

Nicholas Taylor

SmartMessenger: Advanced Instant
Messaging on Smartphones

B.Sc. Computer Science with Software
Engineering

24th March, 2006

“I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the Department’s use of plagiarism detection systems in order to check the integrity of assessed work.

I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.”

Date: 24th March, 2006

Signed:

Abstract

This project documents the development of a prototype client which explores the potential for extending instant messenger clients on smartphones. The application will attempt to surpass the usability and functionality provided by existing clients by allowing the user to connect to both an existing Internet-based messenger service, and a peer-to-peer Bluetooth service developed as part of the project. The system also aims to devising a more flexible presence-reporting system than the global state system currently used in instant messengers, by allowing presence states to be assigned to individual groups, rather than to all users.

Contents

Working documents: <http://www.lancs.ac.uk/~taylorn4>

1. Introduction.....	11
1.1 Background and Motivation.....	11
1.2 Project Aims and Objectives	13
1.3 Development Strategy.....	13
1.4 Overview	13
2. Background and Related Work	15
2.1 Enabling Technologies.....	15
2.1.1 Mobile Internet	15
2.1.2 Bluetooth.....	15
2.1.3 .NET Messenger Service.....	16
2.2 Mobile Operating Systems.....	17
2.2.1 Microsoft Windows Mobile	17
2.2.2 Symbian.....	19
2.3 Mobile Development Platforms	19
2.3.1 Microsoft .NET Compact Framework.....	19
2.3.2 Java Micro Edition.....	20
2.4 Existing Systems	20
2.4.1 Official Messenger Clients.....	20
2.4.2 Multi-Protocol Clients: Trillian	21
2.4.3 Bluetooth Messaging: MobiLuck	23
2.4.4 Active Bluetooth Services: Hypertag.....	23
2.4.5 Other Bluetooth Applications.....	24

2.5 Summary	25
3. Requirements and Design	26
3.1 Requirements.....	26
3.2 Design	27
3.2.1 Network Layer.....	27
3.2.2 Data Layer	28
3.2.3 User Interface Layer	29
3.3 Messenger Service	29
3.3.1 Connection.....	30
3.3.2 Switchboards	31
3.3.3 Advanced Status Protocol	32
3.4 Bluetooth Messaging	33
3.4.1 Connections	33
3.4.2 Presence Protocol.....	34
3.4.3 Messaging Protocol.....	34
3.5 Summary	35
4. Implementation	36
4.1 Networking Layer.....	36
4.1.1 Messenger Notification Server	37
4.1.2 Messenger Switchboard Sessions	38
4.1.3 Bluetooth Networking.....	39
4.2 Data Layer.....	40
4.2.1 Messenger.....	40
4.2.2 Contact List	41
4.2.3 Conversations	43
4.3 User Interface Layer	44

4.3.1 Login	44
4.3.2 ContactViewer	44
4.3.3 ConversationWindow	45
4.4 Summary	46
5. Operation	47
5.1 User Login.....	47
5.2 Contact List Operation.....	48
5.2.1 Managing Contacts	49
5.2.2 Managing Groups	50
5.3 Conversation Screens.....	52
5.4 Summary	53
6. Testing and Evaluation	54
6.1 Black Box Testing	54
6.1.1 Sign-In Procedure	54
6.1.2 Messenger Services.....	55
6.1.3 Messenger Conversations.....	56
6.1.4 Bluetooth	57
6.2 Performance Testing.....	58
6.2.1 Messenger Connection	58
6.2.2 Messenger Contact List.....	60
6.2.3 Bluetooth Service.....	61
6.3 Usability Testing	61
6.3.1 Instructions	62
6.3.2 User Responses.....	62
6.4 Summary	64
7. Conclusions.....	66

7.1 Summary	66
7.2 Overall Conclusions.....	66
7.3 Possible Improvements and Future Developments	68
7.3.1 Extensibility and Additional Services.....	68
7.3.2 User Interface Improvements	68
7.3.3 Cross-Service Conversations.....	69
7.3.4 File Transfer	70
7.3.5 Bluetooth Mesh.....	70
7.3.6 Further Presence Enhancements	71
References.....	72
Appendix A: Project Proposal	73
Appendix B: Messenger Commands.....	82

Index of Figures

Figure 2.1: .NET Messenger communication.....	17
Figure 2.2: Suggested standard layout for Smartphones.....	18
Figure 2.3: Official Messenger clients.....	21
Figure 2.4: Configuring different services in Trillian.....	22
Figure 2.5: Trillian contact list.....	22
Figure 2.6: An advert utilizing Hypertag.....	24
Figure 3.1: System architecture.....	27
Figure 3.2: Conversation records.....	29
Figure 3.3: Connecting to Messenger.....	30
Figure 3.4: Example Advanced Status Protocol messages.....	33
Figure 3.5: Example Bluetooth session.....	34
Figure 3.6: Example Bluetooth message.....	35
Figure 4.1: High-level class diagram.....	36
Figure 4.2: Data layer class diagram.....	40
Figure 4.3: Contact and ContactGroup data classes.....	42
Figure 4.4: Conversation interface.....	43
Figure 5.1: Login screen and options.....	47
Figure 5.2: Connecting to Messenger and Bluetooth services.....	48
Figure 5.3: Contact list and options menu.....	48
Figure 5.4: General Messenger options: changing display name and presence.....	49
Figure 5.5: Contact menu and contact editing dialogs.....	50
Figure 5.6: Messenger group options.....	51
Figure 5.7: Custom group presences.....	51

Figure 5.8: A conversation viewed in SmartMessenger and Microsoft Live Messenger.	52
Figure 5.9: Conversation options and the conversation menu.	52
Figure 5.10: Bluetooth conversations.	53
Figure 5.11: Presence reports in conversations.	53
Figure 6.1: Messenger connection performance testing results.	59
Figure 6.2: Comparison of connection phases between clients.	59
Figure 6.3: Usability testing in progress.	62
Figure 7.1: Cross-service messaging.	69
Figure 7.2: A Bluetooth mesh network.	71

Index of Tables

Table 3.1: System requirements.....	26
Table 4.1: MessengerServer events and causes.	38
Table 4.2: MessengerSwitchboard events and causes.....	39
Table 4.3: Bluetooth networking events and causes.	40
Table 4.4: Messenger class events.	41
Table 4.5: Contact and ContactGroup manipulation methods in ContactList.....	43
Table 6.1: Test cases and results for service sign-in.....	55
Table 6.2: Test cases and results for Messenger functionality.	56
Table 6.3: Test cases and results for Messenger conversations.....	57
Table 6.4: Test cases and results for Bluetooth messaging.	57
Table 6.5: Messenger connection performance testing results.....	59
Table 6.6: Contact list download times for different accounts.....	60
Table 6.7: Detection time test results.	61

1. Introduction

The aim of this project is to develop a prototype instant messenger client which will run on a smartphone platform, which will improve on existing client functionality available for mobile devices, and take advantage of features not usually found in the PC platform where instant messengers originated.

The client will be able to connect to both the .NET Messenger [1] service and a peer-to-peer Bluetooth [2] service which will be developed as part of the project.

In addition, the project aims to extend the presence functionality of Messenger beyond simply reporting a single status to all contacts, and allow different statuses to be assigned to different groups of contacts. The protocol for statuses will be potentially usable by other clients.

Particular attention will need to be paid to the usability of the system, as the constraints on a mobile phone's input devices can potentially lead to confusing user interfaces. The system will need to be intuitive both for users of the platform and users of similar clients, as well as users who are new to both.

1.1 Background and Motivation

Instant messaging allows two or more computer users in separate locations to send messages to each other in real time, essentially holding an online conversation. Like email before it, instant messaging has become a vital communications tool, used by both businesses and home users.

Whereas email is an asynchronous interaction, best equated to the real-world postal system—you send a message, but do not know when it will be received, and may have to wait some time for a reply, instant messaging is better likened to a telephone conversation. When you try to initiate a conversation, you know if the other person is there or not and conversations can be conducted synchronously.

The first instant messenger was ICQ [3], a freeware launched in 1996, which quickly found widespread use. The basic concept involves a server and any number of client applications. The ICQ service simply stores client IP addresses and port numbers centrally and distributes these to clients so that they can communicate with members

of their 'contact list' on a peer-to-peer basis. Other services use the server as a switchboard, which forwards messages on to their intended recipient, with the obvious security advantage of obscuring their IP address.

Many instant messengers include a variety of other services, such as voice and video messaging, SMS, and games. These services will not be addressed in this project.

To be used effectively, instant messengers require a stable internet connection, which is reasonably fast and inexpensive. In the past, this has meant that applications were mostly limited to PCs. However, with the advent of smaller, more powerful mobile devices which improved connectivity, the gaps between mobile phones, PDAs, and fully-fledged PCs are becoming narrower.

For example, between PDAs and computers, tablet PCs now have the computing power and functionality of desktop computers and can easily be taken anywhere. On a smaller scale, so-called 'smartphones' are now building features traditionally found in PDAs into mobile phones. Robust calendar and address book facilities, email, web-browsing and PC synchronisation are just some of the features found in smartphones which were previously confined to larger devices.

As a result, it is now feasible for instant messengers to be run on mobile phones, and some attempts have been made to develop mobile clients for existing services, and to develop new services such as Bluetooth chat.

However, individual clients are not necessarily the best option for mobile devices. Often, a mobile Internet connection will be charged per minute or per megabyte, so a user might want to use more cost-effective connection method, such as Bluetooth, wherever possible. Unfortunately, due to the limited screen resolution and interactivity of mobile devices, it is often difficult to switch between two different applications. Only one window can be viewed at any time, and there may not be an easy interface like the Windows task bar for changing applications.

This project will address this concern by combining a popular Internet-based service and a Bluetooth-based local service into a single client with a common user interface and contact list.

1.2 Project Aims and Objectives

The aims of the project are as follows:

- Develop and implement a protocol for peer-to-peer Bluetooth instant messaging using mobile phones.
- Develop a single messenger application which integrates both the .NET Messenger service and the Bluetooth peer-to-peer service, and provide a common user interface.
- Extend presence functionality in the Messenger system, by allowing the user to select different states for different groups of contacts.
- Improve the usability of the client in comparison to existing instant messenger clients developed for use on mobile phones.

1.3 Development Strategy

The project will use an incremental development strategy, where features of the client will be built up in stages. Development will start with basic connections to the provided services, including maintaining connections and downloading or finding contacts, then add further features such as contact management and status.

Once the system foundations are in place, the messaging service itself and corresponding user interfaces will be added, followed by improved user status settings.

1.4 Overview

The remainder of this report will be structured as follows:

Chapter 2 will be an exploration of platforms, instant messaging technologies and mobile communications technologies which may be relevant to this project, and the potential advantages and disadvantages of the different options. It will also look at similar or related systems, and evaluate their successful features and failings.

Chapter 3 specifies the requirements of the system, the system's overall structure, and the communication architecture that will be used. It also specifies how the client will interact with the server and other clients, and defines the protocols that will be developed for use in the system.

Chapter 4 documents how the design set out in the previous chapter was implemented, and explores particular data structures and algorithms used.

Chapter 5 demonstrates the client in use and highlights features of the user interface, functionality available, and how these features are accessed through the client.

Chapter 6 details the techniques which were used to test the system functionality and ensure that the application performed at a reasonable level. It will also document the results of usability tests carried out with actual end users, their opinions of the application, and improvements which can be made based on their evaluation.

Chapter 7 evaluates the completed system, comparing it to the aims and requirements stated earlier in the document. It will also detail possible enhancements or improvements which could be made to the system, but which were not considered during the design phase, or not implemented due to constraints on time, available technology or scope.

2. Background and Related Work

This section of the report explores technology which is relevant to the project, and existing systems which work on a similar or related premise.

2.1 Enabling Technologies

2.1.1 Mobile Internet

The primary method of Internet connection used in modern mobile phones is GPRS (General Packet Radio Service) [4]. GPRS is a packet-switched standard, meaning that connections are only made as required, and not maintained for the duration of the session—as was the case with the previous popular standard, CSD (Circuit Switched Data) [5]. This is ideal for applications which may only need to transmit intermittently, such as instant messaging, and means that the user is often only charged for the data they download, rather than the amount of time they use the service.

Unused voice bandwidth is used to transmit packets, so transmission speeds are adversely affected by heavy voice traffic in the area, and also by distance from the base station, but bit-rates of 30-80kbit/s are realistic. The use of the phone's regular mobile connection additionally means that GPRS is available whenever they have a signal.

More recently, high-end handsets have become available which provide Internet access using the 802.11g wi-fi standard [6]. While wireless access is not universally available in the way GPRS is, it does provide broadband speeds; typically around 54Mbit/s. Wi-fi connections are increasingly becoming available in public places such as cafes and hotels, with plans in place for city-wide services.

2.1.2 Bluetooth

Bluetooth is a short-range wireless specification designed for networking nearby devices into Personal Area Networks (PANs) or 'piconets'. The range is dependent on a device's class, but can range from only 10cm to 100m.

Popular uses include connection to peripherals such as mice, keyboards and printers in order to reduce clutter from wires. The close proximity and relatively low

bandwidth requirement of such devices means low-powered Bluetooth is an ideal communication medium in these situations, where other wireless specifications (such as 802.11x) would be too powerful.

The specification has become particularly popular in mobile phones and other handheld devices, where it may be used to connect headsets and exchange files with PCs and other devices using the Object Exchange protocol (OBEX) [7].

Active Bluetooth devices can either be 'On' or 'Discoverable'. A discoverable device will transmit data about itself on demand, including its name and available services. Devices which are not discoverable can access nearby devices, but will not provide data about themselves. This information can be used to form connections between devices. They may also form trusted 'pairs', which makes connecting simpler and enables encryption.

On the Windows Mobile platform, the open source 32feet library [8] provides managed code for Bluetooth hardware, including implementations of standard interfaces such as sockets, listeners and connection end points.

2.1.3 .NET Messenger Service

Microsoft's .NET Messenger Service is a protocol and system for Instant Messaging, which deals with presence, notification and messaging for users, and utilises the Microsoft Passport Network [9] for authentication services.

The service has become incredibly popular, partly because users of the free Hotmail service have a Passport account ready to use, and partly through operating system integration. Windows XP had a basic client pre-installed, and the program has long been offered through the Windows Update website.

Unlike ICQ, the main Messenger services do not operate on a peer-to-peer basis, although for performance reasons some newer, advanced features do. Instead of sending messages directly between clients, Messenger relies on switchboard servers to host conversations. Each switchboard operates like a private chat server, to which two or more users can connect. Messages are sent to the switchboard, which forwards them on to all other participants.

The notification server is used to access and modify the user's contact list and monitor the presence of contacts, as well as initiating switchboard sessions.

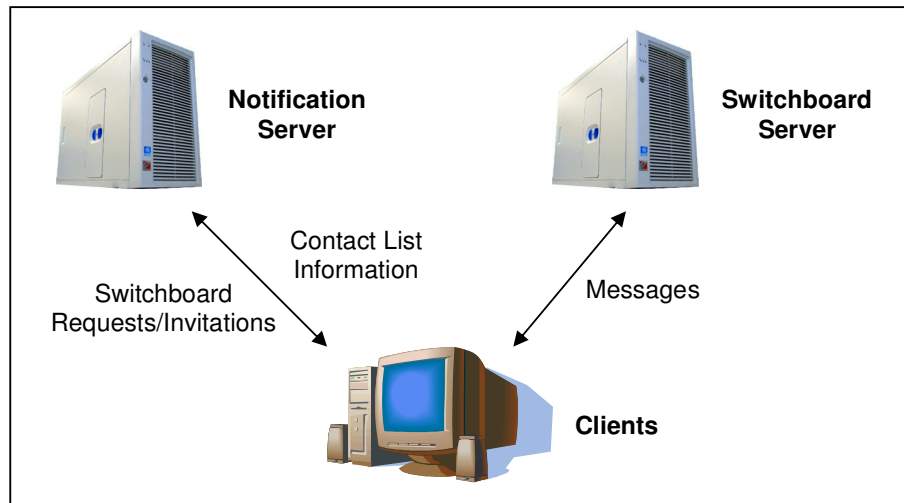


Figure 2.1: .NET Messenger communication.

Messages are sent using the Mobile Status Notification Protocol (MSNP) [10], a proprietary plain-text protocol, over TCP connections, or in some case a HTTP proxy. Messages consist of a three-character code followed by a number of space-delineated arguments, terminated by a new line character. Some commands may be followed by a payload, the length of which is specified in the command arguments. A list of commands is included as Appendix B.

A Messenger client connects to messenger.hotmail.com—the dispatch server—which responds with the address of a notification server. The client and notification server agree on the protocol version to be used and the capabilities of the client, and then the client is required to authenticate itself using a Passport server. Once contacts have been downloaded, the user is able to modify contacts and start switchboard sessions.

The Messenger servers support a number of protocol versions, from MSNP8 upward. The protocol has advanced as far as MSNP13, although public documentation for newer versions is limited.

2.2 Mobile Operating Systems

2.2.1 Microsoft Windows Mobile

Windows Mobile [11] is a compact operating system designed for handheld devices, including Pocket PCs and Smartphones. It has many features familiar to Windows

users, such as the Start menu and stripped-down versions of common applications, including Internet Explorer and Windows Media Player.

Windows Mobile 2003 for Smartphones was created specifically for mobile phones with highly constrained user interaction, particularly where the user would most likely be operating the device with one hand.

For example, Smartphone devices have no touch-screen, and neither physical nor on-screen QWERTY keyboards. Instead, users interact with the system primarily using two 'soft' keys which correspond to two on-screen buttons, a combined four-directional controller and 'select' button, a back key, and a 'home' button. Text input is via a standard 12-button mobile phone keypad.



Figure 2.2: Suggested standard layout for Smartphones.

These strict limitations force developers to follow standard user-interface designs, which in turn mean the end user will be able to learn to use an application quickly and easily [12].

Recently, Microsoft has released an updated Windows Mobile 5.0. This release has improved performance and battery life, as well as offering more multimedia applications and support for Bluetooth hardware.

2.2.2 Symbian

Symbian [13] is another smartphone operating system developed by Symbian Ltd. Designed with resource conservation in mind, Symbian closely monitors memory usage, and attempts to save battery life by switching off the CPU when not in use.

It provides a number of system libraries for various common systems tasks like file handling, and a comprehensive networking and communications subsystem.

The operating system itself does not provide any actual user interface, but instead relies upon platforms developed by third parties, such as Nokia and UIQ Technology, based upon Symbian's user interface framework. This allows Symbian to be used in various form factors and resolutions, and for devices to have their own unique appearance. However, it also means a user of one Symbian device will not necessarily be able to quickly grasp the interface of a device with a different user interface platform.

2.3 Mobile Development Platforms

2.3.1 Microsoft .NET Compact Framework

The .NET Framework [14] is Microsoft's modern, object-oriented software development platform, designed primarily for Windows. Amongst other things, the Framework provides a Common Runtime Engine for platform independence, a Common Language Infrastructure to allow software to be written in any one or more of the .NET languages, and a Base Class Library of classes which encapsulate common functionality.

A subset of the .NET Framework designed to run on the Windows Mobile platform is known as the Compact Framework. This was developed to provide similar features on devices with constrained resources and interaction. As such, it does not contain all the classes and functionality of the full framework, and also includes classes designed specifically for mobile devices which are not found in the full framework.

The OpenNETCF [15] library attempts to address certain limitations of the Compact Framework, and will be used in this project where required.

2.3.2 Java Micro Edition

Java ME [16] is a runtime and API collection created by Sun for devices with constrained resources, including mobile devices. The platform comes in a number of different profiles, each implementing a slightly different subset of the full Java API and providing different specialised APIs.

The profile which is implemented by most modern mobile phones is the Mobile Information Device Profile, which runs a special type of Java application called a MIDlet, packaged into a JAR file for distribution.

The main advantage of Java is what a MIDlet should run on any MIDP implementation, meaning it is not limited to one device or operating system. As a disadvantage, this separation from the operating system means that the user interface is not necessarily what the user is used to, and that interaction with the device hardware may be more limited.

2.4 Existing Systems

Many solutions which involve similar ideas to this project have already been developed both on and off the Smartphone platform. These include various forms of Messenger client, Bluetooth messengers, multi-protocol clients, and other Bluetooth applications. This section of the report will explore several of the most relevant applications.

2.4.1 Official Messenger Clients

Microsoft produces several official clients, the most popular being MSN Messenger. A similar but simpler version called Windows Messenger was distributed along with Windows XP, but development has subsequently ceased. More recently, the MSN Web Messenger was released, which is a browser-based version designed to be run anywhere. Most relevant to this project, however, is the client provided preinstalled on Smartphone devices.

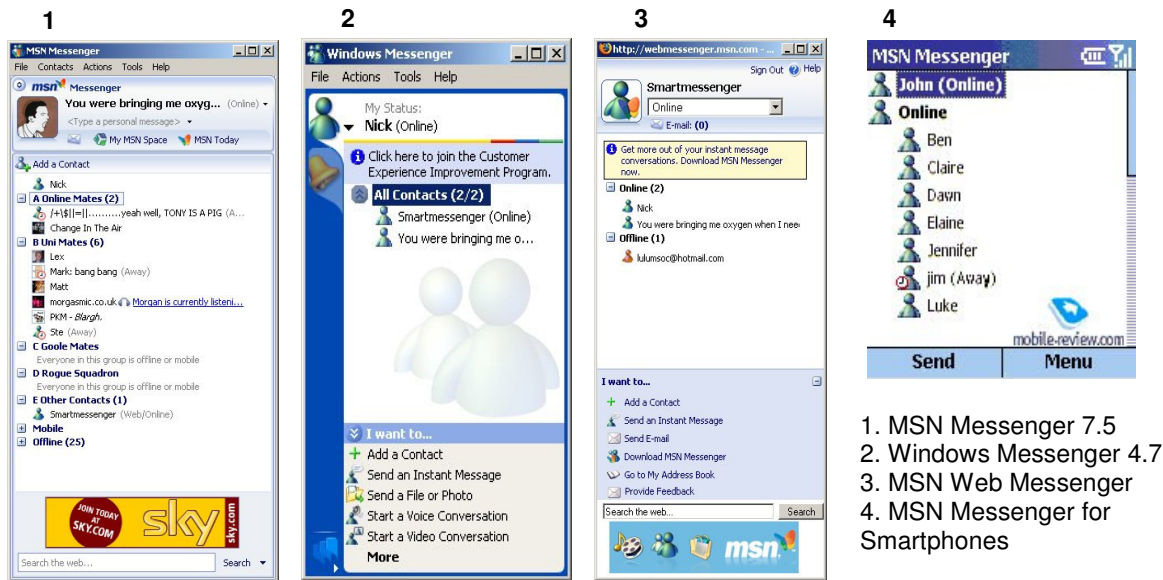


Figure 2.3: Official Messenger clients.

Standard clients rely heavily on user interface elements which are not present on a mobile phone, such as the Windows taskbar. On a desktop client, the taskbar is used to navigate between conversation windows and indicate receipt of a message in a non-focussed window.

This is also a disadvantage to the web client, which is unable to create ‘toaster’ pop-ups on the task bar like the standalone clients do. The web client attempts to overcome this by providing a list of open conversations at the top of the contact list.

2.4.2 Multi-Protocol Clients: Trillian

Trillian [17] is a well-known chat application which is able to connect to many common chat services at once, including MSN, Yahoo, ICQ and AIM, as well as IRC chat. It has remained popular despite various attempts by IM networks to block it, and is now available both as freeware and an enhanced ‘Pro’ version. An API is also provided for developing plug-ins to further extend the client’s functionality.

The client’s most noteworthy features in regards to this project are visual methods of displaying the status of various services, and displaying which contacts belong to which service.

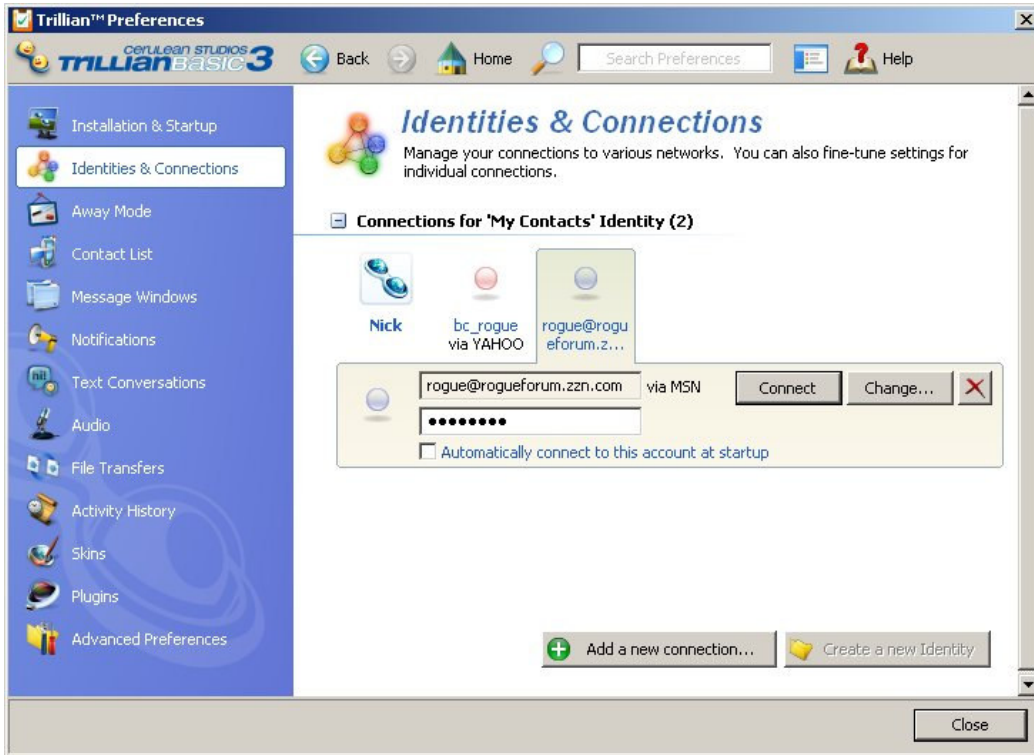


Figure 2.4: Configuring different services in Trillian.

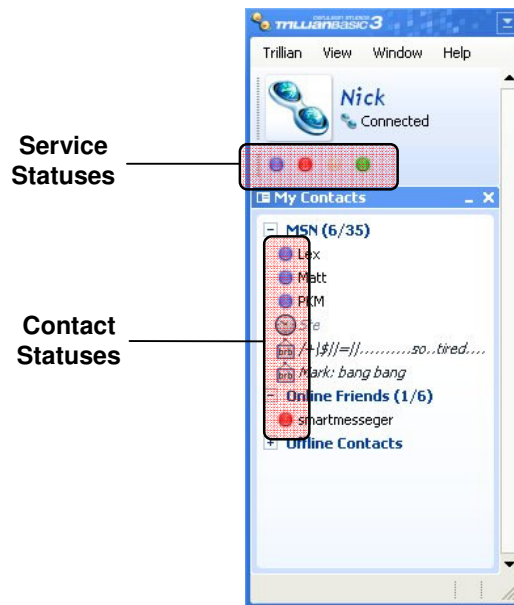


Figure 2.5: Trillian contact list.

Users can enter account details for the various services (see Figure 2.4), which can all be connected at once on start-up if the user chooses. Figure 2.5 shows the main

contact list, which displays all the user's contacts and the current status of the services they have installed.

Each service is colour coded, with a blue status indicator that lights up when the .NET Messenger service is connect, a red light for Yahoo, yellow for AIM and green for ICQ. Individual contacts' status indicators are coded with the same colours, so the user can quickly tell which service the contact belongs to.

2.4.3 Bluetooth Messaging: MobiLuck

Few mobile phones have any form of Bluetooth chat program pre-installed, although sending messages via ad-hoc methods has become popular. Usually this is done by 'Bluejacking', which involves creating an address book entry containing the message and sending it to another phone. Bluetooth services already exist to make this kind of exchange simple and are supported by most mobile phones.

Many actual clients are available, most of which are freeware applications. These range from simple clients which do little more than place a wrapper around the 'Bluejacking' method, to those which implement their own chat protocols. This does mean that different clients are incompatible, which has prevented their use becoming widespread in a field as diverse as mobile computing.

One relatively popular example is MobiLuck [18], which detects Bluetooth devices in the area and allows you to send them a message. MobiLuck works more like SMS messages or email than a true instant messenger, sending one-off messages which often only consist of a phone number for the receiver to call if they wish to meet the sender.

2.4.4 Active Bluetooth Services: Hypertag

Through the popularity of mobile phones, Bluetooth technology has proliferated across the country, particularly to 18-30 year olds with disposable income. This is a lucrative market segment, and heavily targeted by advertisers. Consequently, technologies have arisen to advertise using Bluetooth, on the basic principal that a discoverable device is sent a message or file when they pass within range of an advertising device.

One example which has already been installed in cinemas around the UK is Hypertag [19] technology, where a small Bluetooth device is attached to a film poster and seeks

out nearby devices, then attempts to initiate transfer of a film trailer or other promotional material.



Figure 2.6: An advert utilizing Hypertag

Where most Bluetooth services are passive—the user’s device seeks out a printer rather than the other way round—the Hypertag service actively connects to the user and initiates communication. A Bluetooth chat service, lacking a central server, will need to operate in much the same way by finding and connecting to other clients, although it will still need to accept incoming communications itself.

More importantly, Hypertag demonstrates the willingness of organisations and end users to embrace Bluetooth technology. Businesses see it as a legitimate, usable technology, and mobile phone owners are not afraid to make their device discoverable and accept connections from unknown devices.

2.4.5 Other Bluetooth Applications

Despite Bluetooth technology being readily available to millions of mobile phone owners and completely free to use, surprisingly few widespread applications have arisen to take advantage of this, with the exception of advertising technologies discussed in section 2.4.4.

Several research projects have devised novel applications for Bluetooth devices, in both professional and entertainment domains.

The MobiLenin System [20] is aimed at allowing interaction between public displays and multiple users—and consequently between the users themselves—particularly in

a social context. The prototype system allows users to communicate with a server to vote for one of several choices of music video, and the video with the most votes is played. To encourage interaction, one user is randomly chosen to win a prize, such as a free drink.

A similar system has been designed to allow interaction with public displays situated outside offices [21]. The displays allow owners to leave messages for visitors, indicating that they are out of the office, or that they should not be disturbed. Visitors can likewise leave messages saying they visited while the owner was away, or leave contact details. Originally, the displays were touch screen devices which visitors would physically write on, but the system has been expanded to allow Bluetooth interaction, where the visitor's mobile phone acts as the interface for the system.

2.5 Summary

This chapter has explored the problem domain and found that there is potential for instant messengers to expand into mobile computing, but that current efforts have not necessarily been well-tailored to the platform.

Due to the speed and costs involved in mobile data connections, access to more than one communication method could prove useful. User interface constraints on mobile devices mean that integrating these services into a single client would greatly improve the usability of the device as an instant messenger.

3. Requirements and Design

Based upon the background research from the previous chapter, this section will define requirements for the project, and detail a system design to meet these requirements. The end of the chapter will specify the custom instant messaging protocols that the system will use.

3.1 Requirements

The requirements for the SmartMessenger system, as identified by background research, are listed below:

ID	Description
R1	The client should connect to .NET Messenger and Bluetooth instant messenger services both simultaneously and individually.
R2	The user interface must be intuitive to new users, and familiar both to users of existing clients and users of other software on the platform.
R3	Software should allow users to execute operations quickly and easily, without passing through too many dialogs.
R4	The application should allow the user to send and receive messages through the Messenger service, both to individual users and group conversations.
R5	The application should allow the user to send and receive messages through Bluetooth peer-to-peer connections.
R6	The client should display a contact list containing both Messenger and Bluetooth contacts on a single screen, and allow the user to edit their Messenger contacts.
R7	The client should be fully compatible with the Messenger service, and should be able to communicate flawlessly with official clients.
R8	User must be able to hold numerous conversations at once, and switch easily between them.

Table 3.1: System requirements.

3.2 Design

The system will be comprised of a single client, which will allow the user to connect to the .NET Messenger service and communicate with other nearby clients using Bluetooth. The client will also allow the user to display different statuses to different contacts, based upon which contact list group they are in.

The system will have a layered architecture, split into three distinct layers: network, data and user interface. Each layer will only be aware of and be able to perform operations on the layers below it. Upwards communication will occur through a publish-subscribe model, where upper layers will subscribe to be notified of events occurring in the lower layers.

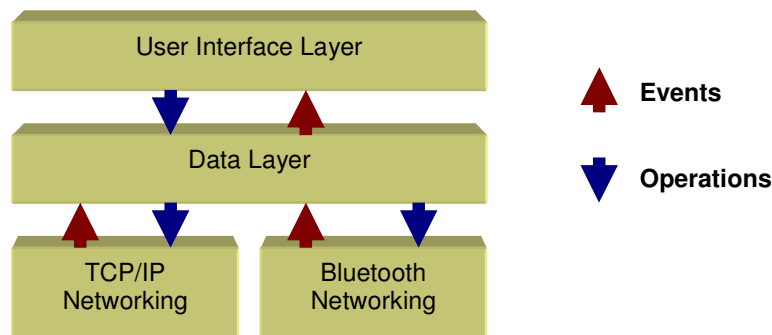


Figure 3.1: System architecture.

3.2.1 Network Layer

The network layer will handle all the sockets and connections required to run the client and all functions related to them, including connecting, disconnecting, and reading and writing from the socket. It will also be able to independently carry out certain tasks which do not require knowledge of the system state and discard commands which are not relevant to the system, making these tasks transparent to the layers above.

For example, the Messenger service occasionally requests that a client computes and returns a hash code based on a random number in order to ensure the client is active. This response will not affect the state of upper layers, and can be carried out by the network layer without any interaction.

The network layer will be split into two distinct segments, one for TCP/IP connections to the Messenger service, and one for Bluetooth connections. At the network level these services need no interaction with each other, so these modules will be completely separate.

3.2.2 Data Layer

The data layer will maintain the state of the messenger services, and provide a unified abstraction over the two network layers. It will also hold information about the user, such as login-in credentials, display name and presence.

Any changes to this layer will actually be formatted into requests for the network layer and sent over the appropriate service. In many cases, such as changing display name or status, the data layer will not actually be updated until the Messenger server has confirmed the change. This will prevent possible discrepancies between the data layer and server in cases where the server rejects the change, or where the change cannot be completed due to connection errors.

This layer will also maintain the contact list, containing records for contacts and groups from the Messenger service and local Bluetooth devices which the messenger has formed connections to. As with other functions, changes made to the data structures which store the contacts will be sent to the server for confirmation before the data is actually updates.

The data layer will also manage conversations, creating new conversation records whenever the user tries to send messages to a contact, and whenever a remote client initiates a conversation through the notification server or Bluetooth. Each conversation record will contain list of members and the conversation history. If the conversation is active, it will also contain a reference to the switchboard in use.

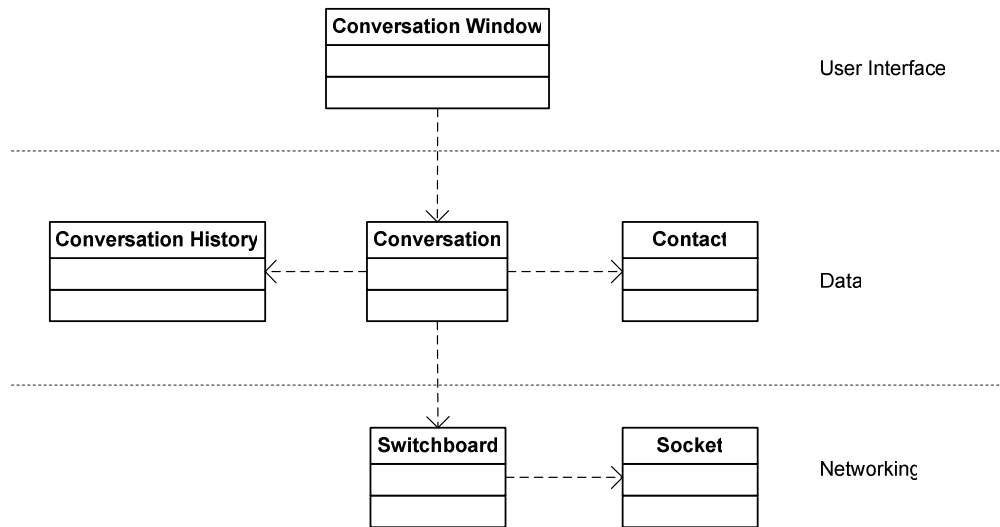


Figure 3.2: Conversation records.

The conversation manager will need to instruct the network layer to request and connect to new switchboard sessions as required.

3.2.3 User Interface Layer

The user interface layer will be responsible for maintaining the user interface itself, and translating user actions into operations on the data layer. It will monitor the data layer for changes and react appropriately, for example, reflecting the current stage of the connection procedure on the login screen and updates the contact list.

The data and user interface layers will share responsibility for input validation. For example, the user interface will limit the number of characters a user can enter for a display name to the number the server allows, but the data layer will also check the length of these values before sending them to the server.

3.3 Messenger Service

On connection, the system will make a connection to the Messenger service first, as this is the more complex of the two services, and the most likely to fail during the connection phase.

A full list of Messenger commands used throughout this section is included as Appendix B.

3.3.1 Connection

Before the client is fully signed in to the Messenger service, communication is essentially synchronous, with the client initiating communication and waiting for a response before continuing. To connect to the Messenger service, the system will connect to the dispatch server at messenger.hotmail.com:1863, sending a 'VER' command which identifies the protocol version to be used.

The system will use MSNP9, as modifications made in subsequent versions have been largely designed to support functionality that the system will not be capable of.

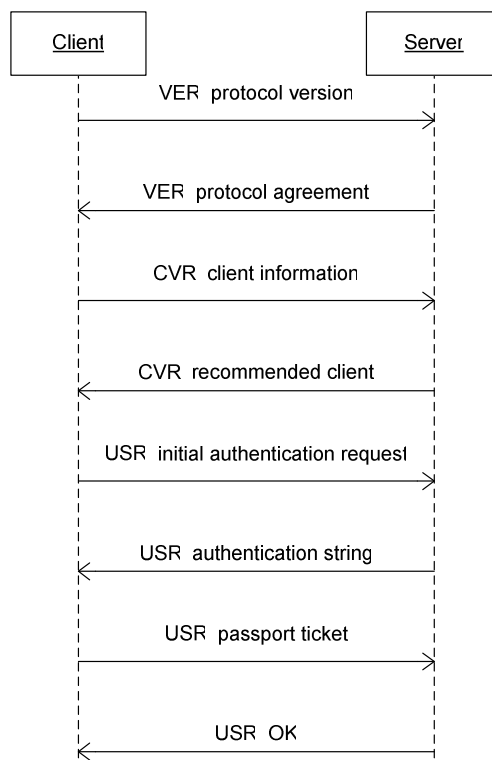


Figure 3.3: Connecting to Messenger.

After a 'VER' response is received, the client will send a 'CVR' command detailing the client operating system, processor architecture, and client program and version, to which the server responds with a recommended up-to-date client program. This is used to identify availability of upgrades and will be ignored by this client.

The client will then send an initial 'USR' command requesting authentication. The dispatch server always responds with an 'XFR' command specifying the address of a notification server to connect to.

The system will then disconnect, open a connection to the specified server and repeat this process. Unlike the dispatch server, the notification server will respond to the initial 'USR' command with a long authentication string. The client will then form a connection to the Microsoft Passport Network, send a request using this string, and receive a token which can be used in a subsequent 'USR' command to the notification server.

The client will then continue to receive and interpret commands asynchronously from the notification server until the server sends an 'OUT' command, or the connection closes. As detailed in section 3.2, some commands will be dealt with at the networking level, while others will be passed up to the data level to process. Some commands may be ignored if they are irrelevant to the client, including those relating to the Hotmail email service.

If any stage of the sign in process fails, the entire sign in operation will be aborted and the Bluetooth service will never be started. The user will be returned to the login screen. Otherwise, the Bluetooth service will be started if required, and the contact list will be launched.

3.3.2 Switchboards

Switchboard servers are used to host a single Messenger conversation each, with two or more participants. Switchboard sessions initiated by the user and sessions set up by other clients to which the user is invited are handled differently, and the system will need to be able to handle both these situations.

Requests to start a conversation with a contact will be sent from the user interface to the data layer, which stores a list of existing conversations. If a conversation with the desired contact already exists and is connected, then it will be returned, otherwise the client will send an 'XFR' command to the server to request a new session. This check will need to be carried out every time a message is sent, as conversations can time out after periods of inactivity.

The server will respond with the address of a switchboard server and an authentication string. The client will connect to this server in a different thread and authenticate itself before inviting the desired contact to join the session. Any messages sent before the server is connected and the contact has joined will be cached at the data level and sent once connection is complete.

Invitations to join conversations initiated by other clients may potentially include one or more existing participants, but the client isn't able to detect this until the switchboard is fully connected. As such, it is not possible to immediately assign the switchboard session to an existing conversation record.

When the client receives an 'RNG' command detailing an existing switchboard session to join it will connect, then wait till details of all participants have been received before advertising its status as 'connected' to the data layer. Until it has been assigned to either an existing conversation or a new conversation the switchboard will not process any incoming messages.

3.3.3 Advanced Status Protocol

This protocol will be required to send custom user presence notifications to members of the Messenger contact list, where different from the main status. As the Messenger service only allows direct communication between clients using switchboard servers, statuses will need to be sent this way.

All message payloads sent through switchboards contain a standard MIME [22] header, which is used to determine the type of the incoming message. Official clients discard any messages with an unknown content type, so a custom content type will be used to ensure status messages are not interpreted by unsupported clients.

Any time a contact comes online, the client will open a switchboard and send a message with the content type 'SmartMessenger/handshake'. Any time the client receives a message with this type, it should return a message with the content type 'SmartMessenger/echo'. This allows any clients using the status protocol to identify each other, and thus avoid having to make redundant switchboard sessions to send status messages to clients which do not understand them.

Once a contact has confirmed that it can understand the protocol, any current status on the contact's group is transmitted using a message with the content type 'SmartMessenger/status'. The body of the message should be a regular Messenger status code, excluding HDN. An empty body indicates that the custom status has been removed, and the client should listen to the notification server for future status updates.

```
Handshaking

MSG 1 U 63\r\n
MIME-Version: 1.0\r\n
Content-Type: SmartMessenger/handshake\r\n
\r\n\r\n

MSG 1 U 58\r\n
MIME-Version: 1.0\r\n
Content-Type: SmartMessenger/echo\r\n
\r\n\r\n

Changing Status to 'Away'

MSG 1 U 61\r\n
MIME-Version: 1.0\r\n
Content-Type: SmartMessenger/status\r\n
\r\n
AWY

Removing Custom Status

MSG 1 U 60\r\n
MIME-Version: 1.0\r\n
Content-Type: SmartMessenger/status\r\n
\r\n\r\n
```

Figure 3.4: Example Advanced Status Protocol messages.

3.4 Bluetooth Messaging

The Bluetooth messenger section of the client will be required to manage its own connections and will need a protocol for communications. In order to maximise similarities between the Messenger and Bluetooth components, the Bluetooth protocol will be similar to MSNP. It will be text-based, using three-character command codes followed by space-delineated arguments and terminated by a new line character.

3.4.1 Connections

Any device running the client will be made discoverable, so that other devices can detect it. Clients will need to constantly try to discover new devices, and listen for incoming connections from other devices.

When the client finds a new nearby Bluetooth device, it will check the device's unique address and connect only if the remote device's address is a lower value than the local device's address. This increases the likelihood of a successful connection, as Bluetooth connection attempts often fail when both devices attempt to connect simultaneously. It also avoids multiple redundant connections being made, as given any pair of devices, only one will attempt to connect to the other.

The client will accept any incoming connections, regardless of the remote device address. If the client already has an active socket for the remote device, it will place it in reserve for use when the first socket closes.

3.4.2 Presence Protocol

Once a Bluetooth connection has been established between two devices, the device that opened the connection is required to initiate conversation. This is done by sending a single HEY command with no arguments, which the second device should echo back.

Both devices should now send an ILN command with a single argument, which is one of the Messenger presence codes, excluding HDN. The absence of a central server capable of hiding a contact's presence means that the Hidden status can not be implemented. This handshaking procedure needs to be repeated for every connection that is made.

Once connected, each device can notify others of a change in status by sending an NLN command with a Messenger presence code.

If either client wishes to close the connection, it should send an OUT command. This means the second client can immediately report the disconnected device as having gone offline, rather than waiting until the socket actually closes.

Client 1	Client 2	Comments
HEY		Client 1 initiates conversation
	HEY	Client 2 responds
ILN NLN		Client 1 sends initial status 'online'
	ILN AWY	Client 2 sends initial status 'away'
	NLN NLN	Client 2 changes status to 'online'
OUT		Client 1 disconnects

Figure 3.5: Example Bluetooth session.

3.4.3 Messaging Protocol

Messages follow the same format as MSNP messages, with the exception of the sender argument. This argument is not required, as connections are dedicated to one device only.

The message command is MSG, followed by a single argument specifying the length of the payload in bytes. This is terminated by a new line as usual, but then

immediately followed by the message payload, which is not terminated by a new line. The client uses the length argument to read the correct number of bytes from the socket.

```
MSG 12\r\n  
Hello World!
```

Figure 3.6: Example Bluetooth message.

3.5 Summary

This chapter has explored the design for the prototype system, including protocol designs, expected interaction with the Messenger service, and a three-tier system architecture designed to obscure unnecessary detail from upper layers. This design will be followed when implementing the system and used when designing test cases.

4. Implementation

This section will discuss the implementation of the prototype system, exploring the classes used within the system and their interactions. The system has been implemented in C#, using the .NET Compact Framework on Windows Mobile Smartphone 2003.

The implementation makes extensive use of events, which essentially add support at the language level for the publish-subscribe model used throughout the design, rather than leaving the developer to implement it (*cf.* Java). Any object can link an ‘event handler’ method to the event, which is invoked any time the original object launches the event.

This rest of this chapter will examine the implementation of each layer, the main classes shown in Table 4.1, and other classes used.

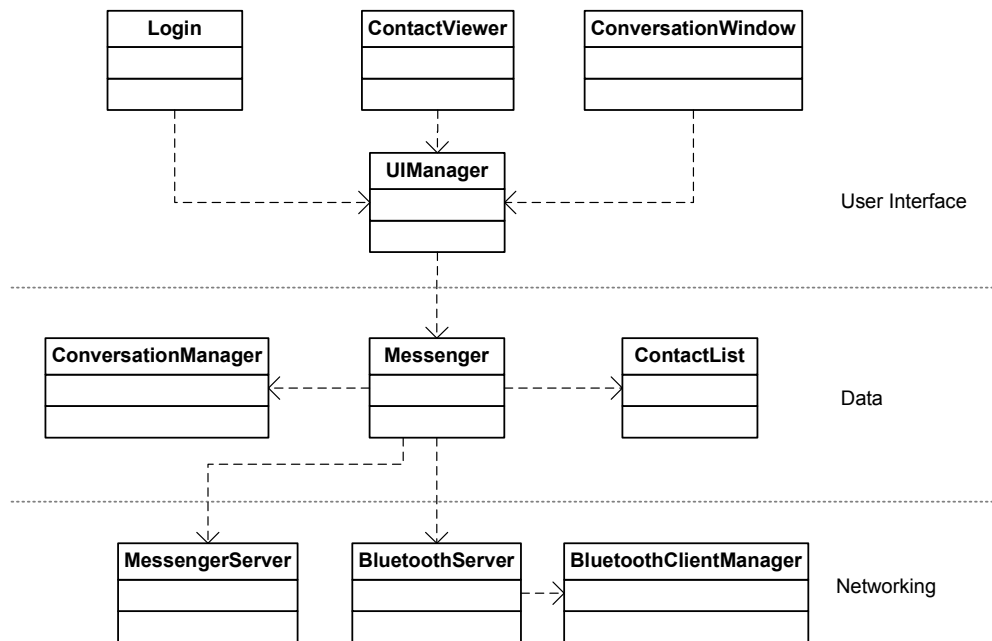


Figure 4.1: High-level class diagram.

4.1 Networking Layer

The networking layer is the bottom-most layer of the system architecture, and is responsible for maintaining and managing actual sockets connected to remote servers and devices.

4.1.1 Messenger Notification Server

TCP connections to the despatch and notification servers are managed by the MessengerServer class. This class consists of methods for connecting, disconnecting and sending commands, and events which correspond to incoming commands from the server which need to be handled at a higher level, such as regular messages, or changes of contact status. The object maintains an enumerated value of its connection status, which may be Disconnected, Connecting, Connected or Disconnected.

MessengerServer is used to communicate with the both despatch and notification servers, as communications with both types of server is the same. When the user signs in, the Connect method is called, passing their Passport address and password as arguments. The Connect method is non-blocking, but in order to prevent simultaneous connection attempts, can only be called successfully when the current connection status is Disconnected. Successful calls start a new Thread, which executes a private connection method.

The first time a connection is started, the MessengerServer does not have the address of a notification server, and defaults to the dispatch server address at messenger.hotmail.com. The method then enters a loop which continues until the client disconnects, breaking up commands into MessengerEventArgs which contain the command type and an array of arguments.

When the dispatch server provides an address for a notification server, this address is stored in an internal variable and the method calls itself, this time connecting to the new address. This can potentially happen numerous times if, for example, the original notification server is overloaded and forwards to a second server. In practice, this rarely happens.

The MessengerServer class is able to handle server transfers, authentication and challenge responses without any interaction with upper layers. Any other commands from the server prompt an event to be launched, as shown in Table 4.1.

Name	Cause
ConnectionError	The connection is closed without the user signing out.
ConnectionStatusChanged	The status of the server changes, for example, disconnecting.
ContactAdded	Details of a new contact have been received.
ContactChanged	A contact's status or display name has changed.
ContactRemoved	Contact has been removed from the contact list.
ConversationInvite	User is invited to a switchboard session.
DisplayNameChanged	A change of name is confirmed by the server.
GroupAdded	Details of a new group have been received.
GroupRemoved	Deletion of a group is confirmed.
GroupRenamed	Renaming of a group is confirmed.
StatusChanged	A change of user status is confirmed by the server.
SwitchboardReady	Address for a requested switchboard session has been received.

Table 4.1: MessengerServer events and causes.

4.1.2 Messenger Switchboard Sessions

The MessengerSwitchboard class is used to manage a TCP connection to an individual switchboard session server. It is constructed much like the MessengerServer class, with Connect, Disconnect and command-sending methods, a connection status property, and events to indicate server commands have been received.

However, switchboard connections are considerably simpler than notification server connections, requiring little authentication and with only six possible incoming commands.

A new MessengerSwitchboard object is created at the data layer any time the address of a switchboard session is received from the notification server, and is configured on construction with the server address, the authentication string received from the notification server and, optionally, a session ID for joining existing sessions.

Unlike MessengerServer, the switchboard does not immediately start processing messages. The Wait property is used to indicate that the switchboard has not yet been assigned to a conversation in the data layer, and any messages received would be lost. This property is modified to 'false' by the data layer once assignment to a conversation is complete.

Name	Cause
Connected	Connected to server, and contacts can be invited in.
ContactAdded	Contact has joined the session.
ContactLeft	Contact has left the session.
Initiated	All contacts are present, and the switchboard can be assigned to a conversation.
MessageReceived	A plain text message has been received.

Table 4.2: MessengerSwitchboard events and causes.

4.1.3 Bluetooth Networking

Bluetooth networking is handled by three classes: `BluetoothServer`, `BluetoothClientManager` and `BluetoothSwitchboard`. These classes rely on the 32feet.NET library, which provides managed Bluetooth interaction for the .NET platform.

As the Bluetooth service is operating on a peer-to-peer basis, rather than the client-server model employed by Messenger, the `BluetoothServer` class is implemented differently to its Messenger counterpart. Rather than forming a connection, the `BluetoothServer` runs two threads, one which constantly attempts to discover nearby devices, and one which listens for incoming connections from devices. As discussed in section 3.4.1, the client will only attempt to connect to a discovered device if the remote device's address is a lower value than the local device's address.

Any connections made, either incoming or outgoing, are passed to the `CreateSwitchboard` method of the `BluetoothClientManager`.

`BluetoothClientManager` is responsible for maintaining `BluetoothSwitchboard` instances, using arrays of `BluetoothSwitchboards` and corresponding addresses. When a connection is passed to the manager, it checks the address against those it has stored. If no match is found, a new switchboard is created and the address and switchboard are added to the arrays. If a match is found, the new connection is passed to the corresponding existing switchboard.

The `BluetoothSwitchboard` class operates much like its Messenger counterpart, connecting to a socket and looping to receive incoming commands until it is told to disconnect.

Name	Cause
BluetoothServer.BluetoothContact	A new connection has been formed with a remote device.
BluetoothServer.BluetoothError	An error occurred with the Bluetooth service.
BluetoothServer.ConnectionStatusChanged	The Bluetooth service has connected or disconnected.
BluetoothSwitchboard.MessageReceived	A message has been received.
BluetoothSwitchboard.StatusChanged	The remote device has changed its status.

Table 4.3: Bluetooth networking events and causes.

4.2 Data Layer

The data layer is responsible for maintaining a unified, abstract representation of the service states, including Messenger service settings, the contact list, and any open switchboard connections for either service.

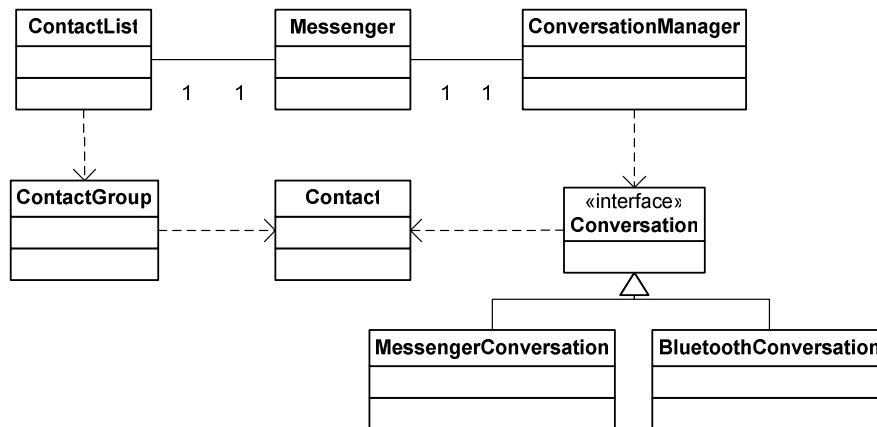


Figure 4.2: Data layer class diagram.

4.2.1 Messenger

The Messenger class forms the core of the data layer, representing the actual service connections and the user's account details and settings. It has methods for connecting and disconnecting, and properties for changing the required services, Messenger display name, and general status.

The Services property is an enumerated value, with values for Messenger only, Bluetooth only, or both services. This can only be changed when the service is fully disconnected. The class maintains two different presence status values, one universal state and one for Bluetooth alone.

The Connect method acts differently depending on the Services option. If both are selected the MessengerServer will be connected first, and the Messenger waits for

connection to complete before starting the Bluetooth service. If only Messenger is selected, the Bluetooth service is never started, and likewise Messenger is not started if only Bluetooth is selected.

When the Messenger service is in use, the `DisplayName` and `Status` properties do not immediately modify the internal state, but cause the Messenger to send a command through the `MessengerServer` object, either a REA command to change display name, or an NLN command to change status. The internal value is only actually changed when the server responds with a success message.

Once the status has been successfully changed, the same status is applied to all Bluetooth switchboards unless a separate Bluetooth status has been set. When Messenger is not in use, the status is sent to Bluetooth switchboards immediately. The Bluetooth status can be set independently, in which case it will also be updated immediately.

Name	Cause
Connected	All requested services have been connected.
Disconnected	All services are disconnected.
Disconnecting	The services are disconnecting, and the user interface should be locked.
ServerMessageChanged	The current status message has been changed.
StatusChanged	The user's status has changed.

Table 4.4: Messenger class events.

4.2.2 Contact List

The user's contact list collects both Messenger contacts and Bluetooth connections into a single class, `ContactList`, which uses data classes to represent individual groups and contacts.

Operations are not actually carried out on the `Contact` or `ContactGroup` classes, which are simply data classes, so these classes do not need to access the service connections themselves. Instead, the `ContactList` class has a set of methods for manipulating contacts and groups, into which the data classes are passed as arguments (see Table 4.5). As with the `Messenger` class, changes to the contact list must be made on the server before the change is reflected in the client. Each method performs error checking to make sure the action is legal, and then sends the appropriate command to the server.

The `ContactList` maintains internal lists of all Messenger contacts, all Messenger groups, all Bluetooth connections and all blocked contacts. These can either be accessed using the methods in Table 4.5, or using a set of enumerators provided as properties.

Whenever a contact comes online, the `ContactList` is responsible for determining if it supports the advanced status protocol by creating a new switchboard for the contact. This automatically starts the handshaking procedure.

Each `ContactGroup` is also able to store a `ContactStatus` when a separate presence status has been assigned to it by the user. When this is changed using the `ChangeGroupStatus` method, a presence message is sent to all contacts in the group who have been identified as 'smart'.

Contact objects themselves also store two different presence properties. One represents the presence reported by the notification server, which the other represents a custom status sent by the user, if any.

Contact	ContactGroup
<pre> +Email() string +DisplayName() string +Name() string +Status() ContactStatus +CustomStatus() ContactStatus +IsSmart() bool +Groups() int +Contact(in name string in email string in status ContactStatus) +Contact(in name string in email string) +AddGroup(in id int) +RemoveGroup(in id int) +Equals(in obj object) bool +GetHashCode() int +CompareTo(in obj object) int </pre>	<pre> +DisplayName() string +ID() int +Name() string +Contacts() ContactEnumerator +Count() int +Status() ContactStatus +ContactGroup(in id int in name string) +AddContact(in contact Contact) +Contains(in contact Contact) bool +RemoveContact(in contact Contact) +Equals(in obj object) bool +GetHashCode() int +ToString() string +CompareTo(in obj object) int </pre>

Figure 4.3: Contact and ContactGroup data classes.

Method	Notes
AddContact(string email)	
AddGroup(string name)	
ChangeGroupStatus(ContactGroup group, ContactStatus status)	Changes status for one group
CopyContact(Contact contact, ContactGroup group)	
GetContact(string email)	
IsBlocked(Contact contact)	
MoveContact(Contact contact, ContactGroup from, ContactGroup to)	
RemoveContact(Contact contact)	Removes contact form list
RemoveContact(Contact contact, ContactGroup group)	Removes contact from group
RemoveGroup(ContactGroup group)	
RenameGroup(ContactGroup group)	
SetBlocked(Contact contact, bool block)	True blocks, False unblocks

Table 4.5: Contact and ContactGroup manipulation methods in ContactList.

4.2.3 Conversations

Each individual conversation is represented by an implementation of the Conversation interface, either BluetoothConversation or MessengerConversation. These classes obscure the underlying network connection, providing methods for sending messages and disconnecting the socket when required. It is also responsible for maintaining a text log of the conversation, and the visibility of the conversation. The visibility property is used to determine if the conversation should be listed on the contact list or not, and is not set to true until the first message is sent or received.

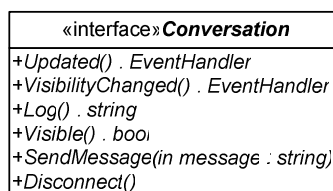


Figure 4.4: Conversation interface.

Conversations are stored in the ConversationManager class, which also creates conversations and requests switchboard connections as required. Most interaction with this class is through the two GetConversation methods, one for each service.

These methods are passed a contact, and check the internal list of existing conversations for any which match the contact. This is returned if found, otherwise a new conversation is created and returned.

4.3 User Interface Layer

The user interface layer is responsible for responding visually to changes in the data layer and interpreting and appropriately restricting the user's interactions with the application.

Three forms are involved in the user interface: Login, ContactViewer, ConversationWindow. Additionally, the UIManager class is used to manage all the forms, and present a main point of interaction with the data layer. The UIManager simplifies interaction in cases where a single data layer event requires a reaction in several different forms—for example, disconnecting requires the contact list to close and the login form to open. By invoking the appropriate methods on the forms itself, it ensures they occur in the correct sequence, and are executed on the display thread.

UIManager also listens to error events on the MessengerSever and BluetoothServer classes and displays error messages as required, bypassing the data layer.

4.3.1 Login

The Login form is displayed when the application starts, and any time the service is disconnected. It consists of text fields for Messenger login details, a label displaying the current connection status (such as 'Disconnected' or 'Starting Bluetooth'), a connect button, and options menu.

The connect button calls either the Connect or Disconnect method in the Messenger class, depending on the current connection status, to either begin or cancel sign-in. The options menu is used to either exit the application or change which services are to be connected. Changing this option modifies the Services property in the Messenger class.

Login also has a method for updating the connection status message, which is called by the UIManager any time a change is signalled by the Messenger object.

4.3.2 ContactViewer

ContactViewer is used to display the user's contact list, and is the main screen used to interact with the application. Visually, it is almost entirely made up of a TreeView user interface component, which is used to display all contacts and conversations in collapsing groups. The menu bar has a button for opening a conversation and an option menu, which contains most of the operations possible within the application.

The class has an Update method which is called by the UIManager any time a change is signalled in the ContactList or ConversationManager. This method redraws the list, adding a node for the list of current conversations, a node for each Messenger group in the ContactList, and a node for Bluetooth contacts, as required by the currently connected services. Each node is then populated with sub-nodes generated by iterating through the appropriate collections.

It also responds to StatusChanged events in the Messenger object via the UpdateStatus method, which updates the selected status in the options menu to reflect the newly changed status.

Pressing the 'chat' button when a contact node is selected calls the GetConversation method in the ConversationManager to retrieve or create a conversation object, then opens the ConversationWindow with that Conversation object (see section 4.3.3, below). When a conversation node is selected, it simply loads the ConversationWindow immediately.

The options menu contains options to sign out of both services, to change Messenger settings such as display name, modify contacts or groups, and change status. Options all invoke methods on the Messenger and ContactList classes, which send the changes to the server. As noted in section 4.2, changes are not actually reflected in the data layer until the server confirms the command has been successful.

4.3.3 ConversationWindow

The ConversationWindow class is used to display both Messenger and Bluetooth conversations. It consists of a message log displaying both incoming and outgoing messages, a text field for entering messages, a send button and options menu.

There is only ever one instance of ConversationWindow created in the application, as multiple windows would be redundant on a platform which can only display one at a time, and use up already restricted memory. Instead, the ConversationWindow uses Conversation instances from the data layer as models, which can be changed at any time using the Conversation property.

The class contains an Update method which is invoked any time the current Conversation object is changed, for example, when a message is received or the

contact changes status. This refreshes the message log with the contents. This method is also used whenever the current Conversation object is changed.

4.4 Summary

This chapter has discussed the implementation of the design set out in the previous chapter. In the next two chapters, the prototype system developed will be used to demonstrate system operation and will undergo functionality, usability and performance testing in order to evaluate the system.

5. Operation

This section of the report will demonstrate the features of the system which are available to the user, using screenshots to demonstrate normal operation of the system.

5.1 User Login

When the application is started, the user is presented with a login screen where they can enter their Passport address and password to log into the messenger service. From the options menu, the user can select which of the services they wish to connect to—Messenger, Bluetooth, or both.

When Bluetooth only is selected, the login form is disabled, as no credentials are required to start the service.

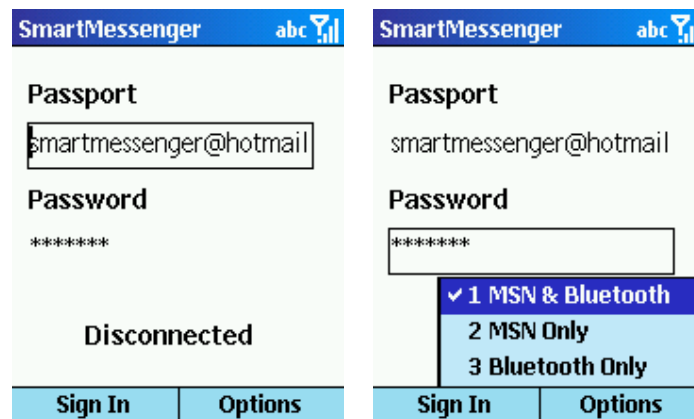


Figure 5.1: Login screen and options.

Pressing the 'Sign In' button begins connection to the services, starting with Messenger, and then Bluetooth as required. At any time during connection the user can press the 'Cancel' button and sign in will be halted.

Throughout the connection process, the user is kept informed of the system's progress, through connection, authentication and contact list download phases of Messenger connection, and starting up the Bluetooth service.

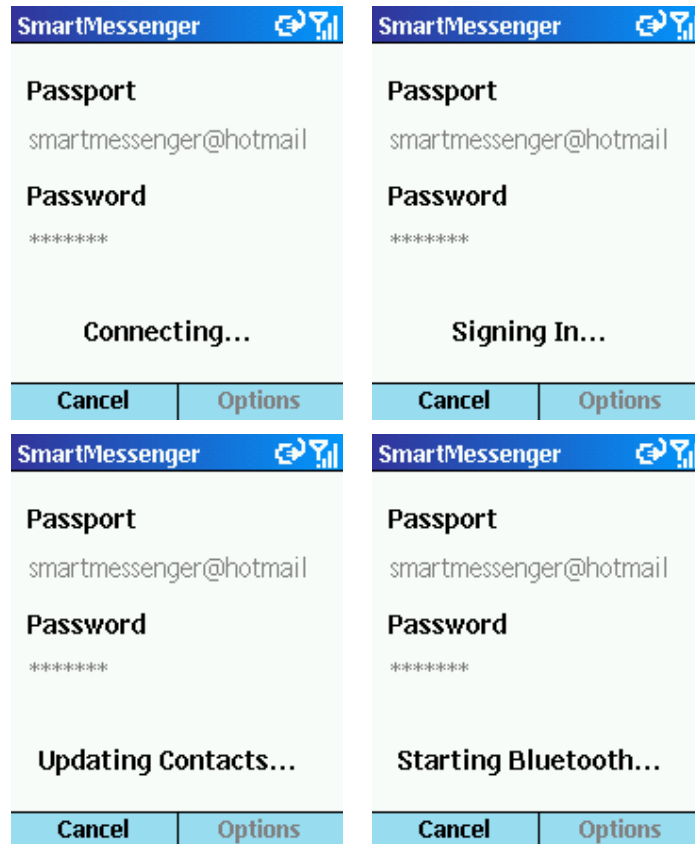


Figure 5.2: Connecting to Messenger and Bluetooth services.

5.2 Contact List Operation

The contact list screen is the main interaction point for the user. In addition to listing both Messenger and Bluetooth contacts, the contact list also contains a list of open conversations and the options menu.

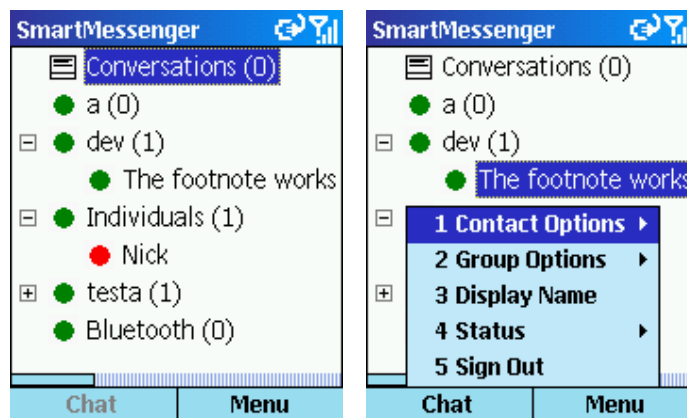


Figure 5.3: Contact list and options menu.

Messenger contacts are sorted into the user's custom contact groups, which are downloaded from the Messenger server. Bluetooth contacts are grouped separately, as are conversations.

The presence of individual contacts is indicated by the icon next to their name. A green icon indicates they are online, red indicates they are offline. A stop sign indicates they are busy, while a clock indicates they are away. These colours and symbols were chosen as they are similar to those used in official clients.

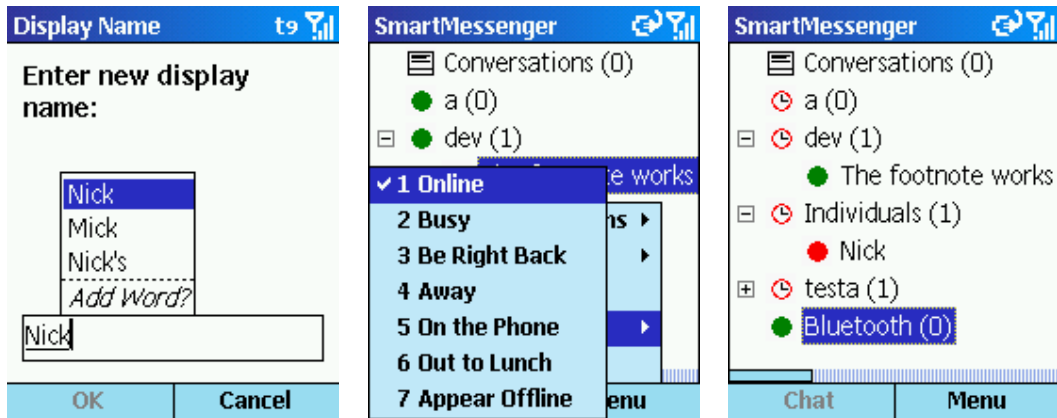


Figure 5.4: General Messenger options: changing display name and presence.

The user's Messenger display name can be changed using a dialog, and the user's presence can be changed through the options menu. The status is indicated by the icon displayed next to each group, which use the same symbols as contact statuses.

5.2.1 Managing Contacts

Contacts are managed through the 'Contact Options' submenu. Most options can only be used when a Messenger contact is selected on the contact list, with the exception of 'Add Contact', which can be used whenever the Messenger service is connected.

When a contact is selected, the usual options found in Messenger clients are available: blocking, deleting the contact from a single group or the entire list, and moving or copying a contact into a group.

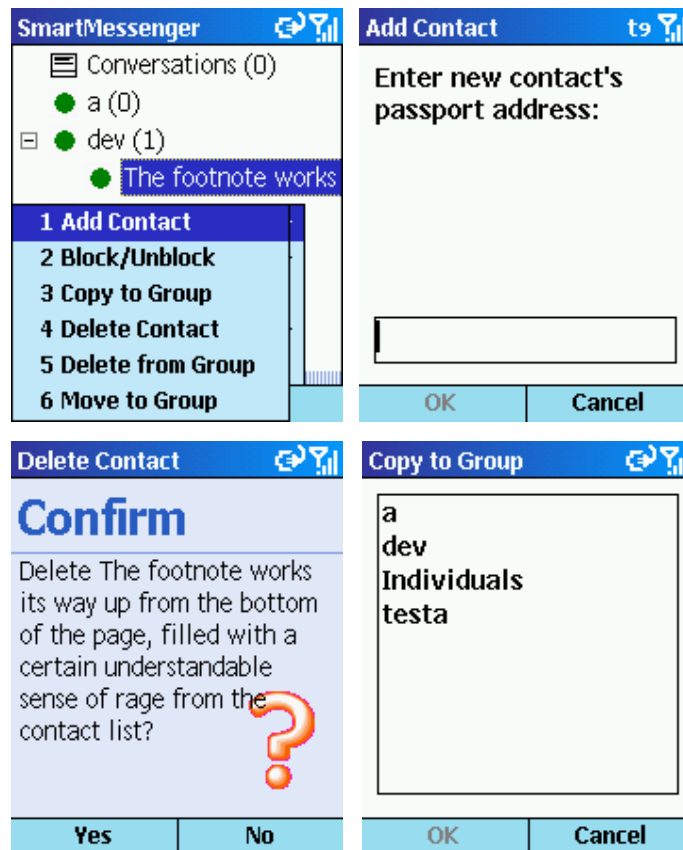


Figure 5.5: Contact menu and contact editing dialogs.

5.2.2 Managing Groups

When connected to the Messenger service, contact groups are managed through the 'Group Options' sub-menu. This menu allows groups to be added, removed and renamed. The 'Individuals' group can never be removed, as Messenger requires at least one group.

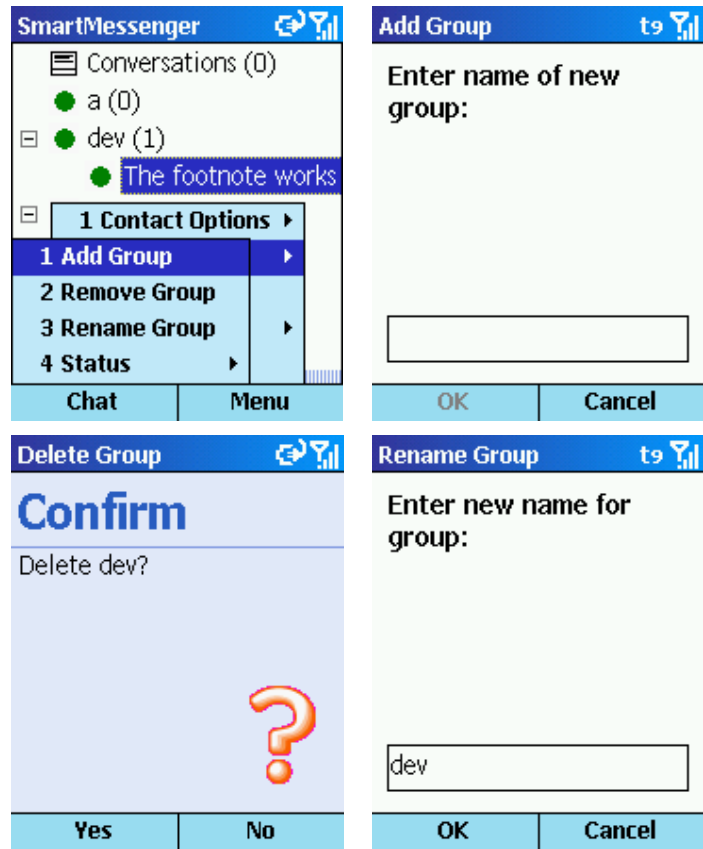


Figure 5.6: Messenger group options.

The group options menu also has a status submenu. When a group is selected, including the Bluetooth group, the status for that group can be changed. Upon changing a group status, the icon for that group changes to reflect the new state.

It is also possible to select 'Use Default', which means the group has no custom presence and defaults to the universal state selected on the main menu.

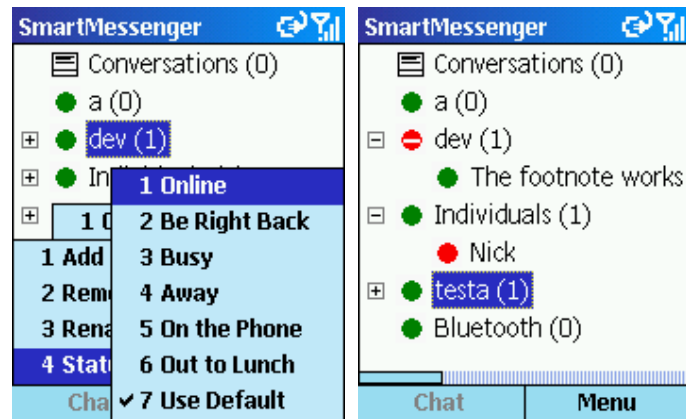


Figure 5.7: Custom group presences.

5.3 Conversation Screens

New conversations can be opened by selecting the desired contact from the contact list and pressing the 'Chat' button.

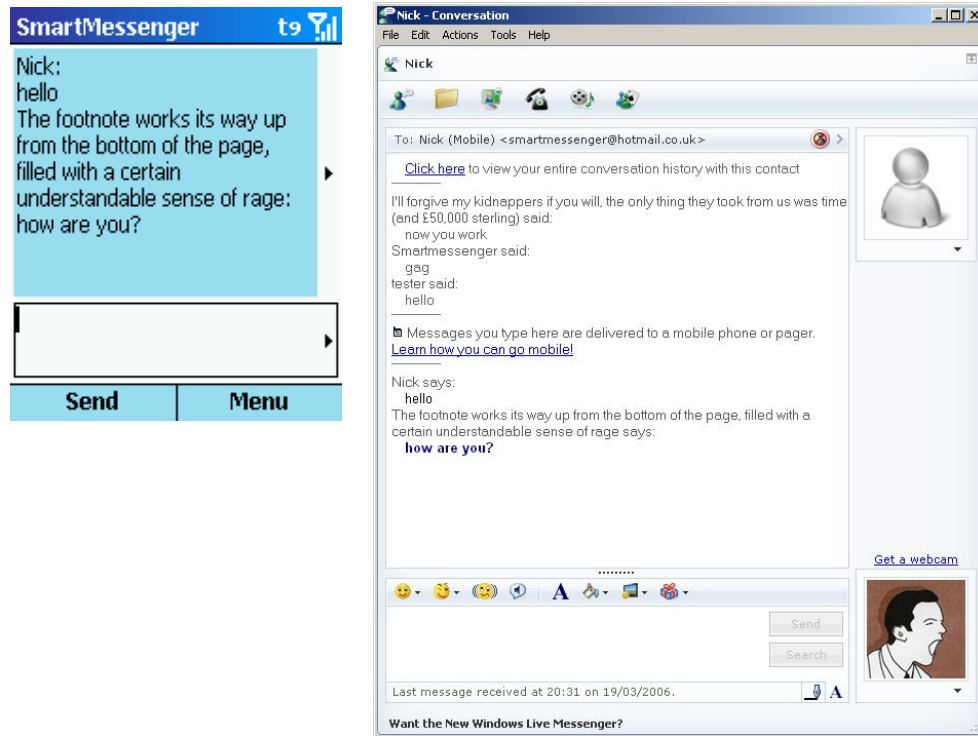


Figure 5.8: A conversation viewed in SmartMessenger and Microsoft Live Messenger.

Once a conversation has been opened, it is listed in the 'Conversations' group of the contact list, and can be reopened either by selecting the original contact and pressing 'Chat' again, or by selecting the conversation from the list and pressing 'Chat'.

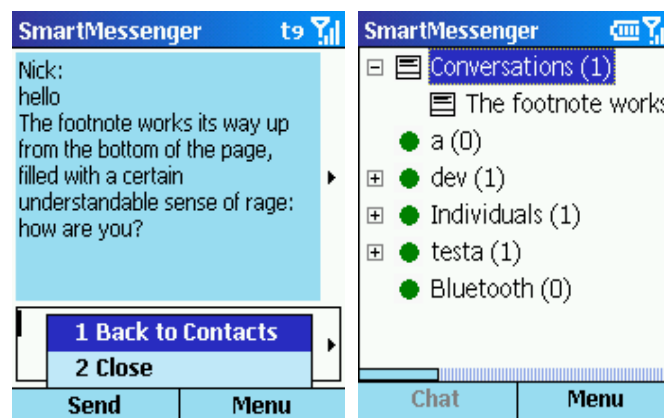


Figure 5.9: Conversation options and the conversation menu.

Conversations can be closed using the menu in the conversation window. It has two options, one to return to the contact list, which preserves the current conversation state, and one to close the conversation entirely. Closing the conversation erases all conversation state and removes the conversation from the conversation menu.

Bluetooth conversations operate in exactly the same way as Messenger conversations.

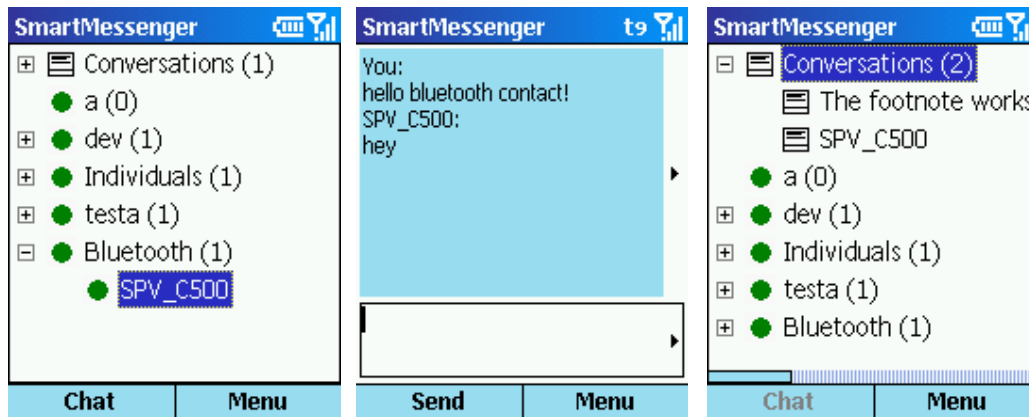


Figure 5.10: Bluetooth conversations.

Changes of state are reported in the conversation window. This may be simply changing to 'Away' or 'Busy', or it may be the contact has disconnected, or in the case of Bluetooth contacts, moved out of range.

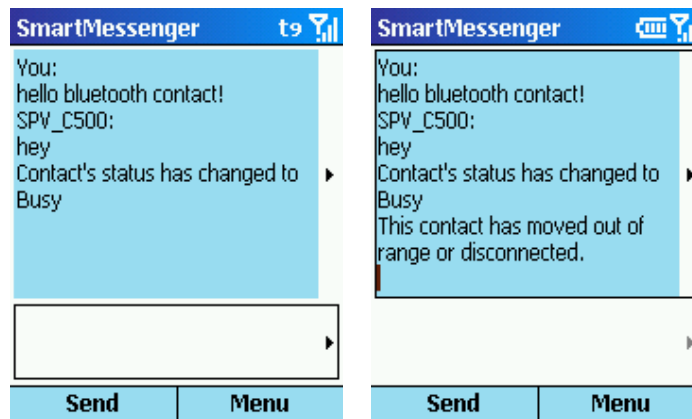


Figure 5.11: Presence reports in conversations.

5.4 Summary

This chapter has explored the functionality of the completed prototype application. The following chapter will test and evaluate the client, before final conclusions can be drawn from the project.

6. Testing and Evaluation

This section will document test cases and results to ensure that the prototype meets its requirements in terms of functionality, performance and usability, which will be evaluated through black box testing, performance testing, and actual use by potential end-users.

For testing purposes, SPV C500 and SPV E200 Smartphones will be used. Where a PC-based Messenger client is required, the latest official client, MSN Messenger 7.5, will be used.

6.1 Black Box Testing

Black box testing will be used to ensure that all features visible to the user perform their function as expected, without causing unanticipated results or undesirable states. Tests have been devised and carried out to test that the client connects as expected and is fully compatible with the Messenger service, meaning it is able to converse with PC-based Messenger clients without unexpected results for either party.

6.1.1 Sign-In Procedure

Test ID	Test and Expected Outcome	Result	Comments
1.1	Sign in to both Messenger and Bluetooth services using correct details. Contact list should load, showing contacts for both services.	PASS	
1.2	Sign in to just Messenger. Contact list should only show Messenger contacts.	FAIL	Bluetooth contact group is displayed, although it is empty and Bluetooth is not actually enabled.
1.3	Connect to just Bluetooth. Messenger login form should be disabled. Contact list should only show Bluetooth contacts.	PASS	
1.4	Use incorrect Passport credentials. Error message should be displayed and sign in cancelled.	PASS	

1.5	Sign in elsewhere once the client is connected. Client should show error message and sign out.	FAIL	Error message is display and sign out process begins, but hangs until 'cancel' button is pressed.
1.6	Sign out. Should return to login screen, and be reported as offline to other clients.	PASS	

Table 6.1: Test cases and results for service sign-in.

6.1.2 Messenger Services

Test ID	Test and Expected Outcome	Result	Comments
2.1	Change display name. Change should be reflected in local and remote clients.	PASS	
2.2	Change status. Change should be reflected in local and remote clients.	PASS	
2.3	Add a group. Group should appear on contact list.	PASS	
2.4	Remove an existing group. Should be removed from the list, and any contacts in it moved to the main group.	PASS	
2.5	Add a contact. The contact should appear in the main group.	PASS	
2.6	Move a contact to the new group. Contact should be removed from the main group and appear in the new one.	PASS	
2.7	Copy the contact to another group. Contact should now appear in both groups.	PASS	
2.8	Delete a contact. Contact should be removed from the contact list.	PASS	
2.9	Block a contact. Should appear offline to the contact's client.	PASS	
2.10	Sign out and back in. Changes made in tests 2.1 and 2.3-2.9 should persist.	PASS	

2.11	Change status for a single group. Status change should be reflected in other SmartMessenger clients, but not conventional clients.	PASS	
------	--	------	--

Table 6.2: Test cases and results for Messenger functionality.

6.1.3 Messenger Conversations

Test ID	Test and Expected Outcome	Result	Comments
3.1	Open conversation with member of contact list. Empty message window should appear.	PASS	
3.2	Send message to contact. Message should be echoed in conversation window and received by contact.	PASS	
3.3	PC client sends message. Should appear in Smartphone conversation window.	PASS	
3.4	Return to contact list. Conversation should be listed under the conversation group.	PASS	
3.5	Reopen conversation from the conversation group. Original conversation should be restored, messages sent as normal.	PASS	
3.6	Reopen conversation from the contact. Original conversation should be restored, messages sent as normal.	PASS	
3.7	Fully close conversation. Should be removed from the conversation list.	PASS	
3.8	Reopen closed conversation from the original contact. Should open a window with no message history.	PASS	
3.9	Send message to client from PC. New conversation should be added to the conversation list.	PASS	
3.10	Open conversation initiated by contact. Messages originally sent should be visible, messaging should work as normal.	PASS	

3.11	Invite third contact from the PC client. Smartphone client should announce their entry.	PASS	
3.12	Send message in three-person conversation. Should be sent to both remote participants.	PASS	
3.13	Remove one PC client from the conversation. Smartphone client should announce their departure.	PASS	

Table 6.3: Test cases and results for Messenger conversations.

6.1.4 Bluetooth

Test ID	Test and Expected Outcome	Result	Comments
4.1	Sign in to Bluetooth service. Device should be made discoverable (check device settings).	PASS	
4.2	Sign out. Bluetooth should be turned off (check device settings).	PASS	
4.3	Activate two devices. Both should shortly appear on the other device's contact list.	PASS	
4.4	Sign out one device. The device should immediately be removed from the other device's contact list.	PASS	
4.5	Start a new conversation from the contact list. Empty messaging window should appear.	PASS	
4.6	Send a message. Should be echoed locally and a new conversation should be opened on the remote device.	PASS	
4.7	Change status. New status should be reflected on remote device's contact list.	PASS	
4.8	Move out of range. Contacts should be removed from list once connection fails.	PASS	

Table 6.4: Test cases and results for Bluetooth messaging.

6.2 Performance Testing

Performance is an important factor in the evaluation of the system, as a user may be less willing to wait for a task to complete on a mobile device than a PC. A PC user may be able to switch to a different task while, for example, waiting for a connection to complete, but mobile phone users generally use only one application at a time. Due to the relative simplicity of mobile phone functions, particularly those which are not smartphones, the phone user is also accustomed to near instantaneous response times, whereas the PC user might expect a complex task to take some time to complete.

Performance testing has been carried out to ensure that connection times are acceptable, and to compare average speeds to those achieved by Microsoft's Smartphone Messenger client. Testing has also been carried out on the discovery speed of the Bluetooth service.

6.2.1 Messenger Connection

Connecting to the Messenger service can be broken down into three distinct processes: connecting and handshaking with the server, authenticating using the Passport service, and receiving your contact list.

Both were tested five times on the same phone to measure the time taken in each section of the connection process. The first is establishing the connection, which is defined as the time from the first VER command to the first communication with nexus.passport.net. The second is authentication, which is measured from the first Passport communication to the first SYN command. The final section is downloading the contact list, which is measured from the SYN command to the final LST command.

For testing purposes, the phone was connected to a PC via USB so that the PC's Internet connection could be used, and Ethereal [23] could be used to observe packets being sent across the connection. It is important to note that GPRS would be considerably slower than this connection, but the speed decrease should theoretically be similar for both clients.

The average results, and range of results, are shown in Table 6.5.

	SmartMessenger		MSN Messenger for Smartphones	
	Mean (s)	Range (s)	Mean (s)	Range (s)
Establish connection	4.16	1.22	5.94	9.89
Authenticate using Passport	4.91	1.15	1.39	0.72
Download contacts	0.24	0.04	0.24	0.04
TOTAL	9.31		7.57	

Table 6.5: Messenger connection performance testing results.

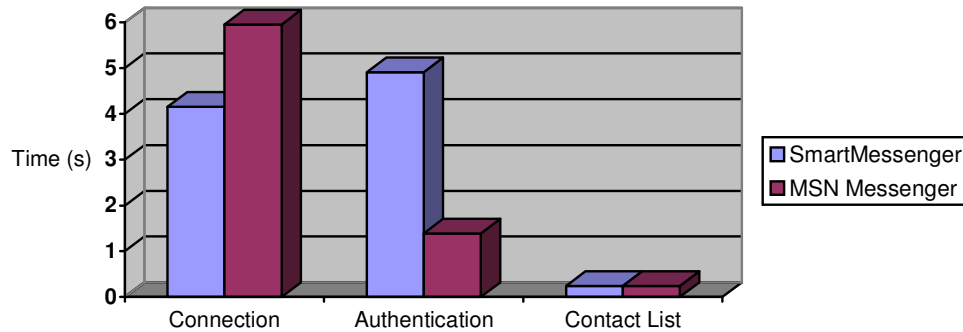


Figure 6.1: Messenger connection performance testing results.

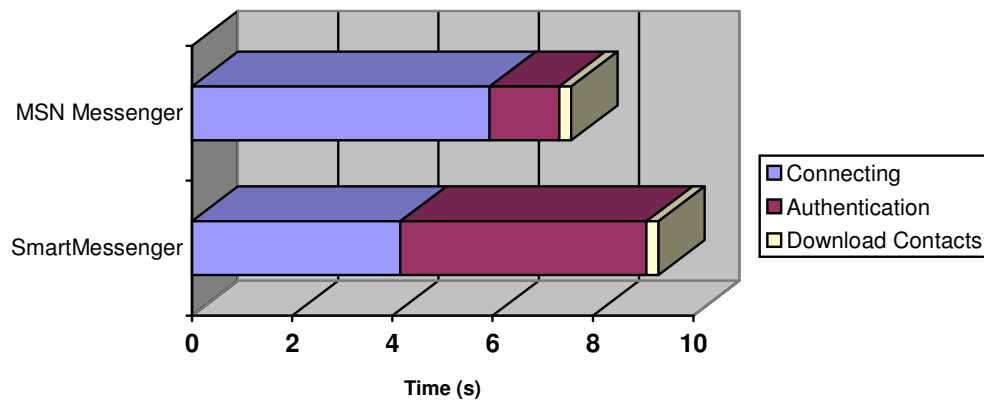


Figure 6.2: Comparison of connection phases between clients.

The results show that MSN Messenger does connect faster on average, although connection times are less predictable. The main differences in efficiency can be found in the connection and authentication phases, with contact download speeds near identical, taking very little time and varying little on both clients.

Analysis of Ethereal logs show that for connection and authentication phases, while SmartMessenger connected to well-known servers (messenger.hotmail.com and nexus.passport.net) before being directed to other servers, MSN Messenger was able to connect directly to the final servers. This could be due to knowledge of the servers,

as both servers and client are Microsoft projects, or caching of past servers to improve performance.

6.2.2 Messenger Contact List

With the exception of Internet connection speeds and server response times, which are beyond the control of a client application, the only factor which varies between sign-in attempts is the number of contacts and contact groups the user must download. Each contact and group is sent as a separate command, so the more contacts the user has, the longer this process will take.

To test the increase in sign-in time which can be expected as contact list size increases, two different accounts were timed on 5 attempts and averaged. It was expected that the account with more contacts would take longer to download contacts, but that time per contact should be roughly the same.

Download time is measured between the receipt of the server's SYN response and receipt of the final contact record. The results are shown in Table 6.6.

	7 items	28 items	137 items
Total Mean Time (s)	0.006	1.096	2.497
Total Range Time (s)	0.006	1.873	0.157
Mean Time per Contact (s)	0.0009	0.039	0.018

Table 6.6: Contact list download times for different accounts.

These results show little correlation between the number of contact list records and the time taken to download them. Although the average time does increase with the number of records, the time taken per contact shows no predictable change between tests. Furthermore, in some tests, such as the 28 item test cases, the results were spread over a large range.

These unexpected results could be caused by several factors, including the internal workings of data structures on the client, or any number of server conditions which it is impossible for this report to consider.

However, in all cases it can be seen that the time per contact is negligible, and the total time of only a few seconds should be acceptable to any user.

6.2.3 Bluetooth Service

Unlike the Messenger service, contacts are not immediately available to the user when the service starts. Nearby Bluetooth devices need to be discovered, and connections must be formed for handshaking before the contact becomes visible to the user and conversations can be started.

To test detection times, the Bluetooth service will be started on one device, followed by a second device, and the time taken for the devices to be displayed on the contact list will be measured. This will be tested five times.

Test #	Detection Time (s)
1	6.78
2	19.53
3	23.69
4	8.27
5	10.14
Average	13.682
Range	16.91

Table 6.7: Detection time test results.

The results of these tests show a wide range of detection times, some of which may be beyond the reasonable length of time a user may be expected to wait. A user might not, for example, wait 20 seconds before deciding that there are no contacts in range.

6.3 Usability Testing

Usability testing has been used to ensure that the prototype system is actually usable and useful to real end-users. This form of testing is particularly relevant to the project, as it relates to one of the projects original aims, which was to “improve the usability of the client in comparison to existing instant messenger clients developed for use on mobile phones”.

The unusual platform of this system, compared to instant messenger applications the average user is accustomed to, also means that ease-of-use is highly desirable.

Test users will be given devices running the client, and asked to sign in to test accounts and interact with both PC-based Messenger clients, and with each other using both Messenger and Bluetooth services.

The instructions detailed below were given to 5 users to carry out, and their responses after attempting to use the system are documented in the following section. Each user was given instructions on the basic operation of the Smartphone platform (soft keys, menus, selection button) before starting.



Figure 6.3: Usability testing in progress.

6.3.1 Instructions

- Sign in to both Messenger and Bluetooth using the credentials provided.
- Change your display name and set your general status to 'Away'.
- Create a new group, and move one of your Messenger contacts into it.
- Change your status back to 'Online' for the new group alone.
- Try to talk to the contact in the group.
- Return to the contact list and start a conversation with a Bluetooth contact.
- Return to the Messenger conversation.
- Sign out and reconnect to the Bluetooth service alone, with no Messenger connection.
- Continue to test the application in any other ways desired.

6.3.2 User Responses

The following responses were received from test users asked to follow the instructions detailed above. Each response specifies the user's usual Messenger client (if any), as well as their mobile phone model and its operating system.

- **User 1:** Live Messenger 8, Samsung E720, proprietary OS

User was able to quickly acclimatise to the Smartphone user interface and follow all instructions easily. The user had no complaints about the user interface, and said it performed as well as he had expected given the hardware platform.

- **User 2:** MSN Messenger 7.5, Sony Ericsson K500i, proprietary OS

The user expected to be able to use selection and back buttons to open conversations from the contact list rather than using the soft keys.

User found the nested contact list confusing, and would have preferred all contact groups to be expanded on start-up rather than closed. The user didn't notice or utilise the conversation group on the contact list when trying to reopen an old conversation, instead opening the conversation by selecting the contact.

He was able to identify how to toggle services, but expected the desired services to immediately connect when he selected it, as his login details were already filled in.

- **User 3:** MSN Messenger 7.5, Nokia 3510i, Nokia ISA platform

The user was able to sign in, change their display name and status without any problems. As with other test subjects, the user initially found navigating difficult, due to lack of experience with the device.

The user was able to create a group without problems, but when asked to modify a group, was unable to find the correct option. As with User 2, he could not identify between contact groups and contacts themselves, and did not realise contact groups could be expanded.

User expected the select button to open a conversation once a contact was selected, but was able to converse easily and switch between conversation and contact screens. When asked to reopen a conversation, the user identified and utilised the conversation group on the contact list.

The user said he found the application relatively simple to use, to the extent he expected on a mobile phone.

- **User 4:** None, None, N/A

This user had never used a mobile phone or a Messenger client, so his experiences with the application are particularly useful in assessing the usability and intuitiveness of the system. Given he has no experience with mobile phones, the user was given a more thorough explanation of the SPV handset layout before starting the trial.

The user was able to sign in, change the display name and status without problems. Again, the tree-view contact list was confusing to the user, who needed prompting to open a group and find contacts.

The user was also confused by standard Messenger terms, for example, when asked to ‘move a contact into the group’ he tried to use the ‘Add Contact’ function.

He said the experience was ‘bewildering’, however he did say the interface was logical, and was clearly able to navigate the application without significant problems.

- **User 5:** MSN Messenger 7.5, Orange SPV C550, Windows Smartphone 2003

This user has experience with both regular Messenger clients, and the Smartphone platform. He was able to follow all instructions with little difficulty, with the exception of moving a contact between groups.

Like other test subjects, he initially had difficulty finding the correct option, as he had not selected a contact first. However, he did eventually work out how to expand groups to select a contact without prompting, where other subjects needed to be given help.

6.4 Summary

This chapter has documented the testing and evaluation of the prototype system in terms of functionality, performance and usability. Black box testing found several flaws in the functionality of the system, which have subsequently been fixed, while performance of the system was found to be comparable to Messenger clients pre-installed on the Windows Smartphone platform.

One notable disadvantage which the application suffered was its inability to cache notification server and Passport server addresses. Examination of performance testing results showed that Microsoft's client does cache server addresses, and gained several second in trials. This is worth considering as an extension to the system.

Usability testing showed that while users with various levels of experience and technical expertise were able to quickly grasp the main functions of the application, they were confused by some secondary features, particularly the 'Contact Options' menu and the collapsed tree view approach to the contact list. This could be because most Messenger clients expand all groups by default, or even ignore contact groups and order contact by their presence.

Additionally, users are not accustomed to contact groups each having their own status indicator, which makes them appear similar to a simple list of contacts when collapsed. These results should certainly be considered for future development and improvement of the system.

7. Conclusions

This section will evaluate the prototype and project as a whole, in order to ascertain whether the aims of the project have been achieved. It will also discuss possible improvements which could be made to the system.

7.1 Summary

This report has explored the domain of mobile connectivity and messaging, and its relevance and usefulness in the modern world, by examining available technologies and similar projects. From this analysis, it was determined that a client integrating more than one messenger service may be a potentially useful tool for smartphone users.

A system offering .NET Messenger and peer-to-peer Bluetooth instant messaging services in a single client application was designed, aiming to successfully integrate the two services, while extending the functionality of the Messenger service to offer better presence reporting and improved usability compared to existing clients on the platform.

This design was implemented on the Microsoft Smartphone 2003 platform using C# and the .NET Compact Framework, as documented in Chapter 4, and tested for functionality, usability, and performance aspects.

7.2 Overall Conclusions

The original aims of this project are restated below, with an evaluation of how the project tried to achieve each aim, and the degree of success to which they were achieved.

- *Develop and implement a protocol for peer-to-peer Bluetooth instant messaging using mobile phones.*

The project has resulted in the development of a reliable, working peer-to-peer Bluetooth instant messaging protocol which allows for devices implementing the protocol to identify each other, exchange presence information, and send and receive messages.

- *Develop a single messenger application which integrates both the .NET Messenger service and the Bluetooth peer-to-peer service, and provide a common user interface.*

The system developed during the project successfully integrated the two services into one application. The prototype application is able to connect to either service, or both at the same time, with contacts for both services appearing in a unified contact list.

Conversations for both services are displayed through a common conversation screen, and presence statuses can be changed together or independently using the same options menu.

- *Extend presence functionality in the Messenger system, by allowing the user to select different states for different groups of contacts.*

The project resulted in the development of a protocol allowing clients supporting the extension to identify each other and exchange presence messages across the Messenger service's switchboard servers. This allows status messages to be sent over a regular Messenger connection, with no support on the server itself.

The presence extension does not affect the client's interoperability with clients developed by Microsoft or third-parties, although these clients are unable to receive presence messages. In these cases, the client's universal presence status is still received.

- *Improve the usability of the client in comparison to existing instant messenger clients developed for use on mobile phones.*

The system attempted to address usability concerns with the addition of a 'conversations' group to the contact list, which would allow the user to easily see their open conversations, where the official client requires the user to access a second screen listing conversations.

However, during usability testing it became clear that users were not always using the conversation menu, preferring to open conversations by selecting the contact instead. This could be because the users were accustomed to the

regular MSN Messenger interface, which does not include such a menu. The element was therefore alien to them, and they were less likely to use it, preferring to rely on familiar interface elements.

They were also confused by the tree view contact list, particularly that it was collapsed on start-up, making groups appear like contacts themselves.

7.3 Possible Improvements and Future Developments

This section will explore possible improvements which could be made to the system given more development time, in order to make the application more usable or more useful to users.

7.3.1 Extensibility and Additional Services

The prototype developed was not designed to be inherently extensible, as this was not one of the project aims. Partly, this was due to the significant differences between the connection models of the chosen services—Messenger utilises a client-server approach, while the Bluetooth service operates in a peer-to-peer fashion, and also involves the unusual ‘discovery’ task.

However, one useful extension of the system would be to implement a modular structure, as used by Trillian. This would facilitate the future addition of support for more services, such as Yahoo or ICQ messengers, or even a plug-in system where the user could select the services they wished to install.

Most instant messengers follow a similar design, with a central server to manage contacts and presence, and secondary connections acting as switchboards. Although these connections may be implemented differently, and will certainly use different protocols, the object-oriented nature of the system means that interfaces could be devised for standard functionality of servers and switchboards, obscuring the actual implementation from the rest of the system.

7.3.2 User Interface Improvements

The single consistent complaint from users during usability testing (see section 6.3) was the nested contact list. The usability of the application could be improved considerably by expanding the tree view on start-up, or simply removing the option to collapse groups.

This is unfortunately not possible with the limited user interface components found in the Compact Framework, however extended user interface elements do exist in open source libraries such as OpenNETCF, so it possible that an appropriate component has been developed. It may also be possible to create a custom component for this task.

Another common usability issue was the tendency to try to use the select and back buttons rather than the soft keys for navigation. It would be possible to capture these button presses and have the user interface react as expected.

7.3.3 Cross-Service Conversations

In the prototype system, Messenger and Bluetooth services are integrated into the same client, but the two services exist completely independently of each other and have no interaction.

One potentially useful feature would be to allow interaction between a device's local Bluetooth contacts and remote Messenger contacts. For example, user A could start a conversation with a Messenger contact B at a remote location, and then invite Bluetooth contact C who is across the room into the conversation. C would not need an Internet connection, as messages could be forwarded to B by A over their normal switchboard connection.

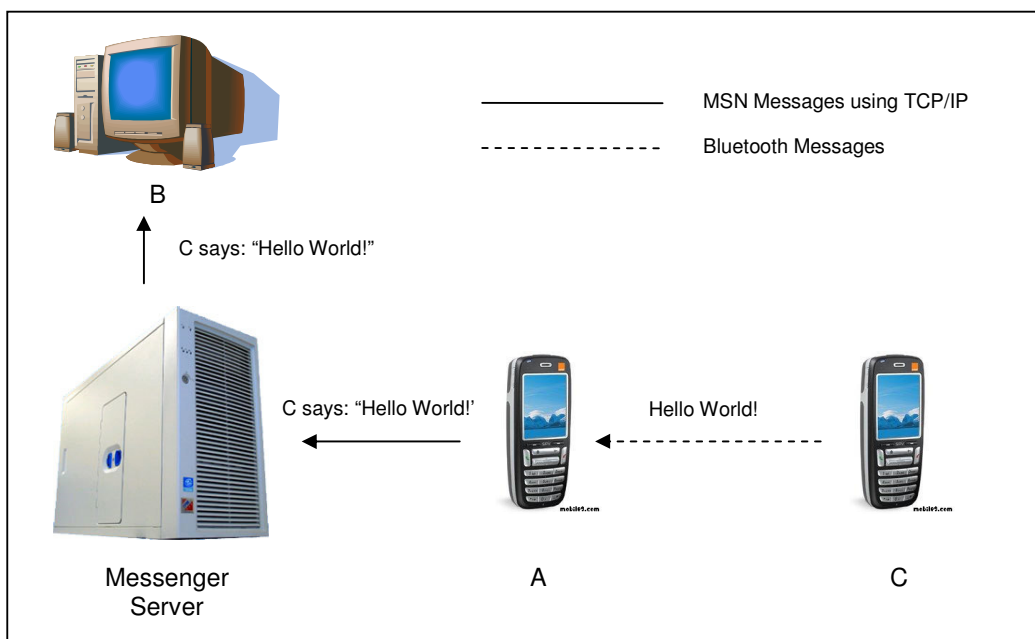


Figure 7.1: Cross-service messaging.

This could be achieved using messages with custom content types, such as those used in this project to send status messages. Clients that understood the protocol could then display the Bluetooth contact and their messages in the conversation as if they were a regular Messenger contact, making the service cross-over transparent to all users. This would, however, require the remote Messenger client to understand the protocol, and would not be compatible with official clients.

At the expense of elegance, this idea could be made compatible with other Messenger clients by forwarding messages inside plain text messages and simply adding text to the start of the message, such as 'Bluetooth contact XX says'.

7.3.4 File Transfer

File transfer is already part of the MSNP protocol and is supported by most clients, so to implement this would add a useful feature to the system. However, more interesting is the possibility of file transfers between Bluetooth devices.

The Smartphone platform has no in-built support for Bluetooth file transfers, so an application which searches for nearby devices supporting a particular protocol could be very useful for users wishing to share files easily between their phones.

File transfers across Bluetooth connections are possible using the OBEX protocol, so this could potentially be used to add file transfer functionality to the Bluetooth IM protocol.

A command could be added to the protocol which invites the user to accept a file transfer. The recipient would be able to decline the file if they wished, or accept, opening a second connection to the remote device using Bluetooth's OBEX profile.

7.3.5 Bluetooth Mesh

The range of Bluetooth messaging could potentially be improved by having each device act as a router, passing messages from one device to another which would normally be out of range.

When two devices connect, they could swap a list of their other Bluetooth contacts (*cf.* GNUtella [24]), and offer to forward messages to them. Messages sent to the middle device with a particular header would be forwarded on to the specified device, without the middle user knowing.

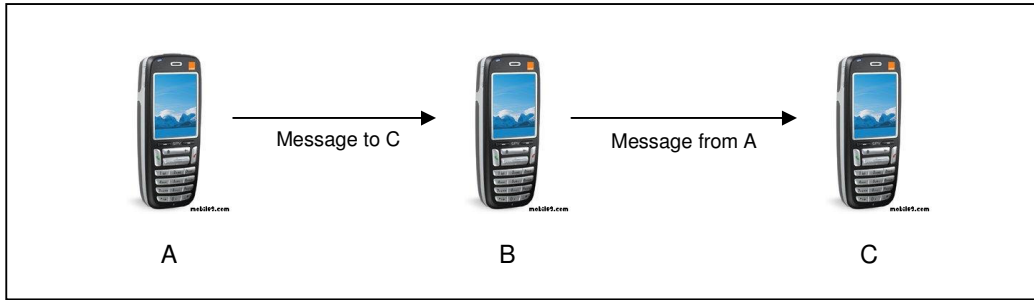


Figure 7.2: A Bluetooth mesh network.

Similar technologies for creating *ad hoc* ‘mesh’ networks of routing devices are currently under development as part of the next generation of Ethernet specifications [25].

However, this would most likely be of limited use, as extending the range would require a string of devices, each within the range of at least one other device. In reality, this is an unlikely circumstance.

7.3.6 Further Presence Enhancements

Although the client developed for this project only allows the user to apply a custom status to entire groups of contacts, there is no reason that a client could not use the advanced status protocol developed to apply a particular status to a single contact.

Furthermore, the protocol could potentially be extended to send completely customised statuses to contacts. Rather than being limited to ‘Away’ or ‘Be Right Back’, the user could send a status message saying ‘in a meeting’ or ‘gone to the shops’, along with the user’s choice of pre-defined icons.

Appendix A: Project Proposal

This section is the original project proposal, submitted in June 2005. The only significant deviation from the proposal regards the use of the DotMSN library, which was intended to provide the Messenger implementation for the project. However, this library was not compatible with the .NET Compact Framework, requiring the full framework to run. As a result, much more of the project has been spent implementing the Messenger Protocol than anticipated.

Abstract

Mobile platforms are an important area in modern communication, and clients to bridge these platforms and their related technologies to existing technologies are in demand.

The proposed system is a usable advanced instant messenger client to run on Smartphones, which will allow communication between the device and the MSN Messenger service, and nearby Bluetooth devices equipped with the same client.

The system will focus on integrating these services seamlessly into the same user interface, and presenting a more robust grouping facility for contacts, which will allow the user to present a different status to each group.

1 Introduction

In the rapidly-developing field of mobile communications, the divisions between personal computers, personal data assistants and mobile phones are quickly become indistinct, to the extent that many believe they will soon become non-existent.

Increasing processing power, memory and connectivity on small devices is enabling applications once confined to the desktop to be carried in the user's pocket. I intend to make use of two popular protocols which are currently available on the Smartphone platform—Bluetooth, and the MSN Messenger Protocol—to develop an instant messenger with significant improvements over the current generation of clients.

The proposed system is a single client which can be used in a variety of circumstances by sending messages over both the MSN Messenger network and across local Bluetooth connections, using the same user interface. In situations where no Internet

connection is available, the user will be able to choose to form only Bluetooth connections, or to form only Internet connections when they don't wish to be open to other users in their surrounding area.

It will additionally allow contacts to be grouped, and the user will be able to present a different status to each group of contacts. For example, a user who was at work might want to appear available to colleagues, but appear busy to friends and family. Conversely they may be attempting to take a break from work and not want to hear from colleagues.

This report will detail the proposed system and investigate existing systems and technical issues which will affect its development. It will lay out the work schedule and specify and justify resources required for its completion.

2 Background

2.1 Instant Messaging

The first instant messenger was ICQ [1], a free client that could be downloaded and used by anybody. It was launched in 1996, and quickly found widespread use.

Whereas email can be equated to the real-world postal system—you send a message, but do not know when it will be received, and may have to wait some time for a reply, instant messaging is better likened to a telephone conversation. When you try to initiate a conversation, you know if the other person is there or not, and conversations can be conducted in real-time.

The basic concept involves a server and any number of client applications. Some networks, such as ICQ, simply store client IP addresses and port numbers centrally and distribute these to clients so that they can communicate directly. In others, such as MSN Messenger [2], the server acts as a switchboard which routes messages to their intended destination.

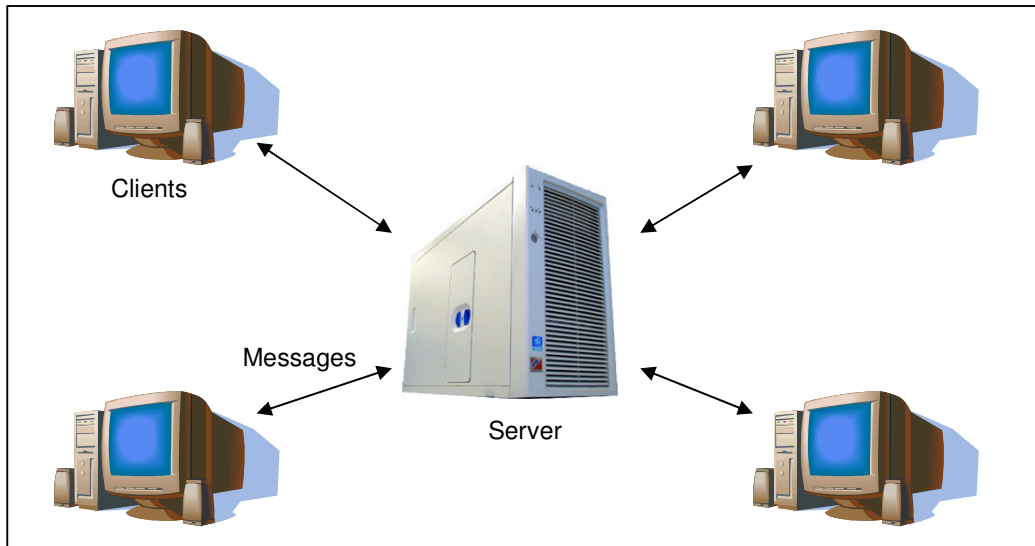


Figure 2.1: Instant Messaging with MSN.

Many instant messengers include a variety of other services, such as voice and video messaging, SMS, and games. These services will not be addressed in this project.

2.2 MSN Messenger Protocol

The MSN Messenger Protocol is a set of commands which can be sent between the client and server. These commands consist of a three letter code, and a list of arguments. All of this forms a string, sent using a TCP/IP connection.

Two servers are involved in Messenger connections. The notification server primarily deals with your status, and provides you with the status of your contacts. By connecting to the notification server, you make others aware of your presence, and likewise by closing a connection

The switchboard deals with messaging sessions between clients. Every session requires a new connection to the switchboard, where a session is essentially a single conversation window.

2.3 Bluetooth

Bluetooth is a short-range short range wireless connectivity standard [3], which is built into most modern mobile devices. It allows *ad hoc* networks ('piconets') to be formed between devices located within ten meters of each other.

2.4 Similar Systems

An MSN Messenger client is already available for Smartphones [4], but it can only be used for MSN connections, and does not have Bluetooth capabilities.



Figure 2.2: MSN Messenger for Smartphones.

Current messenger clients are additionally very limited in terms of how you can present yourself to other users. On most, you can only choose from a set of pre-defined states such as ‘Online’, ‘Away’ or ‘On the Phone’. The most flexible allow you to enter your own messages.

None of them are able to present different statuses to different groups of contacts, despite the potential usefulness of this feature.

3 Proposed Project

3.1 Aims

The key aims of the project are:

- Develop an instant messenger client on the Smartphone platform which communicates using the MSN Messenger network, as well as local Bluetooth networks.
- Improve profiling in messenger clients to allow the user to present different statuses to different contacts.
- Improve profiling to provide more information about the user to their contacts.

3.2 Architecture

The system will be built on Microsoft's .NET Compact Framework [5], which runs on the Smartphone [6] platform. The Compact Framework itself does not include libraries for Bluetooth communication or connecting to MSN Messenger servers, so the system will use two open source libraries: DotMSN [7] and OpenNETCF Bluetooth [8].

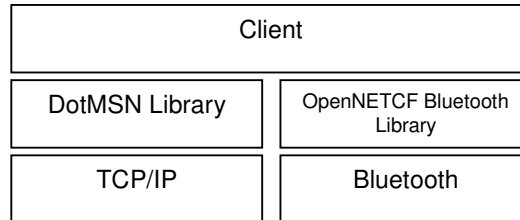


Figure 3.1: Proposed system architecture.

These libraries provide abstractions over the device hardware, making it simpler to deal with the two types of connection.

3.3 Client

The client will provide the same basic functionality as a traditional client by connecting to an MSN Messenger server, maintaining a list of Messenger contacts and showing their online/offline status, and sending and receiving messages from contacts. The user interface will need to be similar to existing instant messenger user interfaces, so that users are familiar with it and can learn how to use it quickly.

The system will also be able to make connections to nearby Bluetooth devices and send text-based messages to any available devices. These devices will be viewed as regular contacts and 'conversations' with a Bluetooth device will appear to be no different to a Messenger conversation.

Users will have the option to connect to either or both of these services, depending on their circumstances. When connected to the Messenger server, messages will be passed back and forth between the client and server. Bluetooth messages will be sent directly between devices.

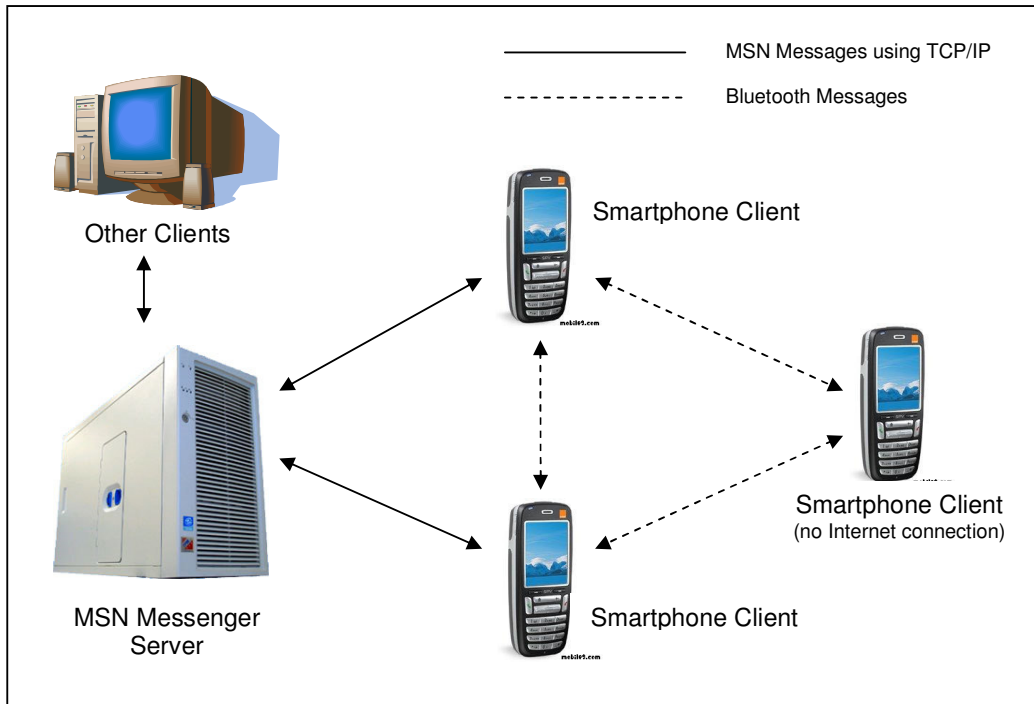


Figure 3.2: Proposed communication architecture.

The user will be able to sort contacts into groups, and present a different profile to each group, rather than being able to set a single profile. Additionally, users will be able to see how many conversations the contact is engaged in, allowing them to judge whether they should be disturbed or not. This is especially important on the Smartphone platform, where only one window is visible on screen at any time.

Where possible, this information should also be conveyed to traditional clients.

3.4 Approach

Development of the client will be incremental, with functions added one at a time. A basic Messenger client will be created first, and testing will be conducted as each client feature is added to ensure that the new functionality works correctly, and has not interfered with existing features.

Once the Messenger client is operational, it will be extended to add the advanced profiling features, and then to add Bluetooth messaging.

4 Testing

Black box testing will be carried out with a variety of valid, extreme and invalid data to ensure that the client responds as expected. Testing will be carried out in all modes of operation (Messenger only, Bluetooth only, and both).

Testing will be carried out as each feature is added, with more in-depth test cases run later.

The Messenger client will need to be tested in conjunction with different version's of Microsoft's client in order to ensure that no behaviour in either client causes unexpected results in the other.

5 Programme of Work

The programme of work can be split into the following main sections:

- 1. Understand Device** (Weeks 1-2)
This stage will involve familiarising myself with the Smartphone platform, in order to learn what the device is capable of, and how the user would normally interact with it.
- 2. Overall design** (Weeks 2-4)
The main architecture of the system will be set out during this, as well as user interface designs.
- 3. Client development** (Weeks 4-7)
Development of the basic MSN Messenger client.
- 4. Profiling extension** (Weeks 8-11)
Extension of the client to involve advance profiling methods.
- 5. Bluetooth extension** (Weeks 12-15)
Extension of the client to utilise Bluetooth connections as well as Messenger connections.
- 6. Testing** (Weeks 16-17)
The system will be thoroughly tested during this phase, according to test strategies set out during the design stages.
- 7. Further development** (Weeks 17-18)
Any development which is found to be required by the testing phase will be carried out in this stage, and further tests conducted to ensure modifications are effective, and do not cause further bugs.

8. Report (Weeks 16-20)

The report will be written throughout the project, but much of it may need to be written retrospectively. This period will be set aside for putting together the final report.

This programme does not include vacation periods, which will allow some flexibility if a section of work takes longer than anticipated.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Understand Device	█	█																		
Overall design		█	█	█																
Client development				█	█	█	█													
Profiling extension								█	█	█	█									
Bluetooth extension												█	█	█	█					
Testing																█	█			
Further development																	█	█		
Report																█	█	█	█	█

Table 5.1: Gantt chart.

6 Resources Required

The project will require Visual Studio .NET 2003, as it is designed specifically for developing in .NET languages such as C# and provides tools to expedite development, such as Window Form Designer.

The Smartphone 2003 SDK will be required to provide the API used by the Smartphone platform, and an emulator for easier testing. This is available for download from Microsoft.

To easily access Bluetooth and MSN Messenger networks, the project will use two open source libraries, OpenNETCF's Bluetooth library and DotMSN for handling MSN Messenger connections. These libraries are also available for free download.

An actual Smartphone device will be required for real-world testing of Internet and Bluetooth connections.

7 References

[1] <http://www.icq.com>

[2] <http://messenger.msn.com>

[3] <http://www.bluetooth.com>

[4] <http://messenger.msn.com/Devices/PocketPC.aspx>

[5] <http://msdn.microsoft.com/smartclient/understanding/netcf/>

[6] <http://www.microsoft.com/windowsmobile/smartphone/default.mspx>

[7] <http://www.xiholutions.net/dotmsn/>

[8] <http://www.peterfoot.net/CategoryView,category,Bluetooth.aspx>

Appendix B: Messenger Commands

This section lists commands from the MSNP9 instant messaging protocol which are relevant to this project. Some commands which are ignored by the project, such as those relating to email notifications from the Hotmail service, have been omitted.

1 Notification Server Commands

Command	Sender	Purpose
VER	Client	Agree on protocol version to be used.
CVR	Client	Specify client details.
USR	Client	Identify user.
XFR	Server	Address of a notification server or switchboard server.
RNG	Server	Invitation to join conversation
OUT	Client/Server	Disconnect.
CHL	Server	Server challenge.
QRY	Client	User's challenge response.
CHG	Client	Change presence.
ILN	Server	Initial presence of a contact.
NLN	Server	Contact presence changes.
FLN	Server	Contact goes offline.
REA	Client	Change your display name.
SYN	Client	Synchronise contact list.
LSG	Server	Contact group record.
LST	Server	Contact record.
ADD	Client	Add a contact.
REM	Client	Remove a contact.
ADG	Client	Add a group.
RMG	Client	Remove a group.
REG	Client.	Rename group.

Table 1.1: Notification server commands.

2 Switchboard Server Commands

Command	Sender	Purpose
USR	Client	Identify user.
ANS	Client	Join a conversation.
IRO	Server	Existing member of a conversation.

CAL	Client	Invite someone to join the conversation.
MSG	Client/Server	Send or receive a message.
OUT	Client	Leave a conversation.
BYE	Server	A contact leaves the conversation.

Table 2.1: Switchboard server commands.