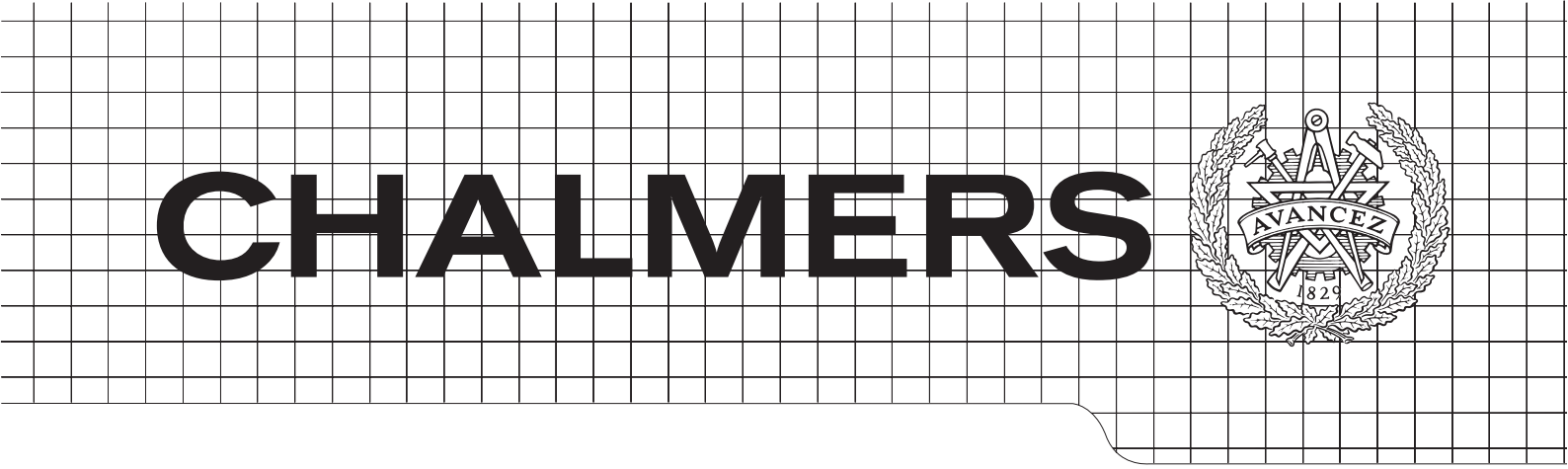

Multi Instant Messaging Client

Bachelor's Thesis

Alexander Ågren
Christoffer Medin
Anton Myrholm
Kim Egenvall
Carl Fredriksson
Niklas Johansson Miglavs

Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden 2015
Bachelor's Thesis DATX02-15-14

Abstract

There is an abundance of chat services and it is not uncommon to run several applications per device that all provide similar functionality. The group started the project to find a solution to this problem and decided to evaluate the possibility of creating a multi instant messaging (multi-IM) client that can connect to all chat services.

The group studied how a multi-IM client could be developed and what services and platforms were reasonable to support in a prototype. The group developed a prototype which supported two platforms and two services and evaluated the results.

The technologies and methods that were used suited the purpose of the project, but the potential of the application is considered greatly hindered by the lack of open Application Programming Interfaces (API) by some of the larger services. These limitations are so large that the future of a multi-IM client is directly dependent on services having open APIs or the developers gaining access to the closed ones.

Sammandrag

Det finns ett överflöd av chattjänster och många har applikationer som alla erbjuder liknande funktionalitet. Gruppen startade projektet för att hitta en lösning på detta problem och bestämde sig för att utvärdera möjligheten i att utveckla en multi-IM klient som kan ansluta sig till alla chattjänster.

Gruppen började med att studera och undersöka hur en sådan applikation kunde utvecklas samt vilka plattformar och tjänster som var rimliga att stödja i prototypen som utvecklades. Gruppen utvecklade sedan en prototyp som stödde två tjänster och två plattformar, och bedömde resultatet.

Tekniken och metoderna som användes visade sig vara lämpliga för projektets ändamål, men potentialen av applikationen begränsades kraftigt av bristen på öppna API:er hos några av de större chattjänsterna. Dessa begränsningar är så stora att applikationens framtid är helt beroende på att chattjänsterna tillhandahåller öppna API:er eller att utvecklarna får tillgång till de stängda.

Vocabulary

Android	Android operating system. Android is a trademark of Google Inc.
API	Application Programming Interface
CLI	Cross-Language Interoperability
Client	Software (or hardware) that connects to a server or service
CPU	Central Processor Unit
DLL	Dynamic Linked Library
FFI	Foreign Function Interface
GPL	GNU General Public License
GUI	Graphical User Interface
IETF	Internet Engineering Task Force
IM	Instant Message. Chat services use IMs for communication.
IOS	Apple mobile devices operating system
IRC	Internet Relay Chat, application layer protocol for text messages
JNI	Java Native Interface
OOP	Object Oriented Programming
OS	Operating System
OSX	Apple computer operating system
P/Invoke	Platform Invoke
Platform	When referring to different platforms one refers to different operating systems
Service	A chat service (i.e. Google Hangouts, Skype^{TM}, etc.)
SO	Shared Object
The prototype	Refers to the prototype that was developed during the project
WPF	Windows Presentation Foundation. Windows C# GUI framework.
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Purpose . . . 1
- 1.3 Problem statement . . . 2
- 1.4 Design overview . . . 2
- 1.5 Project scope and limitations . . . 3
  - 1.5.1 Implemented services . . . 4
  - 1.5.2 Closed services . . . 4
  - 1.5.3 Non-implemented functions . . . 4
  - 1.5.4 Implementing on different platforms . . . 4
- 1.6 Outline of the report . . . 5

2 Project Method **6**
- 2.1 Method of evaluation . . . 6
- 2.2 Scrum . . . 6
- 2.3 Version control . . . 7
- 2.4 Project structure . . . 7
- 2.5 Work division . . . 7

3 Technical Background **8**
- 3.1 Network code . . . 8
  - 3.1.1 C++ and platform support . . . 8
  - 3.1.2 The Boost library . . . 8
  - 3.1.3 XMPP . . . 9
  - 3.1.4 IRC . . . 9
- 3.2 Clients . . . 9
  - 3.2.1 Android . . . 9
  - 3.2.2 Windows . . . 9
- 3.3 Cross-language interoperability . . . 10
  - 3.3.1 Object oriented programming languages and virtual machines . . . 11

3.3.2 Sockets . . . 11
3.3.3 Serialization . . . 11
3.3.4 Shared memory . . . 12
3.3.5 Foreign function interface . . . 12
3.3.6 Other options . . . 13
3.3.7 Cross-Language Interoperability in the prototype . . . 13
3.4 Related work . . . 13

4 Design and implementation **14**
4.1 Network code . . . 14
4.1.1 XMPP . . . 15
4.1.2 IRC . . . 15
4.2 Clients and Graphical User Interfaces . . . 15
4.2.1 Contacts instead of accounts . . . 16
4.2.2 Android . . . 16
4.2.3 Design and implementation on Windows . . . 17
4.3 Cross-language interoperability . . . 18

5 Result and discussion **21**
5.1 Implemented Services . . . 21
5.1.1 XMPP . . . 21
5.1.2 IRC . . . 22
5.2 Non-implemented services . . . 22
5.3 Implemented platforms and their GUIs . . . 22
5.3.1 Windows . . . 23
5.3.2 Android . . . 23
5.4 Function interfaces and wrappers . . . 23
5.5 Project method and its effect on the result . . . 24
5.6 Future development . . . 25
5.6.1 Potential . . . 25
5.6.2 Risks with trusting other services . . . 25
5.6.3 Improvements . . . 26
5.7 Alternative solutions . . . 26
5.8 Experiences gained . . . 27

6 Conclusion **28**

Bibliography **32**

1

Introduction

This chapter serves as an introduction to what the group consider a problem. It provides a background to the problem, the purpose of the project, what challenges the group have to address and what limitations affect the project.

1.1 Background

Chat services attract enormous user bases. There is a large selection of chat services available and many users use them daily. There is however a problem that comes with the extensive usage and amount of services: Many people feel the need to use several chat services to connect with everyone they want [1]. Statistics show that there are nearly 1.48 billion instant messaging accounts in China which is higher than the total population of almost 1.4 billion [2], [3]. This means that even if every single person had one account there would still be some people that have more than one.

The members of the group have personal experience of this abundance of services and have looked at previous attempts at solving this problem. However, these solutions did not satisfy the group's needs in a multi-IM client. The flaws that the group found are discussed in Section 3.4.

1.2 Purpose

The purpose of the project is to find a solution to the problem of having to maintain an application for every single chat service one would like to use. This will be realized by creating a prototype of a chat application that is able to communicate through many chat services. The goal with the application is to remove the need for running several applications and/or browser tabs at the same time.

The goal of the project is to create a prototype (hereafter the prototype) of an application that provides the user with a single graphical interface. The application shall keep the flexibility and reach that using several services gives, while removing the hassle of using many different applications at the same time. The user will choose between people to chat with, rather than between specific accounts at different services.

1.3 Problem statement

The project aims to test the solution of a multi-IM client. This study will evaluate the possibility of creating an application that can solve the problem with multiple chat applications. The application has to work on all popular platforms and if possible with all popular services.

The application needs to replace the default applications of the services if it is going to remove the demand for them. In order to serve as a replacement for these applications it has to provide most of the functionality they offer. To do this the application has to connect to the chat services by implementing the protocols of the services. Also, the application has to work with all of the popular services in order to provide a good solution to the problem. The problem cannot be considered solved if the application merely reduces the amount of applications the user is running.

The application's ability to solve the problem is also largely dependent on it functioning on all popular platforms available. Fully solving the problem can only be done by making the application available on all popular platforms. Otherwise simply switching devices would cause all the benefits of using the application to disappear.

It is of this application the group aims to create a prototype which will be used to test the viability of the application.

1.4 Design overview

In Figure 1.1 there is an overview of how the application is planned to be designed. The network code is the core functionality of the application, which is to interconnect the different chat services. The network code is to be platform independent. See Section 3.1 for a technical background of this part of the application. From the network part of the application there are several branches into platform specific development. A branch is a client targeting a specific platform and adapting the network code via a platform dependent interface. There is a branch for every platform the application is intended to run on. The reason for developing the clients for a specific platform is to get a good look-and-feel on the platform and it is optimal to do this with a platform specific language. See Section 3.2 for a technical background of this part of the application. Also, this design of the application makes the development modular and it is easy to adapt and add platforms. The platform independent network model reduces the amount of work

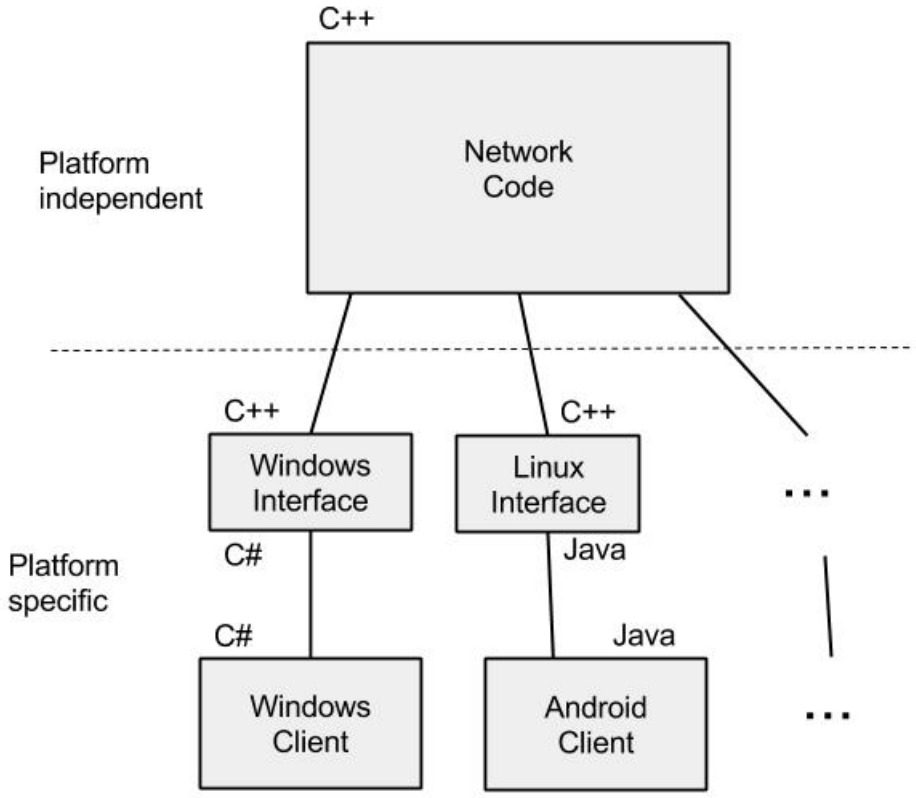

Figure 1.1: Design overview of the application

to port the application to different dependent platforms. Lastly, in Section 3.3, there is a technical background on the intended interfaces between the network code and the clients.

1.5 Project scope and limitations

The group does not aim to develop a full version of the application during this study. Instead, the focus will be to create a prototype of the application that can serve as a proof-of-concept. The goal with this prototype is to test the viability of a multi-IM client and evaluate the technologies and methods used. The prototype will lack in functionality compared to the ideal application. It will however be designed to fit the requirements of the ideal application.

Another limitation would be the avoidance of the GNU General Public License (GPL),

since it enforces dynamically linked libraries in one's project. This poses a problem since not all operating systems allow for dynamically linked libraries. IOS is one of these operating systems, which the group would not want to exclude.

1.5.1 Implemented services

The project aims to implement two services in the prototype to test the viability of several services in the same application. It will also evaluate the viability of implementing the other major services that are not implemented.

The prototype will not implement the entirety of the XMPP protocol due to time constraints. It will implement the basic and mostly used chat features, but not the voice chat, file sharing, roster editing and more, that XMPP supports. More information about XMPP is found Section 3.1.3.

1.5.2 Closed services

Some services have decided to not have open APIs or decided to close their APIs during the timespan of the project. This limited the group in terms of what services could be supported as the group did not consider it worth the time and effort to find a work around to implement these services. These workarounds would most likely include fooling these services that our application was the official one. This is not a viable solution anyway because it would most likely be a breach to their user agreements and that would put us at risk of legal action. It would also put the application at risk of becoming outdated if these services change at all as information about these changes would not be available before they are implemented.

1.5.3 Non-implemented functions

The goal of the application is to interact with several different services and be able to send messages to several services at once. The prototype was limited to sending only plain text messages. Other types of messages, for instance messages containing images, was considered out of scope for this study. The group wanted the focus of the application to be an experience where the user does not have to think about what kind of services they are using by hiding all service related accounts. To do this, the most compatible form of communication is text.

1.5.4 Implementing on different platforms

For an application to be able to properly replace the default applications of chat services it has to be able to run on all popular platforms. Otherwise the original problem would still remain on other platforms.

Implementing on all platforms is however outside of the scope since it would require more time than is available for the project. The goal of the project will be to implement

the prototype on two different platforms in order to test if the application can work both on desktop as well as on mobile devices. If the project was to be extended, it would be easy to introduce the prototype on other platforms since the underlying functionality does not need to be rewritten for a certain platform.

The operating systems chosen were Windows for the desktop version and Android for the mobile platform as they have the largest market shares of their respective markets [4], [5]. Implementing a complete GUI on both platforms is also considered outside the scope for the study. Focus will instead be on creating a functional GUI supporting the basic features of the application. The aim is to create a proof-of-concept that can help evaluate the potential and possibility of creating a full version of the application.

1.6 Outline of the report

This report aims to first introduce the problem, the project's purpose and what approach the group had when trying to solve it. It describes the methods used during the project in Chapter 2. It gives a background on the more technical aspects of the project such as the libraries, languages and previous solutions in Chapter 3. The report discusses the decisions made during design and implementation in Chapter 4, before presenting and discussing the results in Chapter 5. Lastly in Chapter 6 it gives a conclusion.

2

Project Method

This chapter describes the methods that were used during the project. This mainly focuses on development method and techniques used during the development of the prototype. Method of evaluation is also considered.

2.1 Method of evaluation

The group wanted to evaluate the possibility of creating a multi-IM client. In order to get the experience and knowledge needed the group developed a prototype of a multi-IM client. With the experience and knowledge the group gained, it could determine if there was potential for a multi-IM client or not.

2.2 Scrum

The project work was structured using Scrum. Using Scrum the work was divided into sprints, which are small projects in themselves that have a clear goal and timeframe [6]. Before each sprint a planning meeting is held, during which the goals of the sprint are defined. The goals are selected from a product backlog in which all the parts needed for the product are stored and are added to the sprints backlog.

During the sprint each of the goals should meet the Definition of Done which means that the goal is fully met and shippable. The group holds short daily meetings during which each team member presents what they did yesterday, what they are doing today and what issues they have that might affect the sprints goals.

After each sprint the team evaluates the goals of the sprint and determines the status of these. The product backlog is updated with the goals that are done and goals are

changed as needed. Time plan is reevaluated and changed if needed [7].

2.3 Version control

To manage the code during development the distributed revision control system Git was used. Git is distributed so that each user has a copy of the main repository in order to remove any single point of failure that could occur if there was a main server that kept the repository and the users only checked out the code. Git is free software under the GNU General Public License v2 and is one of the largest code management tools on the market [8], [9].

2.4 Project structure

The project started with a research phase. The group researched what services were popular and the possibility of including these in the prototype. Different technologies were researched before the group designed the application and decided upon platforms and services to implement. The group then researched the platforms and what approach to take when creating the prototype for these. Afterwards the group implemented the prototype of the application before evaluating the results at the end of the project.

2.5 Work division

The group was divided to have different responsibilities in the design and implementation phase of the project. There were mainly three areas to be responsible for, they are the network code, platform client development and the interfaces between different clients and the network model. A clear view of the different work areas can be seen in the planning overview of the system design in Figure 1.1. This division of work areas made it possible for the group to work independently on different parts of the application at the same time.

3

Technical Background

The network code is developed in C++ and the GUI in C# and Java for Windows and Android respectively. This chapter gives a bit of background to the libraries Boost and WPF as well as the protocols XMPP and IRC that were used in the project. It also introduces the available options for communication between C++ and Java/C#.

3.1 Network code

3.1.1 C++ and platform support

The network part of the application is to be platform independent. This requires the network code of the prototype to be written in a language that is supported on all platforms. The programming language C++ is supported on all platforms. A study shows that using C++ compiled code can give adequate results when it comes to performance on smaller devices such as Android [10].

3.1.2 The Boost library

Boost is an open source library for C++ [11]. Its goal is to work well with and extend the C++ standard library. Many of the Boost libraries have been standardized [12]. The Boost library that the project uses most frequently is the Boost.Asio library. Boost.Asio is a network programming library that uses the current operating system's socket library in the background, making it cross-platform compatible. The prototype utilizes Boost.Asio for connecting to and communicating with chat servers, for example connecting to Google Hangout servers and communicating through the XMPP protocol.

3.1.3 XMPP

XMPP is an Internet Engineering Task Force (IETF) standard for real time communication [13]. XMPP utilizes XML streams to send XML stanzas. An XML stanza can contain an instant message, presence information, contact list maintenance commands or other types of option commands. Since XMPP is defined as an open standard, it is one of the non-proprietary instant messaging protocols. The prototype uses the XMPP protocol to support the Google Hangout service. XMPP also allows for future support of a plethora of less commonly used services that are built on the communication standard.

3.1.4 IRC

The Internet Relay Chat (IRC) protocol is used for transmitting text messages between a client and a server. IRC is not commonly used as a conversation between two individuals, but instead as a one-to-many conversation where multiple users participate. This kind of conversation takes place in a channel. However it is easy to establish a private conversation between two users in a channel through a certain prefix in the IRC protocol. All messages that are sent to the server is preceded by a prefix that the server interprets in order to perform the corresponding action. The preceding prefix for a message specifies whether it is a server, channel or user message.

3.2 Clients

3.2.1 Android

Android is the most popular smartphone operating system on the market and has 76.6% (2015) of the market [5]. Android is based on the Linux kernel and is open source. Development for Android is mainly done in a Java like language, but does not run a Java Virtual Machine. Instead Android runs its own virtual machine called Android Runtime. Developing for Android has a lot of advantages for the developer that is already used to Java as the differences in the languages are minuscule.

3.2.2 Windows

Windows is the most popular PC operating system on the market [5]. To allow for simple creation of Windows applications, Microsoft has provided Windows Presentation Foundation (WPF). WPF is a framework developed by Microsoft that is used to develop GUIs for Windows applications [14]. WPF uses Direct3D, which is a part of the DirectX family for rendering and allows for both 2D and 3D graphics.

WPF supports graphical interfaces, on screen documents, fixed form documents, images, video and audio. In addition to the support, WPF also provides many base graphical components such as buttons and windows. It also handles all the rendering of these graphical components. Developing applications in WPF is mainly done by defining the

look of the application in XAML (Extensible Application Markup Language) and its behaviour in code behind using either C#, C++, JScript or Visual Basic [15].

3.3 Cross-language interoperability

This subsection reports a small study on cross-language interoperability (CLI). The study focuses on problems with the subject and methods for implementation.

Due to having different parts of the application built in different programming languages, there must be a way to communicate between them. Some object oriented programming languages can interact and use objects created in other languages thanks to the Virtual Machine they run on [16]. For the prototype the CLI problem is about exposing functionality of unmanaged code to managed code. The explanation of unmanaged and managed code follows in the next paragraph.

The network part of the prototype is written in the programming language C++ and is compiled to low level machine code in the form of binary libraries. This type of code is called unmanaged code. The binary libraries can be used by other code when linked to it. The libraries can be linked either in a dynamic or static way. The network code is compiled to a dynamic linked library, called DLL on Windows. The client part is written in the native languages for the different platforms and this type of code is called managed code. The managed code is compiled to intermediate code, instead of machine code, targeting a Virtual Machine. The Virtual Machine interprets the intermediate code to the underlying functions of the platform it is run on. The managed code is checked to be safe and will be contained to the virtual machine. Another advantage with running in a managed environment is that there is garbage collection that manage the memory. The unmanaged code is closer to the machine and therefore often faster than the managed code, but the disadvantage of running the unmanaged code is that it can crash the system or run out of memory. [17]

As listed above, there can be several problems with handling unmanaged code when running it. In addition, there can be several problems interacting with the unmanaged code from the managed code. To begin with, operating between different programming languages is in general a difficult task. When interoperating between managed and unmanaged code there has to be an interface between them to expose the functionality to each other and enable interaction. Furthermore the data types and objects are represented differently between managed code and unmanaged code which has to be interpreted correctly between them. Calling conventions are also an important issue when calling functions or object methods between the different languages. Managing threads and handling exceptions are two other possible problems in this matter. [18]

The following content of this subsection describes general ways how to interoperate between different programming languages and the focus is on the ability to interoperate

between managed code and unmanaged code.

3.3.1 Object oriented programming languages and virtual machines

As mentioned previously, some object oriented programming (OOP) languages can interoperate with each other. For this to be possible the OOP languages have to be adapted or designed to run on a specific Virtual Machine. The hosting Virtual Machine manage the interoperation between the supported languages that is run. For example there are a list of .NET common language interface compliant languages that run on the common language runtime, CLR, virtual machine and there are Java Virtual Machine (JVM) compliant languages that run on the JVM. There is also a version of C++ that adapts the common language interface and can be run on the CLR machine. This technique is not suitable for interoperating between managed and unmanaged code, because the unmanaged code is not run in the different virtual machines but used by the code running in the machines [19], [20].

3.3.2 Sockets

There is a way to communicate between different programming languages using network sockets. To be more specific, the interaction is performed on the internal network of the platform an application is run on. In more detail this means having the different parts of an application that is interacting with each other, acting as either client or server. The network socket is setup to be the address of the own system and for a specific port. The client sends data to the socket and the server is listening and receiving data on the same socket, both parts running on the same platform. The concept of CLI with sockets is simple and controlled. A problem with this method is that the data sent between the languages has to be serialized to be interpreted correctly, see Section 3.3.3. Another problem is that there is an overhead work needed when calling the API for the sockets of the underlying system [21].

3.3.3 Serialization

Serialization of a data type or an object converts it into a stream of bytes representing the data. The process makes it available to store the state of a data type or an object in a file or database or to transmit it in memory or via sockets. Deserialization is the reversed process, that is converting a stream of bytes into a data type or an object.

A method for interoperating is to serialize objects in managed code and deserialize the objects in unmanaged code or vice versa. In most OOP languages there is a built-in mechanism for serialization of data types and objects. The problem is that it is difficult to perform this method between different programming languages and requires much time and work to implement. Also, the process of serializing an object of complex form requires much CPU time, that is a communication overhead [22].

3.3.4 Shared memory

Two different programming languages can interoperate between each other by using shared memory. This method means to transmit data in memory space that is shared between the interoperating languages. The shared memory can either be in form of allocated space in the primary memory or files stored on the hard drive disks. In most cases, some sort of serialization and deserialization of the data is needed when interoperating between different programming languages via shared memory. Shared memory can be used to communicate between managed and unmanaged code. In C# there is a way to allocate global memory that can be accessed by unmanaged code by a pointer to this memory. In Java there are buffers of different data types that can be used to share memory with unmanaged code [23], [24].

3.3.5 Foreign function interface

A foreign function interface (FFI) is a mechanism to make a programming language able to call functions written in another programming language. Some foreign function interfaces also allow to call object methods. In most cases, a FFI is designed to be used by higher level language to invoke functions in lower level language, like C or C++. This design allow managed code to call functions, outside the virtual machine, in unmanaged code. Many FFIs also allow to invoke functions in both directions, that is a called programming language can invoke a function in the calling language. The FFI handles the calling conventions between the different programming languages interacting with each other. Also, the concept of using the foreign function interface mechanism is simple and easy to implement [25]. The problem with this interoperating method is that data types or objects sent between the languages, return values or parameters, has to be marshalled - another word for serialization. In most cases marshalling of simple data types is not a problem, but marshalling more complex data types like objects can be ambiguous [26]. Also, there has to be some overhead work, due to calling conventions and marshalling data, in the runtime machine when calling functions in another language which requires CPU time [27].

The programming language C# has a mechanism named P/Invoke, short for Platform Invocation Services, to call or invoke unmanaged functions in a C or C++ written DLL. The managed C# code imports the unmanaged C++ DLL and is then allowed to invoke the exported unmanaged functions [28].

Considering the Android client written in Java like language, there is a corresponding FFI mechanism called JNI, which is short for Java Native Interface. JNI works in the same way that P/Invoke does, it allows the managed Java client code to invoke functions in the unmanaged C++ code. The managed Java code imports an unmanaged .so file, which is short for shared object file and is the corresponding DLL on a Linux platform [29], [30].

3.3.6 Other options

There are other ways of interoperating between managed and unmanaged code than the ones listed above, but they are not as commonly used or are less supported between different programming languages. For instance the Component Object Model, COM, which is a binary-interface standard for exposing objects to other programming languages and environments than where they were created. The COM interface was introduced by Microsoft and has support for all .NET languages [31]. C++ is well supported by the COM interface, however support on the Linux platform is lacking. Another way to interoperate between managed and unmanaged code, is to build a service in unmanaged code and expose an API that the managed code can use to call into the service. This technique is difficult and time consuming to implement [32].

3.3.7 Cross-Language Interoperability in the prototype

In the prototype the cross-language interoperability method chosen was foreign function interfaces. Design and implementation of this as well as the motivation behind the choice is covered in section 4.3.

3.4 Related work

There have been several attempts at a solution similar to the one in this study. However the ones found by the group did not come up with a good solution to the problem. The three attempts that the group found were Pidgin, Adium and Trillian. Neither of these works on all popular desktop platforms as well as all mobile platforms. Pidgin does not work on any portable devices, Adium only works on OSX and Trillian does not support Linux. All three of them had lack of service coverage which the group considered an important aspect of providing a good solution. This lack of coverage included Skype for Pidgin and Adium as well as Skype on other platforms than Windows for Trillian. Pidgin and Adium are pretty much only GUIs that use the open source library libpurple which does all of the network and logic. Libpurple provides most if not all the functionality the application needs. However libpurple was not suitable since it has a GPL license [33]. This does not work for us, since we don't want to exclude IOS in the future. On IOS all libraries have to be statically linked to, but the GPL license says that you can only link dynamically to the libraries in question.

4

Design and implementation

This chapter presents what features were included in the prototype as well as discussing the decisions made during design and implementation of it.

The ideal application should be able to fulfill our main goals, but it should also have a realistic user experience and therefore design is important. The prototype was designed accordingly to this.

4.1 Network code

The group decided early on that the C++ network code should be stateful which means that it keeps data in memory on the computer and not only provide an interface to the implementing clients. The network code maintains the connections to different servers and store it in its memory. It also stores data regarding connected services as well as the persons with their accounts that the user can communicate with. The GUI also needs to keep data regarding persons and accounts to show the user.

All this makes it easier to develop the application for many platforms because the more functionality the C++ code provides, the less code needs to be rewritten for each client application. The same C++ code will then be used on all platforms. It also makes sense from a software architectural point of view. To do any networking, the network code needs to use low level communication objects (such as sockets and Secure Sockets Layer streams). This make it logical to store these objects and related data in the network code library.

Another early decision the group made was that new services should be easy to add. This decision drives the code architecture towards using many interfaces. In C++ the closest thing to a C# or Java interface is a class with only pure virtual methods. Having

collections of interfaces enables heavy use of polymorphism, making it easy to add new implementations of said interface without having to rewrite code.

4.1.1 XMPP

The group decided early on to do its own implementation of XMPP. The major reason behind this choice is that all well documented XMPP libraries we found (with libpurple being the most common) all have a license that was not suitable for the needs of the project. This is described in Section 3.4.

While implementing parts of the XMPP protocol in the networking part of the application the group made the choice to implement the authentication process as a synchronous sequence of messages to and from the XMPP server in question. This decision was made because the messages that need to be sent and received during authentication has a strict order.

4.1.2 IRC

It was decided to scope the Internet Relay Chat (IRC) protocol to only private conversations on a server. For future expansion, the functionality for one to many conversations is already implemented due to the easy use of the protocol.

The implementation is based on a connection to an IRC server where threads handle the reading and sending of messages. A message that is received is parsed to only contain the actual text of the message which should be displayed to the user. The IRC implementation also handles server messages such as keeping the connection alive automatically.

4.2 Clients and Graphical User Interfaces

When developing the Graphical User Interface (GUI) the group took into consideration what information was important to display to the user and how it can be displayed efficiently. The group also decided to develop a GUI unique to each platform as the targeted platforms provide vastly different possibilities and difficulties.

Smaller devices such as smartphones usually have a touch screen which can make for a GUI that is intuitive and easy to use. They are however limited by their screen size which makes presenting much information at once difficult. A finger is also less accurate than a mouse pointer therefore it is important not to have elements too close to each other, to avoid frustrating the user.

Devices with larger screens such as desktops can present several areas of information at once to the user. Each of these areas can also include more information than what the entirety of a smaller screen could. This allows for showing large amounts of information

to the user. However there is still a limit to how many areas of information that can be presented at once as the screen can quickly become cluttered otherwise.

The GUI was designed so that the user should be able to switch from a platform to another and be familiar with options presented to them. This was not the main focus for the GUI during the project, however it was taken into consideration when designing the GUI.

4.2.1 Contacts instead of accounts

An important aspect of the GUI that applies to all platforms is the abstraction of the different accounts that belong to each contact. That is to say that the user should not have to worry about what service they need to send their message with and should instead be able to fully focus on their conversation. The group still wanted to give the user some idea of what service the messages were sent through. Each message is thus displayed with a list of icons depicting what services was used. This can be seen in Figure 4.2 to the left of the message.

It was also decided that the user should have the option to select what services they want to use for each message and only have the application to provide good default services for each conversation. This is presented differently on the two platforms as the amount of screen space on smaller devices did not allow for a list of check-boxes near the writing area. It was made available as a pop-up checklist that is accessed in the top menu. It was more important to create a functional GUI on each individual platform than to make every aspect of the GUI function the same on all platforms.

4.2.2 Android

When creating a GUI for a mobile device the group was very careful with how the screen space was used as it is very limited. This resulted in the list of contacts not being visible at all times but instead being hid in a so called drawer on the left side of the screen. The contact list drawer also does not take up the entire screen when pulled out as shown in Figure 4.1. Instead it allows the user to still look at the conversation while also looking at the friends list. The list is also slightly transparent which effectively allows for the same screen space to be used twice which is important when the screen is so small.

The start screen is designed to give the user a quick overview of what have changed since they last used the application. This is done by showing the conversations that most recently received a message and the content of it. The start screen also provides the user with some menu options such as creating a new contact.

When selecting services for a message it was decided to hide this a bit more compared to the desktop application as adding more options at other places than the menu bar would clutter the screen too much.

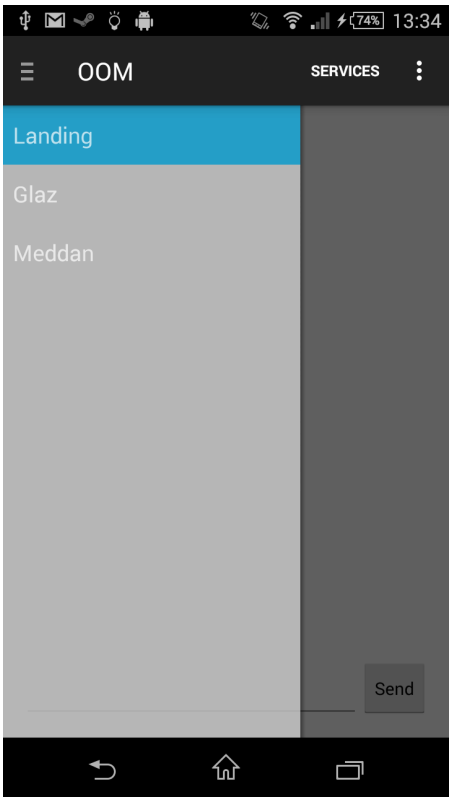

Figure 4.1: The prototype on Android with the contact drawer pulled out from the left.

4.2.3 Design and implementation on Windows

It was decided to use the framework WPF to develop the applications GUI for Windows. WPF was chosen due to it being supported by the Visual Studio IDE (Integrated Development Environment) [32] which the group was already using to develop the network code. WPF is also developed by Microsoft and is well supported on Windows.

The creation of a GUI for desktop provided different problems and possibilities compared to the creation of a GUI for a mobile device. The group had to consider how to use the larger screen space effectively while not making the application cluttered. The result is shown in Figure 4.2.

The larger screen on desktops allows for several windows to be presented to the user at once which allowed for separation of the friends list from the chat window. This would reduce the amount of information presented to the user unless they choose to view it and also allow the user to resize the application window more freely.

Since part of the goal was to reduce the amount of windows and browser tabs running it seemed only natural that the application should not cause unnecessary clutter by having

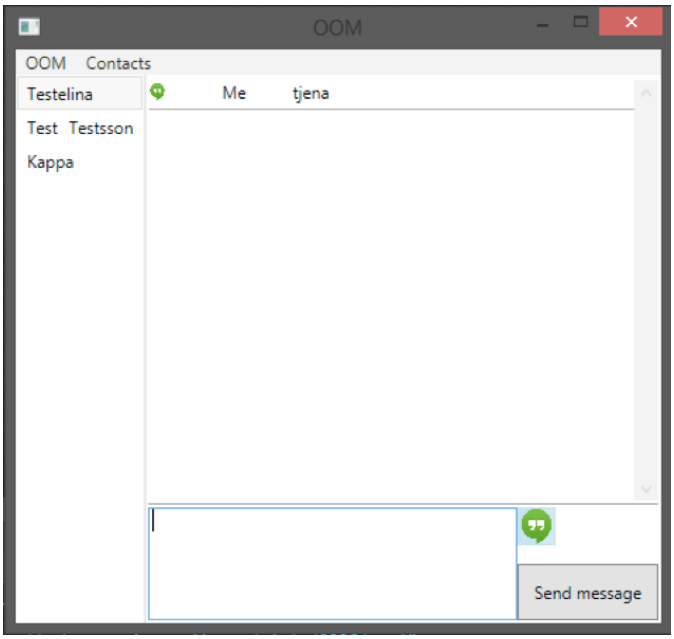

Figure 4.2: The prototype on Windows. Showing a contact list on the left and a conversation with a message on the right. At the bottom the chat box, with the send button and the service selection to it's right.

multiple windows as default. It was considered to include multiple windows as a preference however it was not included in the prototype due to lack of time.

In order to create as much familiarity across platforms as possible the friends list was placed in the same window as the chat. This allowed for the same usage pattern as the smartphone version. Having the friends list and chat in the same window also seemed to fit the project's goal of reducing the number of applications and browser tabs by not cluttering the desktop of the user more than necessary.

Since the desktop application had more space available it was decided to provide the user with a quick and easy way of selecting services. This can be seen in Figure 4.2 where service selection is available on the main conversation screen in the bottom right. This was a good way of using the additional screen space that the desktop platform provided as additional information could be presented in an intuitive way without cluttering the screen.

4.3 Cross-language interoperability

For the prototype the cross-language interoperability method chosen was FFI, see Section 3.3 for a technical background of different CLI methods. The reason for this is that the technique is more straightforward and requires less work to implement than the

other methods. Due to the time limit in the project the amount of work to implement the CLI method was an important aspect.

The technique chosen for interoperating between C# and C++ is P/Invoke and JNI between Java and C++. With the FFI mechanism, the way to interoperate between the different programming languages is by implementing a function interface. In the prototype, the interface defines the functions in the C++ network code that can be called from another language. Even though the two techniques P/Invoke and JNI are similar to each other, there has to be an implementation of the function interface for each technique.

The function interfaces are written in C++ and are adapted to the different programming languages used for the different clients by a wrapping class for each. There are wrapping classes written in C# and Java for the Windows desktop client and for the Android client respectively.

A decision was made that the defined functions in the interfaces handle simple data types as parameters and return values, instead of using more complex data as objects and structures. This makes the implementation of the function interface simple and easy to understand. Simple C types are easily converted to another language. For instance an integer in C# translates to an integer in C++ and the same translation works with the JNI mechanism. Another reason for this choice was to make it easy to know what memory is owned by which part of the application to keep memory related issues to a minimum. Also, this reduces the amount of CPU work spent on overhead when calling the functions, as there is less complex data components marshalled between managed and unmanaged code. When dealing with transmitting messages of various length between managed and unmanaged code, buffer objects were used. A C# string buffer is converted to a null terminated constant character pointer in C++, which basically is a string. For Java buffer objects this is not as easy converted into C++, but one has to consult the JNI mechanism for the proper conversion into pointers. The buffer objects are complex data types but are defined in the FFI mechanisms to be used for transmitting fields of various size of a specific data type. Another way to do this is to call a function returning character by character. The latter alternative is worse in CPU overhead work and therefore not considered for implementation.

Both P/Invoke and JNI have the ability to invoke functions in the calling programming language. This means that C++ can call back into C# or Java. In P/Invoke this is done by delegating a function in C# and pass it as a parameter to C++ where it is marshalled as a function pointer which can be called. In JNI it is easy to invoke an object method of an object reference that is sent by parameter to C++. When implementing the reading of incoming messages in the network code from the client code, there were two design approaches considered. These approaches were either to call back into the C# code when a message had arrived or to poll a function that returns true or false whether a message had been placed in a message buffer or not. Polling is worse with respect to CPU work

but is not as difficult to implement as the other option mentioned. Due to lack of time the polling method was implemented.

5

Result and discussion

This chapter presents and discusses the result of the project. It evaluates the technologies, methods and approach that were used and their effect on the result. It also discusses the potential for further development of the prototype and potential improvements.

5.1 Implemented Services

The services supported in the prototype are IRC and Google Hangouts. They shared the important aspects of having an open API and availability of the API on all our supported platforms. In addition they were both based on open protocols which made them less tied to a single company. This ensured that there was plenty of documentation available during implementation. It was particularly important to the prototype as well as there were only a limited amount of services that could be included in the given timeframe. The prototype would not be able to test the viability of a multi-IM client properly if it could not support two services.

5.1.1 XMPP

Since the prototype has implemented the XMPP protocol it supports all services using the XMPP protocol. The prototype however does not support manual connecting of servers so in practice it only supports Google Hangouts.

The implementation of XMPP was also meant to give support for other large services, mainly Facebook. However Facebook decided to discontinue their XMPP support and with it all their support for access to their chat API. This made supporting Facebook not an option, however there are several other large chat services that make use of the XMPP protocol and a large amount of smaller services.

The group found the inclusion of XMPP support a good idea in an application of this kind, since even though the prototype only supported Google Hangouts, increasing the functionality to include all XMPP services is trivial. The XMPP protocol does not give the group access to the most popular services, but it gives access to a lot of smaller ones.

5.1.2 IRC

IRC was deemed a safe option since it does not have all its users centralized to a single service, thus eliminating the risk of the service becoming unavailable and reducing the risk of the project's effort resulting in a prototype only supporting a single service. This would mean that the prototype became a total failure since it could not give any information about the viability regarding multiple services in a single application.

Similar to XMPP, implementation of one IRC service implies support of all services based on the technology with minor or no adjustments needed.

5.2 Non-implemented services

While trying to create the prototype the group encountered a lot of problems with a lot of the popular chat services on the market. The group had chosen to implement usage of the XMPP protocol in the prototype as that would cover both Facebook and Google Hangouts. As more research was conducted it came to light that Facebook had decided to discontinue their XMPP API and thus Facebook support for the application was ruled out. Another big service that was initially planned to be supported was Skype. Skype however does not provide any other APIs than the desktop API that is only available on Windows. Including Skype was then ruled out as the benefits of supporting the service if only on one platform was outweighed by the confusion and inconvenience it would cause to not have the service available on all platforms.

Several large services were not considered for support when developing the prototype due to a lack of open API. Services such as Facebook and Skype we would want to support, had they provided the needed APIs. The possibility of fooling these services into thinking that our client was the official one was considered. However this would break the terms of use of the services which would put the user's account at risk, making the application a liability for the user.

5.3 Implemented platforms and their GUIs

Listed are the different features implemented in the different GUIs:

- Send message
- Add contact
- Add account to a contact

- Choose service to send a message on
- Change contact information
- Remove contact

In addition to this the Android client is able to display notifications when a message is received.

5.3.1 Windows

The GUI for the Windows platform was developed in WPF. WPF proved to be easy to work with and allowed the group to quickly create a functional GUI without much hassle. A big feature of WPF was that it allowed for data binding between data in the back end code and the GUI. This kind of binding allowed a lot of separation of model and view while still having the GUI be responsive to changes in the model.

Choosing a natively supported framework proved to be a good idea as WPF provided the group with many good default graphical components that made creating an application that looked and behaved like a normal Windows application easy. There were many online resources available when developing which helped the group. However the group found that the official documentation about WPF was lacking.

5.3.2 Android

Android proved to be a suitable platform for the application. The development encountered few issues and creating a graphical interface with the native components using the graphical designer in Android Studio made the development go smoothly [34]. There was also a large amount of information available which helped the group during development.

Developing the prototype for smaller screens turned out to be a challenge, but was overcome with smart design. Design such as the use of a drawer for the contacts list instead of showing it at all times allowed the screen space to be used effectively and create intuitive controls for a touch screen.

The group had some previous experience in working with Android and a lot of experience working with Java which increased productivity as the group did not have to do a lot of research to get work started.

5.4 Function interfaces and wrappers

The result of the study in cross-language interoperability is that the method used in the application prototype was foreign function interfaces, see Subsection 3.3.5 for a technical background. For each platform client developed, there was a function interface with a corresponding language wrapping class implemented.

The function interface mechanism was simple and made it easy to implement communication between different programming languages. The chosen technique adapts well with the application prototype. There were other techniques able to work in the application prototype as well. For instance, using sockets to communicate between the clients and the network code. But due to the time constraints in the project this method was considered not reasonable to implement. Many of the methods in the study in cross-language interoperability are well suited to combine with each other. An example of this is that the foreign function interface could have been combined with usage of shared memory to send complex data objects with help of serialization between the different languages. This combination could be useful if there is a need to send more complex data.

If there would be a further expansion of the prototype, there would most likely be a need for sending more complex data between the clients and the network code. Therefore the aforementioned combination of different CLI methods would be in interest. With a further expansion, the implementation of reading incoming message has to be improved. As it is in the prototype, this is done by polling a function returning true or false whether a message has arrived or not. Polling is not considered a good implementation style in performance aspects. In a further expansion this would be done with callbacks instead.

5.5 Project method and its effect on the result

The group did not use much of the Scrum method as a lot of the work was done separately and it did not seem like a formal development method was needed. This resulted in the project not using almost any of the Scrum method. The parts of Scrum that were still used ended up being the sprints although almost none of the formalities such as sprint retrospective and sprint backlog were used formally. Instead of having daily Scrum meetings the group tried to have meetings twice a week to discuss progress and issues.

Since the group did not use the Scrum method a meaningful discussion about its qualities as a development method cannot be held. However the group felt that as the project proceeded, a more formal way of keeping track of progress and backlog would have helped the group stay on track and make more progress. A more formal development method would also have helped the group keep better track of who's doing what and how much progress has been made in each area.

The group decided to not have the daily Scrum meetings as the work in the project was limited to four hours a day rather than the normal eight hour workday and in addition not all of the time spent working was available for development. The group also had varying schedules which made daily meetings not an option. The group tried simulating these daily meetings by having meetings twice a week. However the group felt that it was not sufficient at times and the time between meetings when one was cancelled or a large part of the group could not attend became too great. When meetings were close to daily

the group had not really made any progress between the meetings. Sometimes some members of the group had done no work at all thus making the meeting non-productive.

The group felt that following a formal development method more closely would have provided the project better structure, improved the overview and increased productivity. However the group felt that the Scrum method of daily meetings and sprints did not fit due to the circumstances of the project.

5.6 Future development

5.6.1 Potential

The group is of the opinion that the application has some potential, however this potential is greatly limited by the larger services lack of open APIs. Creating a multi-IM client that works on all platforms and supports all services is not limited by the technologies available. It is however limited by the large companies not providing open APIs for the general public to use. The fact that both Facebook and Skype were not possible to implement is a huge detriment to the potential of the application as it cannot be considered a good solution if not all the large services are covered. However if these large services would start offering open APIs the potential of the application would be greatly increased.

The application provides some improvements compared to the existing solutions which the group considered valuable. The increased focus on contacts instead of accounts could provide improved user experience for previous solutions. The ability to work on all platforms is something that is needed in the previous solutions, however these solutions would have to switch license entirely which might prove a great hindrance.

5.6.2 Risks with trusting other services

There are some risks involved when developing this kind of application. Most of the features of the application are dependent on the services provided by other companies. This puts the entire functionality of the application at risk should these companies stop providing open APIs. Even if there are deals made with these companies to use their private APIs, there is no guarantee that new companies entering the market would be interested in making these deals.

It is understandable for chat service providers to not provide open APIs. Many of them earn money from advertisement, and this revenue will be diminished if users were using other applications to access their chat service as they would not be exposed to their advertisement.

5.6.3 Improvements

There is much room for improvement in the prototype that was created. However there are some improvements that the group feels are more important for a final version to implement:

- *The coverage of services.* The application that was developed could not support all the popular services due to time constraints. However large services such as Skype and Facebook were not included as the group did not have the ability to support these. In order to create a final version of the application a solution to this issue needs to be found.
- *Sharing data between different platforms.* The application has to synchronize data between platforms such as contacts and conversations. A user would quickly become frustrated if the application did not share this data.
- *Smart messaging.* Storing all contacts and their accounts in a single application is a way of reducing the amount of applications that the user needs. However the user experience can be greatly increased should the application be able to prioritize how to send the messages. This can be realized by having the application check the online status of the different accounts of the recipient and according to this information choose the best option.

5.7 Alternative solutions

The group identified an alternative way of solving the issue with too many chat applications: creating an entirely new chat service to replace all the currently popular ones. This would eliminate the need of other services and thus other applications. However the group found several flaws with creating an entirely new chat service.

The biggest flaw was the fact that the success of a chat service is largely dependent on its popularity. This means that this solution would have to become so popular that no other services would be used. However this is not a realistic goal and by creating another chat service we would further increase the problem. Introducing another service to the market would only split the user base even further and contribute to the problem.

Introducing another service also takes away from the fact that some of the popular chat services such as Facebook have functionality beyond chatting. If a user would still want to access this functionality they would have to still use that service. If the user then wants to chat with their friend on Facebook they will have their conversations with that person split over multiple applications or have to launch another application.

These flaws were large enough to disregard creating a new service as a solution. It would allow people using the application to chat with those who did not. The application would still be useful even if it did not get a large user base as it would only provide improvements

to the user regardless of its popularity. The user could always use whatever application is most convenient at the moment.

5.8 Experiences gained

Developing the network code gave experiences in programming C++, network programming on the socket layer and implementing a protocol from its official specification. In this part of the application, asynchronous multi-threaded programming has been used and learned. In addition, the group has studied interoperability between different programming languages and learned how to implement this.

The implementation of the clients on Android and Windows gave the group experience in designing user interfaces. The development of the Android version gave the group experience in developing for mobile devices. Developing for Windows also gave the group experience working with C# and WPF.

6

Conclusion

The initial plan was to develop a prototype which could support all popular chat services and work on all platforms. The prototype was to support two services and work on two platforms to test this functionality.

After doing some research and starting the creation of the prototype, the group encountered difficulties when selecting services. This was due to several large services withdrawing or never providing open APIs. The services Google Hangouts and IRC provided open APIs and were thus selected for implementation. It was decided to support the platforms Android and Windows as they were the largest on the market.

The group succeeded with the initial goal of testing the viability of a multi-IM client. This was done by creating the prototype which satisfied the requirements of two services and two platforms. The prototype and the process of developing it could then be used to evaluate the viability of a multi-IM client.

The group considered the application itself a success since it could provide the functionality and user experience the group was looking for. The technologies that the group used were considered appropriate and well suited for the task. However the group encountered a lot of troubles with the chat service providers due to the lack of public APIs.

Due to developing in C++ there are no problems supporting all platforms and using the same network code on all platforms can be done by using some form of CLI. The techniques used for CLI in this project would however have to be extended or changed in order to send more complex data between the network code and a client.

The application can function on both desktops as well as mobile devices and the increased focus on contacts instead of accounts was considered a success by the group.

The future looks bleak for an application of this kind as its functionality is dependent on service providers providing open APIs. This causes great risks for anyone creating this kind of application, as their work can be wasted by the service providers. The potential for an application to handle all services at once is definitely there, but without cooperation from the service providers that potential will not be realized.

The group can conclude that an application of this kind cannot provide the solution they were looking for. The multi-IM clients available on the market can be improved by supporting additional services and increasing their focus on contacts instead of accounts. However they are very limited in their coverage of services and supporting of platforms.

Bibliography

[1] J. Perlow and J. Perlow. (2013) Instant messaging clients: Stop the insanity | zdnet. Accessed: 2015-05-12. [Online]. Available: http://www.zdnet.com/article/instant-messaging-clients-stop-the-insanity/

[2] C. Clifford, "Top 10 apps for instant messaging (infographic)," 2013, accessed: 2015-05-12. [Online]. Available: http://www.entrepreneur.com/article/230335/

[3] Worldometers.info, "Population of china (2015) - worldometers," 2015, accessed: 2015-05-12. [Online]. Available: http://www.worldometers.info/world-population/china-population/

[4] Netmarketshare.com, "Operating system market share," 2015, accessed: 2015-05-12. [Online]. Available: http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0

[5] www.idc.com, "Idc: Smartphone os market share," 2015, accessed: 2015-05-12. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[6] Scrum.org, "Scrum.org | the home of scrum > resources > scrum glossary > definition of done," 2015, accessed: 2015-05-12. [Online]. Available: https://www.scrum.org/Resources/Scrum-Glossary/Definition-of-Done

[7] Scrumguides.org, "Scrum guide | scrum guides," 2015, accessed: 2015-05-12. [Online]. Available: http://www.scrumguides.org/scrum-guide.html

[8] Openhub.net, "Compare repositories - open hub," 2015, accessed: 2015-05-12. [Online]. Available: https://www.openhub.net/repositories/compare

[9] Git-scm.com, "About - git," 2015, accessed: 2015-05-12. [Online]. Available: http://git-scm.com/about/distributed

[10] Y.-J. Kim, S.-J. Cho, K.-J. Kim, E.-H. Hwang, S.-H. Yoon, and J.-W. Jeon, "Benchmarking java application using jni and native c application on android," in *Control,*

Automation and Systems (ICCAS), 2012 12th International Conference on. IEEE, 2012, pp. 284–288.

[11] C. Kohlhoff, "Boost.asio - 1.57.0," 2015, accessed: 2015-05-12. [Online]. Available: http://www.boost.org/doc/libs/1_57_0/doc/html/boost_asio.html

[12] Boost.org, "Boost c++ libraries," 2015, accessed: 2015-05-12. [Online]. Available: http://www.boost.org/

[13] Xmpp.org, "About - the xmpp standards foundation," 2015, accessed: 2015-05-12. [Online]. Available: http://xmpp.org/about-xmpp/

[14] Msdn.microsoft.com, "Introducing windows presentation foundation," 2015, accessed: 2015-05-12. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa663364.aspx#introducingwpf_topic8

[15] Msdn.microsoft.com, "Introduction to wpf," 2015, accessed: 2015-05-12. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa970268(v=vs.110).aspx

[16] Msdn.microsoft.com, "Cross-language interoperability," 2015, accessed: 2015-05-12. [Online]. Available: https://msdn.microsoft.com/en-us/library/vstudio/a2c7tshk(v=vs.100).aspx

[17] D. Bolton, "Why managed code is safer - dice insights," 2014, accessed: 2015-05-12. [Online]. Available: http://insights.dice.com/2014/01/29/managed-vs-unmanaged-code/

[18] Msdn.microsoft.com, "An introduction to p/invoke and marshaling on the microsoft .net compact framework," 2015, accessed: 2015-05-12. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa446536.aspx

[19] Msdn.microsoft.com, "Common language runtime (clr)," 2015, accessed: 2015-05-12. [Online]. Available: https://msdn.microsoft.com/en-us/library/8bs2ecf4(v=vs.110).aspx

[20] E. Bruno, "A long look at jvm languages," 2015, accessed: 2015-05-12. [Online]. Available: http://www.drdobbs.com/jvm/a-long-look-at-jvm-languages/240007765

[21] Docs.oracle.com, "What is a socket? (the java tutorials > custom networking > all about sockets)," 2015, accessed: 2015-05-12. [Online]. Available: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

[22] Msdn.microsoft.com, "Serialization (c# and visual basic)," 2015, accessed: 2015-05-12. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms233843.aspx

[23] Msdn.microsoft.com, "Windows presentation foundation in visual studio," 2015, accessed: 2015-05-12. [Online]. Available: https://msdn.microsoft.com/en-us/library/vstudio/system.runtime.interopservices.marshal(v=vs.100).aspx

[24] Oracle, "Package java.nio (java platform se 7)," 2014, accessed: 2015-05-12. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/nio/package-summary.html

[25] C2.com, "Foreign function interface," 2015, accessed: 2015-05-12. [Online]. Available: http://www.c2.com/cgi/wiki?ForeignFunctionInterface

[26] Msdn.microsoft.com, "Interop marshaling," 2015, accessed: 2015-05-12. [Online]. Available: https://msdn.microsoft.com/en-us/library/eaw10et3(v=vs.110).aspx

[27] Msdn.microsoft.com, "Calling native functions from managed code," 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms235282.aspx

[28] Msdn.microsoft.com, "An introduction to p/invoke and marshaling on the microsoft .net compact framework," 2015, accessed: 2015-05-12. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa446536.aspx

[29] Ibm.com, "Java programming with jni," 2015, accessed: 2015-05-12. [Online]. Available: http://www.ibm.com/developerworks/java/tutorials/j-jni/j-jni.html

[30] Docs.oracle.com, "Java se 7 java native interface-related apis and developer guides," 2015, accessed: 2015-05-12. [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/jni/

[31] Msdn.microsoft.com, "Introduction to com interop," 2015, accessed: 2015-05-12. [Online]. Available: https://msdn.microsoft.com/en-us/library/kew41ycz(v=vs.71).aspx

[32] D. Orenstein, "Application programming interface," 2015, accessed: 2015-05-12. [Online]. Available: http://www.computerworld.com/article/2593623/app-development/application-programming-interface.html

[33] Free Software Foundation, Inc, "Gnu general public license," 2007, accessed: 2015-05-26. [Online]. Available: http://www.gnu.org/copyleft/gpl.html

[34] Google, "Android studio overview," accessed: 2015-05-26. [Online]. Available: http://developer.android.com/tools/studio/index.html