Instant Messenger type tool for Virtual Software Development Teams

Jonathan Bailey

BSc Computing (Hons.)

Abstract

Copyright

Acknowledgements

Sam Egerton

Neil McCullen

Andrew Davy

Geoffrey Tasker

Table of Contents

1. Introduction 7
1.1 Problem Identification 7
1.2 Proposed Solutions 7
1.2.1 Off The Shelf Applications 7
1.2.2 Bespoke Software 8
2 Project Plan 9
2.1 Schedule 9
3 Research 12
3.1 Questionnaire .... **Error! Bookmark not defined.**
3.2 Literature Review 12
3.3 Existing Products 14
3.3.1 Windows Live Messenger 14
3.3.2 Yahoo Instant Messenger 15
3.3.3 Trillian 16
4 Refined Problem Definition 17
5 Requirements 18
5.1 Functional Requirements 18
5.2 Non-Functional Requirements 19
6 Design 19
6.1 Proof-of-Principle Prototype 19
6.2 Initial design 21
6.2.1 Graphical User Interface design 21
6.2.2 Functionality design 22
6.2.2.1 Server 22
6.2.2.2 Client 23
6.3 Final design 25
6.3.1 Graphical User Interface design 25
6.3.2 Functional design 27
7 Development and Implementation issues 30
7.1 Threading 30
7.2 Sharing variables 31
7.3 Streaming different objects over a network connection 31

7.3.1 “Serializable” methods........ 31
7.3.2 Streams........ 31
8 Testing........ 32
9 Evaluation........ 37
10 References........ 38
11 Bibliography........ 39

1. Introduction

1.1 Problem Identification

Globally Distributed Virtual Teams encounter many problems and delays because there is often very little or no framework in place to facilitate the informal communication between co-workers (Casey, V. 2006). This can mean that there is none of the collaboration and co-operation that will often take place in an office environment between localised team members. In the case of Globally Distributed Teams, it is often vital to formally arrange communication due to differences in Time Zones, working hours and individual availability. This can lead to lengthy delays in communication between team members, or stop all but the most vital communications due to the complex nature of organising the meetings. This formal organisation also removes the casual aspect. Team members may not discuss implementation or design issues with their team members if they have to go through a lengthy and tedious process of communication.

1.2 Proposed Solutions

1.2.1 Off The Shelf Applications

There is a wide range of Instant Messenger software applications that are available to anyone free of charge. These generic applications will allow the different members of the teams to communicate via a synchronous framework. Many of these applications do not use any of the encryption methods that are widely available. This means that all of the messages sent and received are in plain text format. This could be potentially damaging to any high security project. The conversations held by team members will also be logged on servers that are not under any control of the company to which the employees belong. This can introduce a risk element that many competitive companies will not be prepared to expose themselves to.

These software applications will also allow team members to communicate with people that are either outside the company, or outside the team, such as family and friends. This can possibly lead to loss of productivity amongst the team.

Instant messenger tools such as Windows Live Messenger (Fig1) provide a large range of different functionality that may be suitable for business use. This includes simple file sharing, status changing and group conversation.

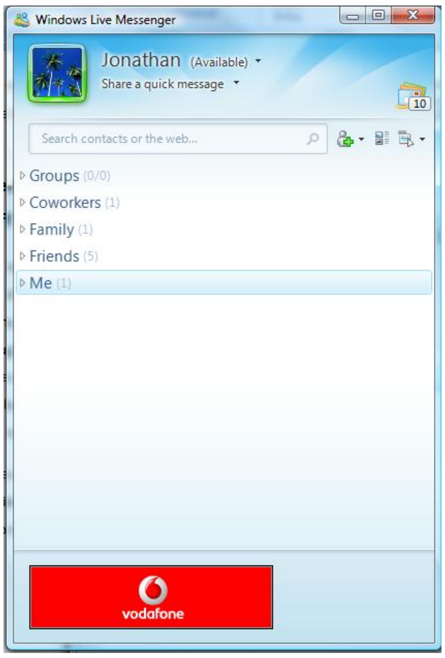


Fig 1: Microsoft's Live Messenger. Build 14.0.8(Beta)

1.2.2 Bespoke Software

Develop a bespoke Instant Messenger application that is private to the company or companies involved in development projects and have Globally Distributed Teams.

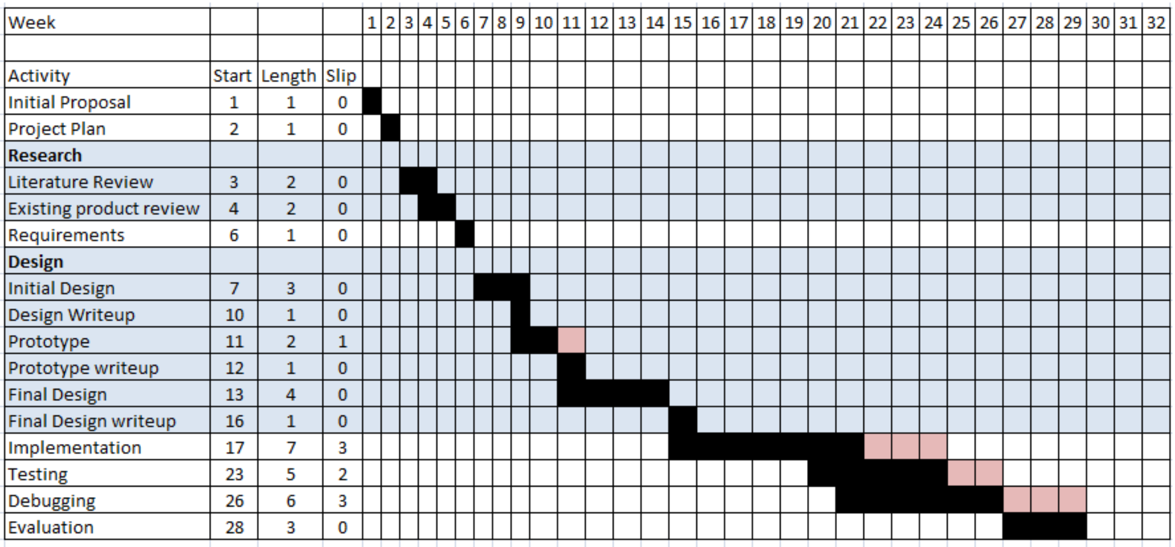
The aim of the Instant Messenger tool is to provide a mechanism for fast, reliable, secure and synchronous informal communication between different members and locations involved in the team. This will help development procedures as well as build relationships between different team members.

Bespoke software will allow the companies involved to exercise some form of control over the ability of their employees to communicate to people who are not a part of the company. This problem of having employees spending time with personal chat and interacting with people that are not involved with the company is a very real concern for various companies. Research has shown that several companies deny their employees access to Instant Messenger tool simply because they are concerned about losing productivity because employees are spending too much time chatting with friends. This will be avoided with a bespoke application, because it will be only available to company employees, contractors, and anyone else that Senior Management gives access to the server and the client.

2 Project Plan

This project is intended to be carried out over a thirty two week period. With a project such as this, with goals to design, build and test a high quality piece of software that can be used in an industrial environment. As well as providing a formal report and evaluation on the processes involved in the project, it is important to ensure that a plan is laid out and followed as closely as possible to ensure that each aspect of the project receives the attention and time that it requires. This plan will also provide a basis for evaluation, considering if the project was completed on schedule or if the time period was too short for all of the work to be completed.

2.1 Schedule



Activity	Start	Length	Slip
Initial Proposal	1	1	0
Project Plan	2	1	0
Research			
Literature Review	3	2	0
Existing product review	4	2	0
Requirements	6	1	0
Design			
Initial Design	7	3	0
Design Writeup	10	1	0
Prototype	11	2	1
Prototype writeup	12	1	0
Final Design	13	4	0
Final Design writeup	16	1	0
Implementation	17	7	3
Testing	23	5	2
Debugging	26	6	3
Evaluation	28	3	0

Fig2:- Project Plan Gantt chart

Figure 2 shows the Gantt chart for this project. The chat lays out fifteen different activities that need to be carried out during the life cycle of this project. It also outlines the order and time scale for each of these activities.

Weeks one and two are set aside to develop an initial proposal for the project, and to outline an initial plan for the 32 week period. This is the beginning of the project, and should not take any longer than the time set aside, which is why there, has been no “slip” period allowed for these two activities.

The research portion of the project is going to include a literature review and a review of the functionality of existing Instant Messenger applications that may be considered suitable for a company to use in the context of communicating between global teams. Although these two activities are unlikely to run beyond their designated time periods, they have both been given two weeks with a week of overlap between them. This is to give enough time to gain a full understanding of the problem domain, and to gather as much information about the existing products as possible. At this point in the project, Research is taking the priority. This is because excellent research will aid with defining the requirements of the application, as well as providing a good understanding of how other products work. This knowledge will provide a good resource when designing and implementing the application further into the project.

The Requirements will then be defined once the research into the area has been completed. This is to provide key milestones and goals for the application to fulfil. Without these it would be difficult to define if the project was a success or not during the evaluation.

The second, larger, section of Design will be started the week that the requirements have been defined. The initial design phase will be used to design a simple chat room client and server application. This is intended to act as a foundation for the finished application. This initial design phase is scheduled to take up to three weeks because it is here that I expect to encounter any problems related to network communication and client – server interactions and their models of implementation.

The prototype will be developed to ensure that the fundamental concept of the Instant Messenger application is achievable. This prototype will be built in accordance to the initial design specification, implementing a chat room client and server, with basic message sending and receiving. The prototype has been given a week of slip to allow for any unforeseen implementation issues that may arise during the build. The documentation for the prototype will be completed during this week of slip, either run in parallel to the prototype implementation if it over runs or completely separate if the prototype is completed on schedule with no issues.

The final design has been scheduled to take up to four weeks to be completed to a satisfactory level. This final design is to include not only the functional design, consisting of Object relationships, database actions and client – server interaction, but also the Graphical User Interface design for the client and associated interfaces such as conversation screens.

The final design write up has been scheduled to be run in parallel to the beginning stages of the implementation of the final application. This was done because the implementation is expected to need as much time as can be spared for it and the write up of the final design is not expected to take a full week to complete.

The implementation of the final solution has been scheduled to take seven weeks with three weeks to allow for any overrun during implementation. However, after the total ten weeks are over if the solution is still not completed, a decision will have to be made on if the application should be signed off in its current state, or if further development can be carried out without negatively impacting the testing and debugging of the solution. At this point in the project, there is scope for three different activities to be run in parallel. This is because during the implementation it is expected that a certain amount of testing is carried out to ensure that core functionality of the solution is working correctly before moving on. It is also expected that this testing of the core functionality will result in some bugs or unexpected behaviour being uncovered, and this leading into the debugging being started before the full implementation of the solution has been completed. This was taken into account when scheduling the project, as these three activities are closely tied to each other, and occasionally dependent upon each other.

The final evaluation phase is expected to take three weeks, with the possibility of the debugging running parallel.

The remaining three weeks are to allow for any final touches, proof reading, adaptation and finalising of both the application and the final report.

It is important for a project to be planned out in detail to ensure clear milestones are set out. The plan will also provide a good structure and flow from one task to another throughout the projects life span. These reasons are part of a list of reasons why a project fails as outlined by Joseph Weiss (Weiss, J.W & Wysocki, R.K 1992).

3 Research

The purpose of this research is to identify the viability of using Instant Messenger applications to aid communication between team members. As well as key aspects of the tools that are considered vital to such a team. This research will help to outline a list of Requirements for a bespoke system, as well as identify any suitable Off-The-Shelf applications that could be considered to fulfil the role of aiding communication within the team.

3.1 Literature Review

The purpose of this review is to outline the various arguments for and against the use of an Instant Messaging tool within the global team environment. The review will take into account different views supported by research papers in the area, and outline, if possible, a clear overall decision on whether the user of such tools is generally considered a good idea or if companies are too concerned with the risks associated with allowing their employees access to such tools. The review will also attempt to outline any particular aspects and functionality that is considered highly advantageous in an Instant Messaging tool when it is being used in this environment.

There is a wide range of research available on the topic of Global Teams, and therefore the difficulties experienced by these teams in communication and collaboration on a project. One such paper (*Canfora. G, 2003*) outlines that distance is almost certain to have a negative impact on any communication methods. The paper goes on to say that the problems with communication could lead to problems such as missing reuse of code or design opportunities within the team because information is not being disseminated fully to all team members. This is confirming similar research into the area that often it is communication that suffers when teams are spread over different time zones and geographical areas (*Kiel, L. 2003 & Herbsleb, J. 2005*). The loss of informal communication between team members has been highlighted as a specific problem for development in the global team environment. Research has suggested that around seventy-five minutes a day is spent developing interpersonal relationships between colleagues in a localised team (*Perry, D.E. 1994*). This kind of interaction between team members is not as easy to cultivate between team members that are separated geographically. The "corridor talk" is often vital to keep team members up to date with the latest updates on the project, to raise issues informally, and to provide help and assistance (*Herbsleb, J.D. Moitra, D, 2001*). When this opportunity for informal communication is missing, there is the possibility that some team members are not kept up to date with the current state of the project, this could allow things like small bugs to be missed due to a team members not sharing information as much as they would throughout a localised team. This inability to cultivate relationships and promote collaboration between different team sites is an opportunity for instant messaging to be implemented effectively. The informal and synchronous nature of instant messaging would simulate a face to face conversation with that team member, and allow fast responses to queries. This would also help reduce the dislike to communicating between different sites with asynchronous methods such as email as there would normally be no time delay between responses (*Herbsleb J.D et al* 2001). This delay between responses is a widely acknowledged problem with asynchronous communication techniques and yet another way in which communication between different sites is slowed down. Espinosa and Carmel (Espinosa, J.A, E. Carmel 2004) argue that the cost of a virtual team's communication can be broken down into several different sections. These include pure communication costs, the cost of time spent writing the actual communication. Clarification costs, the necessary extra communication needed to clarify a misunderstanding etc. Re-work costs, the cost re-working a part of the project when a misunderstanding leads to incorrect work being done. They also argue that synchronous communication technologies reduce the pure communication costs, simply because they are cheaper in terms of time to make and send a communication, rather than a formal e-mail etc.

The argument for against using an instant messenger tool is also quite documented, with many companies stating that they are concerned that allowing team members to use certain software could result in misuse and abuse of that technology. For example (*Carmel, E. 1999 & Casey, V. 2004*) both argue that tools such as e-mail were being abused by team members copying their managers or team leaders into emails that they send or receive from other team members from a different location. This kind of behaviour will quickly cause animosity to grow between the different sites and could cause a company to avoid using less formal methods of communication, fearing that these too would be misused by team members.

Significant geographical boundaries would present difficulties with using synchronous technologies (*Turoff et al. 1993*) because of large variations in working hours and time zones. There may be only a small period of time where both locations are online at the same time. In this kind of environment it is unlikely that the introduction of an instant messenger tool would have a significant impact on the productivity of a team. Relying on asynchronous methods such as email would be one of the few viable options available in this type of situation. Another option would be to organise times when certain team members will work at the same time to allow synchronous communication. This would take time to set up, as it too would rely on emails to organise the shared work period. It also puts added strain onto the employees involved because they may end up working abnormal hours.

In conclusion, although there are some small issues with using an instant messenger tool to help aid communication between team members, such as the possibility of misuse, there is a far stronger case for it to be used in addition to other communication procedures that are in place such as email and video conferencing. The use of instant messaging could fill the gap of informal collaboration and information sharing that seems to be lacking in global teams. It may also help to speed up problem solving as team members can “get together” with each other or a whole group and quickly debug a problem, brain storm a solution or just build trust between team members that is often difficult with asynchronous methods through informal chat.

3.2 Existing Products

There is a huge range of different Instant Messenger products available for use, with some of them even being distributed built into the operating system of modern computers. In order to understand the full range, as well as their functionality and suitability to the problem, some of the most common ones will be outlined here.

3.2.1 Windows Live Messenger

This is possibly the most widely used and recognised instant messenger application available. The application its self is free to download, although there is usually a version installed with every Windows operating system. In order to use it a user simply needs to register a Windows Live account. This is the account that is used to access things like mySpace, hotmail, Windows Live Messenger and Xbox Live. This makes it the most easily available and versatile instant messaging tool for Windows users.

The tool its self provides a wide range of different functionality (*Windows Live Messenger – Features*). This includes simple drag and drop file sharing. Simply drag a file that you wish to share with a person onto an open conversation window with them and the file transfer will automatically commence once the recipient has given their consent to receiving the file. There is also a status changing function that will simply and quickly update a user’s status on every client that the user is added on. There are also options for setting time stamps on messages that have been received, as well as choosing to keep log files of all the conversations that take place at the client side. This would allow very easy reference to past conversations for team members. The application also allows for group conversations to be setup by use of an invite button on the conversation windows. There is also the possibility of having video and voice calls as well with this tool. Simply start a conversation with another person, and invite them to the call to initialise the video call.

All of this is very advantageous for a software team to be able to do. The tool would provide a good, well documented and supported method of communication for global teams, which is also available free of charge, well maintained and supported by Microsoft and available for download for a large number of websites as well as being distributed with the Windows operating system.

However, the application also provides other functionality that may not be suitable for a business environment. The main thing that may be abused in the work place is the "Nudge" system. This is a way of sending a message to a user that makes the conversation window shake on the screen, as well as popping on top of all other windows that the user may have open at the time. This function could be used maliciously between team members where there is a feeling of distrust between different geographical sites (*Casey. V, 2006)*. The application will also allow outside communication with people that are not a part of the team, or even a part of the companies involved in the team. This could lead to team members losing productivity because they are talking to friends and family during business hours via a method that it is not easily detectable that they are doing so. This application also uses a communication system that is not under the direct control of the company. The servers that are used for the application could be located anywhere on the planet. This introduces the issue of maintaining security. This lack of control could mean that team members are restricted in what they can say over the application, especially if they are working on a sensitive project. This would render the use of instant messaging almost pointless, as anything important would need to be sent via a more secure system.

3.2.2 Yahoo Instant Messenger

This is available in two formats. Firstly is the web based, which is a cut down version of the downloadable application. These are both similar application to Microsoft's Live Messenger. It provides a large array of functionality (*Features – Yahoo Messenger*). This includes the standard instant messaging both to individuals and to groups of people. This is achieved in a similar fashion to the Live Messenger application. Yahoo Messenger also provides file sharing although this is limited to 2 Gigabytes. The application also provides a simple mechanism to change the user's current status at the click of a button. This would be useful if a team member has a meeting or is busy. They can simply set their status to "Away" or "Meeting" etc to announce to other members that they are not currently available for chat. Although this does not block messages received. This would allow the user to answer any messages upon their return to the computer.

However, there are other features supplied with both versions that would not be suitable for a business environment. The most non suitable of these is the ability to play games with other people via the client. This could potentially lead to a drop in productivity in the team as members could spend time playing games on the clients rather than working. Another problem with this application is that it too is open to anyone to use. This means that the team members could be talking to friends and family during work hours. There are also options to change things like message fonts. This could provide difficulty for the recipients to read the messages sent, if the font chosen is particularly obscure or inappropriate.

Yahoo Instant Messenger provides similar functionality to Windows Live Messenger (*Features – Yahoo Messenger & Windows Live Messenger – Features*). It also has some of the same issues that make it unsuitable for use as a means for communication between members of a virtual team. These include an offsite server network that is not under the control of the company using it, the lack of control over who team members talk to during work hours, and the issue of these applications also providing connections to gaming sites via means of either the Windows Live ID, or the built in Yahoo Games.

3.2.3 Trillian

Trillian is different to the other applications that have been looked at so far because it provides a mechanism for access to all of the contacts of the other applications as well as access to chat rooms that are available on internet relay chat (*Cerulean Studios – Learn About Trillian*). Internet relay chat or IRC is a method of communicating by using synchronous text messages that are sent via "channels". These channels are servers that have been set up around the globe. This form of messaging was in use after 1988, before instant messaging evolved into the recreational activity that it is today (*Internet Relay Chat*). This is in addition to the functionality that they provide for the instant messaging applications. This application provides all the functionality that Live Messenger and Yahoo – Instant Messenger provide, such as video conferencing, group chats, status changing and message logging. This application also allows users to log into any existing msn or yahoo accounts that they may have for personal use. This contributes to the lack of control that a company can expect to be able to exercise over who their employees are talking to during the working day. As all of the accounts are accessible via the single client, it also becomes difficult to discern who they are talking to simply from observation.

The ability for team members to use this application to log into virtually any server that has IRC capabilities will be a risk that many companies will not be willing to take. It would open up hundreds of chat rooms to team members all from the same client. This ability to connect to any IRC server does mean however that the company could set up its own IRC server in house, set up an IRC channel with restricted access and allow its team members to use that to communicate. For many companies this may be a good idea, but many others may be more concerned with the huge number of available chat rooms that their employees would have access to at any time during the working day.

In conclusion, existing products do supply the functionality that companies are looking for in an instant messenger, but this functionality is included alongside addition functionality that may hinder a company adopting instant messaging into its communications framework. These are things such as access to gaming websites via the client, or access to personal contacts and chat rooms. In order to solve this problem of unwanted functionality it would be best for a bespoke system to be developed with the aim of using it as a means of communication between team members in a globally distributed team.

4 Refined Problem Definition

After carrying out some research into the area of Global teams, their advantages and their problem areas it is possible to clearly outline several problem aspects within the team environment that could potentially be solved with the correct implementation of an instant messaging tool.

It has been made clear by the research carried out in a number of different papers that as the distance between the team members increases, the communication infrastructure becomes more inadequate (*Canfora. G, 2003*). This lack in informal communication between distributed team members when compared to the amount of such communication that

takes place in a localised team (*Herbsleb, J.D. Moitra, D, 2001*) often has a detrimental effect on the productivity of the team. This could be due to the delay in responses when tools such as email are used (*Herbsleb J.D et al* 2001) as well as misinterpretation of messages at different sites (Espinosa, J.A, E. Carmel 2004).
The lack of applications that fit in with the needs of the global team and the companies that employ the techniques allows scope for there to be a bespoke tool developed in order to fulfil the needs of both the global team and the companies that use them. To help provide the needed communication infrastructure an instant messenger tool will be built that is specifically tailored to suit the needs of a globally distributed team.

5 Requirements

By researching and gaining an understanding of the problem domain as well as discussing the problem with the client, it is possible to draw up a list of requirements for an Instant Messaging tool. These requirements can then be used to either identify a product that is already available, or to help develop a bespoke tool that will meet the requirements of Global Teams.

5.1 Core Requirements

The research into this topic has highlighted several key functions that are considered to be high priority to employers when it comes to deciding what Instant Messenger tool to use. These are as follows:-

- Stable Instant Messenger Client – Server application.
- Communication logging on the main server.
- Control over the main server, onsite server.
- Individual conversation.
- Ability to send files to contacts.
- Ability to have group conversations.

5.2 Additional Requirements

The research has also highlighted several functions that whilst may not be central to the operation of the tool, would greatly enhance its efficiency and effectiveness in the global team environment. These requirements are as follows:-

- Ability to send offline messages to contacts.
- Ability to set current status.
- Optional time stamps on messages sent and received.
- User profile thumbnails.
- Optional conversation logs at the client.

6 Design

6.1 Proof-of-Principle Prototype

A prototype was developed early into the development cycle. The aim of this prototype was to test network communication via a private wired Local Area Network, as well as to help understand the complexity of the task of developing an instant messaging application using the selected technologies, for instance Java.

This prototype, whilst extremely basic, provided an insight into issues that might be faced when developing the main application, and also provided an opportunity to test out different methods for overcoming them before they were encountered during the development of the final application.

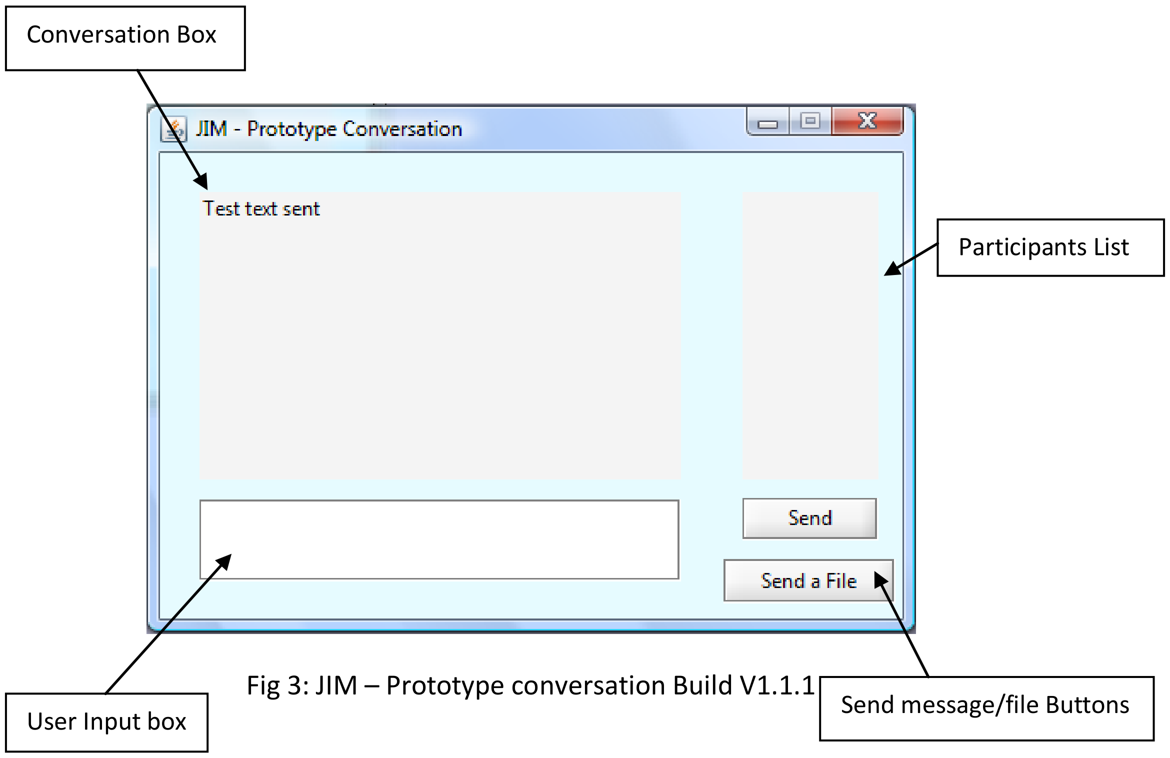


Fig 3: JIM – Prototype conversation Build V1.1.1

The conversation window (Fig3) has a simple java text area for the user input at the bottom of the pane. This field is read and sent to the server, which simply echo's the message back to the client, and to every other client that is currently connected to the server. This is in effect, an anonymous chat room. Any messages that have been sent are displayed in the larger text area. This area will automatically scroll downwards as the conversation develops, and also word warps on the horizontal axis, so that words are not cut in half.

The prototype was also used to develop the desired layout of the client interface. In this case, the participants list and file sending button have no functionality coded for them, but it was felt that they needed to be included to give an appropriate feel for the final layout of the conversation window and allow any changes to be made should the layout be unsatisfactory.

The prototype was also used to test out setting the window to match the environment it is running on by using java's "set-look-and-feel" methods. This is particularly important, as there is the possibility that the application will be run on operating systems other than Windows. In order to test the look and feel methods fully, the prototype was also run on an Apple MacBook

6.2 Initial design

Good coding practise, as well as object orientation dictate that the interface and the actual functionality of an application should be kept separate (McConnel, S. 1993, p71 – p115). Due to this, the interface and the functionality will be outlined separately.

6.2.1 Graphical User Interface design

The initial interface design for the login screen (Appendix A1) shows a simple Username label and text field alongside each other, with a similar layout for the password. The sign in button is below the fields to try and lead the user into filling out the details in a sequential order from top to bottom. There is intention to use a pale colour rather than white, although this has yet to be decided upon. This sign in form will also prompt the user should they enter the incorrect information into the fields with a simple but clear popup (Appendix A2) that will inform the user that the user name or password is incorrect. Once the user clicks on the "Ok" button to signal that they have read and understood the message, the username and password fields are automatically reset to being blank (*Stone, D. et al p212*).

The main menu screen (Appendix A3) that the user will use to start conversations with other team members has been designed to be as simple as possible. The user list is a plain selectable list in the centre of the panel. To start a conversation with another team member that is currently online, the user will simple have to double click on their name on the user list. This will then open a conversation with this person. The user can also single click and then right click on a user name, and a simple popup will appear with that contacts details such as their nick name, given name, location etc (Appendix A5). This user interface was designed to keep it as simple as possible. It would have been dysfunctional to have the main screen having too many options and buttons on it for a user to set or unset. The idea is to have the main screen as a simple tool that a user can glance at and instantly see who is online at that moment in time.

The conversation window (Appendix A4) is where the user will set the options for each conversation independently of the others that they might be having a that time. These are all placed in a simple but discrete options menu at the top left of the window to keep the conversation window as simple as possible without effecting usability. The main conversation panel will update as soon as a new message is received or sent with the option of having time stamps enabled or not. The sending a file button is next to the send a message button to allow for easy sharing of files and folders. There is also a listed non clickable pane to the right of the window that will list the participants of a conversation.

The server has not been given a graphical user interface as it is intended to be run as a service on a dedicated server machine. Having a specific user interface for it would potentially congest the server that it is running on. The server does however have the ability to produce system messages of any activity, and always keeps logs to allow a system administrator to look back over them to diagnose any problems that may have arisen during its operation.

6.2.2 Functionality design

6.2.2.1 Interaction between Server and Clients

The final design for the behaviour of the server and clients is that each client is connected directly to the server and not directly between client and client.

6.2.2.2 Server

The server is designed to use the Java Socket package as its connection method. This Socket uses Transmission Control Protocol to send and receive messages over a network. Because of the way that the java Socket package is built, a lot of the more complex networking is done by the java virtual machine without needing any input from the programmer (*Java Socket API, 2006)*. This is one of the two main reasons why the package was chosen. It allows for very simple and easy to use network connections to be established quickly and simply. TCP was chosen for the method of communication because it provides reliable communications. This is because TCP detects any lost packets due to network congestion and requests that they are resent. This means that the information is guaranteed to get to the destination. TCP also to some extent handles network congestion issues as well as re-arranging old packets (Forouzian, B, A., 2007).

The server is going to wait for a connection to be made by a client attempting to connect on a specified socket. Once a connection is established the server will receive the user’s login details, the user name and the password. The server will then access the SQL database that acts as a storage medium for all of the users that are allowed access to the instant messaging network. The server is to run an SQL query, looking through the database for a user with the same username and password as the one that was provided. Upon finding a user, the server will allow the user to remain connected and add that user to the list of users that are currently online.

Should the database query return an error, an error message will be returned to the client and the connection will be refused.

An accepted connection request from a client will be allocated its own thread that will be dedicated to listening for any messages that that client sends to the server. This dedicated listening thread will also call any relevant message handlers depending on the nature of the messages received.

A thread is simply a split in the execution of a program. It allows a program to execute two different pathways of code seemingly in a synchronous manner (*Java Thread API, 2006*).

The full implementation of the server is available in class diagram form in Appendix B1.

6.2.2.3 Client

The client has been designed with several different screens. Because each of the screens has significantly different functionality associated with them, they have been designed to be split into separate classes, and use java's object orientation to encapsulate the different fields, and pass information via the appropriate getters and setters.

The login screen contains the action listener for the "Sign in" button. This is an inner class which performs method calls when it receives an action event from the "Sign in" button. These method calls allow for the client to connect to the server, setup input and output streams and then send the login details for this user over the output stream.

An inner class was used here instead of removing the button listener and placing it into its own class because this method increases encapsulation, and also provides more readable code (*Java Nested Class, 2006*).

The method to connect to the server creates a socket with a hard coded IP address and port number. This IP address is the address of the server, and the port number on which the server is listening. Once a connection has been established, a global variable is populated with the socket connection for future reference by other classes and methods.

The stream setup method will handle both the input and output stream setup procedures, the output and input stream objects are populated with the connections input and output streams, these are again used to populate global variables that will contain the streams for this client.

```
    this.receiveObject = new ObjectInputStream(this.connection.getInputStream());

    this.sendObject = new ObjectOutputStream(this.connection.getOutputStream());
```

The third method to be called when the action listener receives an action event is the method that sends the login details to the server. This is done by wrapping the username and password along with the clients host into an Arraylist and sending that via the output stream that was created earlier to the server for verification.

The action listener will also create a new listener thread. This is a thread that is designed to listen for any input from the server and handle it appropriately. This will involve calling the required methods depending on the type of the message received. All of the methods needed are encapsulated in the login screen class.

```
Thread t = new Thread(new IncomingReader(receiveObject));
t.start();
```

The remaining methods are all for handling the various inputs from the client, and populating and updating the main menu screen that is created after the user’s details have been verified.

The update and populate methods work by simply receiving either the contact list as it is at login, which is done by the populate method, or by receiving a new user object that is to be added to the contact list that is currently being used.

The populate method will take the contact list and call the main menu constructor. This will create a new main menu screen that will have a list of the currently online contacts.

The login class is also used to store the complete list conversations that are currently connected with this client. This is for later reference by the other components of the client when they are determining a messages destination conversation.

The final method in the login screen class is a message handler that is concerned with sorting out which conversation the revived message is for. This is done with a series of loops that go through each conversation checking the participants host id and their nick name. If there is no conversation found, then a new conversation window is created and the message posted in that conversation.

The main menu class is used to create and show the graphical interface for the main menu. This class also contains a mouse listener and a selection listener. These two listeners are used in conjunction with each other to allow the user to double click on a contact in the contact list to start a conversation with that contact. The mouse listener waits for any mouse click events, which are fired when the use double clicks on a contact, when this happens, the mouse listener will attempt to ascertain which contact was selected by getting the selected contact result from the selection listener.

The selection listener is attached to the JTree that has been used to list the contacts. When a selection is made on the tree, an event is fired that calls the selection listener. The listener will then use that event to determine what has been selected on the list.

The main menu class also contains a method for creating a new instance of the conversation class. This is called when a user double clicks on a contact on the list, and fires an event for both the mouse clicked listener and the selection listener.

The final class involved with the client application is the conversation screen class. This is a basic method which contains the building code for the interface of the conversation as well as an action listener that is assigned to listen to events from the send button. This listener will wrap the String into a Message wrapper and send it back to the login class for sending back to the server. This class also has an update conversation method that will print any new messages received or sent to the conversation window allowing the user to see the latest messages in the conversation.

The client and server applications share a custom package of common utilities. This package contains the definition of the Message object that is used by both applications, as well as the User object which again is used by both applications and a Stream object that is used solely by the server. This package was shared between the two applications to prevent having to duplicate the code for the definitions of the objects in both the server and the client.

6.3 Final design

6.3.1 Graphical User Interface design

The final Design for the Graphical User Interface was designed so that the colours were easy on the eye. This is because the application will be used for a long period of time by its users.

If the interface was too hard on the eyes it would quickly become unusable and therefore worthless to the company that was using it.

Appendix C1 shows the final design for the login screens interface. This was designed to be simple and clear to use. With clear labels on what the text boxes are for, and a simple "Sign In" button for the user to sign into the system. There are also clear instructions for the user to follow, to ensure that anyone unfamiliar with the concept of an instant messenger or this kind of form layout knows what to do at this stage (*Stone, D. Et al. 2005. p212*).

The main menu for the application (Appendix C3) is also designed to be as simple as possible. The current user has their nick name displayed at the top of the window with a status changing menu along side. This nick name is the nick name that was provided in the database, and the field on the main menu is automatically populated with that value.

The status menu will automatically send a status update to the server, which will in turn send the new status to the clients that are currently connected to the server.

There is no dedicated sign out button as closing the main window will automatically sign the user out from the server.

Appendix C2 shows the simple popup menu that appears if the user attempts to login with a set of details that are no currently held in the database. This simple popup also resets the text fields for the login menu so that the user can simply and quickly re-enter their details and attempt to log in again.

This was included in the final application because it was felt that the user will need some prompt to inform them that their login details are wrong instead of just resetting the text fields without explaining to the user why the fields were reset. This prompt has no additional functionality, but is included to improve the overall usability of the application.

The final conversation interface (Appendix C4) is closely based on the one that was developed for the prototype. This was mainly because there was scope for the reuse of code, with only minor modifications to allow the final functionality to be included with the interface. For example the participants list on the far right now populates with the nick names of the users involved in the conversation. The send file button now also works as well, sending the file to the server with a message telling the server to send the file onto the participants in the conversation. Another change made to the conversation was that the menu bar at the top was replaced by two radial buttons along the top to allow the user to easily set if they want to have chat logging enabled or time stamps. There is also an add button at the top of the participants list which will open up a popup window (Appendix C5) that will let the user select what other contacts to add to the conversation.

6.3.2 Functional design

6.3.2.1 Server

The Server application its self is constructed of three classes. This is because each of the three classes carries out very different actions and so to keep the application as loosely coupled as possible it was decided to split the server class into these three components.

The first component, "dbInterface", is the class that is used to handle the connection to the database. This is done by creating an instance of the SQL Driver (*mySQL Connector J*) and using the driver manager (*Java Connection API, 2006*) to connect to the mySQL database that is being used to store the users details. This connection will pass the root accounts password to the database which will allow the server to run the required SQL queries when a client connects and passes the users username and password to the server. The database details are set in a dbconf.txt file. This is to allow the details for the database, such as which table to use and the root password to be easily changed.

```
Class.forName("com.mysql.jdbc.Driver").newInstance();

this.con = DriverManager.getConnection(database, account, password);
```

The connection between the server and the database is a persistent one. Once the connection is established it is kept open until the server is shut down. This reduces the query time on the database by removing the need to connect to the database every time a new client connects to the server.

The class will also generate an SQL query based on the information that the user supplies at the client side. This is the query that will be run on the database to ensure that the user trying to connect to the server is a valid user, and if their password is correct. If either of these conditions is not met, then an error message is returned to the server, which will then notify the client that it does not have access rights to the server, and the user will be prompted to re-enter their details.

The second server component for the server is the thread class. This class is dedicated to listening for incoming streams from the clients and passing the messages onto their respective methods in the main server class. This is determined by the message type that is defined every time a new message object is created. This is to help filter messages to the methods that they are meant for simply and quickly. A new instance of the listener thread is instantiated every time a new client attempts to connect to the server. This thread is then that client's private thread, for handling all of its input. This is to allow the server to be able to process several different inputs that all arrive at the same time, without any apparent slow down in response times to the client.

The third component of the server is the main functionality class. This class contains the different message handling methods, reply streams and connection lists.

Although this implementation is similar to the initial design, the modularity of the server was increased to not only improve the maintainability of the application by allowing modules to be easily changed and updated without effecting other parts of the server, but it also helped improve readability of the code, with each of the classes dealing with a specific function or set of functions.

For a full class diagram showing the design of the server including methods and relationsips, see appendix D1.

6.3.2.2 Client

The final design for the client has nine different classes. This is to improve modularity and to make the code easier to read and maintain.

The main class, JIMClient, is the main class for the client application. This is the class that contains all of the screen independent functionality, and handles the relationships between different screens such as the main menu screen and all the conversation screens that may be open at any one time. This is also the class that will control the sending messages to the serve as well as receiving them, and passing them onto the appropriate classes such as conversation update, file transfers, new contact lists etc.

The client has its own incoming reader thread that is used to read any messages received from the server. This was used because it provides a constant update mechanism that has none of the delays associated with polling for an update every few seconds or so. The reader will call the appropriate methods in the main client class, depending on the type of message received.

The login screen has its own class for drawing the interface on the screen. This interface will then call the login method in the main class once the login button has been clicked. Depending on the returned result from that method, either the client will be allowed to connect to the server and the main menu screen will be initialised or the error popup will be shown, informing the user that the incorrect user name or password has been entered. This class will also reset the values currently being stored in the username and password fields.

The error popup screen is a small class that draws a screen informing that the user has entered incorrect details, and resets the values in the username and password field to null. This popup has no interaction with the main class other than to be initiated by the main class.

The main menu class is again another class dedicated to building and displaying the interface for the main menu screen, with calls to the main client class for any functionality. This class has had the sign out button removed from the interface. This is because it was decided to have the close button at the top of the window call the sign out method instead. This means that the user will close the window as they would any other, and the client would automatically sign them out from the server. This is to remove the possibility of the user closing the interface without signing out. This would result in “dead” users being kept in the servers list of currently online users.

The conversation class is the most changed from the initial designs. Because of the changes to the interface, it is now necessary to have two functional methods in the interface class for the conversation screen. These two methods are for defining if the redial buttons for logging and time stamps have been selected, and for handing file transfers between clients. These methods were kept in the conversation window because the functionality of the methods is dependent on the settings of each of the conversations. The rest of the functionality, such as updating the incoming text area is kept in the main client class.

There are also two other small classes that are required to display two popup screens. The first of these is a screen that is displayed when a user right clicks on a contact in the main menu. This clicking action will call a details request method in the main client class that will display the details of the user that has been selected from the contacts list. The second popup is called from the main class when a user clicks on the add button in a conversation window to add a participant to a conversation. This will call an invite method in the main class that will draw the popup screen with a list of the current users online, the user will simply double click on the users that they want to add to a conversation and an add method will be called in the main class. This method will update the participants list in the conversation class with the selected user. This will allow that user to take part in the conversation that they have been invited to.

7 Development and Implementation issues

7.1 Threading

"*In computer science, a thread of execution is a fork of a computer program into two or more concurrently running tasks. The implementation of threads and processes differs from one operating system to another, but in most cases, a thread is contained inside a process*."

http://en.wikipedia.org/wiki/Thread_(computer_science)

Programming an application to be truly multi-threaded is not a task that I have had to accomplish before. This has lead to having to spend time to learn how to use threads correctly. This use of new technology had delayed the project by several days as Threading introduces several issues which needed to be addressed. Although threads appear to be running at the same time they are not. This is of course only possible in the truest sense when the application is run on a multi-core processor. If this is not the case, the processor will allocate a specific amount of processing time to each thread, and execute as much of the thread as it can in that space of time, before moving onto processing the next thread, this will be repeated until every thread has been completed. The speed with which a processor will carry out this switching between threads will mean that to the general user, it will appear as if the different threads are running in parallel. This also means that if the threads have not been implemented correctly, they will each be making different changes to the same variables and this can result in unpredictable behaviour on the part of the overall application.

7.2 Sharing variables

Java threads are specifically designed to share memory space with other threads. This means that they all have access to the same variables within the program source code. As all of threads used had access to the variables there was often a problem with different threads affecting a variable that another thread was working with. This caused some very complex and time consuming errors that needed to be addressed.

7.3 Streaming different objects over a network connection

7.3.1“Serializable” methods

This is another new concept that has caused implementation delays. In order to send an object via a communications stream between two different applications whatever is being sent needs to be “Serializable”. This means that it can be converted into a sequence of bits so that it can be stored in the communication stream buffer in order to be sent over the stream. During the process of developing this application it was necessary to create several Object types that were unique to the application. As there was no prior experience with streaming data, these unique objects were not declared as being “Serializable”. This meant that they were unable to be sent over the streams.

7.3.2 Streams

Streaming is the method used by java applications to communicate over a network, to between the application and some other object, such as a file. These streams are specific to what they are designed to transfer. So in the case of the “FileOutputStream”, it is designed to output a serialized file only and nothing else. The same is true for sending things like a String.

This dedication of each type of stream caused an issue when developing this application for the simple reason that the application needs to be able to send several different types of Object. Because of the way that java implements connections via sockets, it is only possible to create one stream per socket, and only one socket per client. This would mean that the application would normally have to handle creating and destroying streams depending on what type of stream it needed. This would have lead to dropping of bites from streams whilst new streams were made, or if there was an error and the wrong object type was sent to the wrong stream. These errors would result in the client and the server potentially crashing. In order to overcome this, a unique object wrapper was developed to wrap all of the objects in and allowed them all to be sent over the same stream. This wrapper has been given a header to define the type of object that has been sent, and this then allows the client and server to handle the stream in the appropriate way.

8 Testing

The testing of software is considered to be a difficult task to be completed in full. This is due to the huge range of problems that are faced by a tester. There is the number of different environments that the software could be run in. This is not just Operating System specific, but includes the device drivers, installed software, third-party software and a multitude of other variables that can all have an effect on the stability and usability of a piece of software (*Whittaker, J.A. 2000*). To test all of the combinations of environments is not a feasible testing strategy for testing a small scale application, let alone large and complex software solutions.

There is also a possibility that due to budget or time constraints it is not possible to fully test every piece of code in an application (*Whittaker, J.A. 2000*). This generally leads to testers testing the most likely areas that they consider will hold bugs.

There are a large number of testing practises that can be applied to testing software. These are divided into three groups, white box testing, black box testing and code reviews.

Functional testing, a black box technique, requires that a set of test cases are developed from the requirements specification supplied for the product that is under test. These test cases are developed using techniques such as equivalence partitioning and boundary value analysis (*Fagan, M.E. 1976*). The program is then run, and tested using the test cases derived from the requirements to ensure that the requirements are met. This form of testing could be used as part of an acceptance test to prove to the customer that the product does meet the requirements.

Structural testing, a white box technique (*Miller, E. Howden, W.E. 1981*), involves the tester testing the actual code. This is done by the tester having the code available to them, and deriving the test cases from analysing the code. Structural testing techniques include branch testing, statement coverage and path testing. Branch testing involves covering each branch that the program takes at a decision in the code. Statement coverage involves covering every statement in the program; this can often take a large amount of testing time. Path testing is possibly the most extensive form of testing. This covers every unique path through a program (*Myers et al 2004*). This means that even small programs could have a large number of unique pathways through it. This makes pathway testing a very time consuming method of testing.

Code reviews or inspections use a set of guidelines to allow detection of code standards violations, uninitialized variables, and unreachable code (*Fagan, M.E. 1999*). The times of inspection are planned out at the start of a project, often to review code up to a certain milestone or key feature. The inspections are usually carried out by four people, one being a moderator, one being the programmer, and two others. Any errors found are recorded and classified in terms of severity (*Myers, G.J. 1979*).

Due to the limitation of time, it has been decided that the application will undergo a set of functional test cases developed from the requirements specification of the product. This is to ensure that the product is fit for purpose.

During the testing phase, the application will be distributed to a small group of IT professionals for them to use as an instant messaging application when talking amongst themselves. The members of this team include two software engineers, a network analyst, a mechanical engineer, and a networks security student.

These team members are all separated geographically within the United Kingdom. One member also uses OSX10 operating system instead of a version of Windows. This diversity of users as well as operating environments will provide an insight into how well the application copes with a range of users and environments, as well as gather feedback on improvements and further enhancements that the users may deem necessary.

It is also assumed that a certain level of quality has been maintained throughout the development cycle by the programmer. This means that it has been assumed that there are no uninitialized variables, the variables are named sensibly, and that the code is well commented.

8.1 Test Plan

The aim of the test cases is to ensure that the application that has been developed meets all of requirements that were specified after researching existing products, and discussions with the customer.

Because the requirements have been split into two different groups, the core functionality and the additional functionality, the testing will also be split into two distinct sections to ensure full coverage of both sets of requirements.

Each of the requirements will have its own set of test cases to ensure that the application behaves as expected when being used in different ways. This will include load testing on both the client and server. The load testing will take the form of how well the client handles multiple conversations, and the server will be tested to see how it handles multiple clients connecting.

During the testing the Server will be installed and run on Windows Vista with the following specification:-

- Intel Core2 Duo running at 2.13 GHz
- 2 Gigabytes of RAM
- 64 – Bit Windows Vista Business with Service Pack 1 installed
- MySQL server 5.1
- Java 6 Java Virtual Machine

The Clients will be run on a Windows XP machine with the following specification:-

- AMD
- 1 Gigabyte of RAM
- 32 – Bit Windows XP Professional with Service Pack 2 installed
- Java 6 Java Virtual Machine

Each test case will have an associated test report, outlining if the test was passed or if it was failed. If the test was failed the report will include what the tester was doing, how they did it, the number of clients and or conversations open at the time, symptoms of the error, and suggested fixes.

8.2 Test Cases

Twenty four test cases were designed in order to test the program met its requirements. The first set of test cases (Appendix E1) were designed to cover the ability of the clients to be able to have individual conversations with each other. This was aimed to ensure that the messages were being sent and received correctly at both the destination and the origin. The tests were also designed to make certain that the clients could handle taking part in several individual conversations at the same time. This was a particular concern as the message object had a type field that would hold the type of the message such as a contact list update or an actual message for a conversation.

The client has a built in mechanism for determining which conversation the message is meant for by means of the message source field. This field is compared with the participants of each conversation, and depending on the result, is added to the correct conversation. This was considered to be a high risk method, and so test cases had to be designed for it to be fully tested to ensure that the method was working correctly. This was done with the test cases in appendix E1.

The second set of test cases (Appendix E2) were also designed to test a particularly complex function. This was the ability for the client to send a file to either an individual person or to a group of people in a group conversation. This function worked by breaking down the file into a byte array and sending that array to the clients in a similar way to the normal text based messages. Due to the fact that file sized will vary from file to file, it seemed wasteful to simply create a byte array with the maximum integer value as its size. To overcome this, the length of the file is determined first, and that is used to create a byte array of the correct size. Once the array is created the file is translated into bytes and stored in the array to be sent. The tests were designed to ensure that the file was being correctly stored as a byte array without any parts of the file missing.

The third sets of test cases (Appendix E3) were designed to ensure that the clients were capable of handling group conversations correctly, including inviting addition people to the conversation and people leaving a conversation.

There are also test cases designed to test the user's ability to set time stamps on a conversation (Appendix E6), the user's ability to change their current status and ensure that the status update was delivered to all the clients connected, and that they updated their contact list (Appendix E5). Conversation logging and user profile thumbnails were also tested (Appendix E7 and E8). Both of these were simple functions that required only a small amount of testing time.

Tests were also carried out to make sure that the server was able to keep a constant set of logs with a large number of clients connected at the same time, without any apparent slowdown in the server's responses (Appendix E9). The server has to be able to keep organised logs for every message that was sent and received by the clients. These logs were designed to be ordered by date, user, and conversation in order to be able to quickly and simply find the log that an administrator may be searching for.

The application was also distributed to a small group of Beta testers. This was to analyse the usability and aesthetics of the application. The group were encouraged to note down any problems or improvements they would like to see made, as well as any aspects of the application that they found to be of a high standard.

8.3 Test Results

Out of the twenty four test cases that were executed on the application, twenty two passed with no errors being given. This gives the application a 91% pass rate during the testing phase.

The two test cases that failed were related to the sending of offline messages. This is due to the fact that the design for the application did not include framework necessary for it to work. The fault reports are available in appendix E11. The failed tests were not fixed due to the belief that email and other asynchronous technologies will be able to take the place of the offline messaging. The aim for this application is to provide a comprehensive synchronous communication tool, not a replacement for asynchronous methods.

The high percentage of test passes indicates that the program satisfies all of the core functionality requirements, and most of the additional functionality. It also proves that the application is stable when the server and database is under a degree of load and that the client is in a suitable condition to be used in the intended environment. This is mainly based on the general test cases that were covered in appendix E10. These test cases covered aspects of the program that were not necessarily directly linked to the core functionality that was outlined in the requirements. This included keeping up to date user list, and allow multiple clients to communicate correctly.

The client application was also distributed to members of small distributed team for them to test. This was the beta testing team. This team was chosen because they represented a small subset of the prospective user base that the application is aimed at. The beta testing highlighted several small issues with the client that although not linked to the functionality of the application, helped improve its usability and refine the applications finished state. This included buttons staying in the pressed state after they had been pressed. This was most noticeable on the login screen. The users also pointed out that the incoming text area of the conversation window did not auto scroll to the bottom of the conversation. This was considered to have a significant impact on the usability of the application, as the users were quickly becoming frustrated with having to scroll down to the bottom every time the conversation was updated.

9 Evaluation

9.1 Application

9.2 Project

10 Conclusion

11 References

Canfora, G. *Can Collaborative Software Development Benefit from Synchronous Groupware Functions?* 2003

Carmel, E., *Global Software Teams: Collaboration Across Borders and Time Zones*. 1999

Casey, V and Richardson, I. *uncovering the reality within virtual software team,* 2006

Casey, V. and I. Richardson. *Practical Experience of Virtual Team Software Development* 2004

Cerulean Studios – Learn About Trillian, Available from:- http://www.ceruleanstudios.com/features/ [Accessed on 16/03/2009]

Espinosa, J.A. and Carmel, E, *The Impact of Time Separation on Coordination in Global Software Teams: a Conceptual Foundation* 2004

Features – Yahoo Messenger, Available from:- http://uk.messenger.yahoo.com/features/ [Accessed on 16/03/2009]

Herbsleb, J.D. Moitra, D. *Global Software Development,* 2001

Herbsleb, J. D, Mockus, A, Finholt, T. A, and Grinter, R. E, *An empirical study of global software development: Distance and speed* 2001,

Herbsleb, J.D, Paulish, D.J, and Bass, M. *Global software development at siemens: experience from nine projects* in *Proceedings of the 27th international conference on Software engineering*. 2005.

Internet Relay Chat, Available from:- http://en.wikipedia.org/wiki/Internet_Relay_Chat [Accessed on 16/03/2009]

Kiel, L. *Experiences in Distributed Development: A Case Study*. In *International Workshop on Global Software Development,*. 2003.

Perry, D.E., N.A. Staudenmayer, and L.G. Votta, *People, organizations, and process improvement.* 1994.

Stone, D. Jarret, C. Woodrofe, M. Minocha, S. 2005. *User Interface Design and Evaluation.* Elsevier inc.

Turoff, M., S. R. Hiltz, A. N. F. Bahgat, A. R. Rana. *Distributed group support systems*. 1993

Windows Live Messenger – Features, Available from:- http://www.livemessenger.net/features/ [Accessed on 16/03/2009]

Java Socket API, 2006, Available from - *http://java.sun.com/javase/6/docs/api/java/net/Socket.html* [Accessed on 22/03/2009]

Java Thread API, 2006, Available from - http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Thread.html [Accessed on 22/03/2009]

mySQL Connector J, Available from - http://www.mysql.com/products/connector/j/ [Accessed on 25/03/2009]

Java Connection API, 2006, Available from - http://java.sun.com/javase/6/docs/api/java/sql/DriverManager.html [Accessed on 25/03/2009]

Java Nested Class, 2006, Available from - http://java.sun.com/docs/books/tutorial/java/javaOO/nested.html [Accessed on 26/03/2009]

Whittaker, J.A. 2000 – *What is software testing? And why is it so hard?*

Fagan, M. E. 1976, *Design and code inspections to reduce errors in program development*, vol. 15.

Miller, E., Howden, W.E. 1981 "*A survey of dynamic analysis methods*," in *Tutorial: Software Testing & Validation Techniques*, 2nd edition.

Myers, G.J., Badgett, T., Thomas, T.M. & Sandler, C., 2004. *The Art of Software Testing.* 2nd Ed.. Hoboken, N.J.: John Wiley & Sons.

Fagan, M.E, 1999. *Design and Code Inspections to Reduce Errors in Program Development.* P182-211

Myers, G.J. 1979. *The Art of Software Testing*. John Wiley & Sons

Bibliography

Rice. J, Salisbury. I, 1997, Advanced Java 1.1 Programming, McGraw Hill

Lewis & Loftus, 2005, Java Software Solutions 4th edition, Addison Wesley

Sierra. K, Bates. B, 2003, Head First Java, O'Reilly

Weiss, J.W and Wysocki, R.K, 1992, 5-phase project management, Addison Wesley.

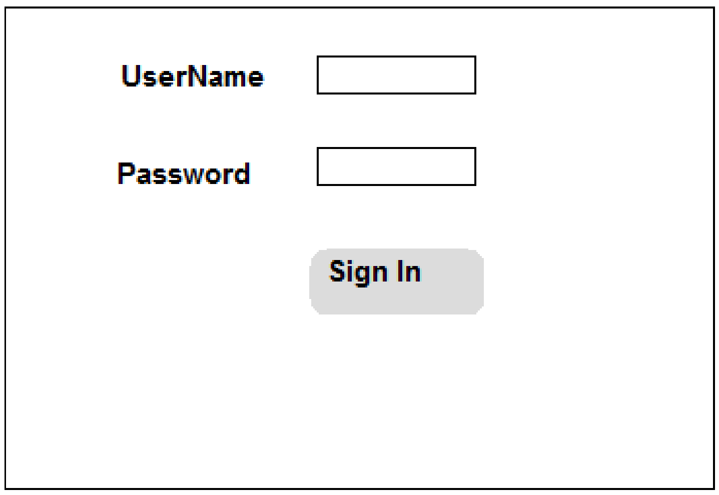
Forouzian, B, A., 2007. Data Communications and Networking. 4th edition. McGraw-Hill

McConnell, S. 1993. Code Complete, Microsoft Press

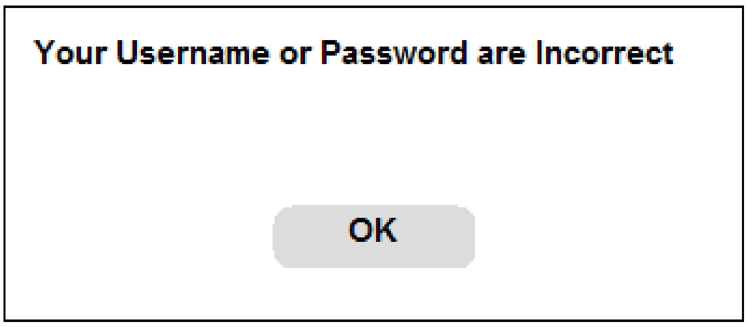
Appendix A

Initial Graphical User Interface Designs

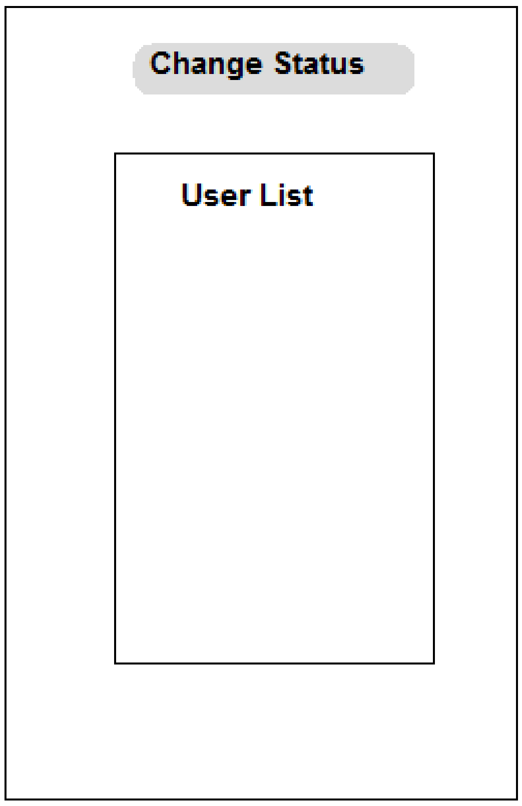
A1 - Login screen



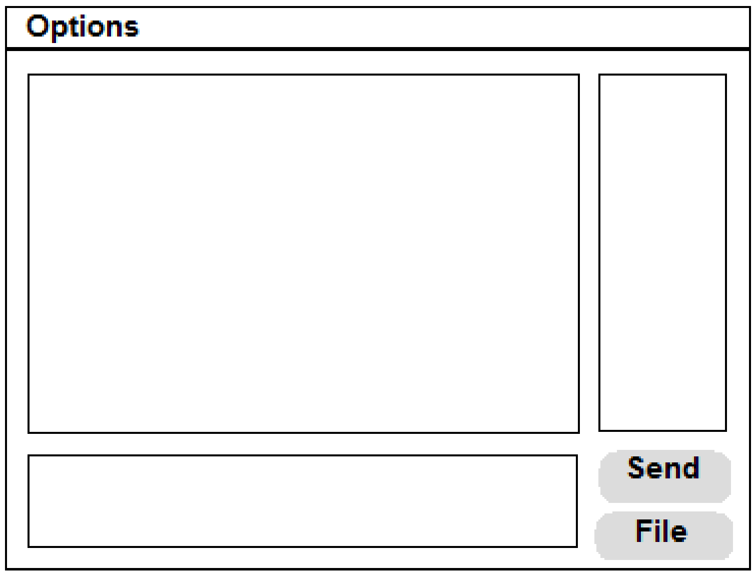
A2 - Login error screen


Your Username or Password are Incorrect

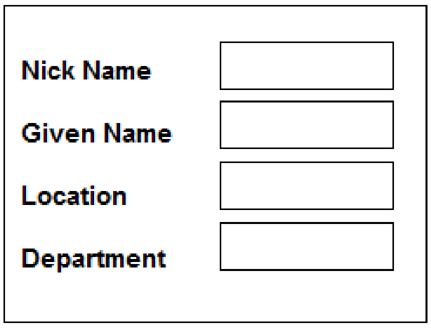
A3 – Main Screen



A4 – Conversation Screen



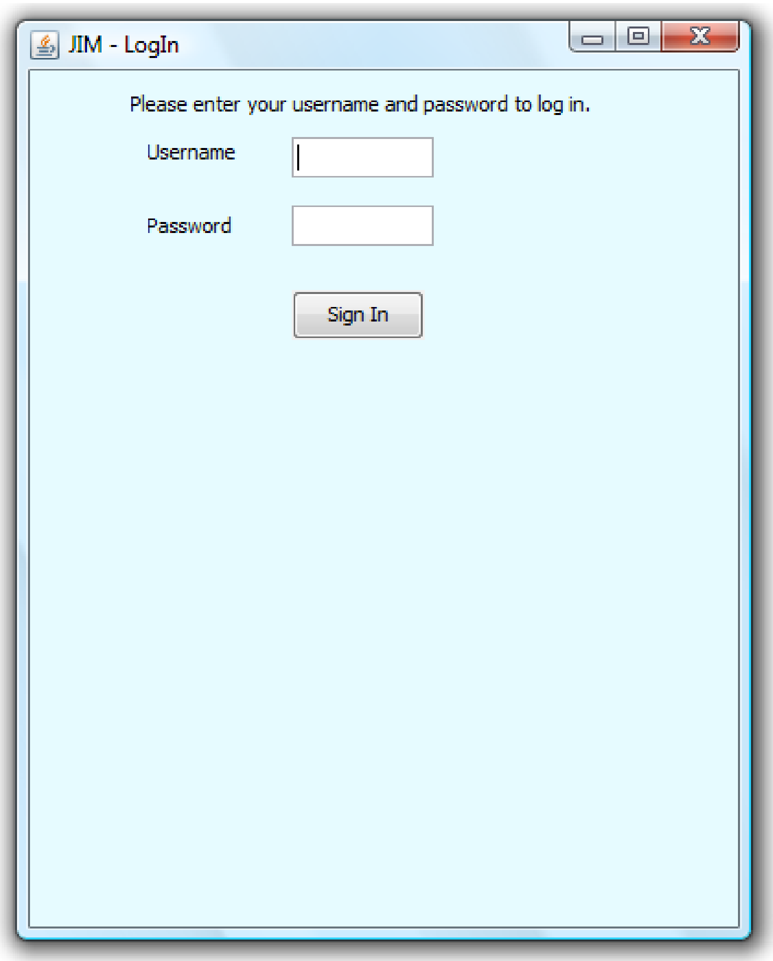
A5 – Contact Details Popup



Appendix B – Initial Class Diagrams

Appendix C – Final Graphical User Interface Designs

C1 – Login Screen

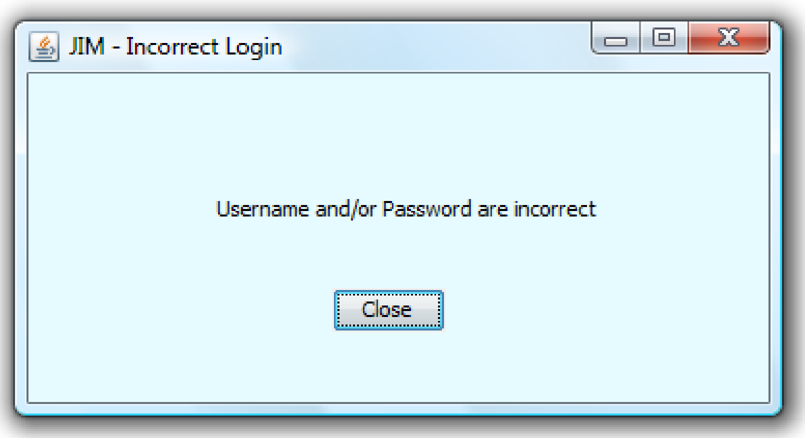


Please enter your username and password to log in.

Username

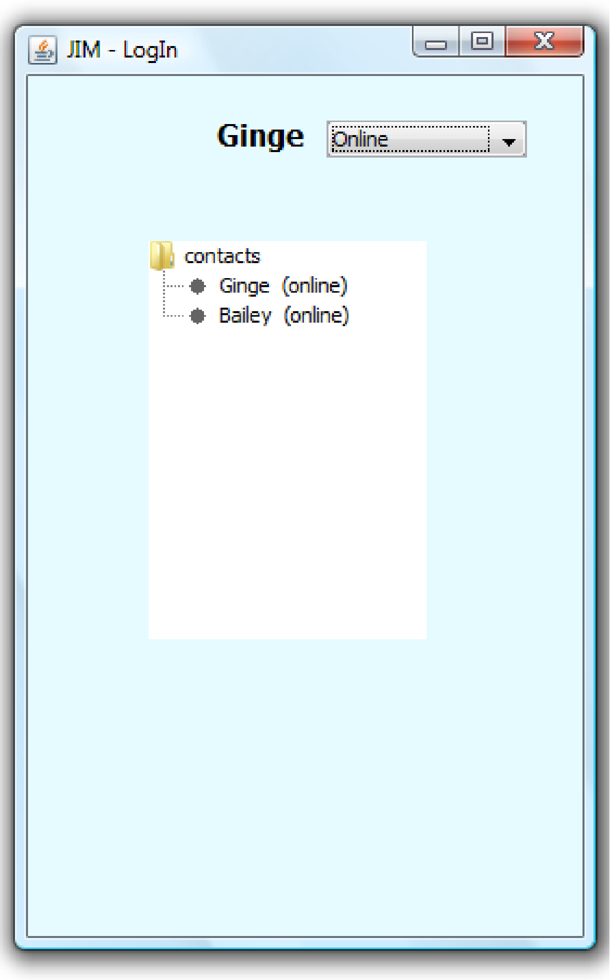
Password

C2 – Incorrect Login Details



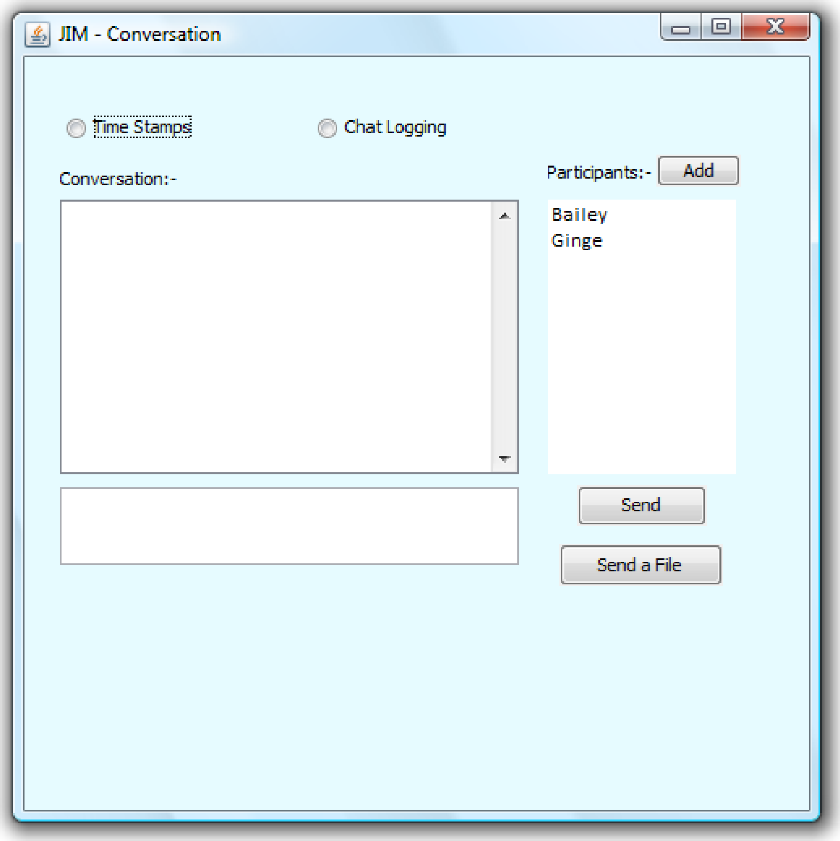
Username and/or Password are incorrect

C3 – Main Menu



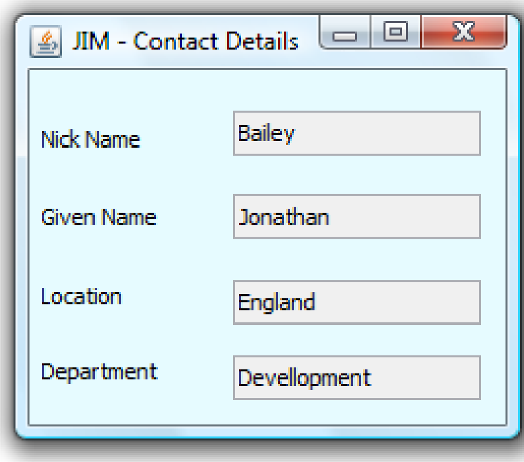
Ginge (online)
Bailey (online)

C4 – Conversation window



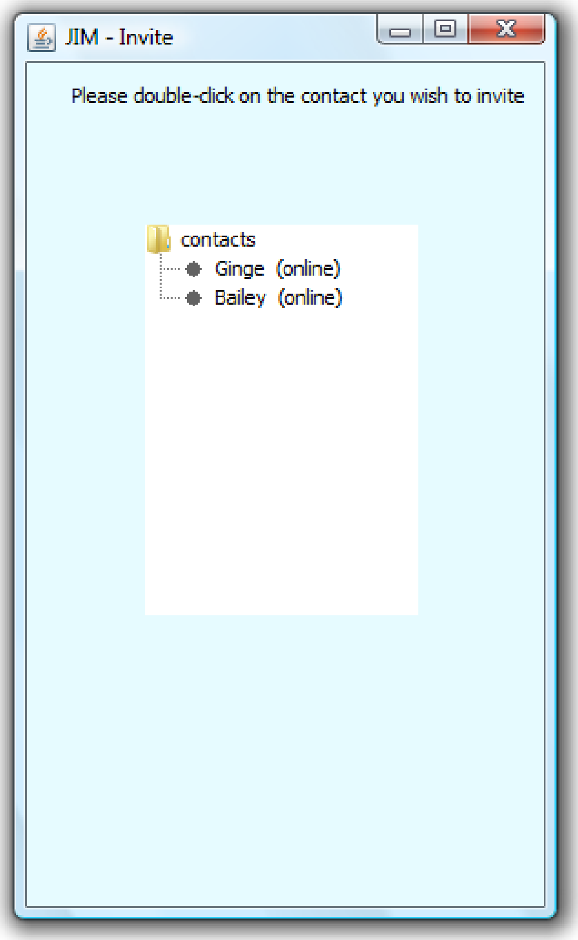
Bailey
Ginge

C5 – Contact Details Popup



Nick Name	Bailey
Given Name	Jonathan
Location	England
Department	Devellopment

C6 – Conversation Invite Popup

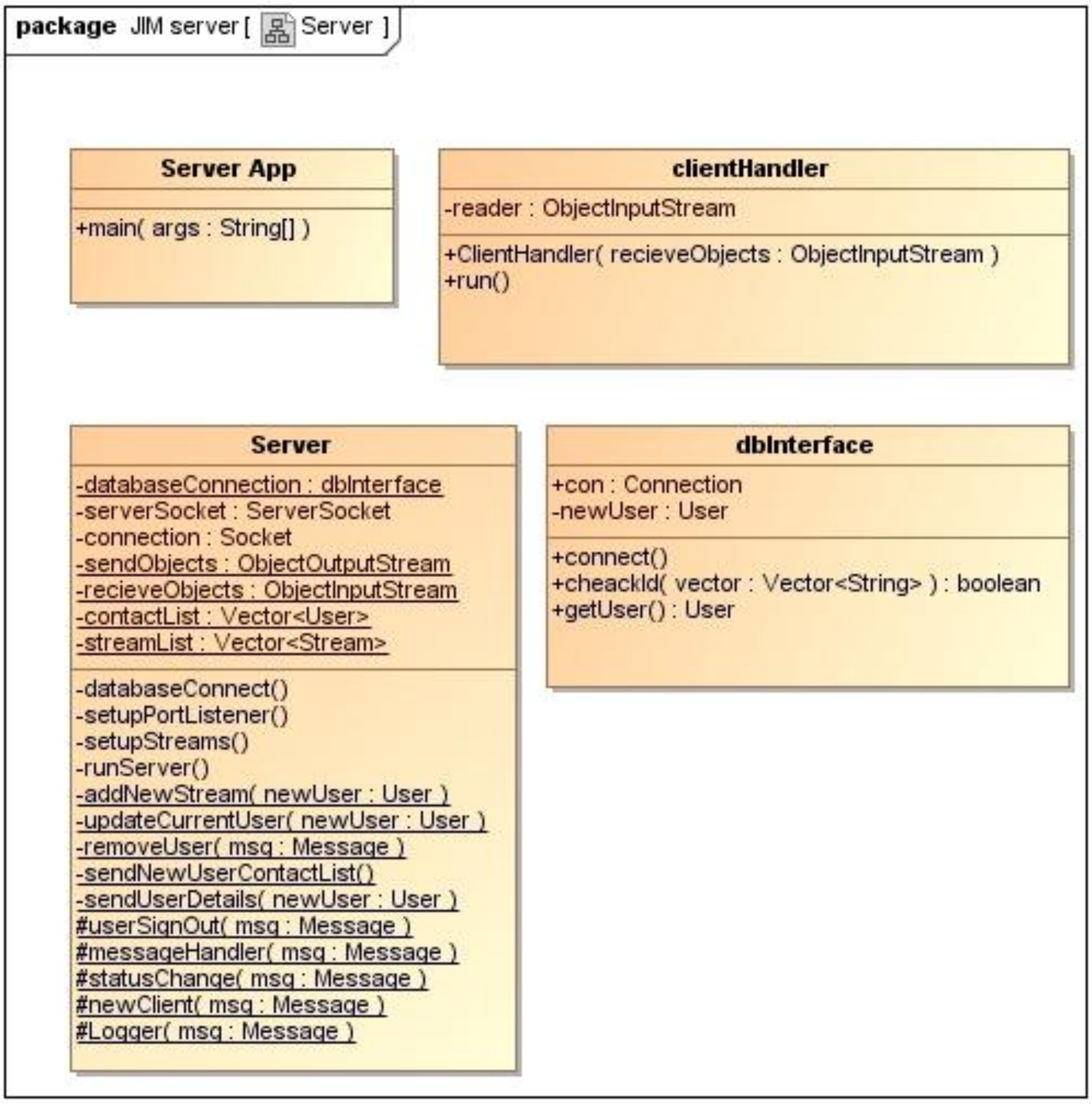


Please double-click on the contact you wish to invite

contacts
Ginge (online)
Bailey (online)

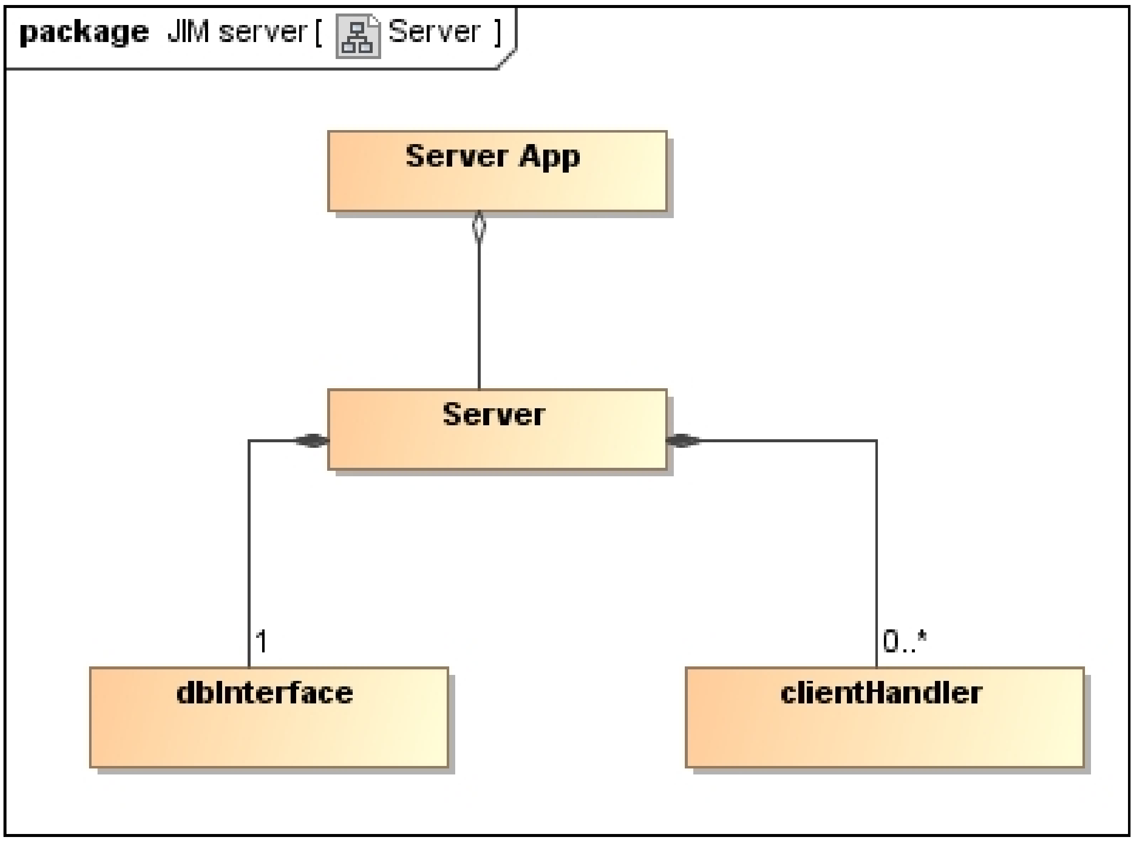
Appendix D – Final Design Class Diagrams

D1 – Server Classes with methods and attributes



package JIM server [Server]
Server App
+main(args : String[])
clientHandler
-reader : ObjectInputStream
+ClientHandler(recieveObjects : ObjectInputStream)
+run()
Server
-databaseConnection : dbInterface
-serverSocket : ServerSocket
-connection : Socket
-sendObjects : ObjectOutputStream
-recieveObjects : ObjectInputStream
-contactList : Vector<User>
-streamList : Vector<Stream>
-databaseConnect()
-setupPortListener()
-setupStreams()
-runServer()
-addNewStream(newUser : User)
-updateCurrentUser(newUser : User)
-removeUser(msg : Message)
-sendNewUserContactList()
-sendUserDetails(newUser : User)
#userSignOut(msg : Message)
#messageHandler(msg : Message)
#statusChange(msg : Message)
#newClient(msg : Message)
#Logger(msg : Message)
dbInterface
+con : Connection
-newUser : User
+connect()
+cheackId(vector : Vector<String>) : boolean
+getUser() : User

D2 – Server Class Diagram showing relationships between classes



Appendix E – Test Cases and Reports

Appendix E1 – Individual Conversation

Test Case 1:- Test the ability to begin an individual conversation

Description: - Verify that a client can send a message to another client.

Requirement under test: - Individual conversation.

Expected Result: - Client B receives the message from Client A in a new conversation window.

Result: - Client B receives the test message from Client A in a new conversation window. (There is not one already open)

Pass/Fail: - PASS

Test Case 2:- Test the ability to reply to an individual conversation

Description: - Verify that a client can send a reply message to another client.

Requirement under test: - Individual conversation.

Expected Result: - Client A receives the reply message from Client B in the current conversation window.

Result: - Client A receives the reply message from Client B in the current conversation window.

Pass/Fail: - PASS

Test Case 3:- Test the ability to have multiple individual conversations

Description: - Verify that a client can handle multiple individual conversations.

Requirement under test: - Individual conversation.

Expected Result: - messages are delivered to the correct recipients and displayed in the correct conversation window.

Result: - messages are successfully delivered to the correct client and to the correct conversation.

Pass/Fail: - PASS

Test Case 4:- Test the ability to restart a conversation

Description: - Verify that a client can close a conversation, and open a new one to the same contact.

Requirement under test: - Individual conversation.

Expected Result: - Client will successfully be able to close a conversation and start a conversation with the same contact. Recipient client is to continue getting updates at the original conversation window.

Result: - Client successfully closes a conversation and starts a conversation with the same contact. Recipient client continues getting updates at the original conversation window.

Pass/Fail: - PASS

Appendix E2 – Send Files

Test Case5:- Test the ability to send a file in an individual conversation

Description: - Verify that a client can send a file to a contact via a private conversation.

Requirement under test: - Ability to send files to contacts.

Expected Result: - Client A will successfully send a complete file to Client B via a private conversation window. And Client B will be able to open the file.

Result: - Client A successfully sent a readable file to Client B via a private conversation. Client B was able to read the file correctly.

Pass/Fail: - PASS

Test Case6:- Test the ability to send a file to all participants in a group conversation

Description: - Verify that a client can send a file to all contacts involved in a group conversation.

Requirement under test: - Ability to send files to contacts.

Expected Result: - Client A will successfully send a complete file to all the participants in a group conversation.

Result: - Client A successfully sent a readable file to all the participants in a group conversation.

Pass/Fail: - PASS

Test Case7:- Test the ability to receive a file in an individual conversation

Description: - Verify that a client can **receive** a file from a contact via a private conversation.

Requirement under test: - Ability to send files to contacts.

Expected Result: - Client A will successfully receive a complete file from Client B via a private conversation window. And Client A will be able to open the file.

Result: - Client A successfully received a readable file from Client B via a private conversation. Client A was able to read the file correctly.

Pass/Fail: - PASS

Test Case8:- Test the ability to receive a file from a participant in a group conversation

Description: - Verify that a client can receive a file from a contact involved in a group conversation.

Requirement under test: - Ability to send files to contacts.

Expected Result: - Client A will successfully receive a complete file from a participant in a group conversation.

Result: - Client A successfully receives a readable file from a participant in a group conversation.

Pass/Fail: - PASS

Appendix E3 – Group Conversations

Test Case9:- Test the ability to receive messages in a group conversation

Description: - Verify that a client can take part in a group conversation.

Requirement under test: - Ability to have group conversations.

Expected Result: - Clients A, B, and C will all correctly receive the messages sent from each of the three participants in the group conversation.

Result: - Clients A, B, and C all correctly received the messages sent from each of the three participants in the group conversation.

Pass/Fail: - PASS

Test Case10:- Test the ability to send messages in a group conversation

Description: - Verify that a client can take part in a group conversation.

Requirement under test: - Ability to have group conversations.

Expected Result: - Clients A, B, and C will all correctly send the messages to the participants in the group conversation.

Result: - Clients A, B, and C all correctly sent the messages to the participants in the group conversation.

Pass/Fail: - PASS

Test Case10:- Test the ability to add a contact to a group conversation

Description: - Verify that a client can add a participant to a group conversation.

Requirement under test: - Ability to have group conversations.

Expected Result: - Clients A and B will be able to successfully add a new participant (Client C) to the group conversation, the participants list for Clients A and B will be updated with the new participant's name.

Result: - Clients A, and B successfully added Client C to the conversation, and the participants list for Clients A and B were successfully updated.

Pass/Fail: - PASS

Test Case11:- Test that a client can be added to a group conversation

Description: - Verify that a client can be added to a group conversation.

Requirement under test: - Ability to have group conversations.

Expected Result: - Client C will successfully be added to a group conversation and have its participant list populated with the participants of the conversation.

Result: - Client C was successfully added to the group conversation and the participants list was updated correctly.

Pass/Fail: - PASS

Test Case11:- Test the ability to leave a group conversation

Description: - Verify that a client can leave a group conversation.

Requirement under test: - Ability to leave group conversations.

Expected Result: - Client A will be able to leave a conversation with Clients B and C, and no longer receive messages for that conversation unless re-added. Clients B and C will have their participants list updates.

Result: - Client A successfully leaves the conversation and receives no more messages for that conversation. The participants list is updated at clients B and C.

Pass/Fail: - PASS

Appendix E4 – Offline messages

Test Case12:- Test the ability to send an offline message to a contact

Description: - Verify that a client can send an offline message to a contact.

Requirement under test: - Ability to send an offline message to a contact.

Expected Result: - Client A will be able to send an offline message to client B.

Result: - Client A fails to send an offline message to client B due to client B not being visible (Appendix F1 – Fault Report 1).

Pass/Fail: - FAIL

Test Case13:- Test the ability to receive an offline message to a contact

Description: - Verify that a client can receive an offline message from a contact.

Requirement under test: - Ability to send an offline message to a contact.

Expected Result: - Client A will be able to receive an offline message from client B.

Result: - Client B fails to send an offline message to client A due to client A not being visible (Appendix F1 – Fault Report 2).

Pass/Fail: - FAIL

Appendix E5 – Setting status

Test Case14:- Test the ability to set current user status

Description: - Verify that a client can set its current status.

Requirement under test: - Ability to set current status.

Expected Result: - Client A will be able to change its current status from online to busy.

Result: - Client A successfully changes its status from online to busy.

Pass/Fail: - PASS

Test Case15:- Test that the clients are being updated with the latest statuses of contacts.

Description: - Verify that a client is being updated with status changes.

Requirement under test: - Ability to set current status.

Expected Result: - Client A will receive an update to its contact list when client B changes its status from online to busy.

Result: - Client A successfully receives an update to its contact list when client B changes its status from online to busy.

Pass/Fail: - PASS

Appendix E6 – Time stamps

Test Case16:- Test that the clients can select time stamps for conversations.

Description: - Verify that a client can set time stamps on conversations.

Requirement under test: - Optional time stamps on messages sent and received.

Expected Result: - Client A will be able to set time stamps on messages sent and received during a conversation with client B. The time stamps are correct to client A’s location.

Result: - Time stamps are successfully added to messages sent and received at client A’s conversation window with client B. The time stamps are correct to client A’s location.

Pass/Fail: - PASS

Appendix E7 – Profile thumbnails

Test Case17:- Test that the clients can activate a profile thumbnail of a contact.

Description: - Verify that a client can activate a profile thumbnail of a contact.

Requirement under test: - User profile thumbnails.

Expected Result: - Client A will be able to select a contact on the main menu screen and right click to activate a details popup for that contact.

Result: - Client A can activate a contact thumbnail by selecting and right clicking on the contact at the main menu screen.

Pass/Fail: - PASS

Appendix E8 – Conversation Logging

Test Case18:- Test that the clients can select logging for conversations.

Description: - Verify that a client can set logging on conversations.

Requirement under test: - Optional time stamps on messages sent and received.

Expected Result: - Client A will be able to select conversation logging for a conversation. Logs should be kept in directories sorted by date. Log files should be named after the participants of the conversation.

Result: - Conversation logs are kept in log files named after the participants in the conversation. Files are stored in directories sorted by date.

Pass/Fail: - PASS

Appendix E9 – Server Logs.

Test Case19:- Test that the server will keep logs of the conversations

Description: - Verify that the server is keeping logs of all conversations.

Requirement under test: - Communication logging on the main server.

Expected Result: - Server will keep accurate logs of all conversations, stored in directories sorted by date and contact.

Result: - Server keeps logs correctly.

Pass/Fail: - PASS

Appendix E10 – General Functionality

Test Case20:- Test that the clients receive updates to contact list.

Description: - Verify that a client can receive updates to its contact list when a new contact logs in.

Requirement under test: - Stable instant messenger client – server application.

Expected Result: - Client A will update its contact list when client B logs in.

Result: - Client A successfully updates its contact list when client B logs in.

Pass/Fail: - PASS

Test Case21:- Test that the clients receive updates to contact list.

Description: - Verify that a client can receive updates to its contact list when a contact logs off.

Requirement under test: - Stable instant messenger client – server application.

Expected Result: - Client A will update its contact list when client B logs off.

Result: - Client A successfully updates its contact list when client B logs off.

Pass/Fail: - PASS

Test Case22:- Test that the server will accept multiple clients

Description: - Verify that several instances of the client program can connect to the server at the same time.

Requirement under test: - Stable instant messenger client – server application.

Expected Result: - Client A and B will be able to sign into the server at the same time.

Result: - Clients A and B successfully connected to the server.

Pass/Fail: - PASS

Test Case23:- Test that the server will update client list at login

Description: - Verify that the client is keeping an accurate contact list at login.

Requirement under test: - Stable instant messenger client – server application.

Expected Result: - Server will print out a list of clients currently online when a new client connects to the server.

Result: - Server prints the correct users from the contact list.

Pass/Fail: - PASS

Test Case24:- Test that the server will update client list at logoff

Description: - Verify that the client is keeping an accurate contact list at logoff.

Requirement under test: - Stable instant messenger client – server application.

Expected Result: - Server will print out a list of clients currently online when a client disconnects from the server.

Result: - Server prints the correct users from the contact list.

Pass/Fail: - PASS

Appendix E11 – Fault Reports

Fault Report 1:- Sending Offline Messages

Test Case: - 12

Application Version: - 1.0

Summary: - Unable to select an offline user from the contact list due to there being no offline users displayed.

Steps to reproduce:-

1. Run the server
2. Run an instance of the client and sign in
3. Attempt to select an offline contact from the contact list

Severity: - Moderate

Fault Report 2:- Receiving Offline Messages

Test Case: - 13

Application Version: - 1.0

Summary: - Unable to select an offline user from the contact list in order to run test case due to there being no offline users displayed.

Steps to reproduce:-

4. Run the server
5. Run an instance of the client and sign in
6. Attempt to select an offline contact from the contact list

Severity: - Moderate