University of Glasgow | Department of Computing Science

GIM - Team G Instant Messenger

Ewan Baird
Heather Hoaglund-Biron
Gordon Martin
James McMinn

Level 3 Project — 28 March 20111

Abstract

The abstract goes here

Education Use Consent

We hereby give our permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name:	________________	Signature:	________________
Name:	________________	Signature:	________________
Name:	________________	Signature:	________________
Name:	________________	Signature:	________________
Name:	________________	Signature:	________________
Name:	________________	Signature:	________________

Contents

Chapter 1

Introduction

1.1 Introduction

Instant messengers are an easy, fast way to communicate over the Internet. Basic instant messengers allow users to type messages to each other on separate computers and have those messages immediately show on the screen, like instant email. More advanced systems have features like audio and video communication. These days instant messengers are becoming more popular, given the way the Internet is helping people from all over the world connect with one another. This has led to an increased demand for quick and easy communication.

These days, its difficult to be original when developing an instant messenger. Programs like Skype, AIM and MSN Messenger are popular, especially with the younger generation. There are many kinds of instant messengers out there, the most prominent of which are always competing against each other to have the newest and most intriguing features.

This project is not about joining that competition. Instead, we are using this opportunity to explore what its like to create an instant messenger from the ground up. As we are part of the younger generation, we use instant messengers almost daily, and its interesting to be able to go behind the scenes and discover how they work for ourselves.

The instant messenger model that we decided to use consists of a server, any number of clients, and a set of rules defining how the server and clients talk to each other, called a protocol. A user of the program is essentially interacting with a client, and the clients interact with each other through the server. That communication is structured using the protocol.

Instead of using the protocol and server of an existing instant messenger and focusing on the client, we decided to create the entire system ourselves. This way were able to learn more about the whole process of creating such a program. We have divided ourselves into two smaller groups: two of us to do the networking and make the server, and the other two to make the client, including the user interface, or UI.

This report is meant for the Computing Science professors at Glasgow University, fellow Computing Science students, and anyone with an interest in instant messaging or the process of completing a large-scale Computing Science project.

The structure of this report is as follows. In Chapter 1, we have this introduction, the problem definition, and an overview of the requirements. Chapter 2 covers the design of the protocol, client, server, and overall networking. In Chapter 3 is a discussion of our implementation of the client and server. Chapter 4 covers evaluation of our finished product, and Chapter 5 is a conclusion and reflective of what we learned.

1.2 Problem Definition

blah blah blah

1.3 Requirements

Chapter 2

Design

2.1 Protocol Design

2.1.1 Protocol Overview

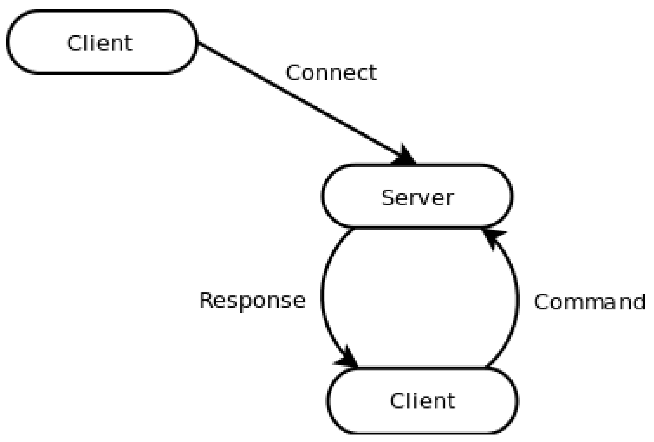

Figure 2.1: The exchange of messages between a client and the server

Gim uses a client-server architecture, where one computer (known as the Sever) acts as a central point which other computers (the Clients) connect. The clients do no communicate directly with each other and all communication takes place between the clients and the server. If a client wishes to send a message to another client it must first got through the server.

In GIM, the Protocol is responsible for enabling communication between the clients and the server in a reliable and consistent manner. A protocol is a set of rules that determine the format and transmission of data between computers. The syntax (the structure, or format) and semantics (the meaning) of the Protocol are discussed in the next section.

At the highest level of abstraction, the GIM Protocol works in a very simple manner. A client connects to the sever and they exchange messages until the connection is closed, as shown in Figure 2.1.

In practice there are several clients connected to the server at once, however as the clients do not directly interact with each other and do not need to know about each other, the entire system can be simplified as described above.

2.1.2 Protocol Specification

2.1.3 Protocol Evolution

2.2 Client Design

In designing the client, we had the high level aim of creating a modular system to allow team members to take responsibility for certain aspects of the system. From an early stage in the design process, we were aware that the client had a large set of responsibilities. We determined these responsibilities to be; to maintain a connection with the server, implement the GIM protocol, provide a user interface to the system and keep a record of up to date information about the users friends. Our first task was to design a set of interacting sub-components to handle these responsibilities.

It was agreed that the MVC (Model-Viewer-Controller) architectural pattern, widely used for applications involving a graphical user interface, was a useful model to base our discussions around. This model abstracts the UI (view) from the back end of the system. When a user performs an action, the controller updates a collection of data associated with the system, and updates the interface to reflect any changes. This seemed appropriate to our needs of keeping and displaying an up to record about the users friend list and creating an interface, and would allow us to split responsibilities between the back and front end of the GUI.

However, we faced challenges in adapting an additional networking component into this framework (to implement the GIM protocol.) We had the choice to conceptually view it as either an additional interface, which viewed changes to the model, or indirectly (by way of the network) the controller component. [wtf... discussion about merits of both] We decided it would be useful to treat the networking code as part of the controller, as it would be modifying the model based on the servers response to its calls, and modifying the GUI to reflect these changes.

Our next challenge was to more formally design the roles of each component, and the boundaries between them.

2.3 Server Design

blah blah blah

2.4 Networking Design

blah blah blah

Chapter 3

Implementation

3.1 Client

blah blah blah

3.2 Server

blah blah blah

3.3 Client-Server Communication

blah blah blah

3.4 Storage

blah blah blah

Chapter 4

Evaluation

4.1 Server Evaluation

blah blah blah

4.2 Client Evaluation

blah blah blah

4.3 User Evaluation

blah blah blah

Chapter 5

Conclusion

5.1 Contributions

blah blah blah

5.2 Reflective

blah blah blah

Appendix A

Appendices