inventory databases to create an e-commerce Web site.
- *Universal data access* is another essential concept in the .NET strategy. If two copies of a file exist (such as on a personal and a company computer), the less recent version must constantly be updated—this is called file *synchronization*. If the separate versions of the file are different, they are *unsynchronized*, a situation that could lead to

errors. Under .NET, data could reside in one central location rather than on separate systems. Any Internet-connected device could access the data (under tight control), which would then be formatted appropriately for use or display on the accessing device. Thus, the same document could be seen and edited on a desktop PC, a PDA, a cell phone or other device. Users would not need to synchronize the information, because it would be fully up-to-date in a central area

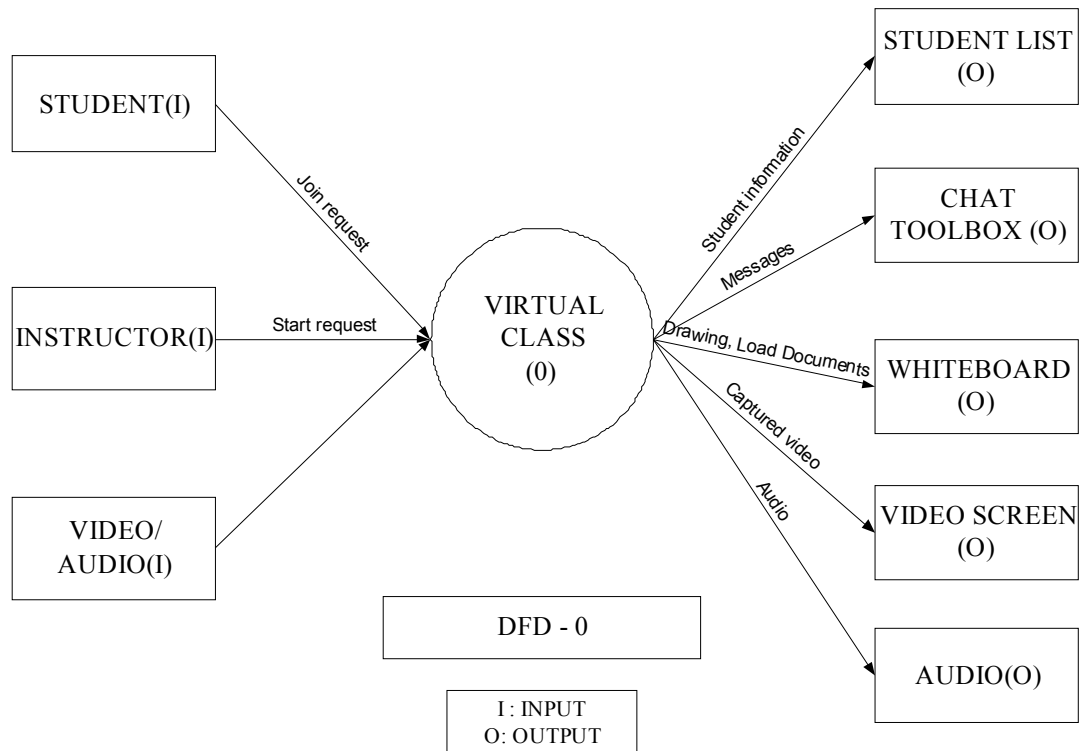
C. DIAGRAMS

C.1. DFD & CFD DIAGRAMS

When the students and also instructor entered this system, all of the activities are controlled by this system. All of the tools and their interaction with the users are managed and all of the responses are given by this system.

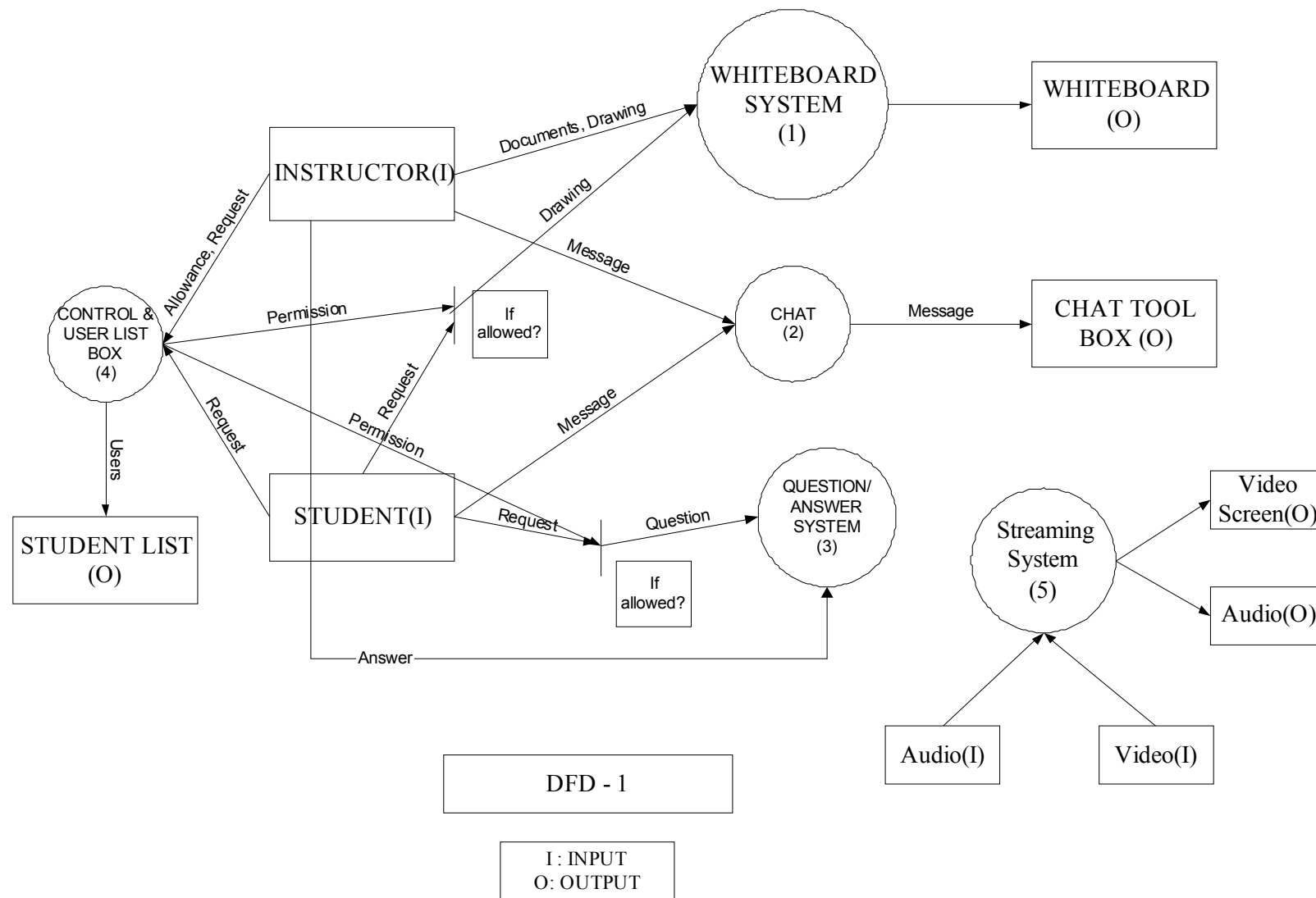
C.1.1. DFD & CFD LEVEL 0

As you can see in the following graph representation of our virtual classroom component, we have 3 inputs and 5 outputs. Besides that, we have one main process of our system, which control all of the tools of the system. All process works under this main process of our system. Our inputs of this component are student, instructor and video/audio streaming device. And outputs are student list, chat toolbox, whiteboard, video screen, and audio. The important processes of this component are Whiteboard, Chat tool, and video/audio streaming system. All of them receive some data from users than all of them is distributed all of the members by this system.



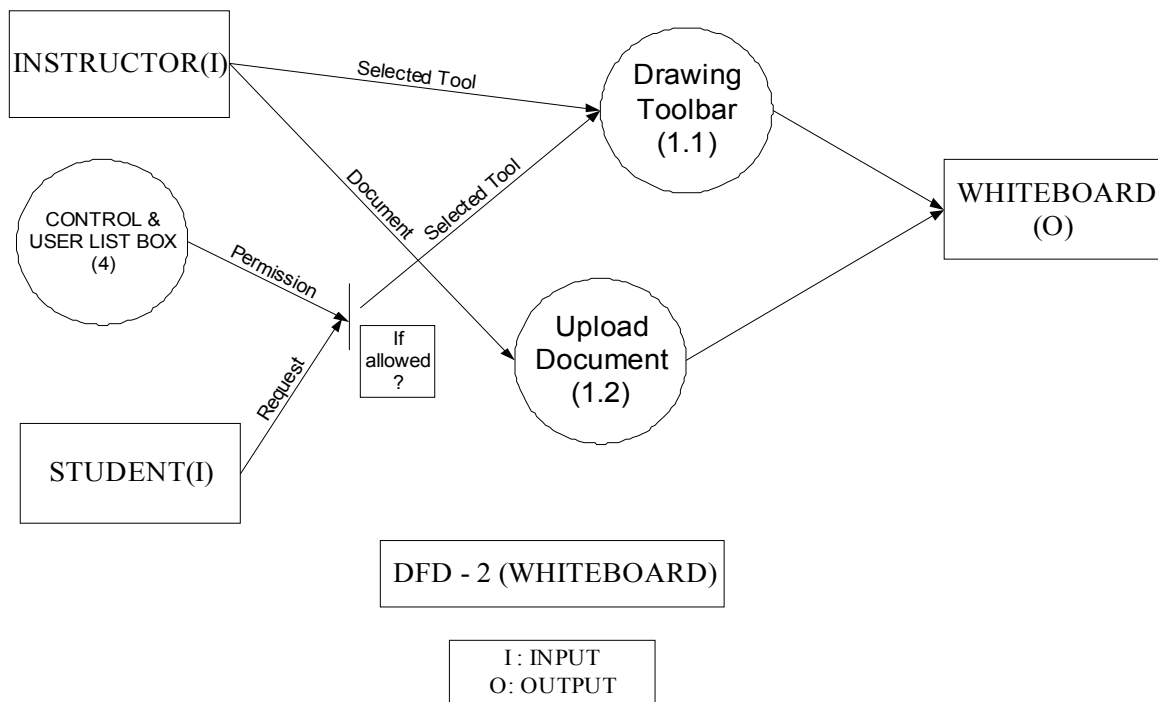
C.1.2. DFD & CFD LEVEL 1

In this system we have 5 main processes, which are whiteboard system, chat, Question/Answer system, Control & User list box, and Streaming system. In the whiteboard only instructor can use this tool but if instructor gives permission to the specific student then the student can also use the whiteboard for asking question. However all of the users can use the chat tool. Question/Answer system provides the communication between instructor and the students. When I student have a question then ask question by using this system. Control & User list box process show the user info and also students' permissions and these permissions are activated/deactivated by the instructor. Streaming system is one of the important processes of this component. The captured video or audio, which is provided by web cam or any camera and microphone, is streamed to students by this system.



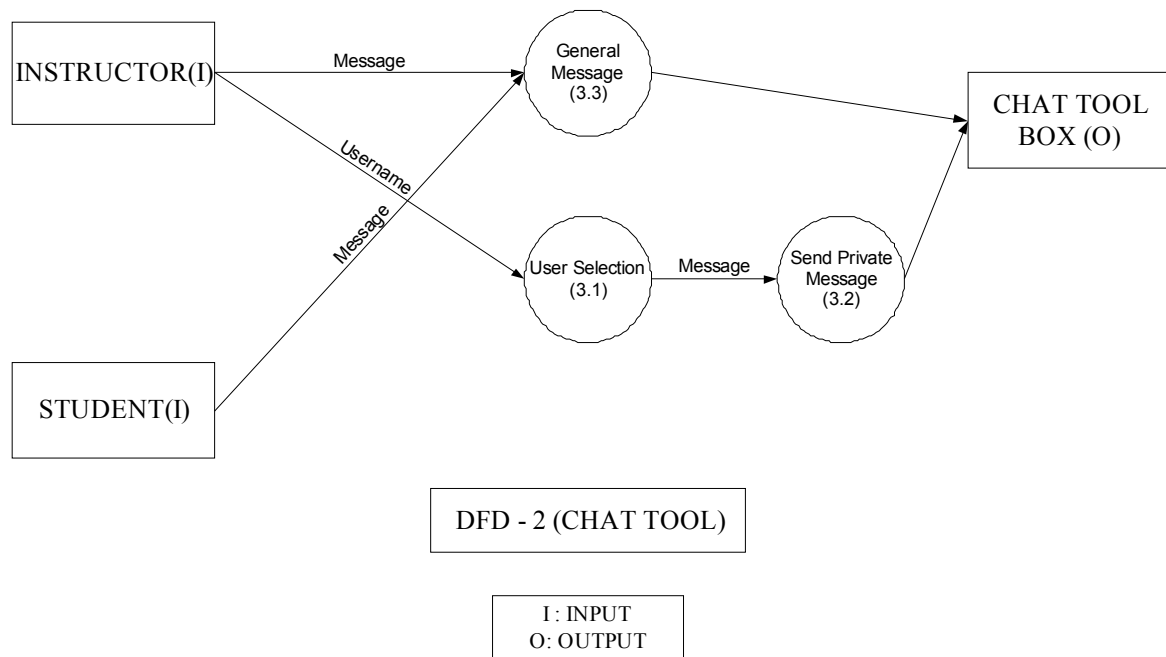
C.1.3.1. WHITEBOARD DFD&CFD LEVEL 2

As you can see the following graphical representation of the whiteboard system, only the instructor can use the whiteboard. All of the drawings and writing is saved in specific place then it is sent all of the users as an output. If the instructor gives permission to the user then user can use white board system. In the corner of the whiteboard there is a drawing toolbar, which makes the drawing easier. Moreover, instructor can also upload their documents, such as power point file, image or text document, on whiteboard.



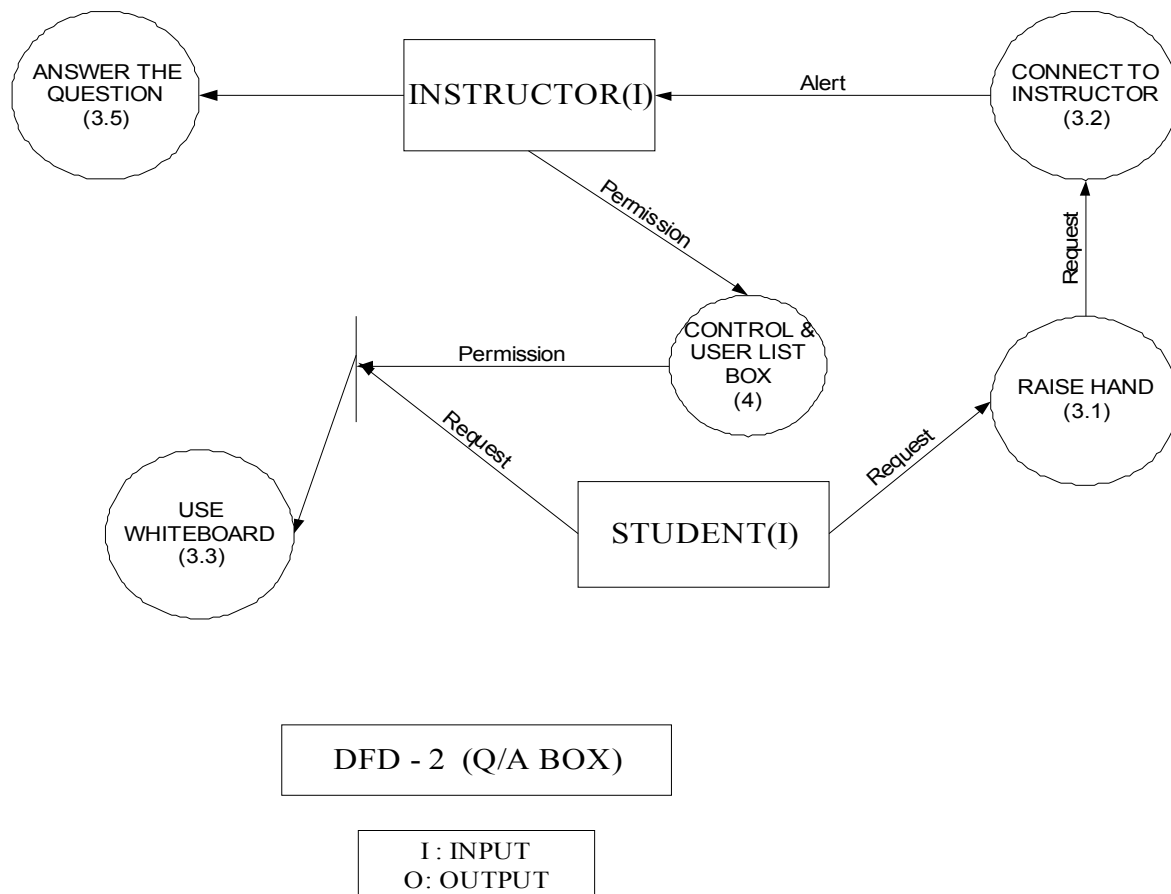
C.1.3.2. CHAT TOOL DFD&CFD LEVEL 2

We have two main processes of this chat tool. These are general message and private message. Only the instructor uses the private message. But general message can be used by all of the users.



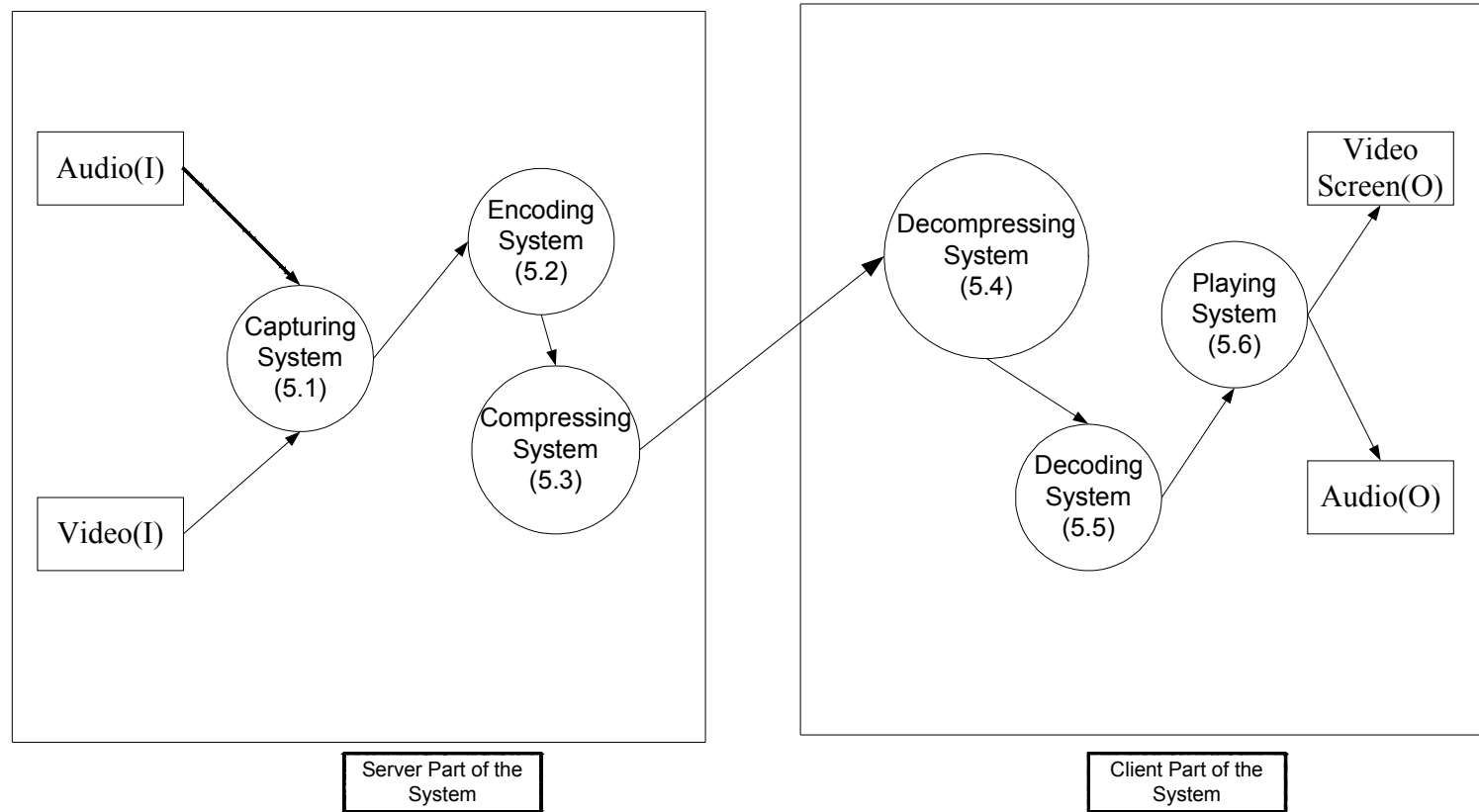
C.1.3.3 Q/A BOX DFD&CFD LEVEL 2

This system provides the interaction between the students and the instructor. When a student wants to ask a question, the system alerts the instructor. Then when the instructor gives the permission to student then this student can use the white board for asking the question. Moreover, instructor can also get the control of whiteboard any time then answer the question.



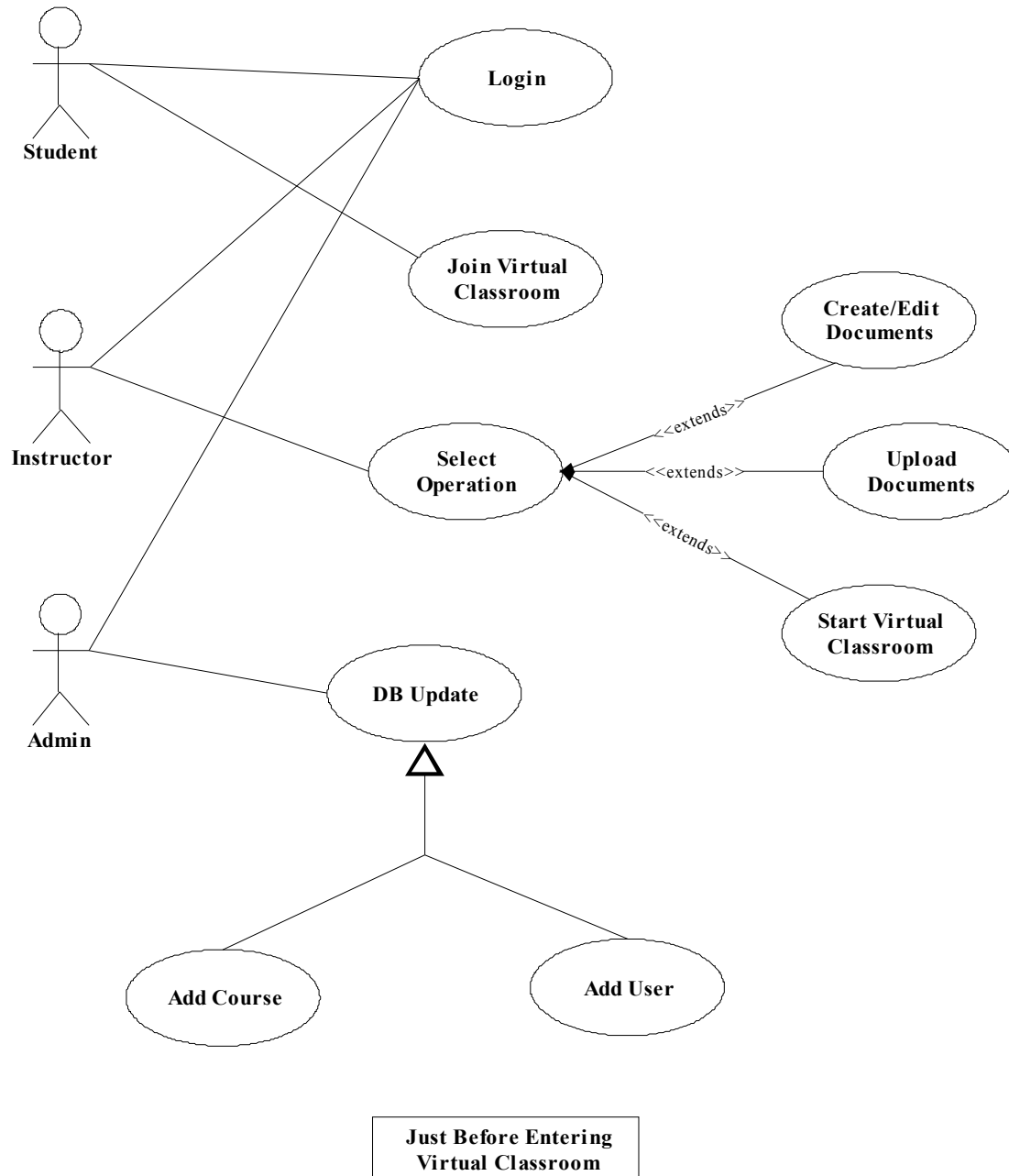
C.1.3.4 STREAMING DFD&CFD LEVEL 2

The main processes of this system are capturing, playing, streaming, compressing, encoding and decoding. This system firstly captures the video and audio from the web cam and microphone and then encodes and then compresses these encoded files. And all of the compressed files are streamed to the other users. When the other users get these files then the client part of the process firstly decompresses and then decodes these files by media player.

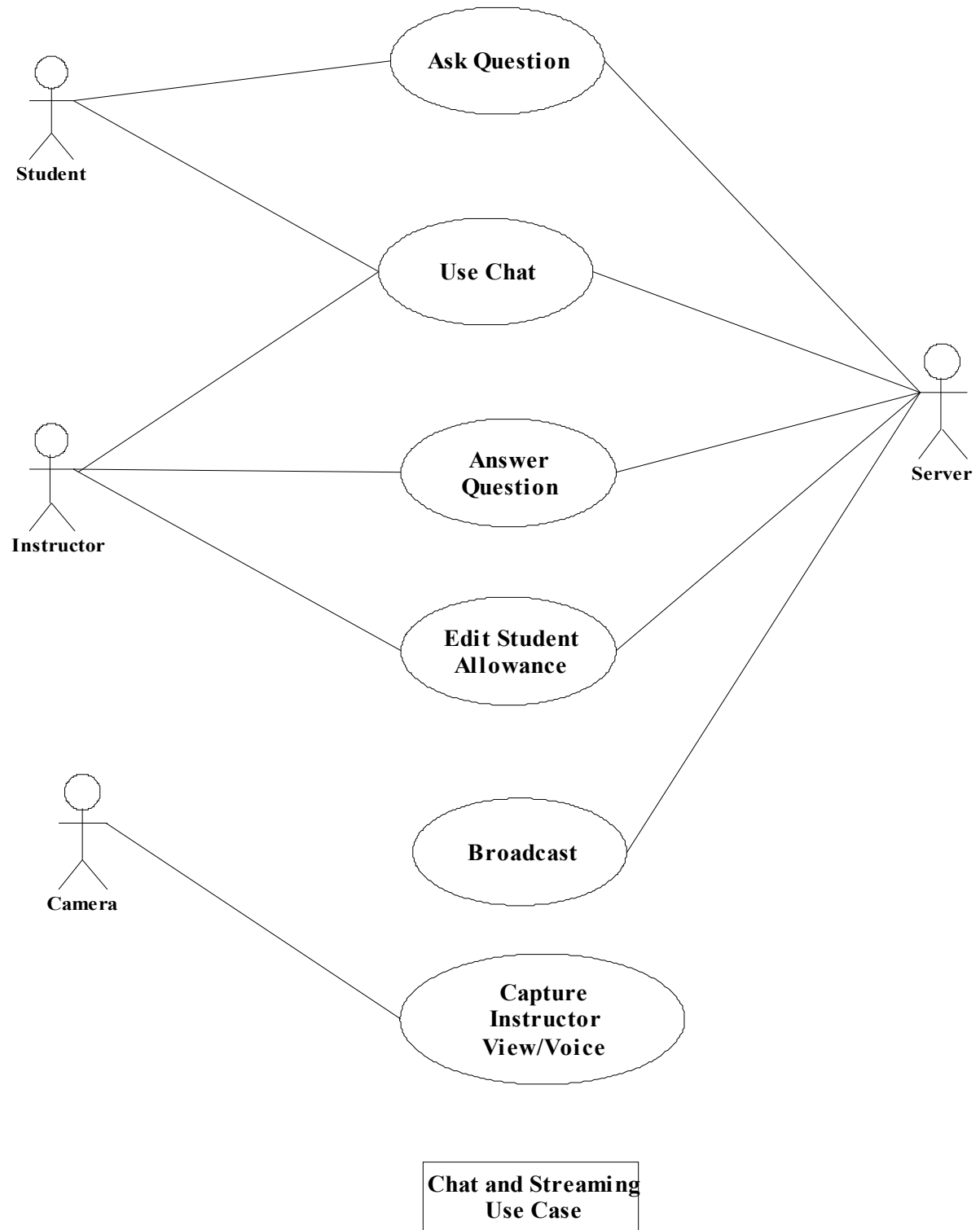


C.2. USE CASE DIAGRAMS

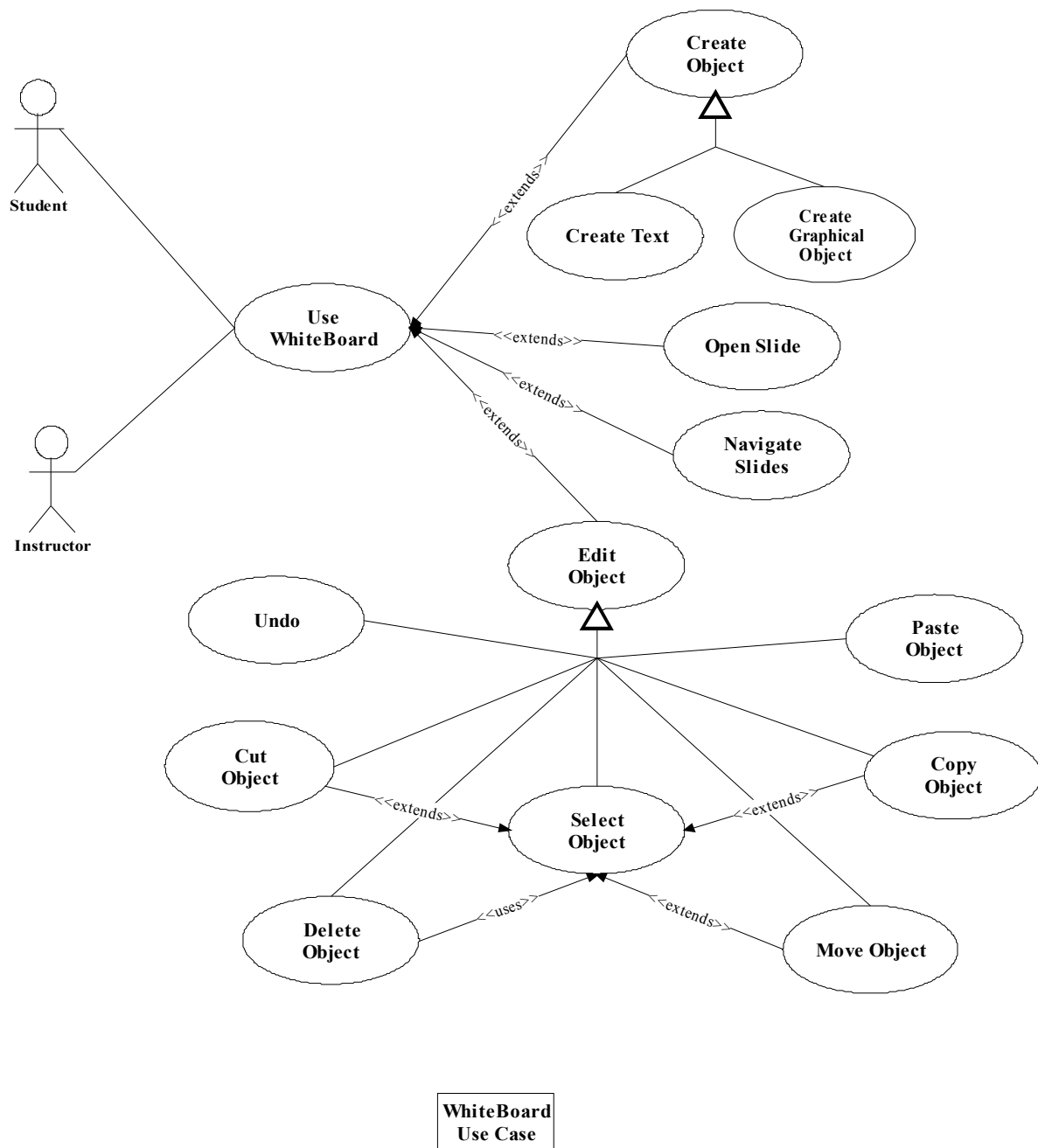
C.2.1 BEFORE VIRTUAL CLASSROOM



C.2.2. COMMUNICATION & STREAMING

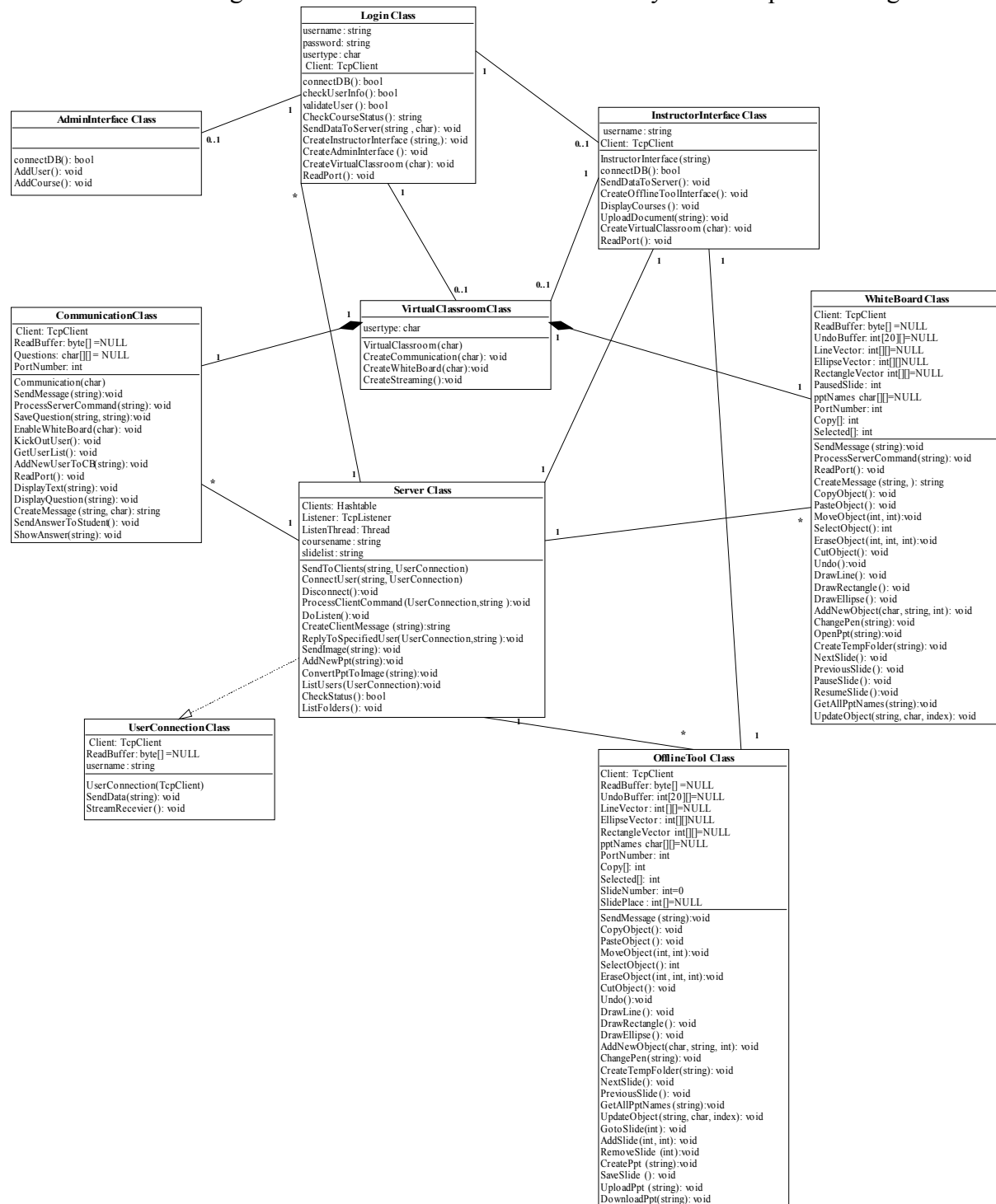


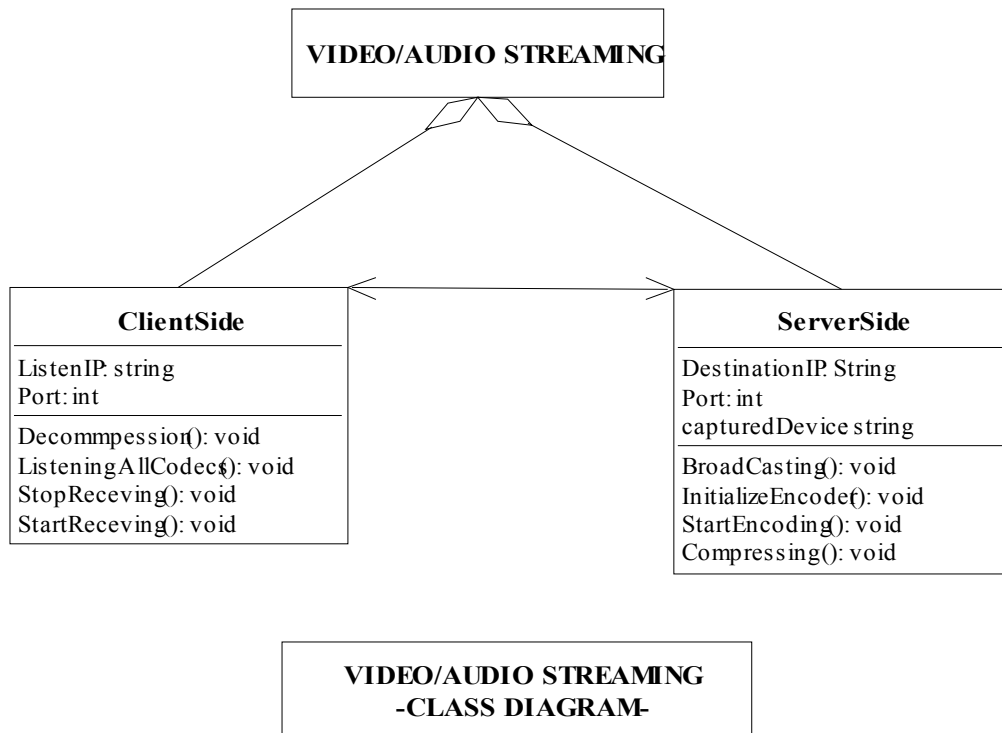
C.2.3. WHITEBOARD



C.3. CLASS DIAGRAMS

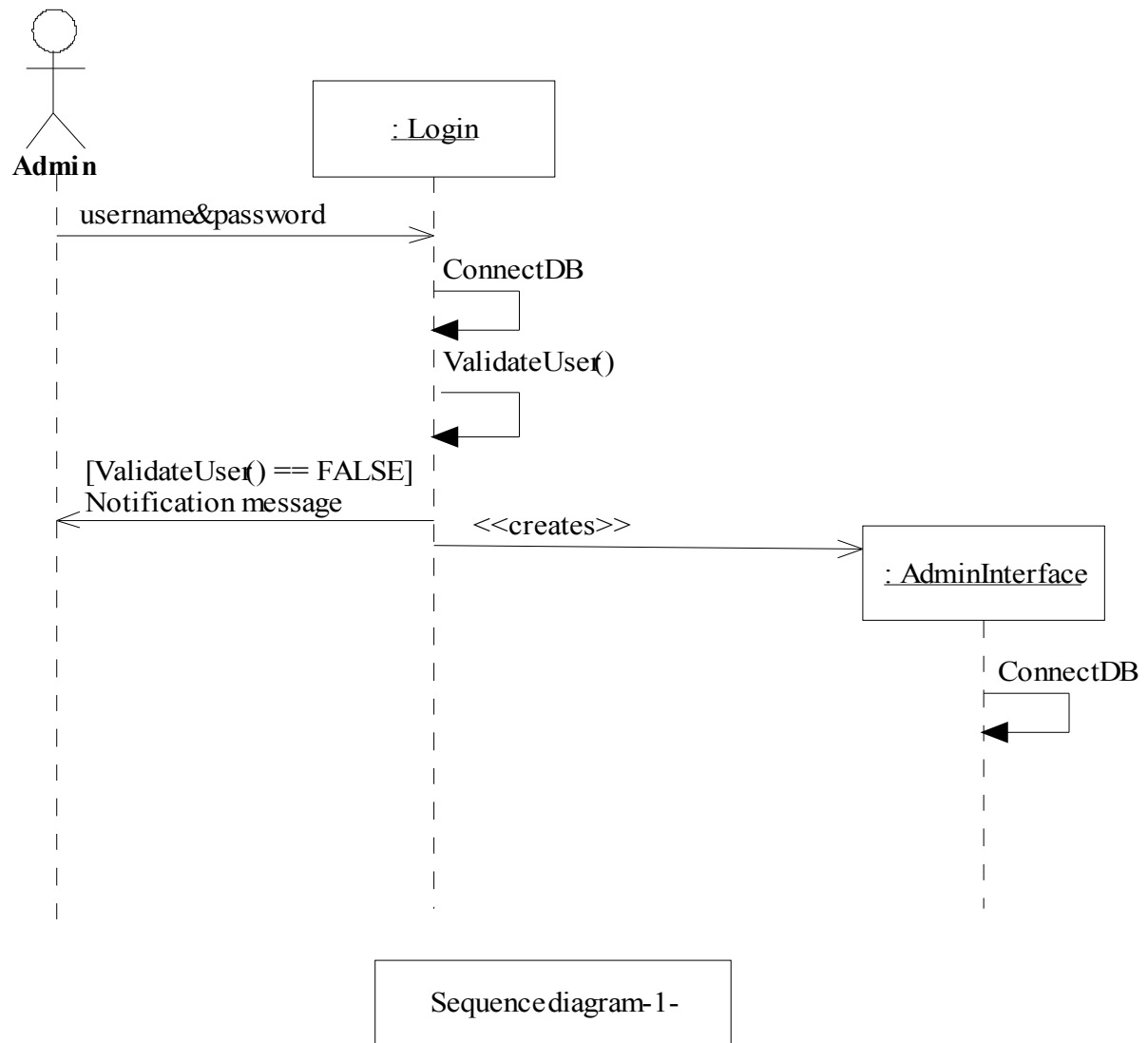
This class diagram shows the whole structure of the system except streaming.



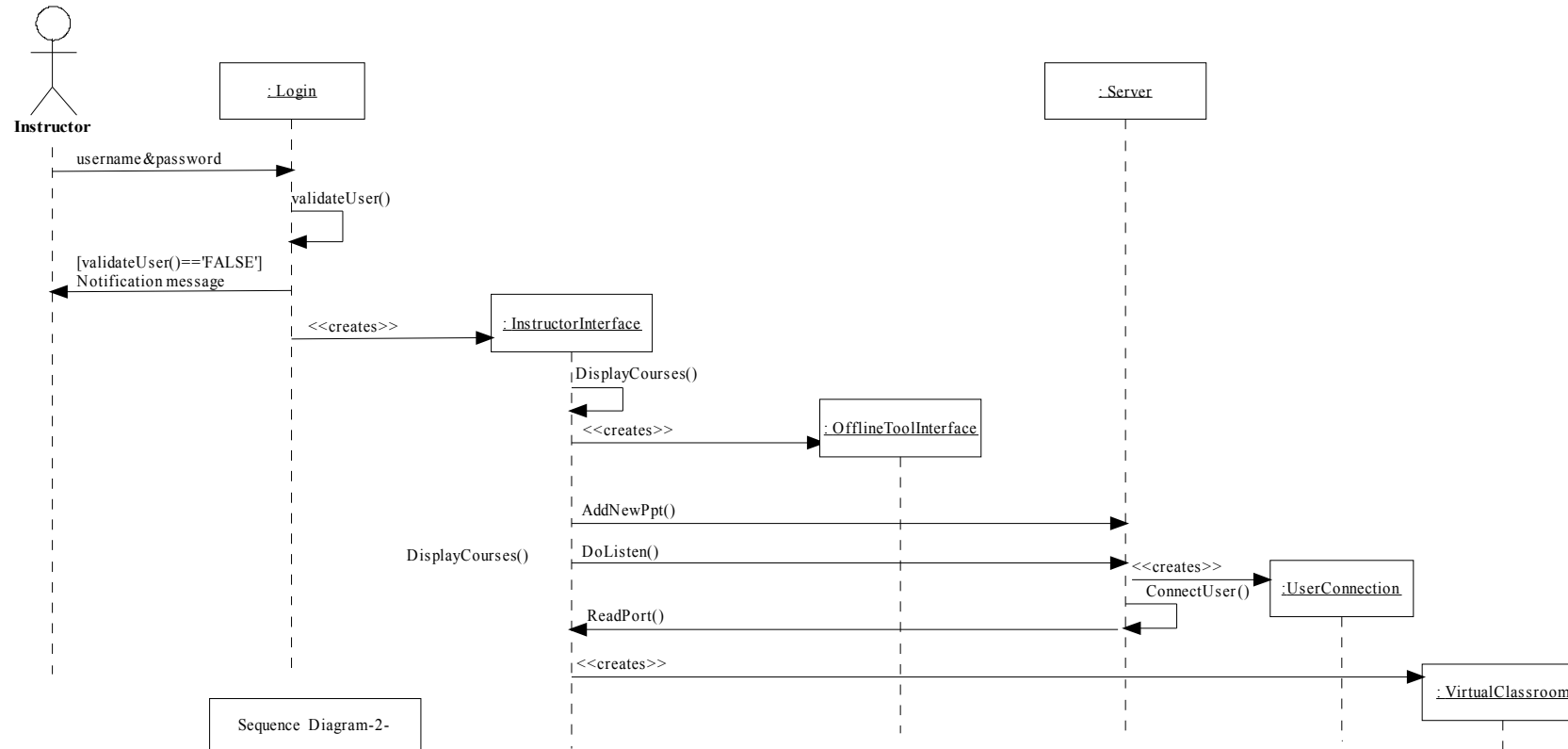


C.4 SEQUENCE DIAGRAMS

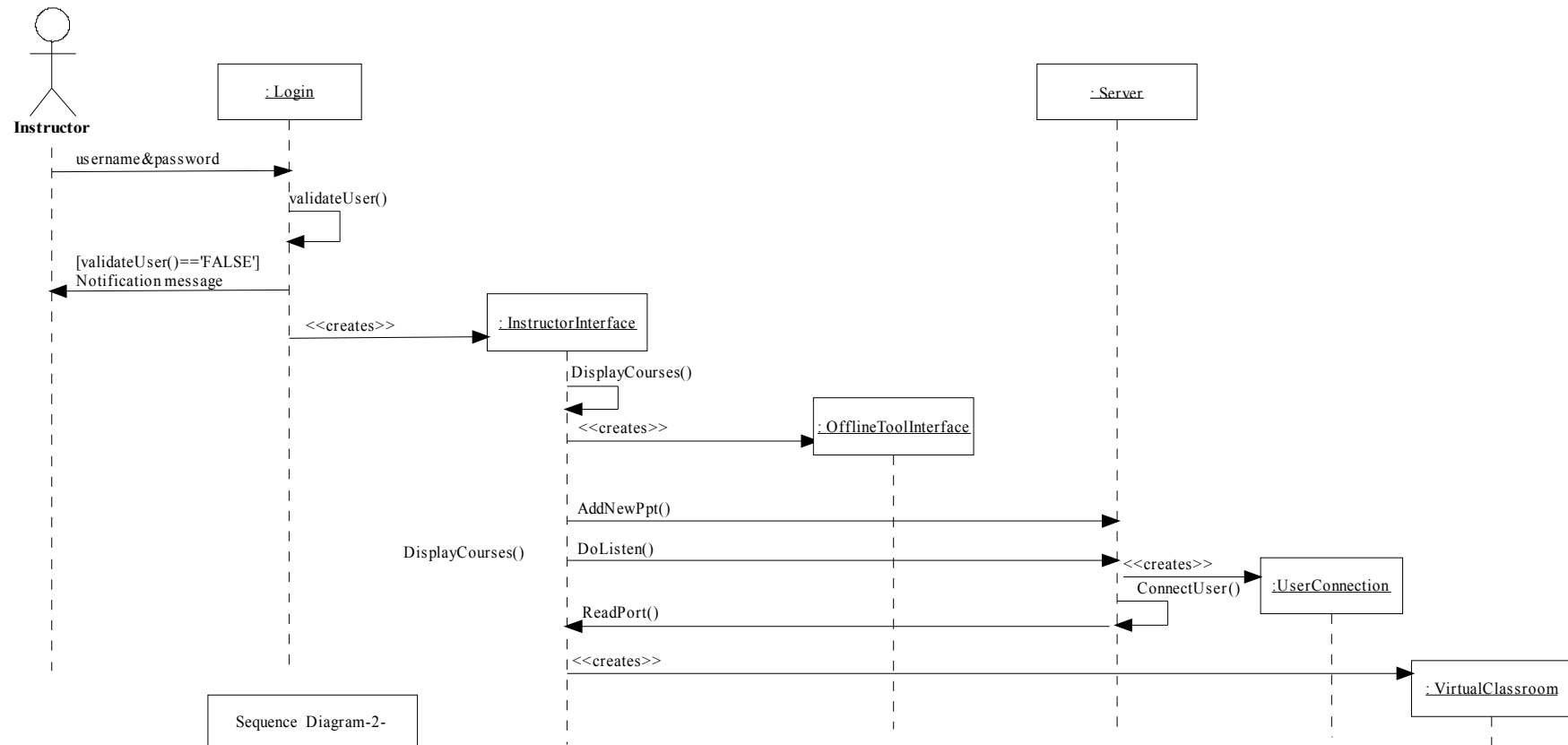
C.4.1 LOGIN MODULE



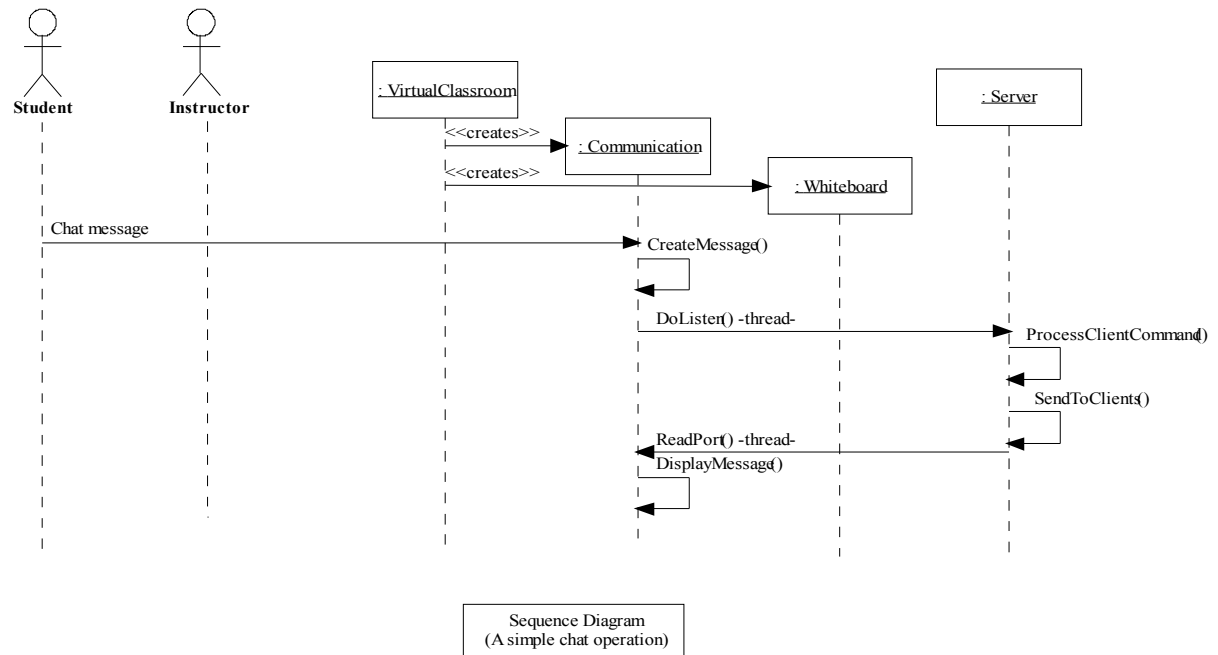
C.4.2 INSTRUCTOR ACTIONS BEFORE VIRTUAL CLASSROOM



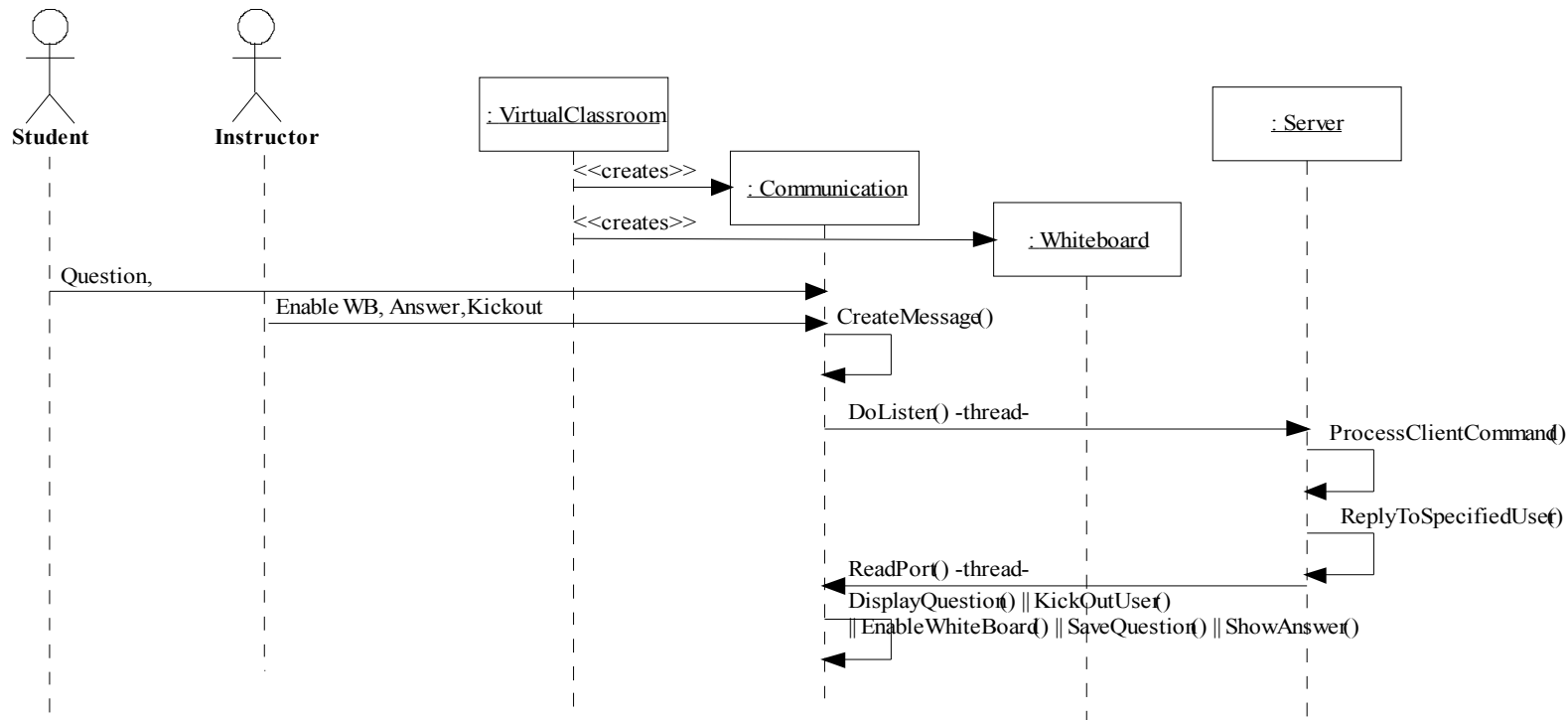
C.4.3. STUDENT ACTIONS BEFORE VIRTUAL CLASSROOM



C.4.4. CHAT

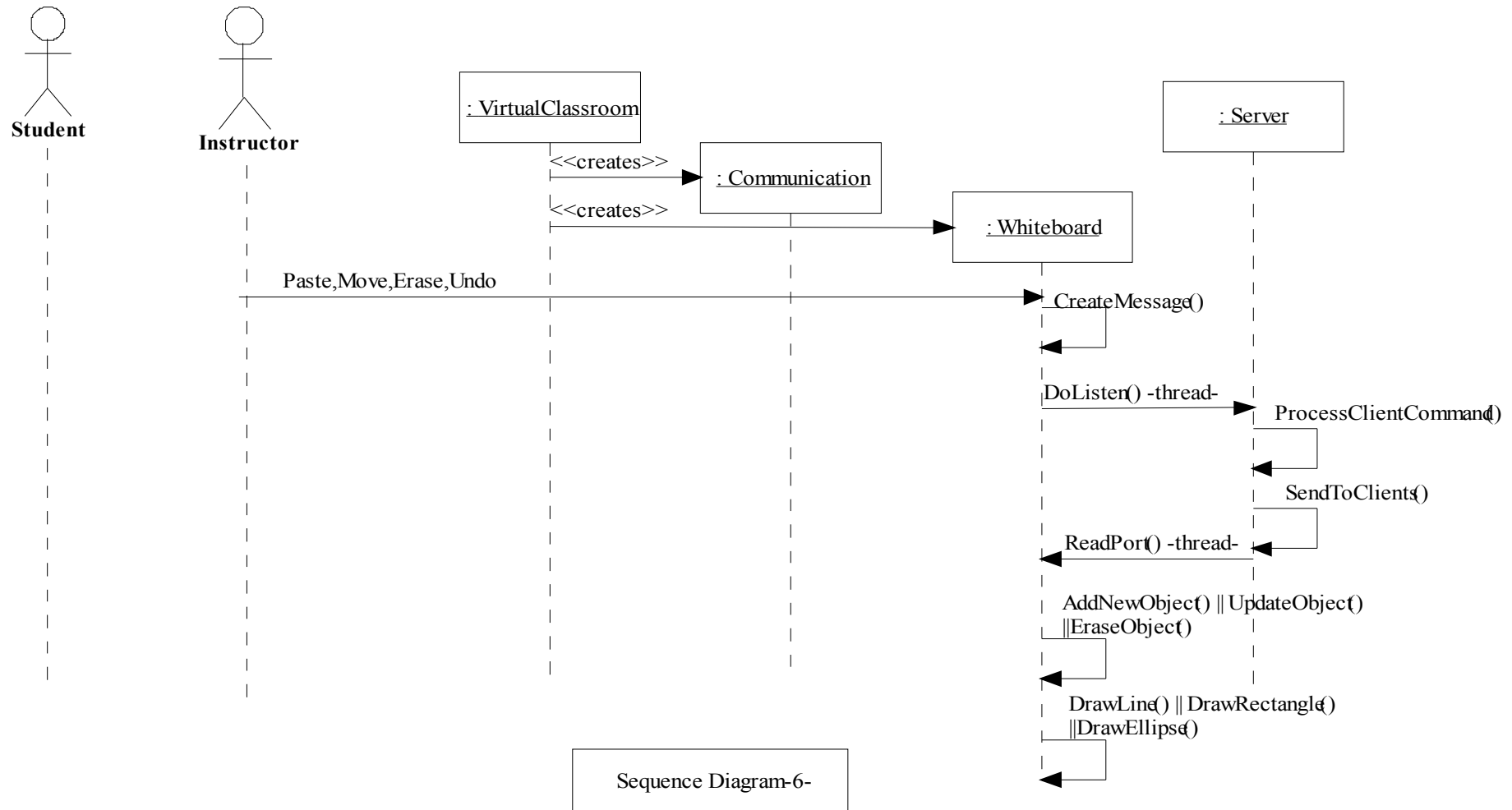


C.4.5. INSTRUCTOR-STUDENT INTERACTION

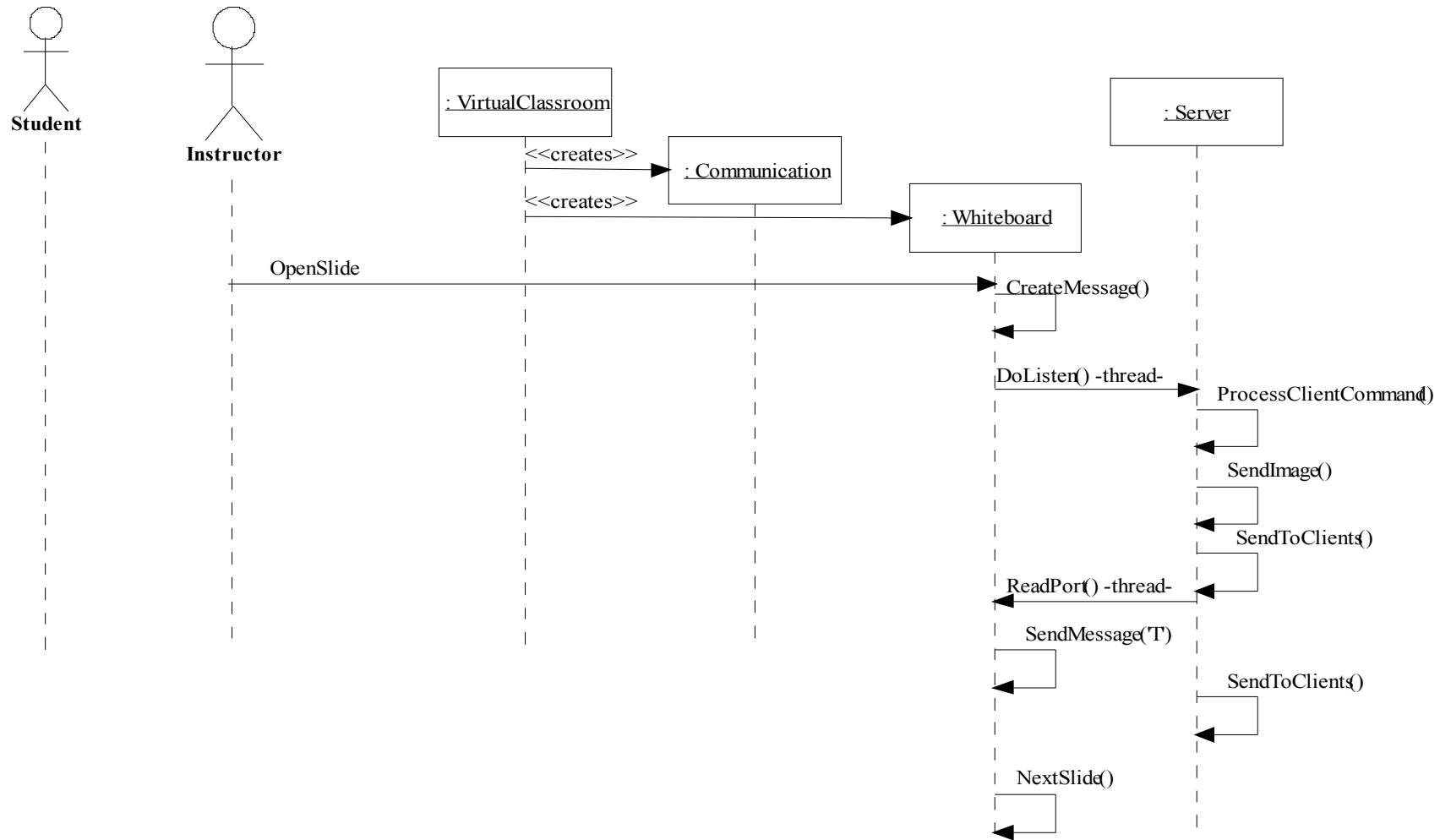


Sequence Diagram-5-

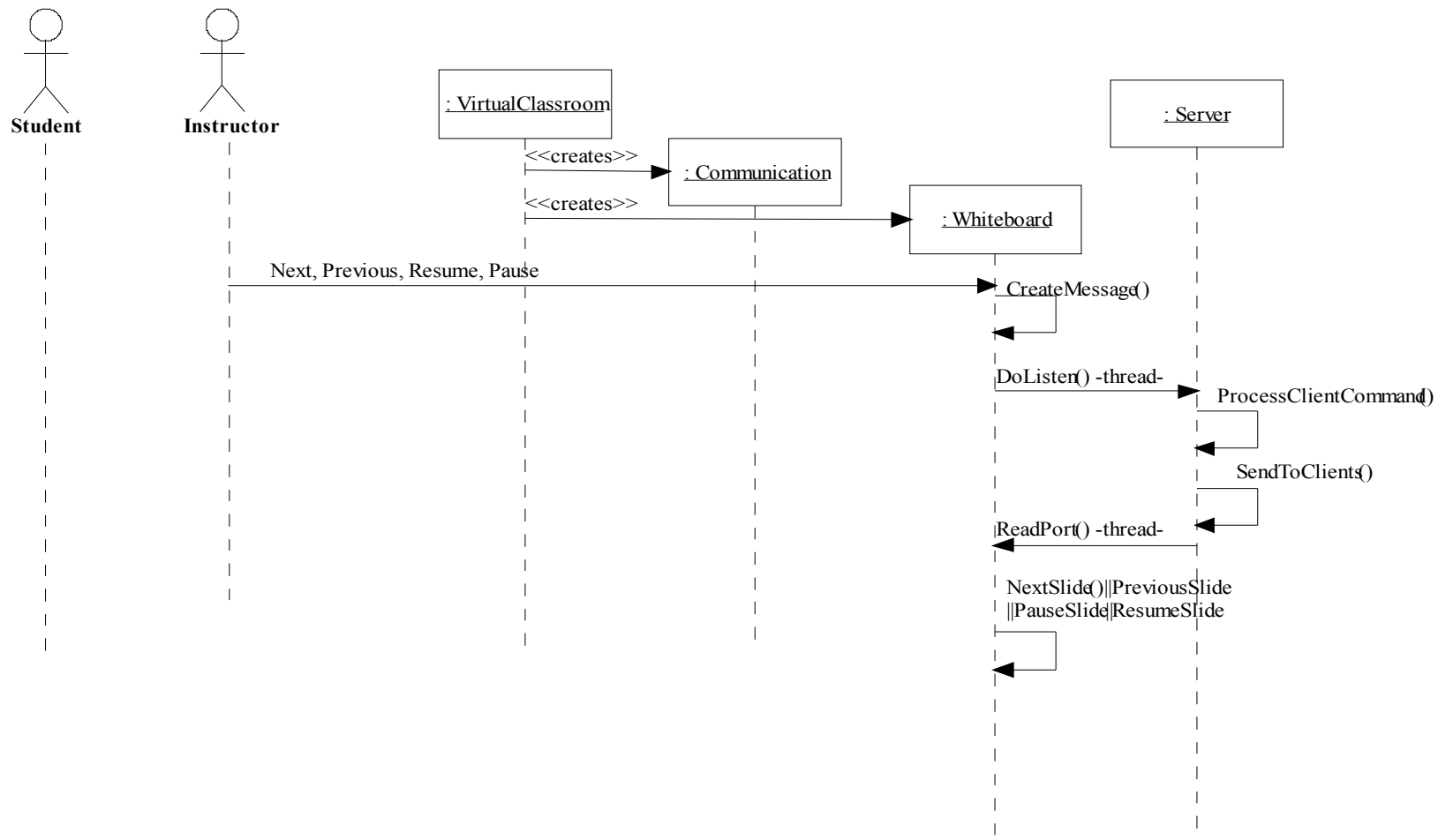
C.4.6. WHITEBOARD OPERATIONS



C.4.7. SLIDES ON WHITEBOARD

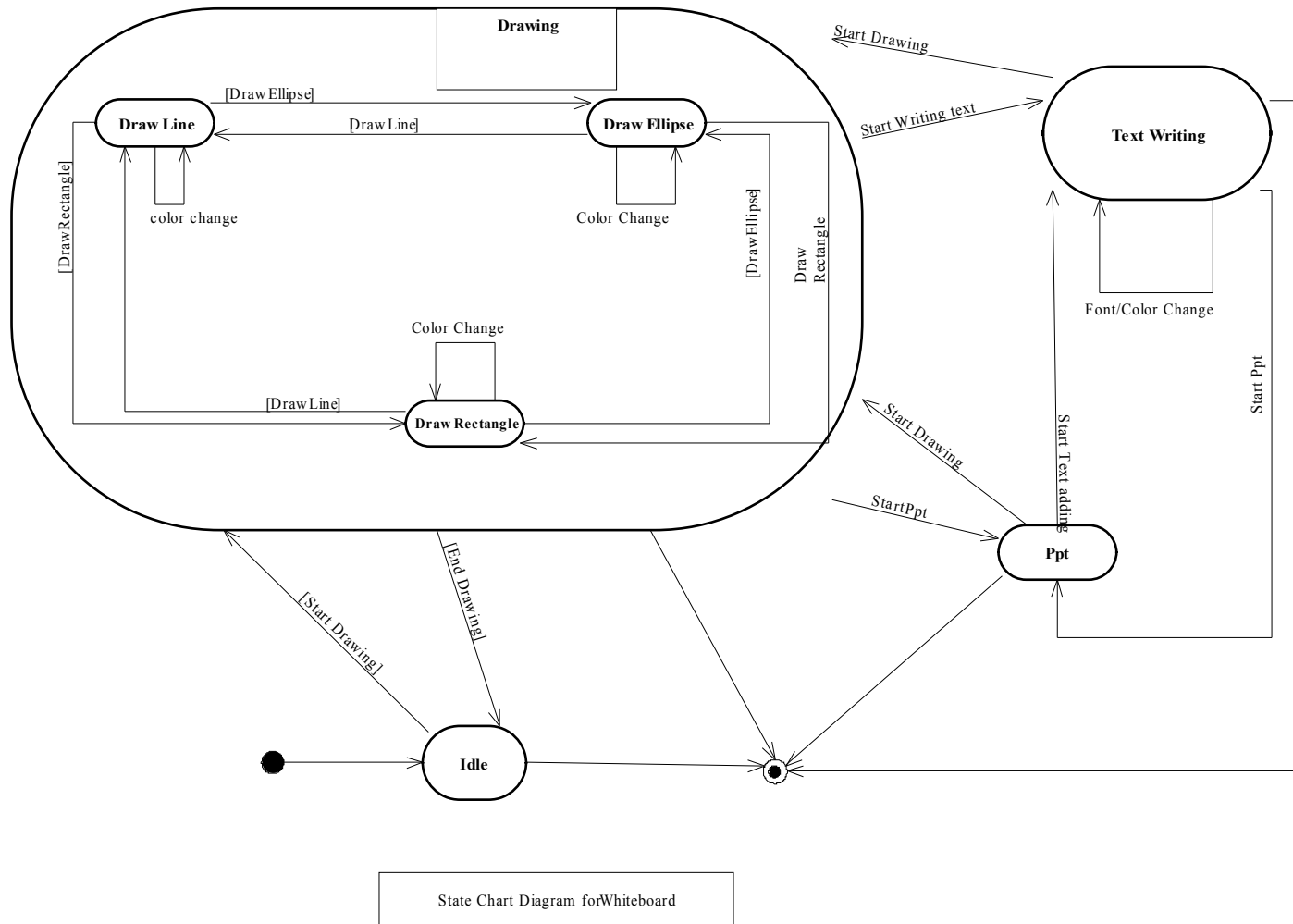


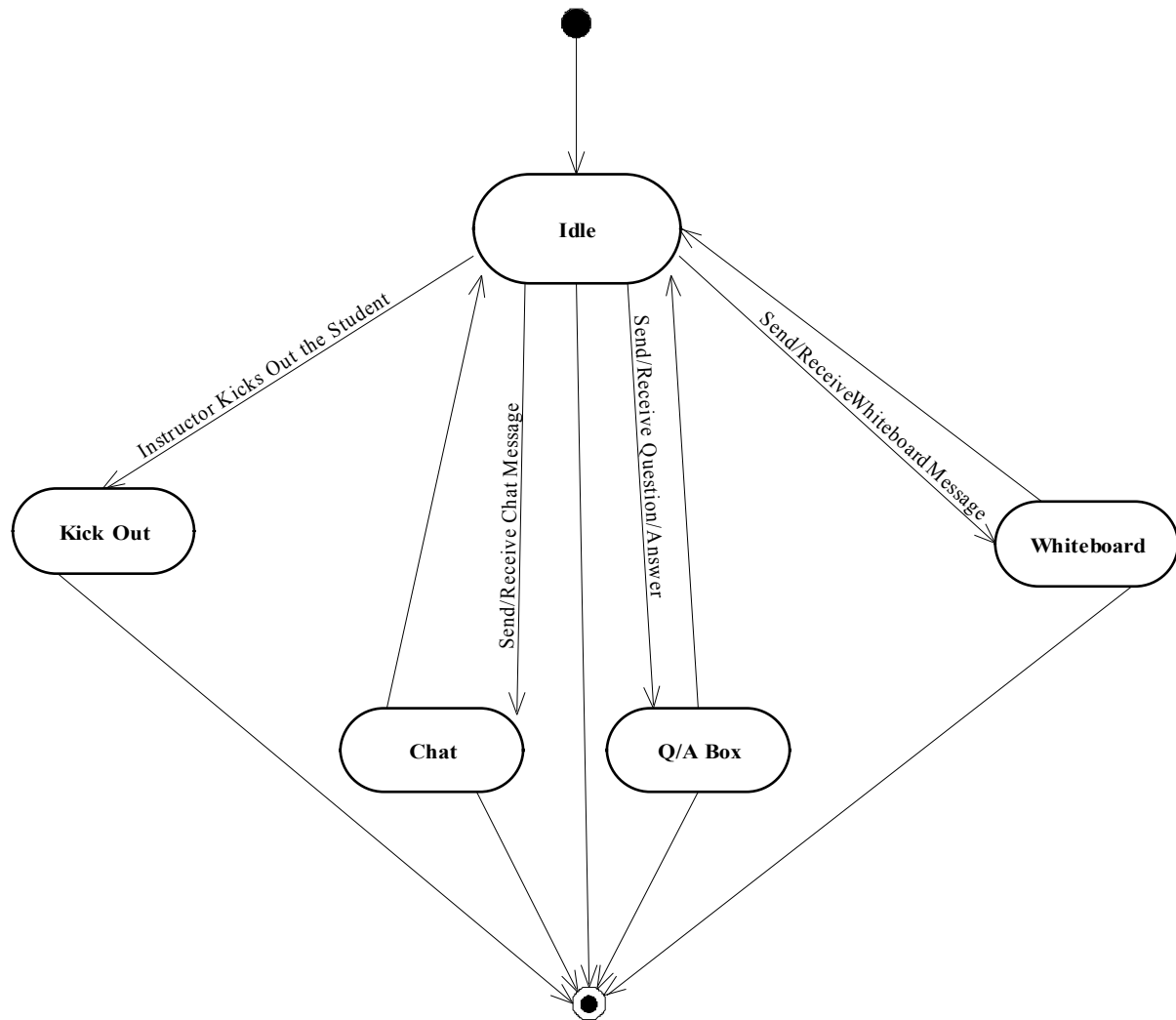
Sequence Diagram-7-
(A simple simulation of operppt)



Sequence Diagram-8-

C.5. STATE CHART DIAGRAM





State Chart Diagram for Communication Tool

D. CLASS DESCRIPTIONS

1. Login Class:

This class will handle all the interaction with users. This class gets the username, password and user type from the user and connects to database for checking the validity of the username and password information. When the user is student then this class checks whether any course is available. If any course is available, it sends the username and user type to server for registering. Then this class creates an instance of the VirtualClassroom class. If there is not

available course then this class returns a notification message to user. In addition to that when the connection is not established, this class returns error message to user. When the user is admin or instructor then this class creates the instances of the AdminInterface or InstructorInterface respectively.

1.1 Members:

- **username (string):** Hold the username information of the user.
- **password (string):** Hold the password information of the user.
- **usertype (char):** Hold the user type.
- **Client (TcpClient):** Used for handling the connection with the server.

1.2 Methods:

- **connectDB ():** This method provides the connection to DB. If connection is refused then it returns false otherwise it returns true.
- **CheckUserInfo ():** This method checks whether the received input is correct, i.e. checks whether the input contains only 'A-Z' or 'a-z' or '0-9' characters. If the input contains special characters then it returns the notification message to the user and returns false.
- **validateUser ():** This method checks whether the username and password is valid. We check all tuples of user table into the database whether the inputs are valid.
- **CheckCourseStatus ():** This method checks whether there is any course is available. In our system, we have only one course is available at a time. Initially all course status in our database is unavailable. When the instructor enters the virtual classroom, this course status is changed. This method gets the name of the available course from course table into the database and returns it.
- **SendDataToServer (string, char):** This method is used when the user is student. By using this method we send a message to server. This message says that a new student login our virtual classroom. This message syntax is given below:
'Login|username|usertype'
- **CreateInstructorInterface (string):** This method is activated when the user is instructor. It creates an instance of the InstructorInterface class by using its constructor. It has one argument, which is passed to the constructor of the InstructorInterface. This argument holds the username of the instructor.

- **CreateAdminInterface ():** This method is activated when the user is admin. It creates an instance of the AdminInterface class by using its constructor.
- **CreateVirtualClassroom (char):** This method is called after the connection is accepted. This method creates an instance of the VirtualClassroom class by using its constructor. It has one argument, which is passed to the constructor of the VirtualClassroom. This argument holds the usertype.
- **ReadPort ():** This method reads the message, which is coming from the server. The server sends a message to the client whether the connection is accepted. This methods reads this messages then give information about connection, if it is accepted then call the CreateVirtualClassroom function, if it is refused then return to user a notification message.

2. AdminInterface Class:

This class provides adding new user and course to database. Only admin can reach and use this class' functions. Admin can add user and course to system.

2.1 Members:

This class has no member.

2.2 Methods

- **connectDB ():** This method provides the connection to DB. If connection is refused then it returns false otherwise it returns true.
- **AddUser ():** This method gets the username and password of the new user then adds this information to user table into database.
- **AddCourse ():** This method gets the course name then adds this information to course table into database.

3. InstructorInterface Class:

This class will handle all of the instructor operations. By this class instructor can see all of the courses, and select one of them then he/she prepare new document by using our offline tool or can upload new power point slides to this course. In addition to that he/she can start the selected course.

2.1. Members

- **username (string):** Hold the username information of the user.
- **Client (TcpClient):** Used for handling the connection with the server.

2.2 Methods

- **InstructorInterface (string):** The constructor of the InstructorInterface class with one argument. This argument specifies the username of the instructor.
- **connectDB ():** This method provides the connection to DB. If connection is refused then it returns false otherwise it returns true.
- **SendDataToServer ():** When the instructor clicks the START button in this interface, this method is called. This method sends the message about the instructor, and course. This message says that a new instructor login for teaching specified course to our virtual classroom. This message syntax is given below:
 ‘Login|username|usertype|coursename’
- **CreateOfflineToolInterface ():** If any instructor wants to create or edit the specified course slide then this method is activated. This method creates an instance of the OfflineToolInterface class by using its constructor.
- **DisplayCourses ():** This method is activated just after the database connection was provided. It gets all of the course names from the course table into the database then shows the instructor. Then instructor can select one of them for making some operations such as creating slides or upload documents.
- **UploadDocument (string):** This function sends the document of specified course to server. This document is uploaded into the specific course folder.
- **CreateVirtualClassroom (char):** This method is called after the connection is accepted. This method creates an instance of the VirtualClassroom class by using its constructor. It has one argument, which is passed to the constructor of the VirtualClassroom. This argument holds the usertype.
- **ReadPort ():** This method reads the message, which is coming from the server. The server sends a message to the client whether the connection is accepted. This methods reads this messages then give information about connection, if it is accepted then call the CreateVirtualClassroom function, if it is refused then return to instructor a notification message.

4. VirtualClassroom Class

This class is the entry point of our virtual classroom. In this class we create the instances of all modules. We can say that this class provides the combination of the all modules (white board, chat, video/audio) of our system.

4.1 Members:

- **username (string):** Hold the username information of the user.

4.2 Modules:

- **VirtualClassroom (char):** The constructor of the VirtualClassroom class with one argument. This argument specifies the usertype.
- **CreateCommunication (char):** This method is called in the constructor of the VirtualClassroom class. This method creates an instance of the Communication class by using its constructor. It has one argument, which is passed to the constructor of the Communication class. This argument holds the usertype. This usertype specifies which tools are disabled in communication tool.
- **CreateWhiteBoard (char):** This method is called in the constructor of the VirtualClassroom class. This method creates an instance of the Communication class by using its constructor. It has one argument, which is passed to the constructor of the Communication class. This argument holds the usertype. This usertype specifies which tools are disabled in Whiteboard tool.
- **CreateStreaming ():** This method is called in the constructor of the VirtualClassroom class. This method creates an instance of the Communication class by using its constructor.

5. Communication Class:

This class contains chat module, question answer module and also user control box module. In the initialization of this module some parts will be disabled according to the usertype. If user is student, then the control box is disabled. If the user is instructor then the question answer box is disabled.

5.1 Members:

- **Client (TcpClient):** Used for handling the connection with the server.
- **ReadBuffer (byte []):** Hold the message coming from the server.
- **Questions (char [][]):** Hold all of the questions asked by students.
- **PortNumber (int):** Hold the port number of the communication module.

5.2 Methods:

- **Communication (char):** The constructor of the Communication class with one argument. This argument specifies the usertype.
- **SendMessage (string):** This method sends the message to server. This message says to server to make some operations.
- **ProcessServerCommand (string):** This method will split the incoming message and will determine the type of the message. According to this message this class makes some operations.

The following are the incoming messages:

1. The server sends the chat message to all users the following format.

‘C|message’

When the clients get this type of message then DisplayText (message) function is called with the received message

2. The server sends the questions to the instructor the following format:

‘Q|question|username’

When the client receives this type of message SaveQuestion (question, username) function is called with the received questions

3. The server sends the whiteboard permission to the following format: (to specified user)

‘E|’

When the client receives this type of message EnableWhiteBoard(‘E’) function is called. (note: E means enable)

4. The server sends message to the following format: (to specified user)

‘D|’

When the client receives this type of message EnableWhiteBoard(‘D’) function is called. (note: D means disable)

5. The server sends message to the following format: (to specified user)

‘K|’

When the client receives this type of message KickOutUser() function is called. Then this user is kicked out the system.

6. The server sends the list of all online users to the following format: (to specified user)

‘L|Users (separated with |)’

When the client receives this type of message AddNewUserToCB function is called for each user.

7. The server sends the answer to the specified student the following format:

‘A|answer’

When the client receives this type of message ShowAnswer (answer) function is called with the received questions

- **SaveQuestion (string, string):** This method saves the received question and username into the Questions array. And also changed the color value of the of question button (into the control box module) of this specified user.
- **EnableWhiteBoard (char):** This method enables or disables the whiteboard according to the argument value. If the argument is 'E' then it enables the whiteboard, or if the argument is 'D' then is disables the whiteboard.
- **KickOutUser ():** This method calls the exit function, which close the Virtual Classroom.
- **GetUserList ():** This method is called in the constructor of this class. This function calls the CreateMessage ("", 'L') that creates the message that is send by SendMessage (string) function.
- **AddNewUserToCB (string):** This method adds the new user name and control button to the control box module to the specified position into the form.
- **ReadPort ():** This method reads the message, which is coming from the server. Then call ProcessServerCommand function with this message.
- **DisplayText (string):** This method will show the incoming or sent message in the clients chat console.
- **DisplayQuestion (string):** When the instructor clicks the question button of the specified user all of the questions of this specified user is displayed.
- **SendAnswerToStudent ():** This method is called when the instructor presses the answer button. In this function we call CreateMessage('answer', 'A') function.
- **ShowAnswer(string):** This methods show all of the question of the student and also show the received answer.
- **CreateMessage (string, char):** This method creates message in defined format. This message can be understand by server than server makes some operations according to this message. After the creation of the message this function call the SendMessage (message) with message argument, which is created by this method.

The following are the created messages:

1. When the user writes some message to chat module and then presses the send button then we call the CreateMessage (message, C) with message and C as arguments.

Then created message is:

'C|message'

2. When the student writes the question into the question/answer box and then presses the send button then we call the CreateMessage (message, Q) with message and C as arguments.

Then created message is:

‘Q|message’

3. When the instructor wants to give the whiteboard permission to the specified user, we call CreateMessage(“username”, E). (Note: E means that enable the whiteboard)

Then created message is:

‘E|username’

4. When the instructor wants to take back the whiteboard permission to the specified user, we call CreateMessage (“username”, D). (Note: D means that disable the whiteboard)

Then created message is:

‘D|username’

5. When the instructor wants to kick out specified user, we call CreateMessage(“username”, K). (Note: K means that kick out the user)

Then created message is:

‘K|username’

6. When the user enters the system, the control box needs the list of the users. We call CreateMessage(“”, L).

Then created message is:

‘L|’

7. When the instructor writes the answer into the question/answer box and then presses the send button then we call the CreateMessage (answer, A) with message and C as arguments.

Then created message is:

‘A|username|answer’

6. WhiteBoard Class:

This class controls all of the operations into the whiteboard. Basically this class has two major operations. One of them is about drawing operations and the other is about the controlling the slide show (such as open, pause, resume, next, previous). Our white board module has drawing capabilities, editing capabilities, and also displaying slides

capabilities. And whiteboard sends all of the command in different port. Besides that our whiteboard is used by one person (instructor or specified student)

6.1. Members

- **Client (TcpClient):** Used for handling the connection with the server.
- **ReadBuffer (byte []):** Hold the message coming from the server.
- **UndoBuffer (int [20][]):** Hold the last 20 operations.
- **LineVector (int [][]):** Hold all of the lines information. Each row represents a line.
- **EllipseVector (int [][]):** Hold all of the ellipse information. Each row represents an ellipse. Each ellipse is represented by two points. These two points create a rectangle (invisible), which indicates the boundary of each ellipse.
- **RectangleVector (int [][]):** Hold all of the rectangle information. Each row represents a rectangle. Each rectangle is represented by two points.
- **PausedSlide (int):** Hold the last slide number.
- **pptNames (char [][]):** Hold all the power point slide names.
- **PortNumber (int):** Hold the port number of the whiteboard module.
- **Copy (int[]):** Hold the information of the copied object.
- **Selected (int[]):** Hold the selected object info.

6.2. Methods

- **SendMessage (string):** This method sends the message to server. This message says to server to make some operations.
- **ProcessServerCommand (string):**

The following are the incoming messages:

1. The server sends message for adding new item to all users the following format.

**‘AL|Points (separated by special char)’ or
‘AE|Points (separated by special char)’ or
‘AR|Points (separated by special char)’**

When the clients get this type of message then AddNewObject (L, Points, 0) function is called.

2. The server sends message for erasing specified item to all users the following format.

‘EL|index’ or ‘EE|index’ or ‘ER|index’

When the clients get this type of message then EraseObject (1 or 2 or 3, index, 0) function is called with the received message.

3. The server sends message for moving specified item to all users the following format.

‘ML|points’ or ‘ME|points’ or ‘MR|points’

When the clients get this type of message then UpdateObject (points, L or E or M, index (which is the first point in the points value)) function is called with the received message.

4. The server sends message for changing the pen properties to all users the following format.

‘CP|properties’

When the clients get this type of message then ChangePen (properties) function is called with the received message.

5.

‘F’-> indicates that all of the slides are sent.

When the clients get this type of message then SendMessage (‘T’) function is called with the received message.

6.

‘T’-> indicates that all of the users are ready to show power point slide and also you can start to show.

When the clients get this type of message then NextSlide () function is called.

7. The server sends message for passing the next slide to all users the following format.

‘N|’

When the clients get this type of message then NextSlide () function is called.

8. The server sends message for passing the previous slide to all users the following format.

‘P|’

When the clients get this type of message then PreviousSlide () function is called.

9. The server sends message for pausing the slide show to all users the following format.

‘X|’

When the clients get this type of message then PauseSlide () function is called.

10. The server sends message for resuming the slide show to all users the following format.

‘R|’

When the clients get this type of message then ResumeSlide () function is called.

11. The server sends message for undoing the last add operation to all users the following format.

‘UAL|Points’ or ‘UAE|Points’ or ‘UAR|Points’

When the clients get this type of message then AddNewObject (L, Points, 1) function is called.

12. The server sends message for undoing the last erase operation to all users the following format.

When the clients get this type of message then EraseObject (1 or 2 or 3, index, 1) function is called with the received message.

‘UEL|Points’ or ‘UEE|Points’ or ‘UER|Points’

- **ReadPort ():** This method reads the message, which is coming from the server. Then call ProcessServerCommand function with this message.
- **CreateMessage (string, char):** This method creates message in defined format. This message can be understood by server then server makes some operations according to this message. After the creation of the message this function calls the SendMessage (message) with message argument, which is created by this method.

The following are the created messages:

1. When the instructor pastes previously copied object we call the CreateMessage (Points, AX) with points (represents new points) and X represents the type of the new object (for example: for line X->L, for ellipse X->E, for rectangle X->R). This means add new X type of object.

Then created message is:

‘AL|Points (separated by special char)’ or

‘AE|Points (separated by special char)’ or

‘AR|Points (separated by special char)’

2. When the instructor erases selected object we call the CreateMessage (index, EX) with index (for example linevector [index]), E means erase and

X represents the type of the erased object (for example: for line X->L, for ellipse X->E, for rectangle X->R).

Then created message is:

‘EL|index’ or ‘EE|index’ or ‘ER|index’

3. When the instructor moves selected object we call the CreateMessage (points, MX) with points (new points of the selected object) E means erase and X represents the type of the erased object (for example: for line X->L, for ellipse X->E, for rectangle X->R). The points are separated with the special characters and the first number represent the index of the object (for example linevector [index])

Then created message is:

‘ML|Points’ or ‘ME|Points’ or ‘MR|Points’

4. When the instructor draws new object we call the CreateMessage (points, AX) with points, A means add and X represents the type of the added object (for example: for line X->L, for ellipse X->E, for rectangle X->R). The points are separated with the special characters .

Then created message is:

‘AL|Points’ or ‘AE|Points’ or ‘AR|Points’

5. When the instructor changes the pen properties we call the CreateMessage (properties, CP) with properties (represents the new properties), CP means change the properties of the pen. The properties are separated with the special characters.

Then created message is:

‘CP| properties’

6. When the instructor want to open selected power point slide we call the CreateMessage (slidename, O) with slidename, O means open new slide

Then created message is:

‘O|slidename’

7. When the instructor want to show next slide we call the CreateMessage (‘, N) with N means next slide

Then created message is:

‘N|’

8. When the instructor want to show previous slide we call the CreateMessage (‘, P) with P means previous slide

Then created message is:

‘P|’

9. When the instructor want to pause the slide show we call the
CreateMessage (‘’, X) with X means that pause the slide show

Then created message is:

‘X|’

10. When the instructor want to resume the slide show we call the
CreateMessage (‘’, R) with R means that resume the slide show

Then created message is:

‘R|’

11. CreateMessage (‘’, S) with S means that get all the slides.

Then created message is:

‘S|’

12. When the instructor undo the last operation, CreateMessage (points, UAX)
with points, U means undo, A means add and X represents the type of the
added object (for example: for line X->L, for ellipse X->E, for rectangle X->R)
is called. The points are separated with the special characters.

Then created message is:

‘UAL|Points’ or ‘UAE|Points’ or ‘UAR|Points’

13. When the instructor undo the last operation, CreateMessage (points, UAX)
with points, U means undo, A means add and X represents the type of the
added object (for example: for line X->L, for ellipse X->E, for rectangle X->R)
is called. The points are separated with the special characters.

Then created message is:

‘UEL|Points’ or ‘UEE|Points’ or ‘UER|Points’

- **CopyObject ():** This method copies the selected object into the copy array. If the selected object is line, the first column of the copy array will be 1, if it is ellipse, it will be 2, if it is rectangle, it will be 3. This method saves only the type of object and the index of this object, which indicates the place of this object in the array. For example; Copy[0] = 1 means that this object is line and Copy[1] = 25 means that 25. line in the LineVector array.
- **PasteObject ():** This method pastes the copied object into the clicked position of the mouse. The all points of the copied object moved according to the position of the mouse. After updating the points we call CreateMessage (points, AX)

function. CreateMessage function prepare message for sending the server. Then server sends an information message to clients, and then all of the clients can see the new object into the desired points.

- **MoveObject (int, int):** This method moves the selected object. And also we call CreateMessage(index, MX). The row of the objects' array will be updated by using this function. If there is not any selected object then this method moves the specified object (according to the arguments, one of the argument shows the type of object and the other represents the index of this object). Then the row of the objects' array will be updated by using this function.
- **SelectObject ():** This method is used for selection of objects. Firstly we should indicate which type of object is selected. For ellipse and rectangle we have a rectangle boundary. This method searches the mouse point in these boundaries. If the mouse position is inside the more than one area then the nearest one is chosen. For line, the intersection point between the mouse position and line points is searched. This method saves only the type of object and the index of this object, which indicates the place of this object in the array. For example; Selected[0] = 1 means that this object is line and Selected[1] = 25 means that 25. line in the LineVector array.
- **EraseObject (int, int, int):** This method erases the selected object. And also we call CreateMessage (index, EX). The row of the objects' array will be deleted by using this function. If there is not any selected object then this method delete the specified object (according to the arguments, one of the argument shows the type of object and the other represents the index of this object). Then the row of the objects' array will be deleted by using this function. If the third argument is 1 then this function returns, else saving this operation into the UndoBuffer is done. If undo stack is full we remove the oldest operation. And add the new operation into the top of the stack. The format of the UndoBuffer for erasing new object is like that:

UndoBuffer [top][0] = type of the operation ('E' means creation)
UndoBuffer [top][1] = type of the object ('L' means line, 'E' means Ellipse and 'R' means Rectangle)

And all the points of the erased objects are added to UndoBuffer.
- **CutObject ():** This method is the combination of the EraseObject (int, int) and the CopyObject ().

- **Undo ():** When the user clicks the undo button this method is called. If the type of the undo buffer is erase (`UndoBuffer[top][0] == E`), then `CreateMessage (points, UAX)` is called for adding this object into white board of all clients. And also add these points into the objects' array. If the type of the undo buffer is add (`UndoBuffer[top][0] == A`), then `CreateMessage(index, UEX)` is called for erasing this object into white board of all clients. And also erase this object from the objects' array.
- **DrawLine ():** This method draws all lines according to the `LineVector` array information.
- **DrawRectangle ():** This method draws all lines according to the `RectangleVector` array information.
- **DrawEllipse ():** This method draws all lines according to the `EllipseVector` array information.
- **AddNewObject (char, string, int):** This method has basically two save operation. One is saving into the objects' array. And the other is saving into the `UndoBuffer`. The first save is done according to the type of the object. The points in the second argument are added to the next row of the objects' array, which is specified by the first argument. The points in the second argument are separated with special character. If the third argument is 1 then this function returns, else the second save operation is done into the `UndoBuffer`. If undo stack is full we remove the oldest operation. And add the new operation into the top of the stack. The format of the `UndoBuffer` for adding new object is like that:
 - `UndoBuffer [top][0] = type of the operation ('C' means creation)`
 - `UndoBuffer [top][1] = type of the object ('L' means line, 'E' means Ellipse and 'R' means Rectangle)`
 - `UndoBuffer [top][2] = index (index of the object)`
- **ChangePen (string):** If the user is instructor, then this method calls the `CreateMessage (properties, CP)`. If it is not instructor then change the properties of the pen according to the received properties. The argument of this class is separated by special character.
- **OpenPpt (string):** If the instructor wants to open the slides, which is created before the lesson, this method is called. In this method, `CreateMessage (slidename, O)` and `CreateTempFolder (slidename)` are called. This method sends to server a message for downloading the ppt which is converted to images from

server. All of the images of this ppt are sent to all clients. And this images are saved into the created folder (by CreateTempFolder (slidename)). Also server sends a special character to each user after sending the images. When the clients got this special character then the clients sends a message to server to indicate that downloading is completed. When the server received this special character from all users then it sends a special character to indicate that 'you can start to show power point slide'.

- **CreateTempFolder (string):** Create a new folder for saving the image, which is received by the server. The name of this folder same as the ppt.
- **NextSlide ():** Show the next slide into the white board as a background.
- **PreviousSlide ():** Show the previous slide into the white board as a background.
- **PauseSlide ():** This method saves the number of the current slide show and also clear white board.
- **ResumeSlide ():** This method displays the last slide into the white board as a background.
- **GetAllPptNames (string):** This method calls the CreateMessage ('', S). This method is called in the constructor of this class.
- **UpdateObject (string, char, int):** This method is used for updating the points of the object. It has three arguments which are indicates points, type, and the index object.

7. Server Class:

This class is the master of virtual classroom that handles all client connections, analyses all incoming messages from clients and responds according to those messages. Mainly, server listens two different port one for chat and login operations, and one port for whiteboard operations. When a connection is established from any client an instance of UserConnection class will be created and other operations will be handled.

7.1 Members

- **Clients(Hash Table):** This data structure will hold all clients as UserConnection objects whenever a new connection is established
- **Listener(TcpListener):** Used for listening the specific port in the ListenThread the incoming connections from clients.
- **ListenThread(Thread):** Thread used for listening the ports

- **Coursename(string):** Used for holding the current online course name
- **Slidelist(string):** Used for holding the names of the subfolders under current course's folder.

7.2 Methods

- **SendToClients(string, UserConnection):** This subroutine sends a message to all attached clients except the sender. The first argument is the message to be sent and the second argument is used for sending to all clients reside in Clients method. While sending the message the **SendData** method of UserConnection objects is used.
- **ConnectUser(string, UserConnection):** This method checks to see if username already exists in the clients Hash table, if it does sends a "Refuse" message otherwise confirms with "Join" message send to the client who is trying to log into the classroom.
- **Disconnect(UserConnection):** This subroutine notifies other clients that sender left the chat, and removes the name from the clients Hashtable.
- **ProcessClientCommand(UserConnection,string):** This is the event handler for the UserConnection object, when it receives a full line message. This method parses the cammand and parameters and take appropriate action.
 1. If the incoming first part of the message is "Login" the server will call ConnectUser(message,sender) where message holds the information whether "Refuse" or "Join" and sender is the object of the UserConnection class who tries to log into the class.
 2. If the incoming first part of the message is one of the messages from "C" – "L" – "E" – "R" – "EL" – "EE" – "ER" – "ML" – "ME" – "MR" – "AL" – "AE" – "AR" – "CP" – "N" – "P" – "X" – "R" – "UAL" – "UAE" – "UAR" – "UEL" – "UER" – "UEE" – "T" then the message does not require to be manipulated and must be transmitted to all clients except the sender, therefore SendToClients method will be called with the received message.
 3. When the first part of the message is "Q" or "A" then messages must be sent to the specific users. If the first part of the message is "Q" than server will transmit the message only to the instructor; therefore ReplyToSpecifiedUser(message, instructor) where message is the received message and instructor is the object of the UserConnection class

that represents the instructor. If the first part of the message is the “A” then message will be transmitted to the specified user that is indicated in the message. Then in this case ReplyToSpecifiedUser(message, student) will be called

4. When the first part of the message is “D”-“E” or “K” then this message is a specific message then ReplyToSenderMessage will be called where the specific user will be parsed from the remaining of the message. For example message is in the format of this type as mentioned above D|username.
5. If the first part of the message is “L” then the ListUsers method will be called.
6. If the first part of the message is “O” then SendFolder will be called with the folder name that will be parsed from incoming message.
7. If the first part of the message is “N” – “X” – “R” then these messages are also transmitted to all users except the sender; therefore SendToClients method will be called.
8. If the first part of the message is “CL” than ListFolders() method will be called.
9. If the first part of the message is “T” then CheckStatus method will be called

- **DoListen():** This method is used as a background listener thread to allow reading incoming messages.
- **CreateClientMessage(string):** This method will create message that will send to the clients or a specific client. It takes only one argument that specifies the operation it may be “Finished”, “Completed” or “Concatenate”. “Finished” means that image sending process is finished and “Completed” indicates that all clients received the slides. If the argument is Finished than the method will return “F|” else if argument is Completed then it will return “T|”. If the argument is Concatenate then it will return SL|slidelist as the message.
- **ReplyToSpecifiedUser(UserConnection, string):** this method will send the message only to the specified user who is indicated as the first argument to this method.

- **SentImages(string):** This method will be called when instructor wants to open slides on the whiteboard. The argument specifies the name of the folder. When this method is called it will open the folder specified as argument and it will send all the files to all clients on a specific stream. When the last image is transmitted to a client the message “F” will be sent to this client indicating that all images are sent.
- **AddNewPpt(string):** This method is responsible for creating a folder with the name that is specified as the argument. This function will create a stream and get the files into this folder. This folder will be created in the same directory with the program.
- **ConvertPptToImage(string):** This method is responsible for converting slides to image files. The argument specifies where these slides reside.
- **ListUsers(UserConnection):** This method concatenates all the client names and sends them to the user who requested user list.
- **CheckStatus():** This function will determine whether all the images of the slide show is sent all the clients successfully. It returns true or false. When returns true “T” message will be sent to the instructor that will indicate that all client received the images.
- **ListFolders ():** This method will investigate the current lectures folder and concatenate all the subfolders’ names and put it into the member slidelist

8. UserConnection Class:

This class is used for handling the connections to the server. When a new connection is found an instance of this class will be created in server class and this object will handle all operations. This class encapsulates the functionality of a TcpClient connection with streaming for a single user.

8.1 Members

- **Client (TcpClient):** Used for handling the connection.
- **ReadBuffer (byte []):** Used for storing the data received
- **Username (string):** Used for storing the username

8.2 Methods

- **UserConnection (TcpClient):** Start the asynchronous read thread and saves the data to the ReadBuffer.

- **SendData (string):** This subroutine uses a StreamWriter to send a message to the user. In this routine it locks the stream for no other threads to use the stream at the same time
- **StreamReceiver ():** This is the callback function for TcpClient.GetStream.Begin. It begins an asynchronous read from a stream.

9. Offline Tool Class:

This class gives instructor the options of creating and editing power point slides for the selected course. He/she can download an existing document and make some changes on that document such as adding new slides, adding some annotations to the existing slides or he/she can create a new ppt slide. This tool can use the whiteboard's drawing functions so it will help the instructor to create annotations on the slides. Only the instructor can use this class' functions.

9.1 Members:

- **Client (TcpClient):** Used for handling the connection with the server.
- **ReadBuffer (byte []):** Holds the message coming from the server.
- **UndoBuffer (int [20][]):** Holds the last 20 operations.
- **LineVector (int [][]):** Holds all of the lines information. Each row represents a line.
- **EllipseVector (int [][]):** Holds all of the ellipse information. Each row represents an ellipse. Two points represent an ellipse. These two points create a rectangle (invisible), which indicates the boundary of each ellipse.
- **RectangleVector (int [][]):** Holds all of the rectangle information. Each row represents a rectangle. Two points represent a rectangle.
- **PortNumber (int):** Holds the port number of the whiteboard module.
- **Copy (int[]):** Holds the information of the copied object.
- **Selected (int[]):** Holds the selected object info.
- **SlideNumber (int):** Holds the number of slides.
- **SlidePlace (int []):** Holds the list that shows places of the slides.

9.2 Methods:

- **SendMessage (string):** This method sends the message to server. This message says to server to make some operations.

- **CopyObject ():** This method copies the selected object into the copy array. If the selected object is line, the first column of the copy array will be 1, if it is ellipse, it will be 2, if it is rectangle, it will be 3. This method saves only the type of object and the index of this object, which indicates the place of this object in the array. For example; Copy[0] = 1 means that this object is line and Copy[1] = 25 means that 25. line in the LineVector array.
- **PasteObject ():** This method pastes the copied object into the clicked position of the mouse. The all points of the copied object moved according to the position of the mouse. After updating the points we call CreateMessage (points, AX) function. CreateMessage function prepare message for sending the server. Then server sends an information message to clients, and then all of the clients can see the new object into the desired points.
- **MoveObject (int, int):** This method moves the selected object. And also we call CreateMessage (index, MX). The row of the objects' array will be updated by using this function. If there is not any selected object then this method moves the specified object (according to the arguments, one of the argument shows the type of object an the other represents the index of this object). Then the row of the objects' array will be updated by using this function.
- **SelectObject ():** This method is used for selection of objects. Firstly we should indicate which type of object is selected. For ellipse and rectangle we have a rectangle boundary. This method searches the mouse point in these boundaries. If the mouse position is inside the more than one area then the nearest one is chosen. For line, the intersection point between the mouse position and line points is searched. This method saves only the type of object and the index of this object, which indicates the place of this object in the array. For example; Selected [0] = 1 means that this object is line and Selected [1] = 25 means that 25th line in the LineVector array.
- **EraseObject (int, int, int):** This method erases the selected object. And also we call CreateMessage (index, EX). The row of the objects' array will be deleted by using this function. If there is not any selected object then this method delete the specified object (according to the arguments, one of the argument shows the type of object an the other represents the index of this object). Then the row of the objects' array will be deleted by using this function. If the third argument is 1 then this function returns, else saving this operation into the UndoBuffer is done.

If undo stack is full we remove the oldest operation. And add the new operation into the top of the stack. The format of the UndoBuffer for erasing new object is like that:

UndoBuffer [top][0] = type of the operation ('E' means creation)

UndoBuffer [top][1] = type of the object ('L' means line, 'E' means Ellipse and 'R' means Rectangle)

And all the points of the erased objects are added to UndoBuffer.

- **CutObject ():** This method is the combination of the EraseObject (int, int) and the CopyObject ().
- **Undo ():** When the user clicks the undo button this method is called. If the type of the undo buffer is erase (UndoBuffer [top][0] == E), then CreateMessage (points, UAX) is called for adding this object into white board of all clients. And also add these points into the objects' array. If the type of the undo buffer is add (UndoBuffer [top][0] == A), then CreateMessage (index, UEX) is called for erasing this object into white board of all clients. And also erase this object from the objects' array.
- **DrawLine ():** This method draws all lines according to the LineVector array information.
- **DrawRectangle ():** This method draws all lines according to the RectangleVector array information.
- **DrawEllipse ():** This method draws all lines according to the EllipseVector array information.
- **AddNewObject (char, string, int):** This method has basically two save operations. One is saving into the objects' array. And the other is saving into the UndoBuffer. The first save is done according to the type of the object. The points in the second argument are added to the next row of the objects' array, which is specified by the first argument. The points in the second argument are separated with special character. If the third argument is 1 then this function returns, else the second save operation is done into the UndoBuffer. If undo stack is full we remove the oldest operation. And add the new operation into the top of the stack. The format of the UndoBuffer for adding new object is like that:

UndoBuffer [top][0] = type of the operation ('C' means creation)

UndoBuffer [top][1] = type of the object ('L' means line, 'E' means Ellipse and 'R' means Rectangle)

UndoBuffer [top][2] = index (index of the object)

- **ChangePen (string):** If the user is instructor, then this method calls the CreateMessage (properties, CP). If it is not instructor then change the properties of the pen according to the received properties. The argument of this class is separated by special character.
- **NextSlide ():** Show the next slide into the white board as a background.
- **PreviousSlide ():** Show the previous slide into the white board as a background.
- **UpdateObject (string, char, int):** This method is used for updating the points of the object. It has three arguments which are indicates points, type, and the index object.
- **GotoSlide (int n):** This method is used to show the n^{th} slide. Shows the slide specified by SlidePlace [n].
- **CreateTempFolder (string):** Create a new folder for saving the power point document with the given name, which will be received by the server if the instructor decides to upload the document. The name of this folder same as the ppt.
- **AddSlide (int n, int y):** This method is used to add a new slide to the current document with the specified index n. It increments the SlideNumber by one. Then it does:

Start with $i = \text{SlideNumber} - 1$

For each integer i such that $n \leq i < \text{SlideNumber}$

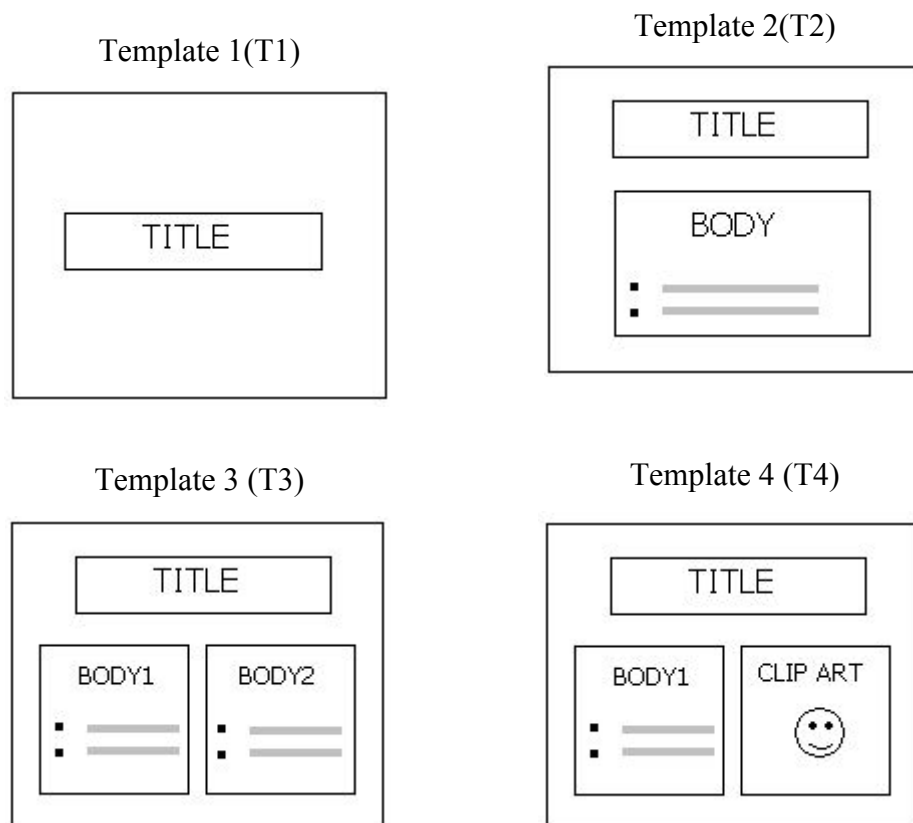
Do {

SlidePlace [i] = SlidePlace [i-1]

Decrement i by 1

}.

Then it stores the SlideNumber to the SlidePlace [n]. Finally it calls GotoSlide (n). Note that n is not allowed to be greater than SlideNumber+1 and less than 1. Here y specifies the template property of the current slide. We have 4 different templates for a slide: (The value in parenthesis identifies the template and will be defined as a macro like #define T1 1, #define T2 2, etc.)



- **RemoveSlide (int n):** This method is used to remove a slide from the current document with the specified index. Then it does:

Start with $i = n + 1$

For each integer i such that $n \leq i < \text{SlideNumber}$

Do {

$\text{SlidePlace}[i - 1] = \text{SlidePlace}[i]$

Increment i by 1

}.

It decrements the SlideNumber by one. Here n is not allowed to be greater than SlideNumber+1 and less than 1.

- **CreatePpt (string):** This method is used to create a new power point document with the given name. It first calls the method CreateTempFolder (string) to create a temporary folder that the document will be saved to, sets SlideNumber to 0, and then it calls the method AddSlide (1) to add a first slide to the created document.

- **SaveSlide ():** This method is used to save the current slide. If the instructor made changes on the current slide, and then decided to go to another slide or he/she finished with the document, the current slide is saved to the folder that was created to behave as a power point document. (The slide is saved as an image file) The instructor cannot make changes on another slide before he is finished with the current slide.
- **UploadPpt (string):** This method is used to upload a power point document that is stored at the instructor's computer to the server. The given string specifies the path of the document.
- **DownloadPpt (string):** This method is used to download the power point document specified with the given name. It is called when the instructor wants to edit an existing document that is stored at the server.

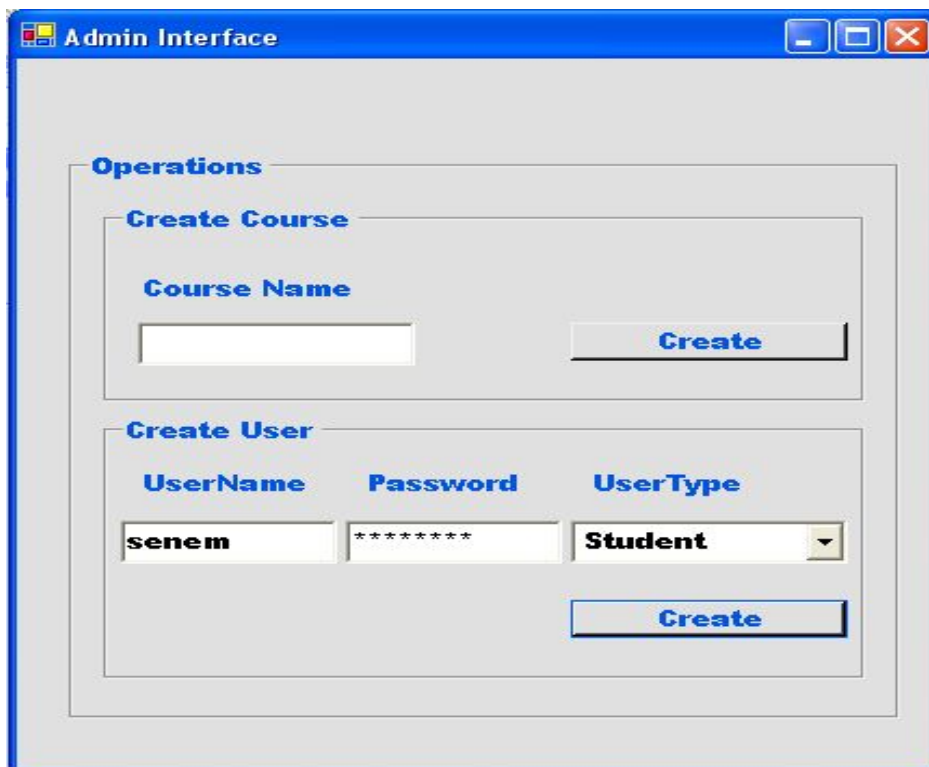
E. GRAPHICAL USER INTERFACES

E.1. LOGIN INTERFACE



The image shows a screenshot of a Windows-style application window titled "Login". The window has a blue title bar with standard minimize, maximize, and close buttons. The main content area has a light gray background. At the top center, the word "Welcome!" is displayed in a blue, italicized font. Below this, there is a section titled "Login" in blue. This section contains three labels: "User Name", "Password", and "User Type", all in blue. To the right of each label is a corresponding input field: a text box for "User Name", a text box for "Password", and a dropdown menu for "User Type" which currently shows "Student". At the bottom of the "Login" section, there are two buttons: "Clear" and "Sign In", both with blue text and black borders.

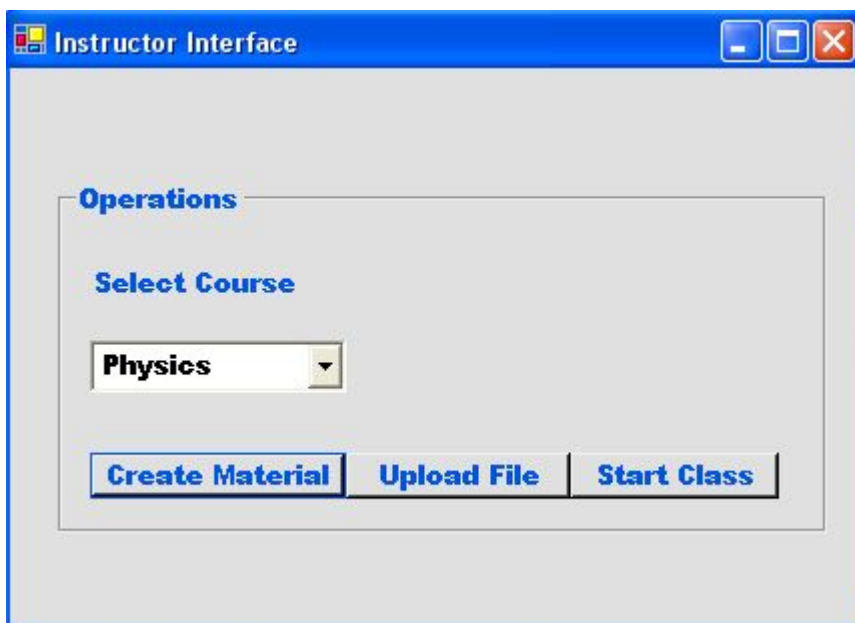
E.2 Admin Interface



The Admin Interface window has a blue title bar with the text "Admin Interface" and standard window controls. The main content area is light gray and contains a section titled "Operations" in blue. Inside "Operations", there are two sub-sections: "Create Course" and "Create User". The "Create Course" section has a text input field for "Course Name" and a "Create" button. The "Create User" section has three input fields: "UserName" (containing "senem"), "Password" (containing "*****"), and "UserType" (a dropdown menu showing "Student"). There is a "Create" button at the bottom right of the "Create User" section.

Operations		
Create Course		
Course Name	<input type="text"/>	Create
Create User		
UserName	Password	UserType
senem	*****	Student
		Create

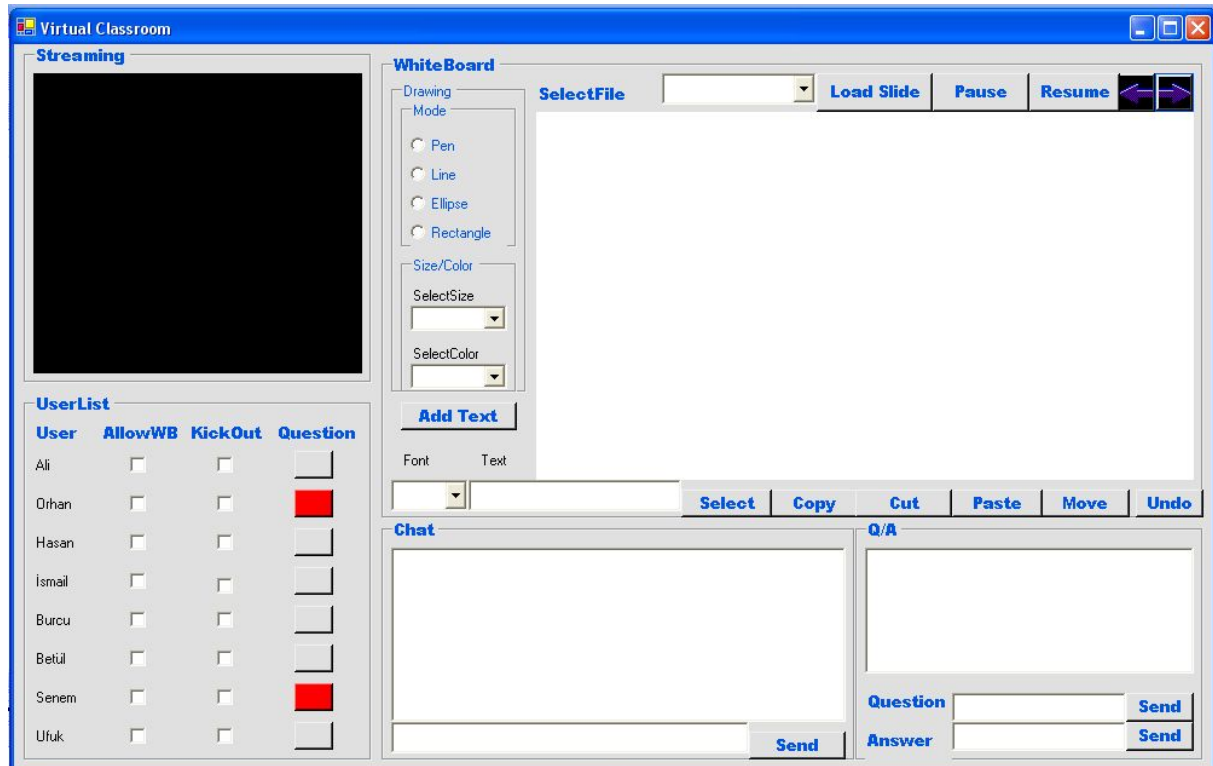
E.3 Instructor Interface



The Instructor Interface window has a blue title bar with the text "Instructor Interface" and standard window controls. The main content area is light gray and contains a section titled "Operations" in blue. Inside "Operations", there is a "Select Course" section with a dropdown menu showing "Physics". Below this, there are three buttons: "Create Material", "Upload File", and "Start Class".

Operations
Select Course
Physics
Create Material Upload File Start Class

E.4 Virtual Classroom



F. CONCLUSION

Now we completed the design of the project. The design report of the project made us to understand the details of our project. We first started with analyzing the system. At this point we met many difficulties but we did not give up. After synthesizing the requirements of the system we moved into design phase. We first constructed the use case and the scenarios and actors corresponding to them. Then making use of these use case we designed our data structures. Then we observed the interaction of the different modules with each other. After that we accomplished the architecture design of the system. Then we came to the user interface design of the project. We tried to design a simple and user friendly user interface. From this point on our next duty is to implement this system. We are sure that we will construct the system that we designed. We rely on ourselves.

Throughout the design step of the project, we collected information, searched for the resources of alternative implementation ways and tried to find out how to make the best design for a student registration system. With all the work we have done, we understood the complete roadmap of building a project, not only with its implementation but also with its

design and research steps. We have done interviews, diagram designs, user GUI's and many other things in the design step to construct this project. This design steps made us sure that, a well designed system is quite easy to implement.

With the courage of making a well designed system, in the next semester we believe that it will not be a lot hard to implement this project since we think our design is quite complete and successful.

G. APPENDIX

Task	Schedule
Dates:	26 Sep 2004 10 Jan 2005
1. Project Definition Phase	
1.1 Announcements of Project Topics	
1.2 Determining the preferences	
1.3 Assignment of the topics	
MS: Project is defined	
2. Survey Phase	
2.1 Survey on the project topic	
2.2 Preparing the proposal	
MS: Survey is finished	
3. Requirements Phase	
3.1 Searching on background of the project	
3.2 Communicating with related people	
3.3 Discussion on requirements	
MS: Requirements are defined	
4. Initial Design Phase	
4.1 Database	
4.2 Server	
4.3 Login System	
4.4 News Group	
4.5 User Management System	
4.6 Course Management System	
4.7 Virtual Class	
4.7.1 Student Lists	
4.7.2 White Board	
4.7.3 Chat Tool	
4.7.4 QA Box	
4.7.5 Video Screen	
MS: Initial Design is finished.	
5. Detailed Design Phase	
5.1 Login System	
5.2 Admin Interface	
5.3 Instructor Interface	
5.4 Server	
5.5 Virtual Class	
5.5.1 Communication Class	
5.5.2 Whiteboard	
5.6 Offline Tool	
5.7 User Connection	
5.8 Video Screen	
6. Prototype	