

CSE 681 Project 3

Remote UML Diagrammer

Operational Concept Document

Philip Pratt-Szeliga
Syracuse University
Software Modeling and Analysis
October 17th, 2007

Table of Contents

1. Introduction	3
1.1 Executive Summary	3
2. Overall System Architecture	3
2.1 Uses	4
2.1.1 Client Uses	4
2.1.2 Server Uses	6
2.2 User Interface	7
2.2.1 Client User Interface	7
2.2.2 Server User Interface	10
2.3 Top Level Tasks	12
2.3.1 Client Top Level Tasks	12
2.3.2 Server Top Level Tasks	15
2.4 Program Context	17
2.5 Threads	18
3. Analysis of Critical Issues	19
3.1 Latency of two directly connected clients	19
3.2 Locking issues with two directly connected clients	20
3.3 Maintainability	21
3.4 Server goes offline	21
3.5 Security	22
3.6 Communication errors	22
3.7 Disk space	22
3.8 Port number allocation	22
3.9 Future communication over the internet	23
3.10 Server-side client handling	23
3.11 Viewing Synchronization	23
3.12 File transfer	24
4. Partitions	24
4.1 Client Partitions	24
4.2 Server Partitions	26
5. Conclusion	28
6. References	29
7. Appendix A. Messages used in Remote UML Diagrammer	30
8. Appendix B. MD5 encryption	32

1. Introduction

The Remote UML Diagrammer is a tool to edit and view UML Diagrams remotely. The suite of applications includes a client application to edit and view UML Diagrams and also a server application to store UML Diagrams and to help initiate client connections with each other. In addition the Remote UML Diagrammer Suite will support file transfer. This Operational Concept Document (OCD) describes the software architecture of the Remote UML Diagrammer. It will discuss the uses, user interfaces, top level tasks and context in Section 2. In Section 3 this OCD will describe the critical issues of implementing the software architecture. The critical issues are latency of two directly connected clients, locking issues with two directly connected clients, maintainability, server goes offline, security, communication errors, disk space, port number allocation, future communication over the internet, server-side client handling, viewing synchronization and file transfer. Section 4 will detail the various modules and partitions present in the design. Section 5 will draw conclusions on the previous sections. Lastly, section 6 will list the references and sections 7 and 8 are appendices.

1.1 Executive Summary

The Remote UML Diagrammer is a tool for users to collaborate to create UML Diagrams. It is an extension of the UML Diagrammer in which users loaded and saved UML diagrams and created and edited diagrams in either a text editor or a graphical environment. The Remote UML Diagrammer consists of a server executable and a client executable which will be developed concurrently. The server will feature a user login system in which a username and password is verified against the known good. After a user is logged in they will be placed in the main lobby. In the main lobby they will see a list of the available UML diagrams on the server and who is editing them. The main lobby will also allow a user to edit their password, upload/download and delete files and do file differences. From the main lobby the user can choose to create or enter any number of rooms that allow for editing and viewing of UML files. Inside one room there can be two concurrent, directly connected UML Editor users and any number of UML Viewer users. In this document we discuss the system architecture, the critical issues and the partitions of the Remote UML Diagrammer. After reading this document the reader should have a clear understanding of how to create a high level and detailed design for the Remote UML Diagrammer.

2. Overall System Architecture

The Overall System Architecture describes the Uses, User Interfaces and Top Level Tasks of the Remote UML Diagrammer. The Uses Section describes how the software will be used and who will use it. The User Interfaces Section describes how the users will interact with the software in a graphical manner. The Top Level Tasks Section describes each and every function that is accessible to the user in detail. Each section will be divided into a Client Section and a Server Section because the Remote UML

Diagrammer will have two distinct executables. One executable will be for a client and the other will be for the server. To help the reader I have included a system architecture diagram below.

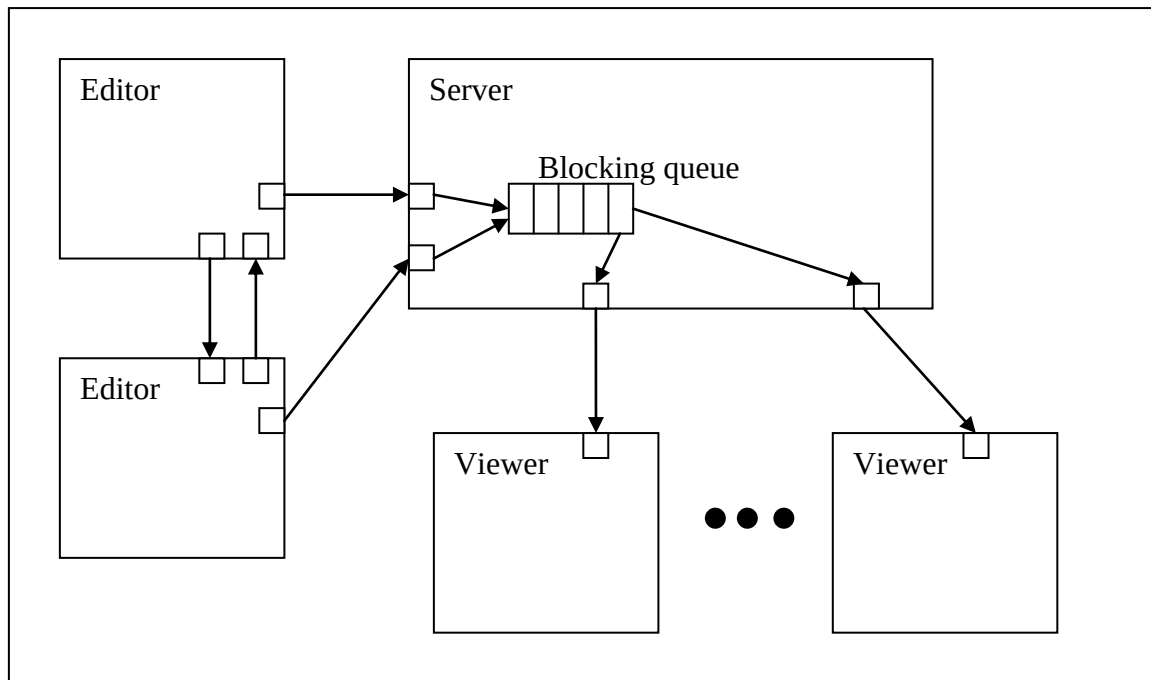


Figure 2.1 – System Architecture Diagram

Shown in the diagram above is the general system architecture for one room. This diagram is basically repeated N number of times to achieve the functionality of a server. In one room there is a server thread that has a blocking queue. The editors connect to the blocking queue through sockets to send edit and save messages. The viewers connect to the server to receive the edit messages. The small boxes represent a socket. Lastly, there is a direct connection between the two editors. This peer to peer connection will be used to send messages (edit, chat, etc) as well as to do file transfers. In the final design there may be sockets from the viewers to the server and sockets from the server to the editors for chat and file transfer broadcast purposes. Those sockets have been left out of this diagram for clarity.

2.1 Uses

The uses of the software architecture are how people and other software interact with the system [Fawcett1]. The client and the server both have different uses and I will talk about them below.

2.1.1 Client Uses

The client program has use cases for two users. These users are a UML Editor and an UML Viewer. The client use cases diagram is shown below.

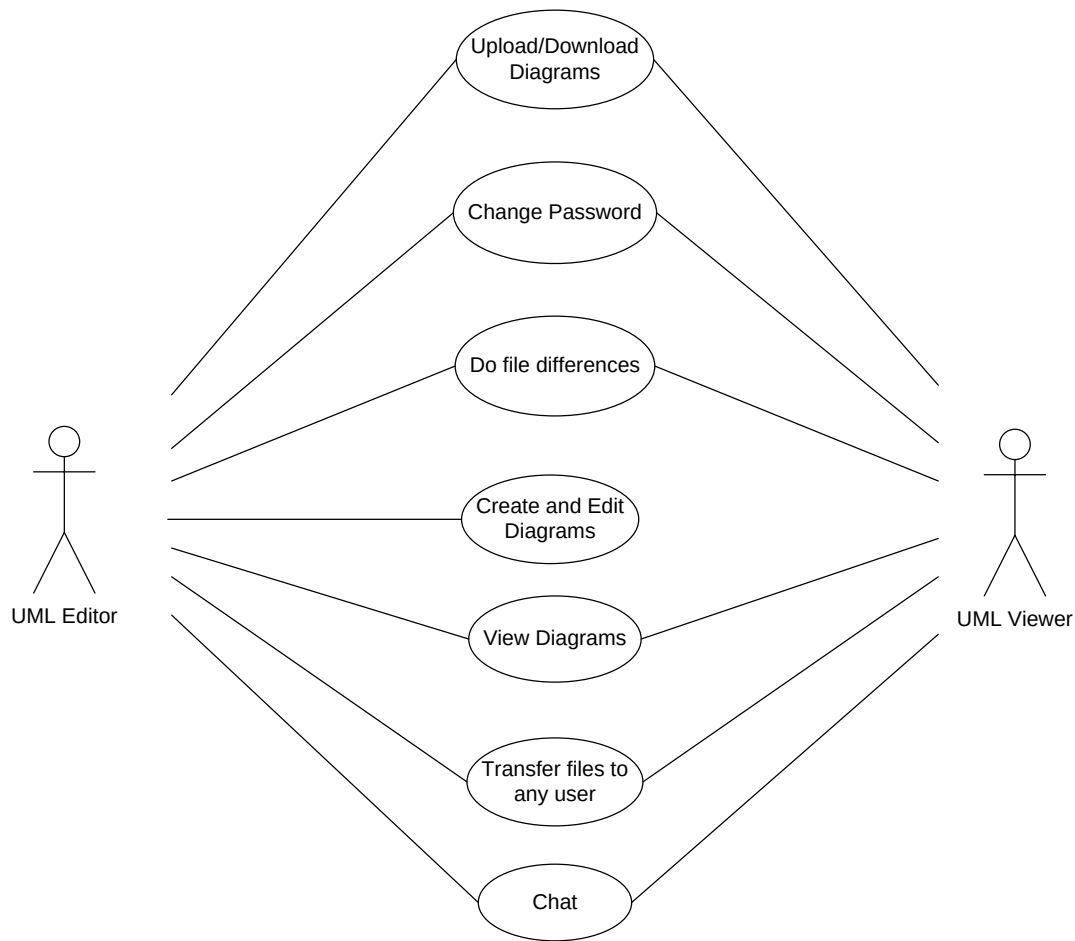


Figure 2.1.1.1 – Uses cases for the Remote UML Diagrammer Client

The UML Editor and the UML Viewer can be a Software Developer, a Manager, a Student, a Teacher or anyone who wants to work with UML Diagrams. The UML Editor is someone who can create and edit UML Diagrams. This user will be able to directly connect to another UML Editor to collaborate. The UML Viewer is someone who can view diagrams as read-only. He or she can view a collaboration taking place or view static files.

The UML Diagrammer Client will have three main modes or operation. The first mode is main lobby. When in this mode the user can upload and download diagrams, change the password, do file differences and chat. The second mode is direct connection. In this mode the user can create and edit diagrams, transfer files to any user and chat. The last mode is viewing. When in the viewing mode the user can view diagrams, transfer files to any user and chat.

Uploading and downloading diagrams consists of choosing a file or files on the remote or local machine and then transferring them over a socket. Changing the password consists of sending the md5 encrypted password over the socket to the server. Doing file differences consists of viewing two files with the differences highlighted.

Creating and editing diagrams consists of manipulating files either directly connected to another UML Editor user or not. Viewing diagrams consists of viewing static or changing diagrams. Chatting consists of sending text to the server. The server will broadcast the text to all users in the room.

2.1.2 Server Uses

As shown below there is only one user type in the server. This is because there is little code in the server related to user interface.

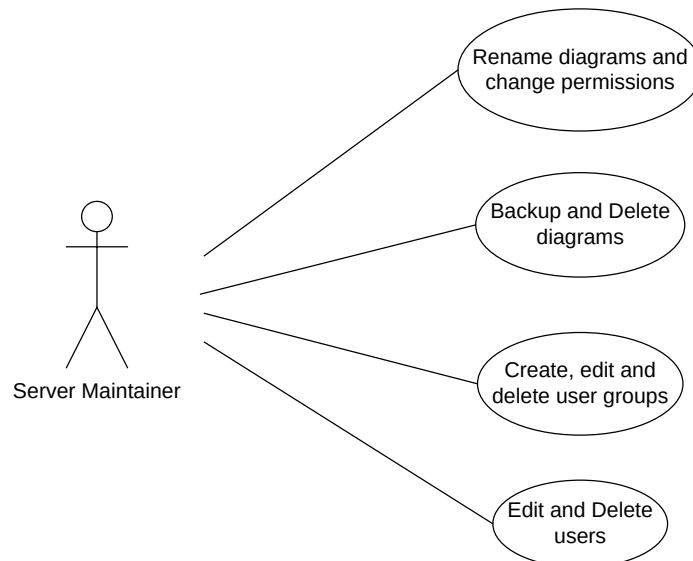


Figure 2.1.2.1 – Use cases for the Remote UML Diagrammer Server

The server maintainer is the only use case for the server. The reason that there is any use case at all is because users sometimes do things like forget their passwords, or create diagrams and then leave them on the server for extended periods of time.

Renaming diagrams is to change the name of a diagram. This is useful because a client can name a diagram and then go on vacation. Normal users do not have permission to rename the users diagram so the server maintainer must do this. Also, the server will check to see if anyone is using the file before it is renamed. Changing permissions involves changing the owner of a file and changing who can access the file.

Deleting diagrams is to remove the diagram from the server. The server maintainer has the option to backup diagrams as well. Backing up and deleting diagrams is needed because users sometimes create a lot of diagrams for projects and then never remove them from the server when the project is done. This causes the server's disk drives to become full. As with renaming, the server will check to see if anyone is using the file.

Creating, editing and deleting user groups involve the Unix-like file permission system of the Remote UML Diagrammer. These permissions allow for the protection of user's documents while allowing someone to access someone else's document.

Editing and deleting users allows for the server maintainer to change users. Editing is changing the user password, because sometimes this must be reset since users can forget their password. Deleting users involves removing an old user from the system.

2.2 User Interface

The user interface is a graphical method for the user to interact with the software. The user interface for the client and server will be discussed separately below.

2.2.1 Client User Interface

The client user interface consists of three main portions. The first portion is the welcome screen. The second portion is the main lobby and the third portion is the diagram editor/viewer. The welcome screen is shown below.

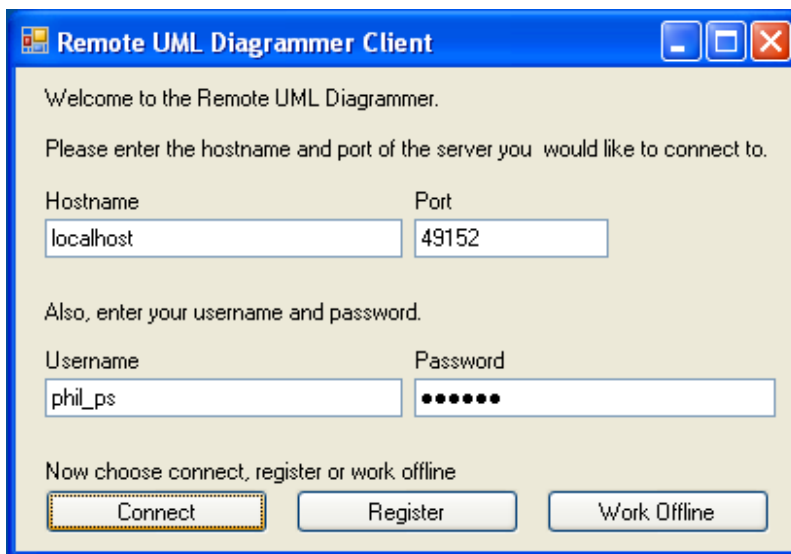


Image 2.2.1.1 – Image of Welcome Screen

From the welcome screen the user can enter the hostname and port of the remote server. Then he or she will enter a username and password. If the user does not have a username and password he or she can make one up and click register. If the user clicks work offline the user will be presented with the interface of project 2, the UML Diagrammer and he or she will edit the diagram offline. If the user clicks connect, a popup dialog will be shown that says "Connecting to server" with a progress bar. Once the user is connected to the server and the password has been verified as correct, the screen shot below (Image 2.1.1.2) will be shown.

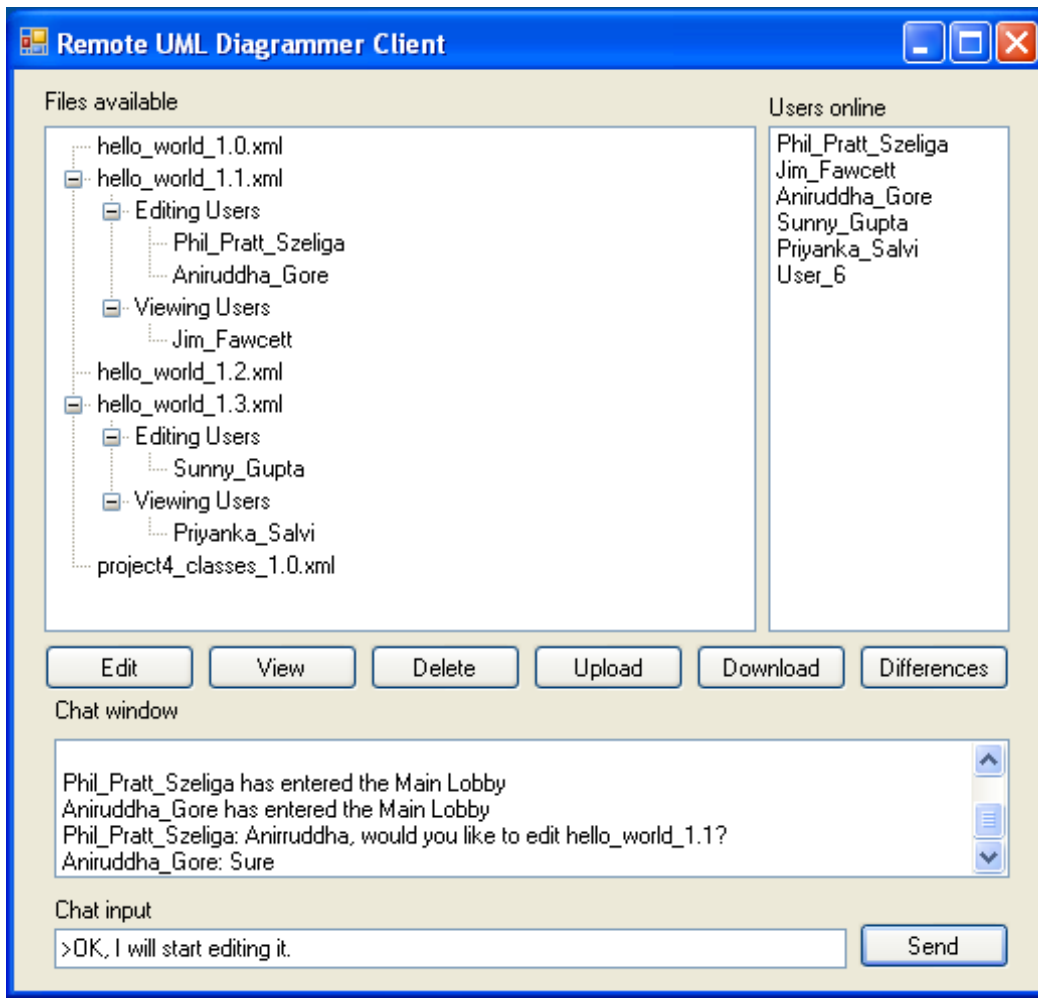


Image 2.1.1.2 – Image of Main Lobby

Above is an image of the main lobby. This is a place where the client can communicate with other users who are in the main lobby, and also view what files are being edited and viewed. As seen in the image there is a listing of all files on the server (Files Available), a listing of the users that are online (Users online) and a chat window (Chat window). From here the user can edit, view, delete, upload, download or do differences on a file. I will discuss each action separately.

If the user clicks edit, the server will check how many users are editing the file. If there are no users editing, the diagram editor/viewer will be displayed (Image 2.1.1.3) and the user can begin editing. If there is one, the diagram editor/viewer will be displayed and the two users will become directly connected to each other. If there are two, the user will be told that he or she can only view the file right now. The editor and viewer consist of the same code and GUI. Editing features (except panning) are disabled when the user is viewing. Below I will show and discuss the editor/viewer GUI and then I will begin talking about the other options that the user when he or she is in the main lobby.

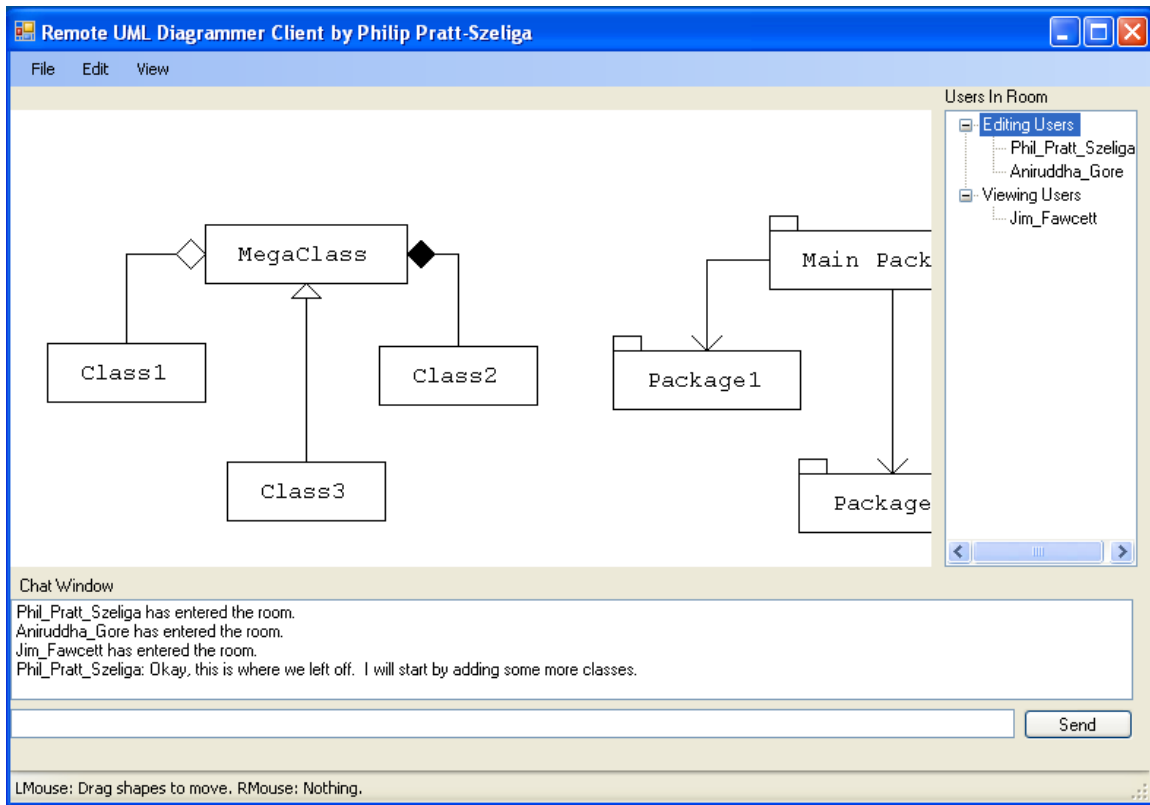


Image 2.1.1.3 – Image of Editor/Viewer

Pictured above (Image 2.1.1.3) is the editor/viewer window. In this window we will have an editing/viewing panel, a tree control showing the users in the room and a listbox showing the chat conversation. The editor/viewer will be based on the project 2 code and then extended to support chat, file transfer and remote diagramming. In the editor/viewer window there will be one extra option in the File Menu to send a file. When the user selects the Send a File option he or she will be presented with a standard .NET 2.0 OpenFileDialog where he or she will choose any number of files to send to the people in the room.

I will now return to discussing the additional options in the main lobby that have not been discussed. These options are file delete, file upload, file download and file differences. To do file delete the user will select a file from the Files Available Listbox (Image 2.1.1.2) and choose delete. He or she will be presented with a confirmation dialog and then the file will be deleted if the user owns the file.

To upload a file the user will be presented with a dialog to choose the file pattern. Once this is done the standard .NET 2.0 OpenFileDialog will be shown and the user can choose which files to upload. If a conflict occurs the user will not be able to upload that file. To download a file the user will choose the file from the Files Available Listbox and if it is not being edited, he or she will receive the file in a local directory specified by him or her.

Lastly, to do file differences, the user will select two files from the Files Available Listbox. If these two files are not being edited a viewer window will appear that shows the common content with black pen, the content only in file one with red pen and the content only in file two with yellow pen.

2.2.2 Server User Interface

The server user interface consists of one main dialog with three screens that are activated with a tab control. Below the first tab is shown.

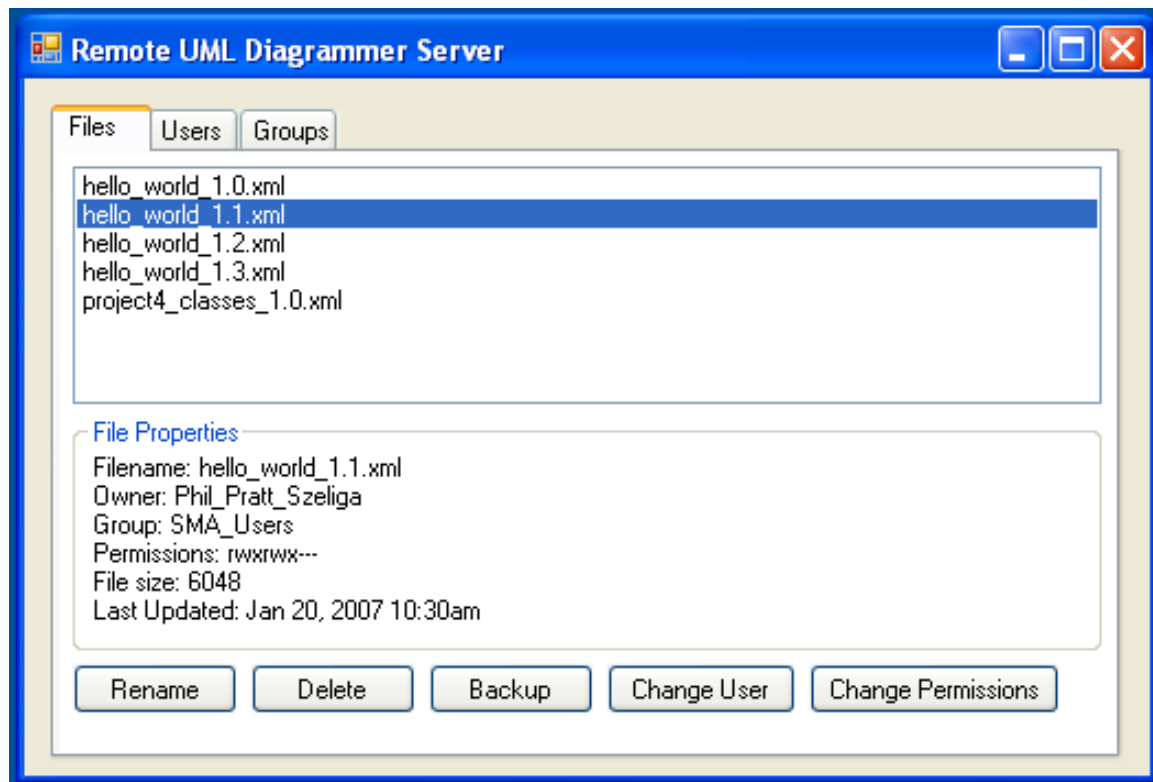


Image 2.2.2.1 – Files tab control in Server Main Window

As shown above there is a listing of all the files on the server. Once the user clicks a file the file properties will be updated with the owner, group, permissions, file size and modify date.

From the files tab control the user will be able to rename, delete and backup files. In addition he or she will be able to change the user of a file and change the permissions.

Below is an image of the users tab control. It has a list of users in the top. Below the list of users it will show what groups each user belongs to. The system administrator will have the options to add a new user, rename a user, change the user's password, add groups associated with a user and remove a group associated with a user. Note the add group and remove group simply changes what groups a user belongs to. It does not

remove or add a group from or to the system. That is done in the third server image, the groups tab control (Image 2.2.2.3).

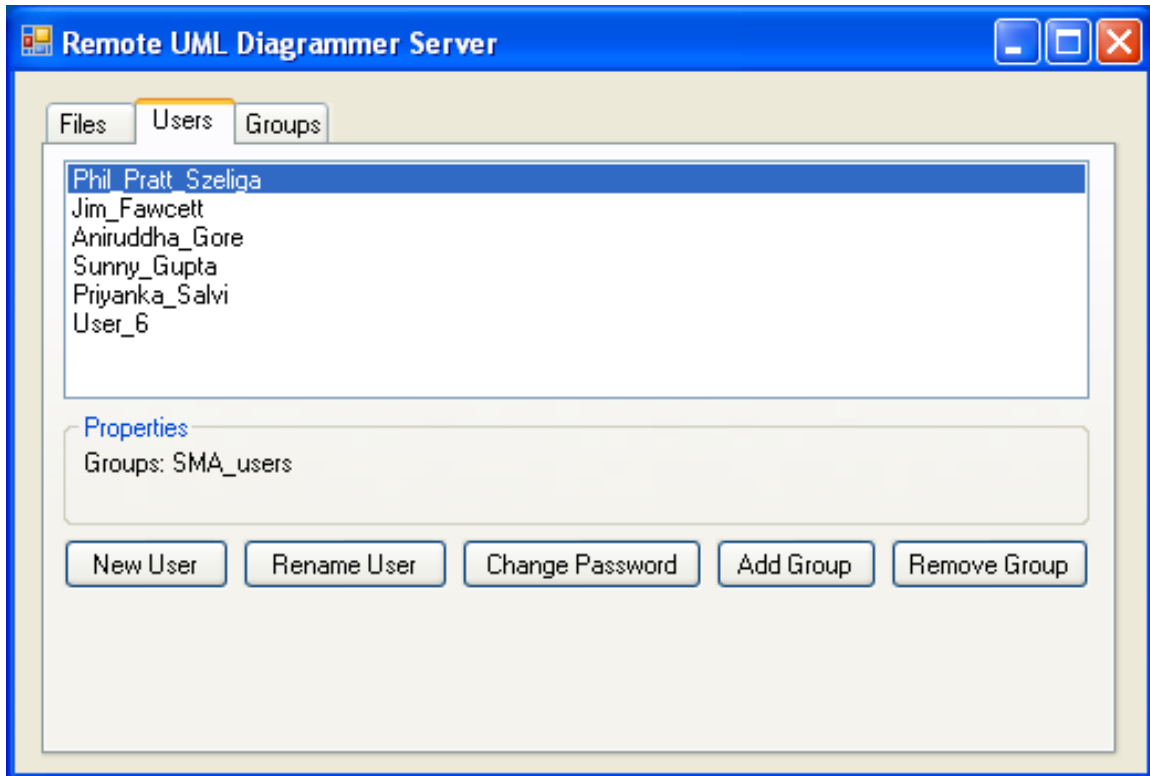


Image 2.2.2.2 – Users tab control in Server Main Window

Shown below is the groups tab control. It is a very simple page that shows all the groups in the system. The administrator has the options to add a new group or remove the selected group from the system.

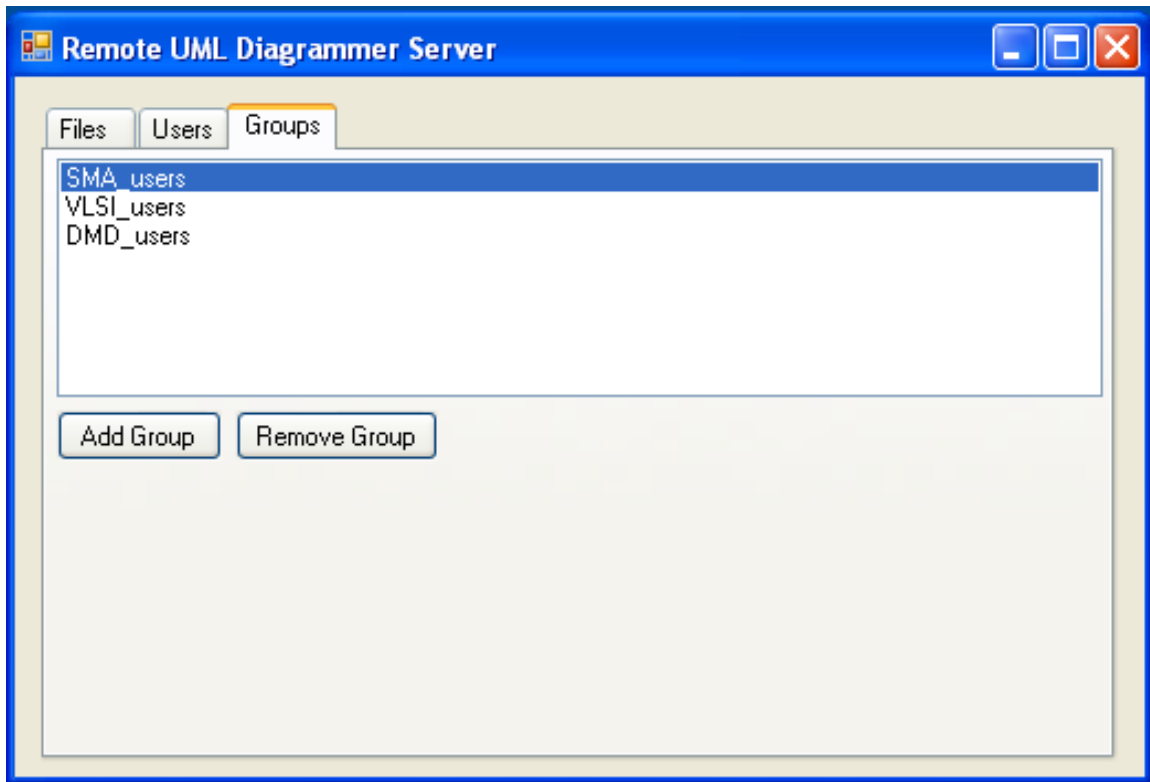


Image 2.2.2.3 – Groups tab control in Server Main Window

2.3 Top Level Tasks

The top level tasks of a software architecture are functions that the user or thread can execute. Since there are two executables in the Remote UML Diagrammer, I have divided the top level tasks into two sections. These sections are client and server.

2.3.1 Client Top Level Tasks

First I will talk about the client top level tasks of a Professional Remote UML Diagrammer System and then I will talk about the subset of features that will be implemented in project 4.

The professional top level tasks are:

- Login to server
- Upload/Download files
- Open/Create/Save/Delete files
- Edit/View UML Diagrams
- Networked undo
- Networked cut/copy/paste
- Panning
- Transfer/Receive files to/from any user – Includes encryption of file transfer.

- View documents that have been received from remote users
- Send and Receive Chat
- Check file differences
- Archive files with version control system

The additional features of the Professional Remote Diagrammer that are not top level tasks are:

- Chat with encryption
- Chat with private messages
- Chat with spam checking
- Encryption of file transfer
- Encryption of login password

The Project 4 top level tasks are everything in the professional diagrammer, except:

- Networked undo
- Networked cut/copy/paste
- Check file differences
- Archive files with version control system

In addition Project 4 will not implement these features:

- Chat with encryption
- Chat with private messages
- Chat with spam checking
- Encryption of file transfer

Below is an activity diagram for the client application.

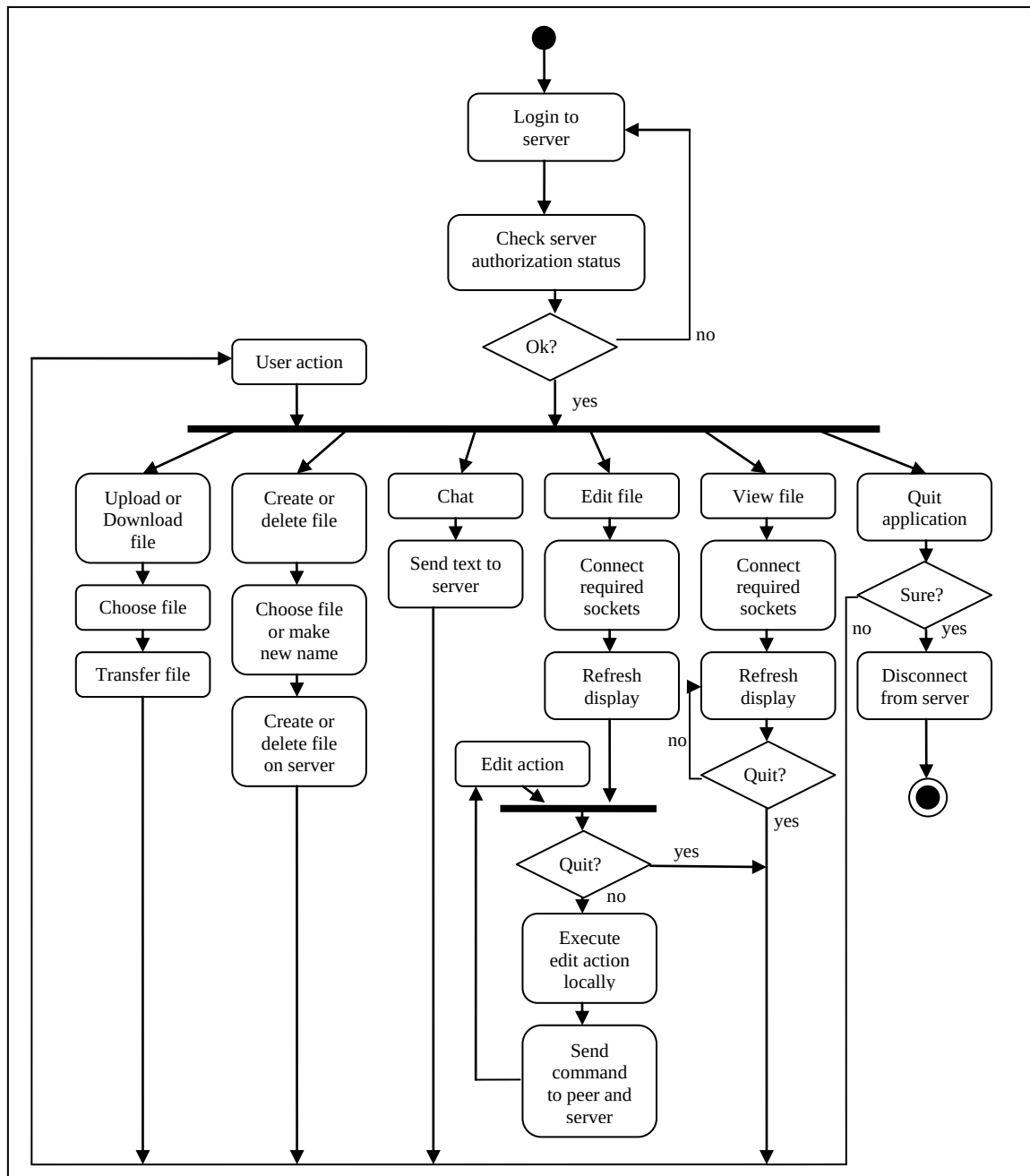


Figure 2.3.1.1 – Activity diagram of Remote UML Diagrammer Client

The program starts with the client logging into the server. If this is successful, the user is shown the GUI for the main lobby. In the main lobby the user can upload, download, create or delete files. In addition he or she can chat with other users in the main lobby and choose files to edit or delete. I will discuss each of these tasks in detail below.

The upload file task will open a file dialog in which the user chooses a file or files that will be uploaded. The files selected will then be sent over a socket in 1024 byte blocks. Once this is done the client will wait for further user actions. Similarly, the download task will download the file that is selected using a socket.

The create file task will ask the user to choose a new filename and then create it on the server. Similarly, the delete file task will delete the selected file from the server. Another simple task is the chat task. The user will simply send a message over a socket to the server.

The edit task is a more complex task. With edit, the user will choose which file he or she would like to edit. Once that is done the edit dialog is opened and the program connects to the server and the peer. The display is updated with the initial file contents and then the program waits for a user action. If the action is not quit, the action is performed locally and then transmitted to the peer and the server. The display is then refreshed and execution repeats. A similar task to editing is viewing. In viewing the user will also choose which file he or she would like to open. A socket to the server is opened and the viewer refreshes the display whenever messages are received.

Lastly, the quit action simply exits the program if the user is sure he or she would like to quit.

2.3.2 Server Top Level Tasks

The server is a multithreaded application that handles client requests and also allows a system administrator to maintain the server. The top level tasks and activity diagram of the client handler mirror the client application, so much of the information is redundant. Therefore, I will not repeat that information here and I will simply talk about the top level tasks visible to the system administrator.

The top level tasks that we will implement in the professional server are:

- Rename file
- Delete file
- Backup file
- Change user of file
- Change permissions of file
- Add a new user
- Rename a user
- Change a user's password
- Add a group to a user
- Delete a group from a user
- Add a group to the system
- Delete a group from the system

In the project 4 server we will not implement these top level tasks:

- Backup file
- Change user of file
- Rename a user

Below is an activity diagram for the system administrator thread.

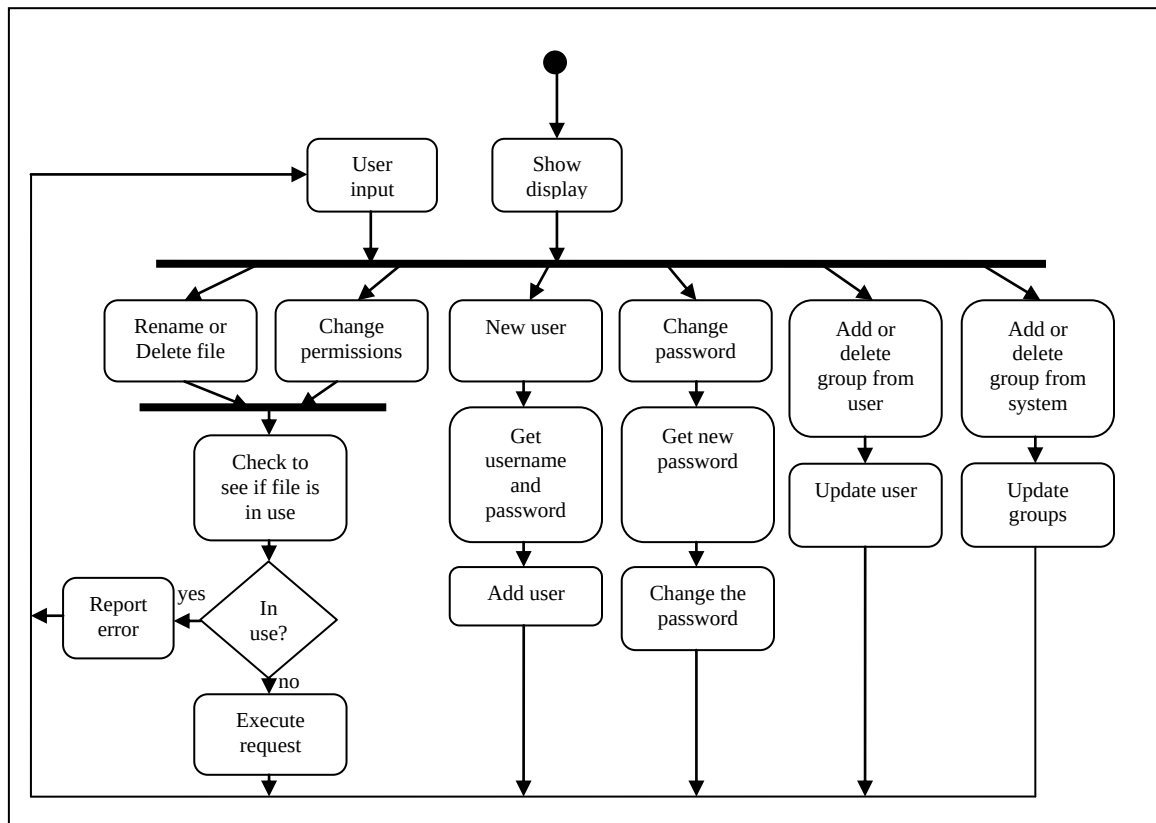


Figure 2.3.2.1 – Activity diagram of Remote UML Diagrammer Server, System Administration Tasks

Shown above is the activity diagram of the Remote UML Diagrammer Server (System Administration Tasks). I will now discuss each task in detail.

The rename or delete file task checks to see if the file is in use and then, if it is not, renames or deletes the file. Similarly, the change permissions task checks to see if the file is in use and then changes the permissions.

The new user task prompts the system administrator to enter the username and password of the new user and then updates the users.xml file. The change password task prompts the system administrator to enter the new password and then also updates the users.xml file.

The add or delete group from user prompts the system administrator to choose a user and allows the system administrator modify his or her group settings stored in the users.xml file. Lastly, the add or delete group from system prompts the system administrator to choose a group or type in a new name to update the groups.xml file.

2.4 Program Context

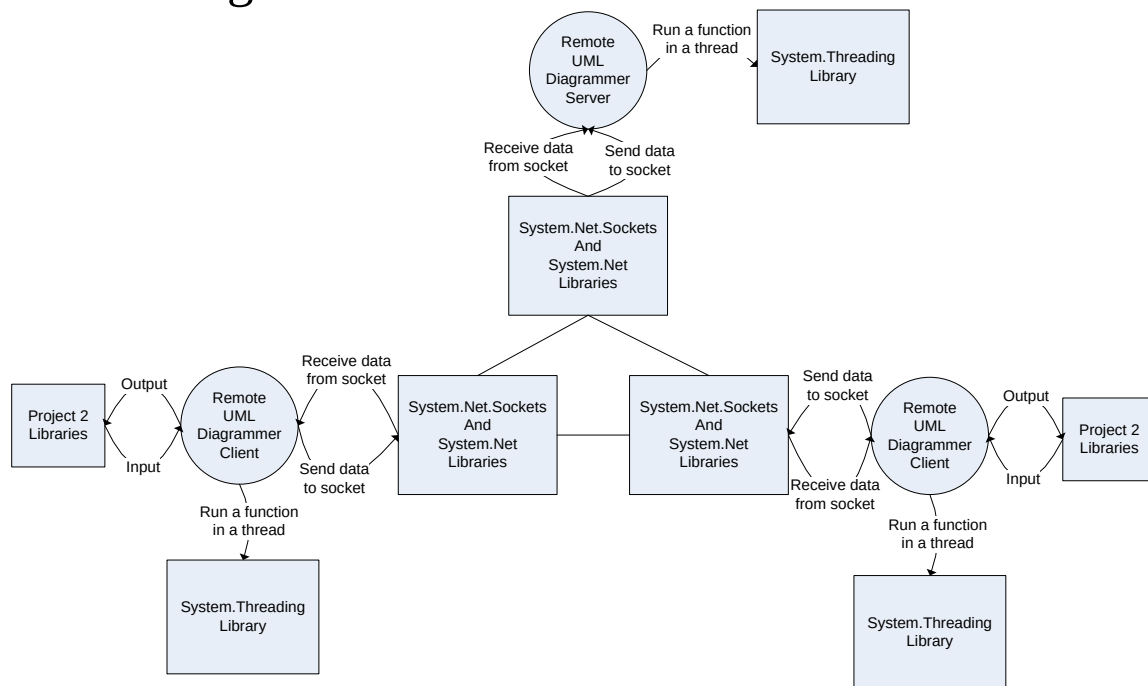


Figure 2.4.1 – Context diagram for Remote UML Diagrammer

The figure above details the context for the Remote UML Diagram Client and Server. On the bottom there are two Remote UML Diagrammer Clients and on the top there is the Remote UML Diagrammer Server. I have not included the viewing clients in this picture. The viewing clients will be the same as the clients shown, except they cannot edit the diagram. They will be connected to the server as well.

The libraries in both the clients shown consist of the project 2 libraries and also System.Threading, System.Net.Sockets and System.Net. The project 2 libraries consist of System.IO, System.Xml.Serialization, System.Collections.Generic, System.Collectors and System.Windows.Forms. One additional library that will be used is the System.Security.Cryptography library. This was left out for clarity.

The System.IO library combined with the System.Xml.Serialization library will provide the functionality to serialize data objects into XML files. They will also provide the functionality to deserialize XML files into data objects. These two libraries will still be used in the Remote UML Diagrammer even though I have decided to use custom XML message passing as discussed in the critical issue 'latency of two directly connected clients'. The libraries will be used for loading the shapes.xml and connectors.xml files used in project 2 and also new files called users.xml and groups.xml. Users.xml and groups.xml will store information for the Unix-like permission system.

The System.Collections.Generic library will be used due to the List<T> class from project 2 and also due to the addition of the Queue<T> class for project 4. The Queue<T> class will form the basis of the BlockingQueue<T> class.

BlockingQueue<T> implements a blocking queue and is essential for two threads to reliably communicate with each other.

The System.Security.Cryptography library will be used to call functions in the MD5CryptoServiceProvider class. This class is the basis of MD5 encryption and will be used to encrypt passwords. I have discussed this further in section 3.5.

The System.Net.Sockets library will be used to support socket communication in both the client and the server. Sockets will be used extensively in this project to facilitate process to process (and machine to machine) communication using message passing. The TcpClient, TcpListener and Socket classes will be used as the basis for the Sender and Receiver classes. System.Net is required because the class IPAddress with the function Parse will be used.

Lastly, the System.Threading library will be extensively used in project 4 to facilitate threading. Threading allows for two portions of one application to execute pseudo-concurrently. Pseudo-concurrently means that two portions of code operate at almost the same time. The machines that I will be using have a single core, single processor architecture, so all threading will be pseudo-concurrent due to timesharing of the single processing resource. Threads will be created for anything dealing with sockets so that we can have sending and receiving tasks operating almost concurrently with the graphics display.

2.5 Threads

Since we are using sockets, threads will be used frequently in the Remote UML Diagrammer. For every receive object there will be two threads and for every send object there will be one thread. The receive object has a thread that listens for incoming connections and then spawns a ClientHandler to handle the client. It also has a thread that receives the individual characters from the socket and enqueues a framed message into a blocking queue. The send object simply has a thread that dequeues from the sending blocking queue and sends the message out a stream. I have included an event diagram below that details the communication system showing the threads involved.

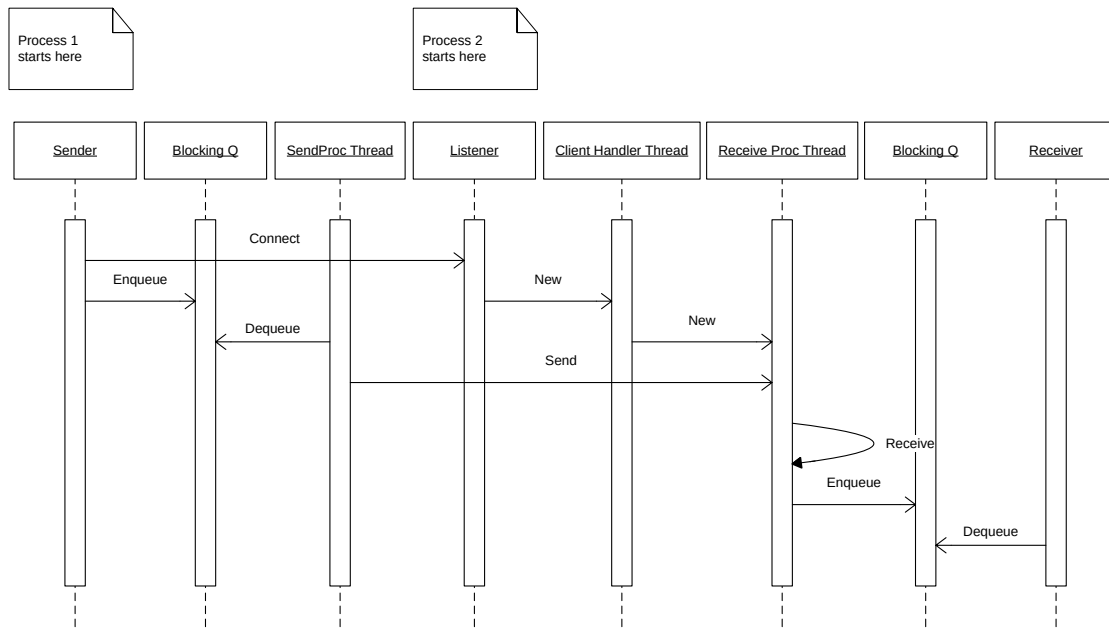


Figure 2.5 – Event trace diagram of communication system showing threads.

3. Analysis of Critical Issues

Critical issues are possible risk items that could cause the implementation of the project to fail. Therefore, since I do not want our implementation to fail, I have determined the critical issues for this software architecture. These issues are latency of two directly connected clients, locking issues with two directly connected clients, maintainability, server goes offline, security, communication errors, disk space, port number allocation, future communication over the internet, server-side client handling, viewing synchronization and file transfer

3.1 Latency of two directly connected clients

In the project purpose for this project it is mentioned that the latency of two directly connected clients should be addressed. Therefore, I have listed this as a critical issue in the design. Latency is the time delay between the moment something is initiated, and the moment one of its effects begins or becomes detectable [Wikipedia1]. We are specifically interested in the latency of editing UML diagrams. To do this we will be sending messages between the directly connected clients.

To achieve minimal latency I have devised a message passing scheme that uses very little text to represent the commands to be sent. In this message passing scheme we will only send the required parameters of a class, for instance. We will assume that each client has the same shapes.xml and connectors.xml that describe how to build the shapes, so we will not have to transmit each individual segment. We will start each message with the

<msg> tag and end the message with the </msg> tag. This is so that we can properly frame the message. Inside the message body we will include all the parameters needed to describe the command. These parameters will be in comma separated variable format to minimize the number of characters being sent over the socket. An example message is shown below.

```
<msg>ap,10,1,100,100</msg>
```

This message is sending five parameters. The parameters that are being sent are add polygon command, polygon identifier, polygon type, location x and location y. I have shown below in a table the format of the rest of the commands.

Command	Command Code	Extra Parameters
Add polygon	ap	Poly_id, type, location
Delete polygon	dp	Poly_id
Add connector	ac	Conn_id, poly_id1, poly_id2, type
Delete connector	dc	Conn_id
Rename polygon	rp	Poly_id, new_text
Move polygon	mp	Poly_id, new_location_x, new_location_y

We will use a common library to read and write xml messages in this way. The full listing of all the commands used in the Remote UML Diagrammer is shown in Appendix A.

The only problem with this message passing scheme is that the move polygon command could generate many messages. If this is a problem I will only send the command when the user releases the mouse button, rather than on every MouseMove event.

3.2 Locking issues with two directly connected clients

In the Remote UML Diagrammer there is a shared resource that is being shared over the network. This shared resource is the uml diagram itself. We must lock this shared resource somehow or else one client can be adding a connector to a shape while another client is deleting the shape. This will cause the connector to be invalid. No errors will occur when there is an invalid connector in my project 2 code, but they are not deletable in the graphical editor. In order to delete an invalid connector someone must go in and edit the xml file that represents the uml diagram.

In order to solve the locking issues we will be using a token passing scheme. In this scheme the client with the token is the only client who can edit. The client with the token does the edit operations and after he or she is done the token it is passed to the other client. If the other client has no operations to do the token is immediately passed again. While this method will increase the network bandwidth usage it should not decrease the

latency of the peer to peer communication. Automatic token passing is a little more elegant than asking the user to push a button to pass the token, however, I would not recommend operating this way when working over the internet. When we are working over the internet the time for the token to travel will probably slow down the system. Therefore, when working over the internet, we must have the user signal when he or she is done for the moment or possibly use some other method.

Note it has been mentioned recently in class that we will not be required to lock the uml diagram in this manner. I have left the text above as an example of how we could lock the uml diagram.

3.3 Maintainability

From my experience in the past, I have found that networking client/server applications often have a lot of code to do relatively simple tasks. This is because networking applications have to implement features such as blocking queues and threading to make them robust. Additionally, there must be code for serialization and deserialization. While developing the application, the serialization and deserialization code may go through many changes. Due to the requirement that the two separate applications have to have compatible versions, I have decided to list maintainability as a critical issue.

In order to keep the software versions identical, I will create a new version for both the client and the server whenever I am editing either program. In addition to careful version control I will strive to create common libraries for shared code that can be used by both subsystems. For instance the BlockingQueue, Sender and Receiver class will be common to the server and client. I plan to keep these in libraries. Also, the message system will use custom serialization and deserialization functions to convert from object to XML and back again. I will create one library to handle both serialization and deserialization so that I am sure to edit both functions whenever I am editing one.

What I have discussed in this section becomes even more important for systems that many users will be contributing to. A more elaborate versioning system would be required in this case and also the use of a version control system.

3.4 Server goes offline

In my architecture the server is required for directly connected clients (peer to peer clients) to find each other and establish connection. If the server is down they cannot communicate. After discussion in the help sessions it was determined that there is nothing that can be done if the server goes down except to wait for it to be restored. Therefore, I will try to maintain backward compatibility with the project 2 file format and this client can be used for offline editing. If backward compatibility cannot be achieved there will be an offline editor available as part of the project 4 application suite.

3.5 Security

Since this is a multi-user application and information could possibly be sent over untrusted networks in future versions, we must address the issue of security. Security is defined as the condition of being protected against danger or loss [Wikipedia2]. In this context we would like to protect information about the user and also the information the user sends.

To protect information about the user we will be storing the user passwords using md5 encryption on the server. A user's password is actually a very valuable piece of information because often users use the same password for every account they own. To test out md5 encryption I have included source code in Appendix B.

In the professional version of the Remote UML Diagrammer it would be desirable to support encryption of the chat messages and the file transfers. This way it would be harder to intercept chat messages or file transfers over the internet. Specifically, the professional version will support SSH [Wikipedia3] either directly or through tunneling.

3.6 Communication errors

Since this is a networked application suite, there exists the possibility of communication errors. A communication error could be a file payload sent incorrectly or a socket connection disconnected. In order to solve communication errors when sending files we will include a checksum at the end of the transmission (discussed in section 3.12) that the clients must validate. In the student version of the Remote UML Diagrammer it will simply ignore the file. In the professional version more elaborate things can be done such as resending the file if there is corruption.

Additionally, I think it should be required in the student version of the Remote UML Diagrammer for the program to not crash on the event of the network connection becoming unplugged. Therefore, I will impose this requirement on project 4.

3.7 Disk space

In our application suite disk space is a concern. We would like the Remote UML Diagrammer Server to be able to host many files. Previously, the UML Diagrammer for project 2 was designed so that each user file uses the least amount of space (i.e. not storing each segment for each shape). Therefore, this critical issue has been avoided.

3.8 Port number allocation

In a professional application you must always be aware of how your program interacts with the surrounding protocols and constraints. One constraint that is imposed on us when we would like to communicate over the internet is the port number. When viewing the allocated port list [Iana1] we find that the port numbers for dynamic or private ports is in the range 49152-65535. Therefore, in server we will allocate the main listening port in this range to avoid conflicts with other programs. The other listening ports will be dynamically chosen by entering a value of 0 for the port. When 0 is sent to TcpListener the system chooses a port for us.

3.9 Future communication over the internet

It is mentioned in the project description that we must address future communication over the internet. Throughout the critical issues I have tried to address these issues by discussing latency, locking, security and port number allocation. In addition to these concerns we will be using Webservices when working over the internet rather than sockets.

3.10 Server-side client handling

In the server there exists a complex interaction when handling all the various clients connected to it. I will call this complex interaction server-side client handling.

The basic sequence of operations that happen on the server is a client comes in and a new socket is created for the client. The handle to this new socket must be stored somewhere in the server. I have divided the server into a main lobby and any number of rooms. In each of these classes there will exist a list of sockets that exist in the room. What is required for implementation is a list of the delegates that will be used in order to transfer the socket pointer from one room to another.

The following delegates will be used:

Delegate Name	Description
New_client	Called by the listener when a new client enters the system
Client_verified	Called when the client's password is validated
New_user	Called when the client wishes to make a new username
Create_room	Called when the client would like to create a room
Enter_room	Called when the client would like to enter a room
Leave_room	Called when the client would like to leave a room
Delete_client	Called when the client would like to disconnect from the server

Table 3.10.1 – Delegates to be used in server-side client handling

From my quick prototype code, I think these delegates cover all the possible cases. However, this list may grow as implementation proceeds.

3.11 Viewing Synchronization

There exists a problem where the viewer joins the room after the editing has begun. What will happen with a simple architecture is that the viewer client will simply receive the stored file. What is needed is for the viewer client to receive the stored file and the updates that have not been saved yet.

To solve this critical issue we will keep a queue of the messages that have been sent since the diagram has been opened. Upon client connect the stored file and the update queue will be sent to the user. This method will reduce the overhead of saving a file at every message, thus making the server faster.

3.12 File transfer

In order to achieve file transfer over our communications link we will have to do a number of things. First we will send a file begin message (Appendix A) telling the file name, and the number of payloads in the file. One payload will be defined as 1024 bytes. The last payload will be less than 1024 bytes. After the file begin message is sent we will proceed to send each payload in a message as shown in Appendix A. Lastly we will send the checksum of the file.

The checksum will be calculated by have a counter that is reset to zero on the file begin message. During each payload the counter is incremented by a specified amount based on the contents of the payload. If the counter exceeds the `int.MaxValue` value it will simply overflow to become `int.MinValue`. No exceptions are thrown in C# so this method will work. Since each system will have the same value for `int.MaxValue` the checksum should be calculated the same on each machine. To be sure that all compilations of our code use a 32 bit integer for this variable we will specify `Int32`.

In addition to chunking the file into payload sections, we also must convert each section into a Base64 encoding. This is to be sure that all characters can be transmitted over our text stream. If we did not convert to Base64 there may be special control characters in a binary file that would halt the transmission of the system.

4. Partitions

The partitions of a Software Architecture are the modules and objects that make up the system [Fawcett1]. The Remote UML Diagrammer is divided into Client Partitions and Server Partitions. I will discuss each separately.

4.1 Client Partitions

The client is divided into 11 partitions or modules. I have included a figure to display these modules (figure 4.1.1). These modules are Executive, User Interface, UML_Client,

Communications, Sender, Receiver, Blocking Queue, Storage, Polygon, Connector, and Geometry.

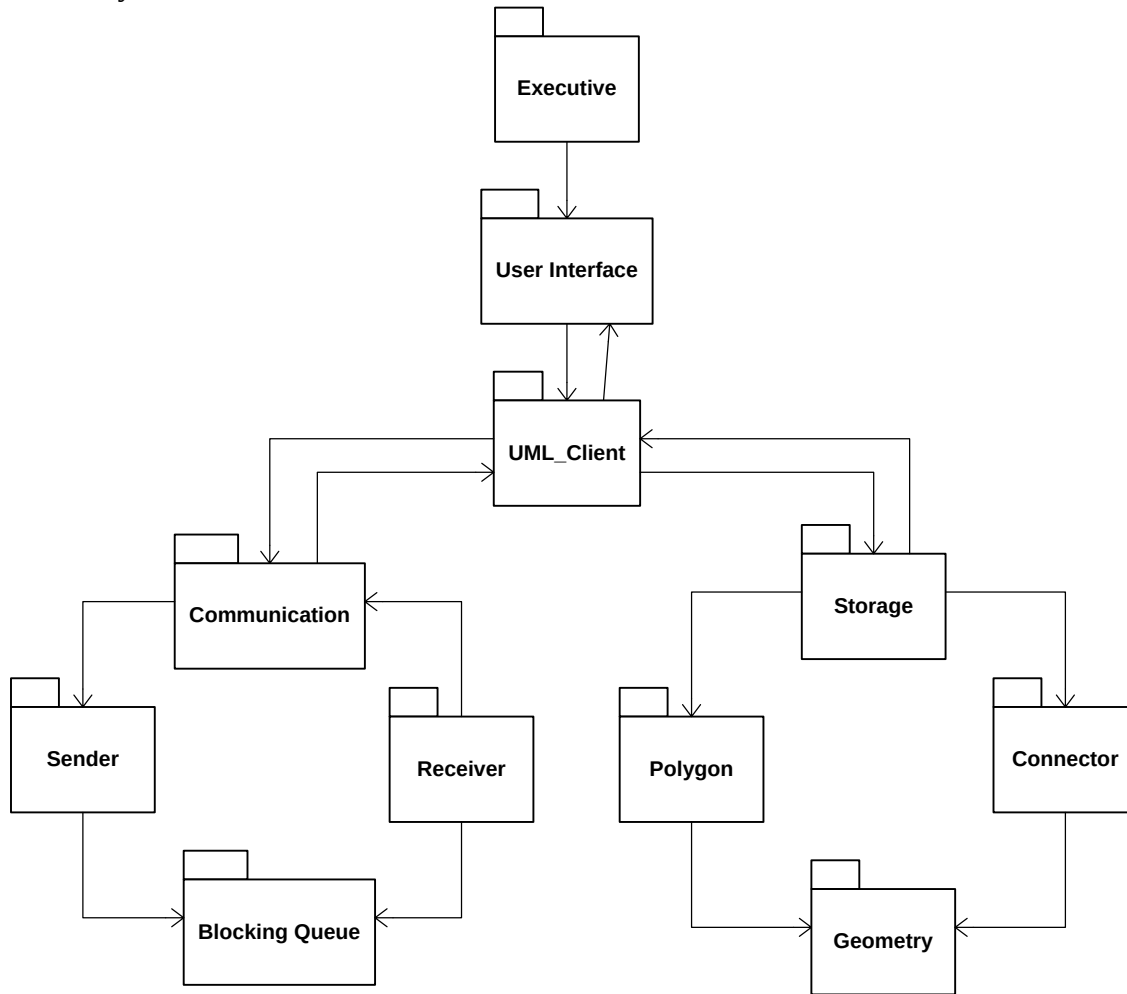


Figure 4.1.1 – Client Partitions

Shown above are the client partitions. I have already discussed Storage, Polygon, Connector and Geometry in project 1 so I will not discuss them again. They will be nearly the same in project 4 as they were in project 2. There may be a slight change to handle multiple clients panning the display and looking at different parts of the diagram at the same time.

The executive module is the main module. It contains the user interface module. The program main is located in the executive module. The executive module will interact with the user interface module, draw the main dialog box and allow the user to manipulate the program. The executive module is a very small module.

The user interface module will consist of all the classes that derive from Form. There will be three main classes in the client; they will be login, main_lobby and editor. Much of the code will be reused for the editor, but there will be new code and resources for the login and main_lobby classes. As discussed in section 2.2.1, we will be utilizing the tree

view for the display of data. Other than that the user interface will consist of buttons, textboxes, listviews and labels.

The UML_Client module will be a module that separates functionality from the user interface module. It will consist of one communications object and one storage object. Messages will be sent to and from the communications object from and to the storage object.

The communications module will be a module that encodes and decodes messages and also sends and receives them. I anticipate that this module will consist of several senders and receivers grouped together in classes. This class will use delegates to get data from the receiver and also to send data to the UML_Client module, which in turn sends data to the storage module.

The sender module will consist of a socket, a sending thread and a queue. Messages will be sent to the sender module with the PostMessage function from the communications module, which gets them from the UML_Client module. Messages will start to be sent out by enqueueing the message in the send queue and then later passed out of the socket inside the sending thread. The basis of this module will be from the handout code.

The receiver module will consist of a listener, a client handler, a receiver thread and a queue. Messages will initially come in to the receiver thread and placed in the queue. A delegate will be called when a complete message has been received and then the client handler takes the message from the queue. Lastly, the receiver module will send the message up to the communications module (using a delegate) to be parsed. The basis of this module will be from the handout code.

Lastly, the blocking queue is a simple module that implements a templated queue which has blocking properties. Much of the code from this module will be from the handout code.

4.2 Server Partitions

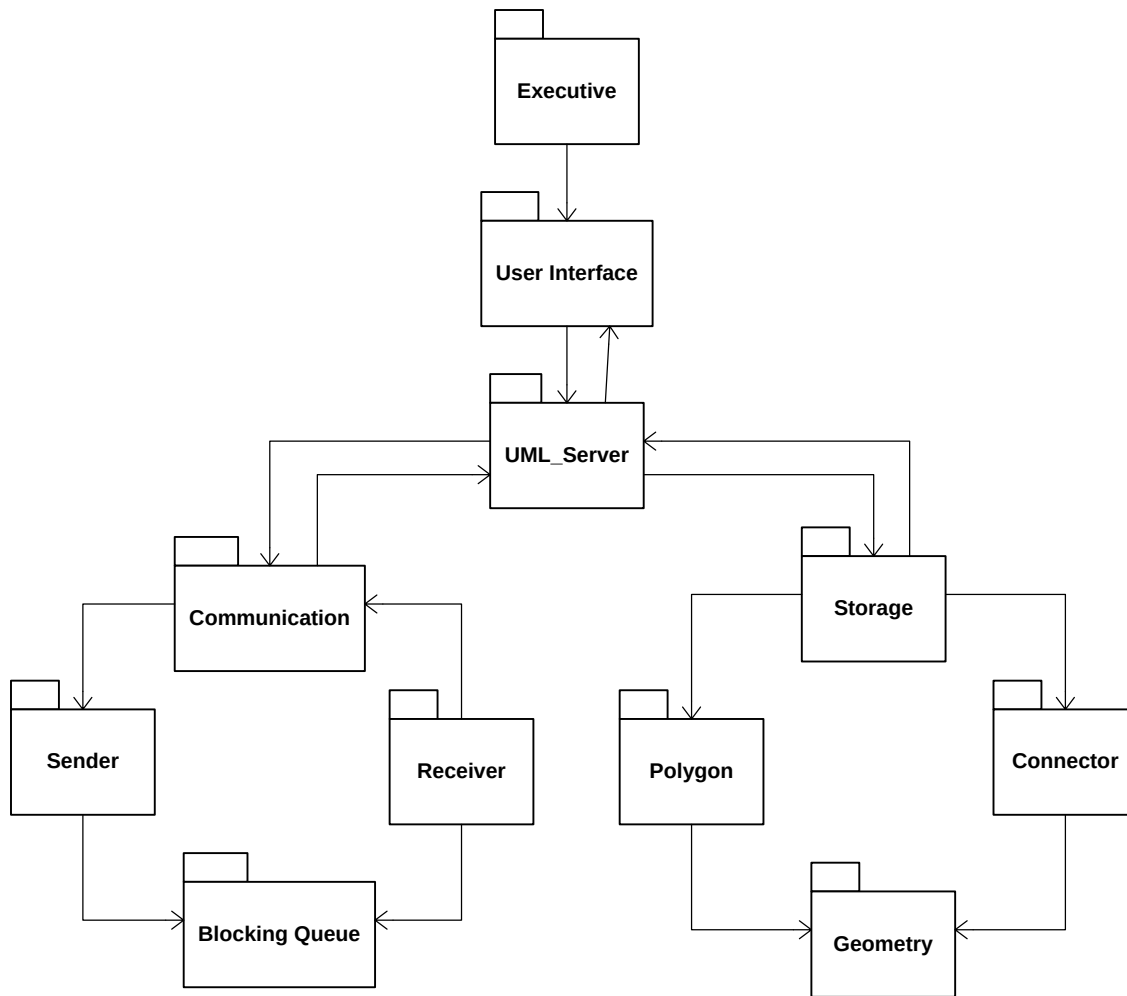


Figure 4.2.1 – Server Partitions

The server partitions are very similar to the client partitions. In this section I will mention what is different from the client.

The main thing that is noticeably different is there is a `UML_Server` module rather than a `UML_Client` module. Note we still require `Polygon`, `Connector` and `Geometry` in the server since it will be using the same data structures when saving files. The other main difference is that communication will be specifically tailored for the server application. Since we will try to share all the other libraries, `UML_Server` and `Communication` will be the major difference between the client and the server.

The `UML_Server` module is a complex module that handles the login of clients, the main lobby, and the rooms. The login of clients consists of verifying a password against a previously stored password using MD5 encryption. If the login is successful, the `UML_Server` module adds the `ClientConnection` object to the `main_lobby_clients` list. This list is held in the `UML_Server` module allowing for chat messages to be broadcasted to everyone in the main lobby.

All the features of the main lobby are implemented in the UML_Server module. These features are file upload/download/delete available files request and chat. There will be functions to handle all of these features. The UML_Server will also handle setting up viewing and editing clients. Once a ClientConnection object messages saying that it would like to become a viewing or editing client several steps are followed. First, a room is created (if none is open) and a new set of lists for broadcasting chat is activated. Second, the client is told what port to make a direct connection to in order to achieve peer to peer communication. Third, the two sockets are connected between the client and the server to pass messages.

Inside a room of the UML_Server there will be a queue that sends messages to all the viewing clients and also a queue that stores all the file changes since the last save (changes queue). Upon entering the room a client will receive the xml file and also messages from the changes queue using the peek method. Upon save the server will remove from the changes queue using the dequeue method.

Also inside the room there will be handles to all the members of the room. These handles will be used to broadcast messages to all members.

The remaining modules that have not been mentioned are identical to the client.

5. Conclusion

In conclusion, the Remote UML Diagrammer will allow a user to remotely create, edit and view UML Diagrams. It will also allow him or her to upload and download UML diagrams from and to a server. In addition the user can send chat messages and transmit files to other users for collaboration purposes. This document laid out the architecture for the Remote UML Diagrammer that can be used in creating a detailed design and then implementation.

6. References

Fawcett1 -

[http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/presentations/SoftwareArchitecture.ppt#257,5,Architectural Issues](http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/presentations/SoftwareArchitecture.ppt#257,5,Architectural%20Issues)

Iana1 -

<http://www.iana.org/assignments/port-numbers>

Wikipedia1 -

[http://en.wikipedia.org/wiki/Latency_\(engineering\)](http://en.wikipedia.org/wiki/Latency_(engineering))

Wikipedia2 -

<http://en.wikipedia.org/wiki/Security>

Wikipedia3 -

http://en.wikipedia.org/wiki/Secure_Shell

7. Appendix A. Messages used in Remote UML Diagrammer

The messages in the Remote UML Diagrammer Client will be in the following form:

```
<msg>command,parameter1, parameter2, parameterN</msg>
```

The client will be in any one of three states at one time. These states are login, main lobby and editing/viewing. I will list the messages for each of these states separately.

Login Send:

Command	Command Code	Parameters
Login	login	Local ip, listening port, Username, password
New user	new_user	Local ip, listening port, Username, password

Login Receive:

Command	Command Code	Parameters
Login Reply	login_reply	Login status (ok, notok)
New user reply	new_user_reply	New user status (ok, notok)

Main Lobby Send:

Command	Command Code	Parameters
Get files	Gf	-
Get users	gu	-
Upload Begin	up_begin	Filename, Number of payloads
Upload Payload	up_payload	Filename, Payload number, payload
Upload End	up_end	Filename, checksum
Download	down	Filename
Delete	del	Filename
View	view	Filename
Edit	edit	Filename
Chat	chat	Text
Quit	quit	-

Main Lobby Receive:

Command	Command Code	Parameters
Get files reply	gf_reply	Filename, editing users, viewing users
Get users reply	gu_reply	users
Upload Reply	up_reply	Status (ok, notok)
Download Begin	down_begin	Filename, Number of payloads
Download Payload	down_payload	Filename, Payload number, payload
Download End	down_end	Filename, checksum
Delete Reply	del_reply	Status (ok, notok)
View Reply	view_reply	Status (ok, notok)

Edit Reply	edit_reply	Status (ok, notok), peer ip, peer port
Chat Reply	chat_reply	Status (ok, notok)
Peer info	peer_info	Ip, port

Editing/Viewing Send:

Command	Command Code	Parameters
Connect	connect	Senders listening ip, senders listening port
Add polygon	ap	Poly_id, type, location
Delete polygon	dp	Poly_id
Add connector	ac	Conn_id, poly_id1, poly_id2, type
Delete connector	dc	Conn_id
Rename polygon	rp	Poly_id, new_text
Move polygon	mp	Poly_id, new_location_x, new_location_y
Upload Begin	up_begin	Filename, Number of payloads
Upload Payload	up_payload	Filename, Payload number, payload
Upload End	up_end	Filename, checksum
Get users	gu	-
Chat	chat	Text
Send token	token	-
Save file	save	Filename
Quit	quit	-

Editing/Viewing Receive:

Command	Command Code	Parameters
Reset file	rf	-
Add polygon	ap	Poly_id, type, location
Delete polygon	dp	Poly_id
Add connector	ac	Conn_id, poly_id1, poly_id2, type
Delete connector	dc	Conn_id
Rename polygon	rp	Poly_id, new_text
Move polygon	mp	Poly_id, new_location_x, new_location_y
Download Begin	down_begin	Filename, Number of payloads
Download Payload	down_payload	Filename, Payload number, payload
Download End	down_end	Filename, checksum
Get users reply	gu_reply	Editing users, viewing users
Chat	chat	Text
Send token	token	-

8. Appendix B. MD5 encryption

Listed below is some sample code to support MD5 encryption for passwords on a .Net 2.0 machine.

```
using System;
using System.Security.Cryptography;
using System.Text;

class Example
{
    // Hash an input string and return the hash as
    // a 32 character hexadecimal string.
    static string getMd5Hash(string input)
    {
        // Create a new instance of the MD5CryptoServiceProvider object.
        MD5CryptoServiceProvider md5Hasher = new MD5CryptoServiceProvider();

        // Convert the input string to a byte array and compute the hash.
        byte[] data = md5Hasher.ComputeHash(Encoding.Default.GetBytes(input));

        // Create a new StringBuilder to collect the bytes
        // and create a string.
        StringBuilder sBuilder = new StringBuilder();

        // Loop through each byte of the hashed data
        // and format each one as a hexadecimal string.
        for (int i = 0; i < data.Length; i++)
        {
            sBuilder.Append(data[i].ToString("x2"));
        }

        // Return the hexadecimal string.
        return sBuilder.ToString();
    }

    // Verify a hash against a string.
    static bool verifyMd5Hash(string input, string hash)
    {
        // Hash the input.
        string hashOfInput = getMd5Hash(input);

        // Create a StringComparer to compare the hashes.
        StringComparer comparer = StringComparer.OrdinalIgnoreCase;

        if (0 == comparer.Compare(hashOfInput, hash))
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    static void Main()
    {
        string source = "Hello World!";
        string hash = getMd5Hash(source);
        Console.WriteLine("The MD5 hash of " + source + " is: " + hash + ".");
        Console.WriteLine("Verifying the hash...");

        if (verifyMd5Hash(source, hash))
        {
            Console.WriteLine("The hashes are the same.");
        }
    }
}
```

```
    }  
    else  
    {  
        Console.WriteLine("The hashes are not same.");  
    }  
}  
}  
// This code example produces the following output:  
//  
// The MD5 hash of Hello World! is: ed076287532e86365e841e92bfc50d8c.  
// Verifying the hash...  
// The hashes are the same.
```