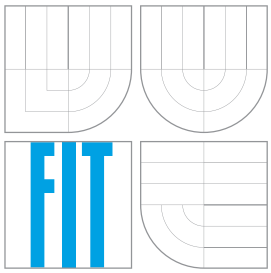
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VOIP V JABBER KLIENTU

VOIP IN JABBER CLIENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. VOJTĚCH KULIČKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. JOZEF MLÍCH
SUPERVISOR

BRNO 2011

Abstrakt

Práce se zabývá možnostmi implementace VoIP do existujícího XMPP programu se sdílenou tabulí. Analyzuje možnosti využití současných technologií pro podporu VoIP. Cílem je nahrazení stávajících komunikačních knihoven klienta za telepathy. Dále také přidání VoIP.

Abstract

This thesis tackles the issues of implementing a VoIP support into an XMPP based IM application. The state of the art is analyzed to find a suitable technology to base the VoIP on. The work's goal is to port the existing client application to network framework telepathy and implentation of VoIP.

Klíčová slova

VoIP, IM, sdílená tabule, XMPP, telepathy, RTP, farsight, Makneto, SIP

Keywords

VoIP, IM, Shared whiteboard, XMPP, telepathy, RTP, farsight, Makneto, SIP

Citace

Vojtěch Kulička: VoIP in jabber client, diplomová práce, Brno, FIT VUT v Brně, 2011

VoIP in jabber client

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana inženýra Jozefa Mlícha.

.....
Vojtěch Kulička
May 24, 2011

Poděkování

Děkuji svému vedoucímu Ing. Jozefu Mlíchovi a Ing. Jaroslavu Řezníkovi za odbornou pomoc. Dále bych chtěl poděkovat své přítelkyni a rodině za neomezenou podporu. Zvláště má přítelkyně mi byla neocenitelnou oporou. V neposlední řadě jsem vděčný receptu na pečené žampiony, s kterými jsem přečkal mnoho náročných chvil při studiu. Nejdříve žampiony rovnoměrně nakrájíme, následně naskládáme do zapékací mísy a promícháme s olivovým olejem, protlačeným česnekem, solí a oreganem a vložíme do trouby. Po upečení je nejlepší směs nechat v lednici uležet. A nakonec děkuji svým přátelům Michalu Čamborovi a Marku Černému za neutuchající morální podporu v průběhu studia. Díky výše zmíněným bude studium příjemnou vzpomínkou.

Contents

Introduction

Human is a social creature and likes to chat, share feelings and ideas. At first we managed to do so by making simple sounds. Those sounds later on developed into words. Then much later the human race started to feel the need to record what we were thinking. We made up symbols and started to write. As the society grew and spread, we wanted to communicate with people from other tribes and villages. At first we would travel and use spoken words, but as the distances grew we figured we can have our thoughts delivered in writing. Mail was born. In 1844 telegraph was invented by Samuel Morse followed by telephone in 1874 by Alexander Graham Bell. And finally in 1969 the Internet was created. All of these inventions aimed to provide means of communication to satisfy the needs of the evolving society.

In the early days of the Internet email was the main means of communication. And just like regular mail people would have their electronic mailboxes to which the emails were delivered. Email was a huge step forward for it provided a way to almost instantly deliver text from one place to another regardless of the distance for free. The main disadvantage of email is that people had to check their mailboxes read new mail and then reply. It is just neither fast nor convenient enough for team cooperation when team members are far apart. For those and other purposes like chatting Instant Messenger programs were introduced.

An IM program offers real-time communication between two people via text messages that are delivered from one user to another instantly. Instant messengers became very popular and started adding on features like multi-user chat, various games and most importantly VoIP(Voice over IP) support. VoIP capable IM like skype have become extremely popular at first for making it possible for people to call each other for free over the internet. Later video conferencing capability was added, so you could talk and see you colleague at the same time. One more thing comes in extremely handy when working in a team - a whiteboard.

At this time there is no usable IM providing VoIP and shared whiteboard for GNU/Linux. This thesis aims to add VoIP support to an existing XMPP client with shared board called Makneto. Makneto was created by Jaroslav Řezník as a master's thesis in 2008. At this point it is using iris library for XMPP communication. The shared board data is also transferred over XMPP. One of the goals of this thesis is to port Makneto to telepathy, which is now a very reliable and robust library for communication for numerous protocols.

This work is done for Red Hat Czech. I have never worked for a software company and moreover on a project that has such a great potential. I have a chance to improve my programming skills under the leadership of experienced programmers.

The following chapter 1 gives a detailed description of the current version of program Makneto. We will find out about it's architecture, strengths and weaknesses.

Chapter 2 focuses on XMPP/Jabber communication protocol. It talks about it's features and limitations. The chapter contains concrete examples that give the reader an idea

of how it works and is it so popular.

Chapter 3 is about Telepathy communication framework, how it works, what it consists of. There is also a description of a IM client application Empathy based on Telepathy.

Current audio and video streaming protocols are listed and discussed in chapter 4. Based on this chapter a suitable protocol is chosen for the implementation. This chapter also talk about VoIP and what should the implementer have in mind when writing a VoIP application.

The port to Telepathy and new Makneto's architecture are in chapter 5. It talks about design decisions and reasoning behind them.

Chapter 6 explains how was the VoIP implemented, what problems were encountered and what is the result.

Chapter 1

Makneto

The collaborative application Makneto is the main topic of this chapter. First we take a look at the architecture and what it is based on. The following section explains why port Makneto to Telepathy and add VoIP. The main sources are [48, 29, 44].

1.1 Current architecture

Makneto is an instant messenger with a shared board capability. It was written by Jaroslav Řezník as his master's thesis in 2008. Makneto is written using Qt version 4 and KDE 4 libraries. Qt is a application framework extending C++ and adding dynamic functionality like signals and slots mechanism or property system. Signals and slots are used instead of callback function, because they are as opposed to callbacks type-safe. The dynamic property system allows objects to have properties added at runtime without previous specification in their header file. In order to use the extended functionality the objects must directly or indirectly inherit QObject, which all the framework's classes are derived from. These mechanisms also require source files to be compiled by the Meta Object Compiler before any standard C++ compiler.

Qt runs on all major computer platforms like GNU Linux, Microsoft Windows and Mac OS X. More so it is supported by mobile device platforms such as Symbian and Microsoft Windows Mobile. There is also a port to Google Android called Lighthouse. It was developed by company named Trolltech and was available under two licenses. First was a commercial license allowing companies to write proprietory applications for a fee. Second was GNU General Public License, which was of course free. Thanks to the commercial license the Qt documentation is one of the best among any software available under GNU GPL. In June 2008 Trolltech was acquired by Nokia that decided to make the source code available so that anyone could contribute. In January 2009 another licensing option was added - Lesser General Public License [29, 31].

Besides Qt Makneto utilizes functionality provide by KDE4 [23] libraries, which makes Makneto unable to run on a different operating system than GNU Linux. KDE is a desktop environment created by Matthias Ettricht in 1996 for he felt the need for a good quality window manager for Linux. It is currently in version 4, which brought great features. There is a new desktop called Plasma, which allows users to display widgets called plasmoids directly on the desktop. That allows users to have a TODO list, calendar, translator, weather forecast, system monitor and much more right in front of them on their desktops without having launch any of those to get the information they need. It makes users' work

easier and more efficient. KDE and GNOME are the most common window managers in Linux. GNOME offers stable useful environment and is influenced mostly by large businesses using it. KDE is more flexible and works more with the look and feel.

Makneto communicates using XMPP/Jabber protocol implemented in Qt-based C++ library called Iris. All of the Iris is primarily used by Psi instant messenger. It's development is still quite active and it supports all of Jabber's key features as well as several extensions. Iris' downside is very poor documentation. It's wiki is very brief and all to all gives one example. The only way to find out how it works is browsing through Psi code [35].

Makneto's shared whiteboard is based an official extension of XMPP SVGWB by Joonas Govenius [12]. That is implemented in Psi and Makneto utilizes their code. SVGWB defines all the necessary actions like whiteboard session initiation including invitation and mainly method of sending and receiving information about the graphical objects using XMPP XML. The actual graphical object representation is defined by SVG[1] [43] by W3C. It describes graphics objects using vectors in XML format. Makneto uses just a subset of SVG called SVG Tiny [44] as it does not need all of the features. The subset includes two-dimensional vector graphics and raster graphics and multimedia. To sum up Makneto's whiteboard features, it allows you to draw lines, rectangles, ellipses, circles, sketch using a paintbrush and input images in jpg and png formats. Graphical objects are resizable, can be rotated and copied.

Makneto runs in one window represented by class `MaknetoMainWindow` as shown in the class diagram in figure 1.1 and displayed in figure 1.2 . User's contact list (`RoasterView`) is on a panel on the left hand side as a part of stacked widget along with MUC[2] (`MUCView`) tab and tab for controlling presence (`ConnectionView`). The whiteboard and chat session are initiated through the contact list and are located on the right side of the window. Makneto handles more sessions at once each in a separate tab. Each tab is represented by a `SessionView` object and the whiteboard within it by `WbWidget`. The subject to change is class `Connection` that is built on top of iris library. Using the application is very comfortable although it crashes from time to time.

1.2 Motivation for porting and adding VoIP support

Makneto has a potential to become a great collaboration tool. The tasks of today are more and more difficult so the need to solve the task in teams is greater and greater. Especially in computer science the teams are often from all over the world and it is important to discuss current issues. Meeting in person is not an option and that is where Makneto comes in. The shared whiteboard is very useful for sharing thinkmaps and visualizing problems and solutions, but the days when people had the time and patience to write are gone. Everybody wants to talk these days.

Makneto at this point has couple design flaws. First it uses iris library for communication in XMPP network. Iris is very poorly documented and every change in the implementation would reflect in Makneto. Second Makneto depends on KDE libraries, which at this point is not necessary. Getting rid of this dependency will help increase Makneto's portability.

The current implementation of Makneto is UI and backend in one large application. The goal is to separate the backend and UI. Backend will take care of communication.

[1] Scalable Vector Graphics
[2] Multi-User Chat

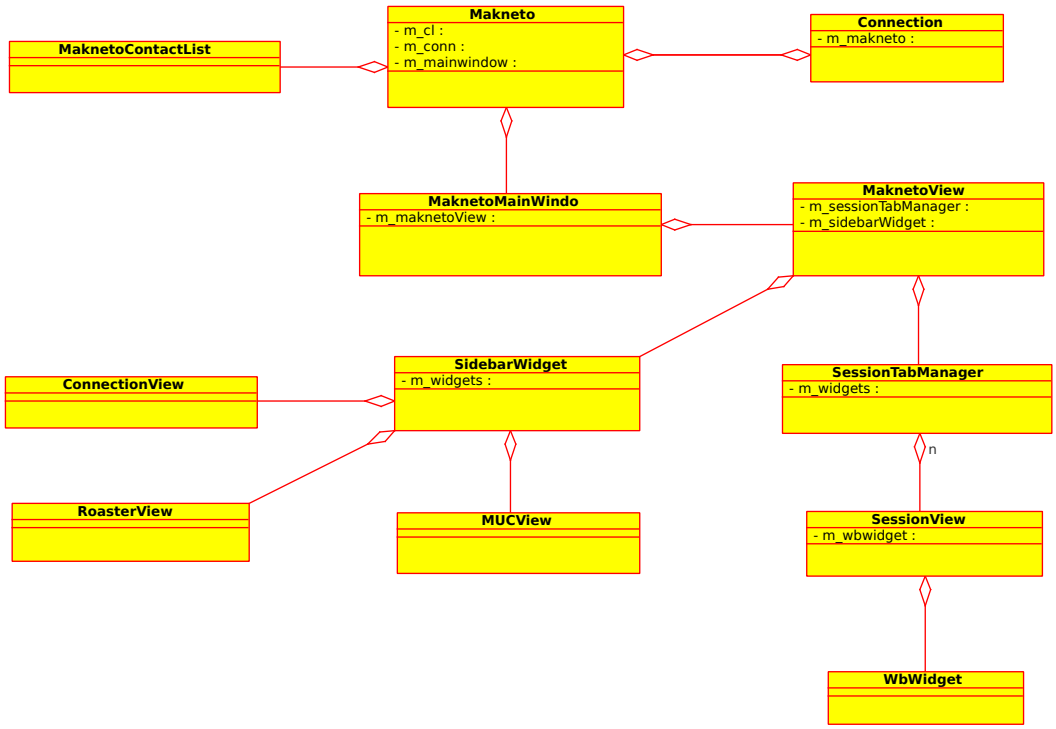


Figure 1.1: Makneto class diagram

That means text messaging, shared whiteboard and after this thesis is finished also VoIP. Backend will use communication framework telepathy described in chapter 3. Telepathy supports various protocols as well as voice and video calls. Makneto will be able to use XMPP, ICQ, IRC, MSN etc. with one implementation of a client.

Frontend shall be a subject to change based on target device. Makneto for desktop will have a Qt4 frontend and Makneto for smartphones and tablets will use the Qt Quick UI. With rapidly increasing number tablets and smartphones made and sold it would be shame not to plan to support them.

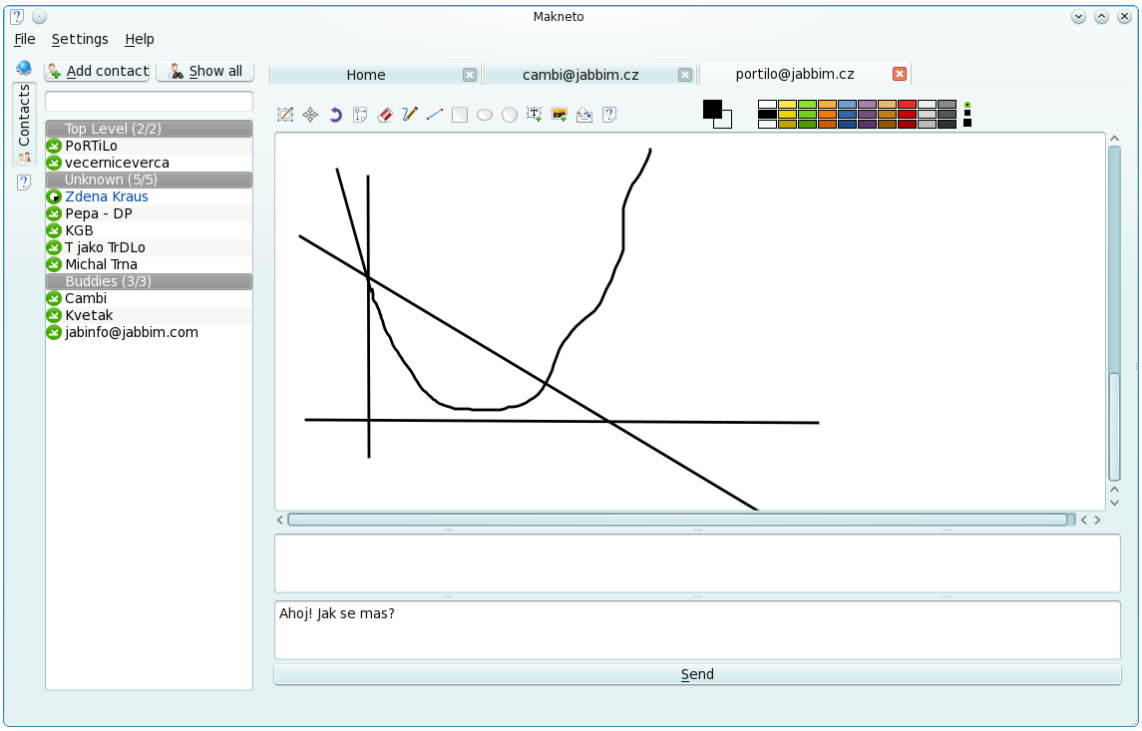


Figure 1.2: Makneto before any work was done on it.

Chapter 2

XMPP

This chapter talks about Extensible Messaging and Presence Protocol [9]. The first section lists key features and explain how the protocol works. Next section is about extensions of XMPP, mainly Jingle designed for media session signaling. The information is mainly acquired from [39, 37, 38].

In 1999 Jerremie Miller created protocol called Jabber. Jabber was an open protocol based on XML as opposed to existing protocols like ICQ [16] or AIM [32], which were proprietory and owned and controlled by private companies. The first attempt to make Jabber a standard failed. The second attempt did not use the name Jabber, but eXtensible Messaging and Presence Protocol instead. IETF[1] approved and XMPP was standardized in 2004 in RFC 3920 [37] and RFC 3921 [38] called XMPP Core and XMPP IM respectively. Development of XMPP is still active as it is based on XML it is possible to add functionality without jeopardizing the compatibility of existing implementations.

XMPP is a robust, scalable, secure, open and extensible protocol that has been well tested over the years passing all of the test without any sign of trouble. It has been very well thought out and altogether it is a great protocol. The description given bellow is in some of the parts simplified. Full description is out of the scope of this thesis.

2.1 XMPP protocol

XMPP is defined in [37] and [38], which describe all the key features of the protocol. Authors of XMPP aim for a scalable, extensible and in every way powerful protocol. Years later with XMPP still around and more and more popular we can safely say they succeeded. The key features of XMPP include:

- **Decentralized architecture** - XMPP network does not rely on one server. The network consists of servers and clients. Every client may run his or her own server. Clients register with a server of their choice. The figure 2.1 demonstrates how it works. There are several clients connected to three servers. If a client on Server 1 wants to communicate with another one on Server 2, then Server 1 sends the message to Server 2 and Server 2 forwards it to the client. It works much like email. If a client is not satisfied with the server he or she registered with he or she may simply register with a different server or run his or her own. There is also no way to take the whole

[1] Internet Engineering Task Force

network out with DoS[2] or DDoS[3] attack. There simply is not a small number of target to attack.

- **Open nature** - XMPP is an open standard that can be used by anyone without having to pay, sign any agreement or behave by any restrictive policies. Also it's fate is not in hands of one company but rather in the hands of everyone. Anyone who wishes to contribute can do so by cooperating with XMPP Standards Foundation.
- **Extensibility** - thanks to XML based nature of XMPP it is very simple to add new features without discontinuing support of the old ones. This feature makes XMPP very flexible for it can easily add new functionality and it has been already done several times. More on this is later in this chapter.
- **Security** - with built-in support for TLS and SSL there is no need to worry that the messages might be read by a third person using man in the middle attack. Companies using XMPP for communication inside their network might run their server locally with no access to the outside world.

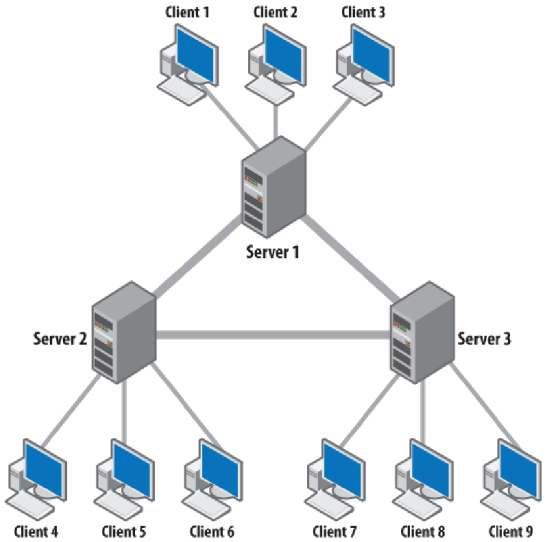


Figure 2.1: XMPP decentralized architecture [39]

Now let's look at the protocol and how to use it.

Client-server communication

XMPP is basically streaming XML documents. The stream is an unbounded XML document which contains another XML documents from both client and server. The XML documents enclosed in the `stream` tags with a depth of 1 are called stanzas. Stanza is a basic unit of communication like a packet. The following stanzas are defined:

[2]Denial of Service
[3]Distributed Denial of Service

- **message** - used for getting information from one place to another in a push manner. The message stanza is not acknowledged and no answer is expected. It is used for instant messages, alerts, notifications, group chat etc. The nature of the message is specified by `type`.
- **presence** - stanza for acquiring another user's presence. XMPP honors user's privacy and to get someone's presence status he or she needs to authorize it first by adding the querier to his or her contact list. There are several types of presence just like in any other IM protocol.
- **IQ** - is an abbreviation for Info/Query. This stanza is used for everything else. IQ works on question-answer basis. It is used for example for getting remote client's capabilities. IQ stanza must always receive a reply.

First a TCP connection is established between client and server. Once established a stream is opened by the client by sending the server `<stream>` tag. For illustration have a look at example 2.1 a stream opening. Line 2 and 3 contain used namespaces, line 6 language used for any human-readable XML character data and line 7 version for signaling support for stream-features.

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <stream:stream
3   xmlns='jabber:client'
4   xmlns:stream='http://etherx.jabber.org/streams'
5   to=\"jabbim.cz\"
6   xml:lang='en'
7   version=\"1.0\">
```

Example 2.1: Opening communication with server

Server answers with a second `stream` back to the client, which is shown in the example 2.2. The server assigns the session a unique id (line 5), that is used for preventing domain spoofing in case encryption is not set up in the following parameter negotiation. The id ought to be both unique and unpredictable.

```
1 <?xml version='1.0'?>
2 <stream:stream
3   xmlns='jabber:client'
4   xmlns:stream='http://etherx.jabber.org/streams'
5   id='856661962'
6   from='jabbim.cz'
7   version='1.0'
8   xml:lang='en'>
```

Example 2.2: Server's reply to client's opening stream

The next step is to negotiate properties of the stream. The server sends an XML enclosed in `stream:feature` tags, informing the client about features it supports. The most important property is by far encryption and authentication. Preferred encryption method

is TLS, but SSL is also an option. However some servers may require usage of TLS, which is recommended for both client-server and server-server communication. Authentication is a key responsibility of the server. It must ensure that users attempting to connect to it are who they say they are. The server acts as a gateway to the entire network and must not allow identity spoofing. Authentication is done via SASL[4] and options supported by the server are enclosed in `mechanism` tags. The example 2.3 shows the supported encryption on line 3 (TLS) and mechanisms for authentication are on lines 4-11.

```
1  <stream:features>
2    <starttls
3      xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
4    <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
5      <mechanism>
6        PLAIN
7      </mechanism>
8      <mechanism>
9        DIGEST-MD5
10     </mechanism>
11   </mechanism>
12   ...
13 </stream:features>
```

Example 2.3: Server sends supported encryption options

Once communication parameters are set, client can request presence, set his or her own presence or communicate with other clients. Sending a message to a fellow user is accomplished via `message` stanza and may look like XML in example 2.4.

```
1  <message
2    from='vtheman@jabbim.cz'
3     to='kuba86@jabber.cz'
4    xml:lang='en'>
5    <body>Hey, how are you?</body>
6  </message>
```

Example 2.4: Chat message

It must include `to`, `from` attributes and `body` opening and closing tags. The first specifies who is the message addressed to using a Jabber ID. Jabber ID consists of
`user name @ domain name [/ resource]`. Resource was added to JID to allow user to connect from different device/locations without having to log out. An example of resource is `home` or `work`. A simple Jabber ID is `vtheman@jabbim.cz`. Attribute `from` naturally contains a jabber ID as well, but this time an ID of the sender. The `body` element contains the actual message.

Next stanza is `presence` and it is needed for following presence of contacts in users' roster. To get presence of a user he or she must approve of it. Once the user has been authorized to get another user's presence they have both subscribed to get each other's pres-

[4]Simple Authentication and Security Layer

ence. After connecting to the server the user sends initial presence stanza: `<presence/>`. From that moment on the server takes care of the presence. Whenever the user's presence changes, it sends notification to all subscribers in the user's roster as shown in example 2.5. Similarly if someone else's presence changes the user gets notified by his or her server.

```
1 <presence from="vtheman@jabbim.cz" to="kuba86@jabber.cz">
2   <show>dnd</show>
3   <status>Working on my thesis!</status>
4 </presence>
```

Example 2.5: Presence publication

Presence stanza contains elements `show` and `status`. `show` can be either `chat` meaning available and ready for chat, `away` meaning the user is not at the PC at the moment, `xa` indicates the user will be gone for a longer period of time and finally `dnd`, which stands for do not disturb.

The last of stanzas is Info/Query shortly IQ. IQ is very similar to HTTP in methods and in the query-answer nature. IQ queries use method `get` for requesting information and method `set` for making requests based on provided information. Answer to a query is IQ stanza of type `result` and contains information requested by `get` method or acknowledge in case of `set` method. The last type of IQ is `error` and is used to indicate that something went wrong. A good example of an IQ stanza is acquiring a roster shown in figure 2.6. The type of stanza (IQ) and method (`get`) are specified on line 1 and 2 says what exactly the client ask for. The server's answer is in figure 2.7. We can see it is of type `result` and lines 3-5 contain the requested data - JIDs of people in the roster.

```
1 <iq type="get">
2   <query xmlns="jabber:iq:roster"/>
3 </iq>
```

Example 2.6: Requesting roster from server

```
1 <iq type="result">
2   <query xmlns="jabber:iq:roster">
3     <item jid="kuba86@jabber.cz"/>
4     <item jid="rezza@jabber.cz"/>
5     <item jid="imlich@jabber.fit.vutbr.cz"/>
6   </query>
7 </iq>
```

Example 2.7: Roster sent by the server

The stream ends with `</stream>` tag and that means end of the communication.

XMPP Extensions

As mentioned earlier XMPP is easily extensible thanks to it's XML based architecture. XMPP extensions are published by XSF[5] as XEP[6]. Basic XMPP functionality defined in the RFC 3920 [37] and RFC 3921 [38] ought be supported by every XMPP server and client. Functionality described in XEP is optional. These are the most popular extensions:

- **Multi-User Chat** defined in XEP-0045 [36] allows users to create virtual rooms just like in IRC, invite their contacts to join the room and thus communicate with multiple people at once.
- **Service Discovery** registered as XEP-0030 [20] defines a way of finding out what capabilities have one's contacts.
- **Entity Capabilities** defined in XEP-0115 [21] adds client's capabilities to presence information, so that if a XEP-0115 capable client requests presence it receives a list of supported features as well.

There are tens of extensions either standardized of waiting for becoming a standard defining various handy features.

2.2 Jingle

XMPP Extension defined in XEP-0166 [41] known as Jingle is a signaling protocol that initiates, manages and terminates media sessions via XMPP. Jingle was first used in Google Talk [11] for Voice call signaling in 2005. The idea was to use an existing XMPP communication channel to setup a peer-to-peer media session that uses a different means of transporting data, e.g. RTP[7] for voice or video and TCP for file transfer.

Figure 2.2 shows how the protocol works. Session initiation starts when the initiator sends `session-initiate` with Application type, e.g voice call, and Transport method, e.g UDP are described. As you can see in the figure Jingle uses an IQ stanza so the initiator immediately receives a IQ result acknowledging the invitation reception from the responder. Next all the necessary application type and transport type parameters of the session are negotiated. In case of voice call the application type parameters might be audio codec and sampling frequency. Transport type parameters include the peers' IP addresses and ports and transport method. When all the parameters have been set up the responder either accepts or declines the invitation. If accepted the data start flowing between the two peers just as it was agreed during the initiation phase. Jingle can also be used to adjust parameters of an existing session if necessary. And final one of the peers sends the other `session-terminate` to end the session.

[5] XMPP Standards Foundation
[6] XMPP Extension Protocol
[7] Real-time Transport Protocol

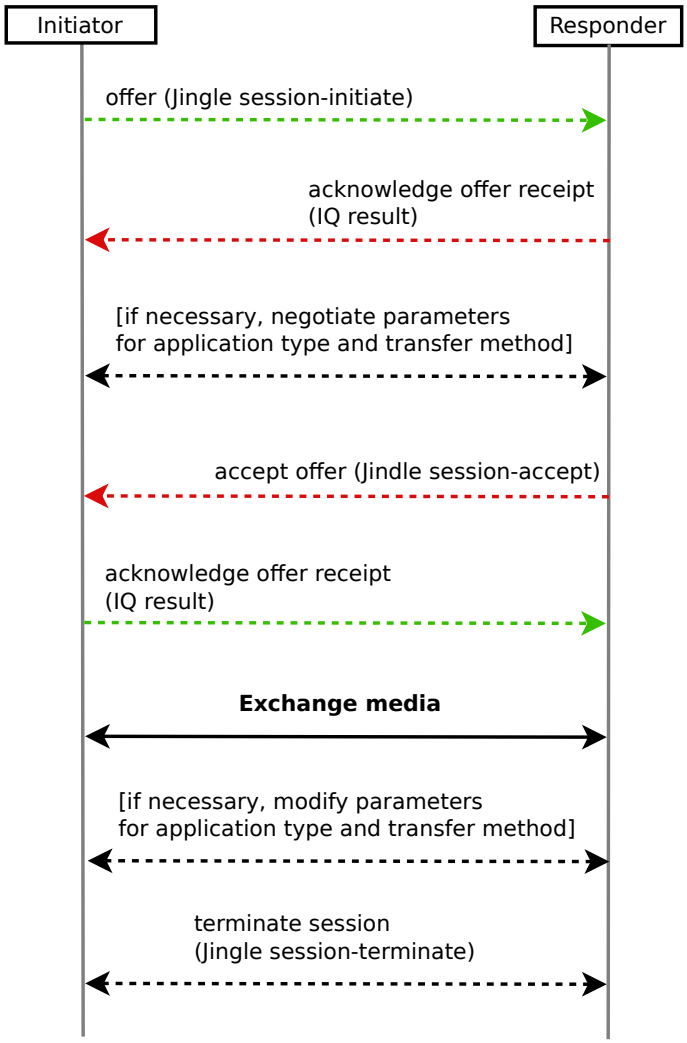


Figure 2.2: Data flow in media session initiation, management and termination using XMPP Jingle [39]

Chapter 3

Telepathy

In this chapter the framework to which Makneto shall be ported is looked at. First basic terms and basic idea of how is the framework designed what can it do. Then we describe D-Bus and the relationship with Telepathy and have a detailed look at components of Telepathy. Finally we talk about an existing client built on Telepathy. The chapter takes information mainly from [7, 3].

Telepathy [5] is a modular communications framework for building real-time communication applications. It supports numerous communication protocols as plugable backends e.g XMPP/Jabber(telepathy-gabble), SIP(telepathy-sofiasip), MSN(telepathy-butterfly), etc. Each of telepathy's components runs in a separate process as desktop service and communicates via D-Bus. The components are shared by telepathy clients. To get a better idea of how this concept works take a look at figure 3.1 [7]. XMPP represents Connectin Manager Gabble that creates a connection for an account. Chat client can use that connection for text messaging and Voice client can use the same connection for establishing a call using Jingle. Or one client client can use more Connection Managers simultaneously. The VoIP client can without any adjustments support calls to XMPP, MSN and SIP networks thanks to the high level of abstraction.

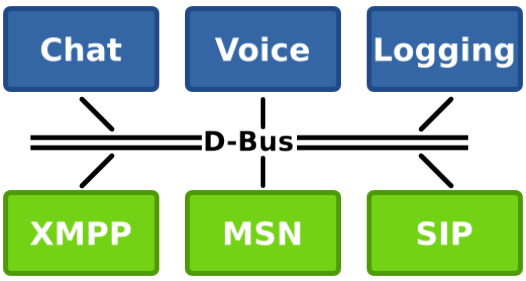


Figure 3.1: Telepathy architecture [7]

There are several features making telepathy very useful as a communications framework.

- **Robustness** - all the components are independent. If one crashes, others will not be affected
- **Ease of development** - the components can be replaced without having to stop the service

- **Language independence** - since telepathy components use D-Bus for communication among themselves, any language that has D-Bus binding might be used to write them
- **Desktop independence** - D-Bus is present in both main Linux window managers GNOME and KDE, so the same telepathy components could be backend for appropriate frontends.
- **Code reuse** - the client applications do not have to worry about protocol specifics, which are handled by Telepathy. The client can use more protocols by making no or small alterations to the code.
- **Connection reuse** - more than one Telepathy client can use the same connection simultaneously:

3.1 Telepathy architecture

Telepathy is a very powerful framework and as such it is also complicated. To successfully write programs using Telepathy we need to know what telepathy consists of and what it is based on. This section tries to put all the terms into context of this thesis.

D-Bus

D-Bus is a kind of inter-process communication. It allows two applications running in different processes, written in different programming language communicate. More so these applications may communicate directly, without having to go through message bus daemon. There two types of D-Bus. First is a system bus used for events such as "USB device disconnected" or "printer out of paper." Second type is per-user-login-session bus, which is used by user applications. D-Bus low level API is represented by libdbus and it requires XML parser (libxml or expat) to work. Higher level language bindings such as Qt, GLib, Java etc. are built on top of libdbus and offer more convenient way of using D-Bus, although they add more dependencies [3, 7].

Each process that wants to communicate over D-Bus will need to use most the following depending on it's nature:

- **Unique name** - is an unique id (e.g. `:2.1`) assigned by D-Bus daemon to the client application. Unique name is similar to a public IP address.
- **Well-known name** - is similar to a DNS name. If a process wants to make a service available to other processes it requests a well-known name. If another process wants to access the service it uses the well-known name to do so. Well-known name might look like this: `org.freedesktop.Telepathy.ChannelDispatcher`.
- **Object path** - is a path to an object that is exported by process running a service.
- **Interface** - is a way of requesting a service using signals or methods. Each D-Bus client must register at least one interface and each interface provides at least one method or signal. Every interface needs to have to name like a well-known name.
- **Method** - is implemented in the object specified by object path and exposed in the interface for that object for other processes to use.

- **Signal** - is a D-Bus signal client process can connect to it's callback function. If a signal is invoked the callback function is called.
- **Property** - is used for exposing D-Bus object's properties. To do so the objet must implement org.freedesktop.DBus.Properties interface.

The figure 3.2 shows an example of two programs connected to D-Bus to be able to communicate with each other. Program B provides a service called well-known name (`org.freedesktop.foo.Bar`) and it's id is `:1.3`. Program A does not provide any service and thus does not need any well-known name. It just needs an id 1.2 to use other programs' services [7].

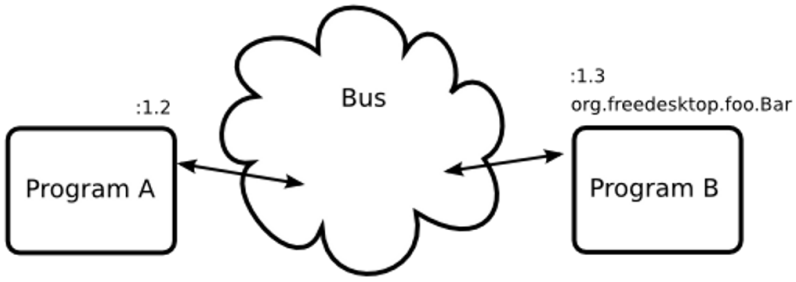


Figure 3.2: D-Bus id and well-known name example [7]

The figure 3.3 shows an overview of all of the terms described above in a simple diagram.

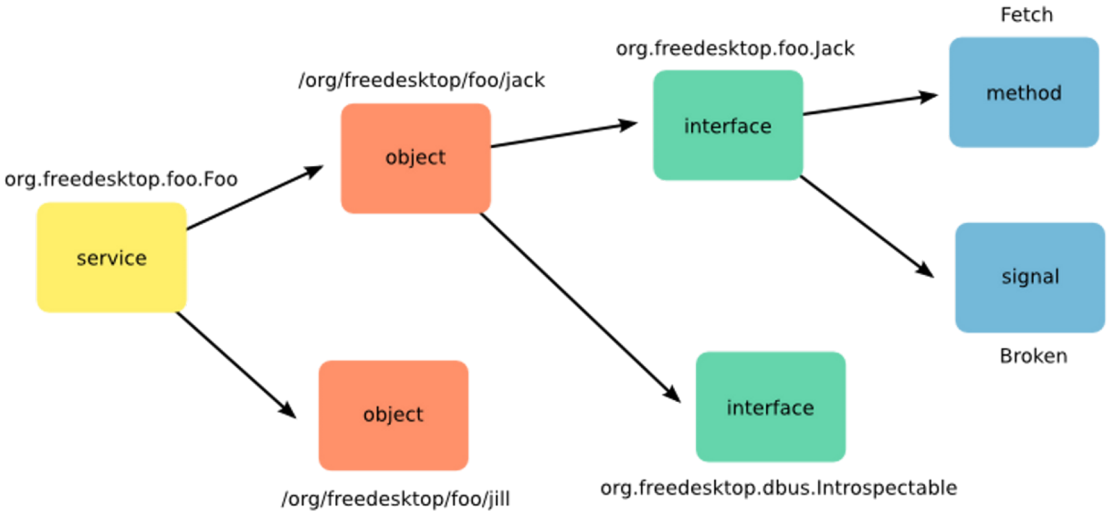


Figure 3.3: D-Bus architecture [7]

D-Bus is a key component of Telepathy framework. Telepathy supports many protocols all of which might provide different capabilities. For example IRC does not support avatars while XMPP does. Even though avatar feature is supported by XMPP protocol it might not be supported by the server we are connected to or by the opposite client in case of

peer-to-peer connection. The available features are exposed by D-Bus Properties Interface. That is an easy way of determining the protocol, server or client capabilities [3].

Mission Control

Mission Control is a Telepathy component that implements Account Manager and Channel Dispatcher and it's primary purpose is to encapsulate those two. There always only one running on the system. It is very important to check version running for they are backwards incompatible [7]. For better understanding of the role of this and the following components have a look at figure 3.4. It illustrates the relationship among the the key components.

Account Manager

Account Manager is responsible for handling accounts (e.g. XMPP, ICQ, MSN etc.). It is accessible by well-known name on D-Bus - `org.freedesktop.Telepathy.AccountManager`. A client application first creates an account using the provided methods, supplying it with Connection Manager, protocol and display name. Account Manager creates and then as long as the Account is active maintains Connection to that Account. The Accounts are persistent and the next time the application runs it does not need to create them again.

To create a Connection a Connection Manager is called by Telepathy. An Account may be valid or invalid and also enabled or disabled. Valid ones may establish a Connection, whereas invalid can't. Enabled and disabled property are user's way of telling the application which one should be ignored and which not. The lists of valid and invalid accounts are accessible via appropriately named methods [7].

Account

Accounts are created via Account Manager. They are only created once and then stay on the system and can be accessed by any Telepathy clients. An active Account object registers with D-Bus and has an object path `/org/freedesktop/Telepathy/Account/CM/PROTOCOL/ACCDN`. `CM` stands for Connection Manager (e.g. gabble, salut, buttefly, etc.), `PROTOCOL` is substitution for a protocol name and `ACCDN` is Account's Display Name. The Account object implements `org.freedesktop.Telepathy.Acc` interface. Features supported by this interface depend on the protocol used and the server-side software.

The Account settings are done via `org.freedesktop.Telepathy.Properties` interface. All available attributes can be obtained by calling a single method provided by the properties Interface. The method is very convenient for it returns all the properties at once in single D-Bus call. Setting all properties at once works exactly the same. Some features are available via specified interface, e.g. avatar. Avatar used to be a property, but now it has it's own interface. The Interfaces property of the account lists all interfaces of additional features [7].

Connection Manager

Connection Manager supplies Account Manager with Connections. It is not directly used by the client program. Account Manager requests connection for active accounts. There is always only one Connection Manager for each protocol running at any time.

Connection Manager is a protocol-dependent Telepathy component. Different protocols need different Connection Managers. For example if the client application wants to communicate using XMPP/Jabber it has to use Telepathy-gabble and for MSN Telepathy-butterfly is required. Some Connection Managers can communicate via more than one protocol, for example Telepathy-haze. To see what protocols are supported there is an appropriate method implemented by the Connection Manager [7].

Connection

Connection represents an active protocol session. It is associated with an Account and is created by Connection Manager based on a request of the Account Manager. Connection implements org.freedesktop.Telepathy.Connection interface and additional interfaces depending on the protocol. List additional interfaces available can be retrieved by checking the Interfaces property. The most common interfaces are listed below [7]:

- **Contacts** - used to get as much information about a contact as asked in one D-Bus call.
- **Aliasing** - serves for setting aliases for contacts and checking if the contacts have changed their alias themselves.
- **Avatars** - interface to one of the most popular protocol features. Allows users to set their avatars and retrieve other users' avatars.
- **ContactCapabilities** - retrieves capabilities of contacts' Clients to see what features they support. Checks for example for VoIP or file transfer support.
- **Location** - lets user publish his or her current location as well as find out his or her contacts' whereabouts.

Channel Dispatcher

This component handles Channels incoming from active Connections of valid Accounts. Channel Dispatcher monitors available or activatable Telepathy Clients through D-Bus. Clients register with user's session D-Bus and provide a `CLIENT\_NAME.client` file. Both of those serve as a way to publish Client's properties including a channel filter. The Channel Dispatcher based on these properties knows what kind of a client it is and what type of channels it is interested in (channel filter). If the Client is running then the properties are acquired via the Client interface. If the client is not running and is activatable then the .client file is used by Channel Dispatcher to pre-look up the properties and if they match the incoming Channel, the Client is activated. So providing the .client file only makes sense for activatable Clients.

When a Channel comes in from one of the Connections Channel Dispatcher notifies appropriate Clients. There are three kinds of clients - Observer, Approver and Handler detailed description of which is given later in this chapter. The Channel is dispatched to all Observers and all Approvers with a matching channel filter. The Approvers choose Handler to handle the Channel. Should the Client fail, Channel Dispatcher may recover from such error and look for another Handler [7].

Channel

Channel allows the local client to exchange various kind of data with a remote server. It is associated with a Connection and always implements at least two interfaces. The first is `org.freedesktop.Telepathy.Channel` and the second depends on the Channel type. Channels for text messaging will be of type Text and will implement `org.freedesktop.Telepathy.ChannelType.Text` interface. The following list shows most common types of Channels [7]:

- **ContactList** - used to get information of contacts in user's contact list.
- **Text** - designed for exchanging text messages.
- **Call** - used for VoIP and video calls.
- **FileTransfer** - Channel for sending and receiving files.
- **ContactSearch** - is used when a user wants to find a contact on a server.

Channels are created using two methods - `CreateChannel()` and `EnsureChannel()`. These methods are implemented by both Channel Dispatcher and Connection. When calling either of those methods on Channel Dispatcher the resulting Channel will go through the procedure of looking for handler as described above. When using directly the connection the calling application must handle the Channel itself as the Channel Dispatcher will not interfere. It is also possible to supply the Channel Dispatcher with a preferred handler and thus achieve the same effect. It is better to use the Channel Dispatcher for if the client should fail it may dispatch the Channel to another handler [7].

Both of the methods provide a Channel. The difference between the two is that `CreateChannel()` creates actual new Channel whereas `EnsureChannel()` will attempt to reuse an existing Channel with the same properties. The first is typically used for file transfer and contact search and the latter for text, streamed media and contact list Channels [7].

Client

Client is an application that wants to use Telepathy. It needs to register a well-known name in org.freedesktop.Telepathy.Client namespace, e.g. Empathy registers org.freedesktop.Telepathy.Client.Empathy. Then it provides a .client file where purpose of which is described above. Telepathy defines three types of clients - Observer, Approver and Handler. All of these need to provide appropriate channel filter, e.g. Observer provides ObserverChannelFilter. Based on the published filter the Channel Dispatcher dispatches an incoming Channel to the Client or not [7].

Observers are called upon a creation of a new Channel. They monitor Channels and provide the acquired information to user. The observers have different functions based on the type of observed Channel, e.g. Text Channel observer might serve as a logger and FileTransfer observer as a file transfer progress monitor. Observer is must not interfere except for when the user interaction like hitting the cancel button in a file transfer progress window [7].

Approver is a Telepathy Client that is supposed to accept the incoming Channel and decide, which Handler it is dispatched to. The Channel Dispatcher provides Approvers with a list of possible Handlers. Approver notifies the user of a new Channel and lets him or her decide whether to accept or reject it. Similarly the user is allowed to choose which Handler

will handle the Channel. Handler might also be chosen by the Approver itself. Approver does not call methods just like the Observer. Calling methods is up to the Handlers. For example if there is an incoming file transfer the Approves lets user decide whether to accept it or not, but the `AcceptFile` method will be called by the chosen Handler [7].

The last client is Handler. Handler does all the interaction with the Channel. A typical example of a Handler is chat-window. It displays messages and allows the user to send text messages back [7].

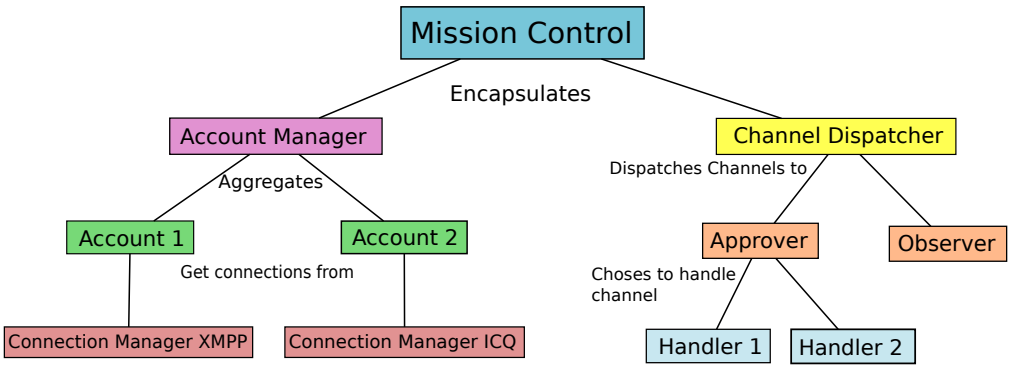


Figure 3.4: Simplified Telepathy architecture

.

The figure 3.5 represents typical setting using Telepathy. There is one Account Manager and one Channel Dispatcher for those are unique in the system. Next we can see Connection Managers, which allow the clients use numerous protocols. In general the number of supported protocols equals the number of present ConnectionManagers. Finally there are clients, which may an Observer for logging, Approver and a Handler. But it as well may be three handlers. Also one client may represent Handler, Approver and Observer at the same time. All of these entities communicate via D-Bus.

3.2 Empathy

Empathy is a multiprotocol instant messaging application based on Telepathy. In the terms described above Empathy is a Telepathy client. It is written in python using Telepathy-python bindings. Empathy registers a well-known name with D-Bus and communicates with Telepathy components to provide the communication services. Empathy supports text messaging, file transfer, voice and video calls over various protocols. Supported protocol include Google Talk, XMPP/Jabber, MSN, IRC, AIM Facebook, Yahoo!, Gadu Gadu, and ICQ. For some of those protocols like Google Talk and XMPP voice and video calls implemented. The support of the protocols depend on Telepathy Connection managers installed. Additional functionality includes sharing users' whereabouts among themselves, automatic reconnection when internet connection is reestablished and automatic changes of presence to away and extended away [34].

The current stable version of Empathy is 2.32.2 and it is a default communication application in GNOME releases since version 2.24 instead of Pidgin. Empathy also replaced

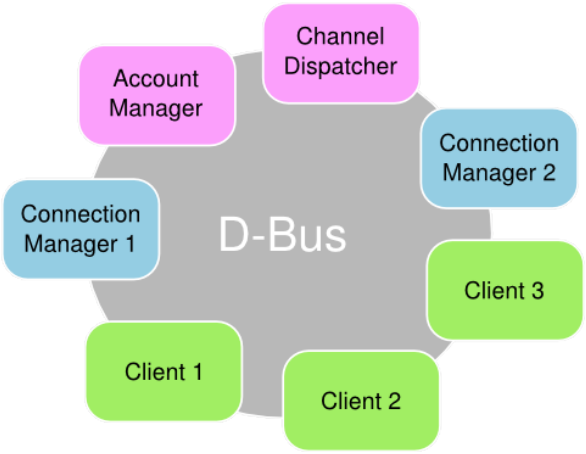


Figure 3.5: Telepathy components registered with user session D-Bus [7]

Ekiga - program for voice calls and video-call. It became an ultimate free communication tool. Empathy's GUI is takes after Gossip, which is an older IM application for GNOME.

Chapter 4

Existing solutions

The first section of this chapter talks about existing applications supporting both VoIP and whiteboarding. The listed solutions were chosen based on relevance in the context of this work. The level of technical details of each of the applications depends on whether it is an open source or proprietory. The next section explains how to transfer multimedia over the internet. Finally the last section aims at VoIP and what must be taken into account when implementing it. The chapter is based on information from [40, 10, 25].

There is a number of existing communication applications offering voice calls and shared whiteboard. Some more popular, more advanced, more user-friendly, offering more features or better support than others. Some of those programs support video call and some even conference calls. We shall go over the existing solutions that implement both shared whiteboard and voice calls. Among the described are Skype [42], Windows Live Messenger [28], Brosix [2], Yahoo! Messenger [47] and AIM [32].

4.1 Applications with whiteboard and VoIP support

Skype

Skype is the most popular and most used common VoIP application of all. Skype was released in 2003 and was developed by KaZaa [22]. It is available for all major computer platforms (Microsoft Windows, Mac OS X and GNU Linux) as well as mobile platforms like Android and even Apple's iOS. Although skype is available for Linux it is not well supported. The latest version of skype for Linux is 2.1.0.81 Beta while skype for Windows is of version 5.1. The polarity of skype demonstrates the fact it is built-in on more and more TVs.

Skype communicates using Skype protocol, which is proprietory. Recently much effort is being put in reverse engineering the skype protocol although the first attempt dates back to 2004 and was done in [40]. Skype encrypts the communication end-to-end with 256 bit AES so the amount of information acquired by packet sniffers is very limited. The motivation for recent efforts are simple. Skype is used by tens of millions of users every day, but the support for Linux is at this point almost non existing. Linux users have had enough and plan to create an open source client capable of communicating with skype. The Wikipedia skype protocol page [45] is filling up with details.

While most of IM programs utilize client-server communication scheme, skype uses peer-to-peer model. The skype network consists of nodes, supernodes and login servers. See figure 4.1, which was taken from [40]. Nodes are clients. Each client keeps addresses of

a number of supernodes. Supernodes are clients with good-enough bandwidth, public IP address and enough memory and CPU power. Supernodes forward traffic to clients behind device with network address translation or restrictive filters. It is believed that skype uses something like STUN, which helps overcome NAT and was first defined in [17] and then superseded by [18]. It seems that nodes themselves determine whether they are behind address translating device or firewall. If two nodes want to communicate and either of them or both is behind NAT then a supernode is used to forward traffic between them.

Skype call signaling is done via TCP and UDP is primarily used for the voice transfer. If a skype client finds out it is behind a firewall that forbids UDP, the speech is transferred using TCP. The codec used is unknown although there are couple candidates, but it is almost certainly of wideband type.

The decentralized architecture seems to be working well although there was an outage on December 22nd 2010. Skype officials claim it was due to lack of supernodes [1].

Features of skype include calling landlines and cell phones and sending text messages and vice versa - SkypeOut and SkypeIn. The feature list continues with voice and video calls, multi-user chat, conference calls, voice mail and screen sharing. The newest and long expected feature of video conference was introduced in may of 2010 in skype 5.0.

Though closed program, skype provides an API for developers who want to create extensions called skype extras. Skype extras include a wide range of utilities that might be plugged into skype. The extras uSeeToo, TalkAndWrite, WhiteBoardMeeting and Sketch Pad all provide shared whiteboard each in their own way.

Windows Live Messenger

Windows Live Messenger was first released in July 1999 as MSN Messenger and offered just text messaging with users of AOL Instant Messenger [32]. Due to AOL's constant effort to block Microsoft from it's network Microsoft gave in and removed the feature. Since then MSN Messenger could only connect to MSN Messenger Service. In 2001 with the release of Windows XP, MSN Messenger 4.6 came out with voice call support. The last version of MSN Messenger, version 7.5, featured video calls. Windows Live Messenger 8.0 was released in June 2006 and that was the end of the name MSN Messenger.

Windows Live Messenger utilizes client-server model and communicates using Microsoft Notification protocol over TCP. The first 7 versions of MSNP were disclosed to public, but since version 8 the details have been kept a secret. MSNP does not use encryption so even though MSNP's description was not published it was not hard to put it together using packet sniffers.

The Live Messenger is available for Windows, Mac OS X and recently was integrated into Microsoft's game console Xbox 360. It features social network integration, offline messaging, games and applications, voice and video calls and standard IM features. An interesting feature is Multiple points of presence allowing user to be connected on two devices. Shared whiteboard is available as an extra application and is not capable of multi-user session.

Brosix

An award winning application first released in 2006. Brosix features voice and video chat and multi-user chat, basic IM functions and couple advanced functions. Brosix's whiteboard is an Microsoft Paint like window shared among the participants and won Best IM Feature 2009 award from about.com. Next great feature allows users share screen, including mouse

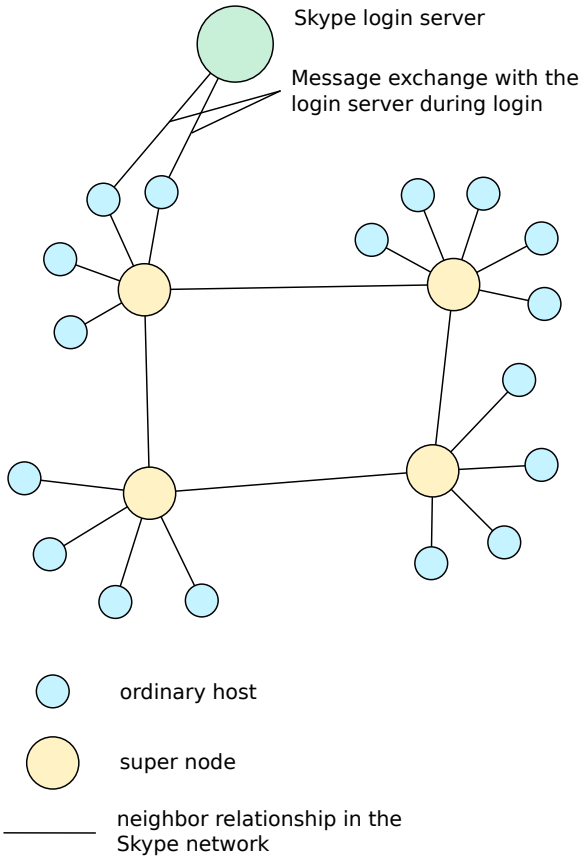


Figure 4.1: Skype architecture [40].

and keyboard much like VNC. Finally Brosix implements co-browsing where users share a browser window.

Brosix is available for Windows, Max OS X and GNU Linux in commercial and personal (free) version. There is little known about used technologies and it is almost impossible to reverse-engineer using packet sniffers as Brosix uses 256 bit AES encryption.

Yahoo! Messenger

Yahoo! Messenger is just like all of the above a closed program though some information has leaked [46]. Yahoo! Messenger protocol(YMSG) uses TCP on port 5050 or a different one if default is not available. To get to clients behind firewall HTTP is utilized. Video and voice supposedly use SIP and H.323.

Besides the standard set of IM functions Yahoo! Messenger can call PSTN, send SMS and handle voice conference. Whiteboard feature is called Scribbler and is a plugin. Linux is missing in the list of supported platforms while Windows and Mac OS X are not.

AOL Instant Messenger

AIM is yet another IM with proprietory communication protocol. Though AIM is a bit of an exception for it supports two protocols. First is just for simple text messaging called TOC and has been disclosed to public. Second protocol that supports all of the advanced features is being kept a secret.

AOL's messenger is available for Windows and Mac OS X and features voice and video calls as well as whiteboarding. Whiteboard is available as a plugin for AIM called IM Whiteboard.

4.2 Multimedia streaming protocols

The following section lists the most popular protocols for VoIP and explains how they work. The first part is about signaling protocol SIP and the second about streaming protocol RTP.

SIP

Session Initiation Protocol is standardized by IETF and was first defined in RFC 2543 [27]. The latest definition is in RFC 3261 [19]. SIP is used in VoIP for negotiating the details of the call. The parameters of the call are described using Session Description protocol described in RFC 4566 [26]. It can be used for establishing, negotiation and terminating any type of session whether it is between two users or it is a multi-user session. Among the features of SIP is also instant messaging, presence or any kind of event notification. SIP uses primarily UDP on port 5060, but can also use TCP on the same port and 5061 for TLS secured SIP. The syntax is similar to HTTP.

The clients are identified by URI which looks like this: `sip:username@hostname`, to be concrete `sip:bob@biloxi.com`. First the client must register with a proxy, which in Bob's case is `biloxi.com`. Now let's say Alice wants to call Bob. To do that she needs to know his URI. She sends an `INVITE` to her proxy, by which it is forwarded to Bob's proxy and finally delivered to Bob and his phone or computer start ringing. If he picks up an appropriate numeric code is sent to Alice. The example 4.1 shows the scenario presented.

```
1  INVITE sip:bob@biloxi.com SIP/2.0
2  Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
3  Max-Forwards: 70
4  To: Bob <sip:bob@biloxi.com>
5  From: Alice <sip:alice@atlanta.com>;tag=1928301774
6  Call-ID: a84b4c76e66710@pc33.atlanta.com
7  CSeq: 314159 INVITE
8  Contact: <sip:alice@pc33.atlanta.com>
9  Content-Type: application/sdp
10 Content-Length: 142
```

Example 4.1: SIP invite from Alice to Bob [27]

Since SIP serves only to initiate and control the session it needs to cooperate with a protocol that does perform the actual data. That protocol is Real-time Transfer Protocol.

RTP

Real-time Transport Protocol is an IETF standard for transporting data that need to be delivered in real-time rather than reliably. Therefore UDP is used on the transport layer. RTP was first defined in RFC 1889 [13] in 1996 and then later updated in RFC 3550 [14] in 2003. It is primary protocol for streaming audio and video over the internet. It is used for VoIP for transporting the voice while SIP or another signaling protocol negotiates parameters of the transport. More and more TV stations have been converting to the internet and they use RTP as means of distribution. RTP uses unicast as well as multicast when streaming to multiple subscribers[1].

RTP is used in conjunction with Real-time Transport Control Protocol. RTCP monitors QoS, statistics of the transfer and helps with synchronization when streaming to multiple destinations. The volume of RTCP traffic should be around 5% of the volume of the stream [33].

Unlike the circuit switched network, where the QoS is ensured by it's nature, the packet switched network does not have a way of ensuring short or not even constant delay or sufficient bandwidth (it is possible, but very rarely and in private network with proper SLA[2]). RTP defines mechanisms for making the most of the packet switched network. It is important to note that in the internet packets of the same stream might take different path to their destination. That and network congestion are the reasons for varying delay commonly referred to as jitter. RTP labels all the packets with sequence numbers. The implementation of RTP at the destination has a buffer to compensate for jitter and out-of-sequence delivery. If the packet arrives too late it is dropped. Dropping packets to some point might not even be noticeable by the user.

RTP sends and receives data on even port numbers and the associated RTCP uses the next higher odd port number. An example on an RTP packet follows.

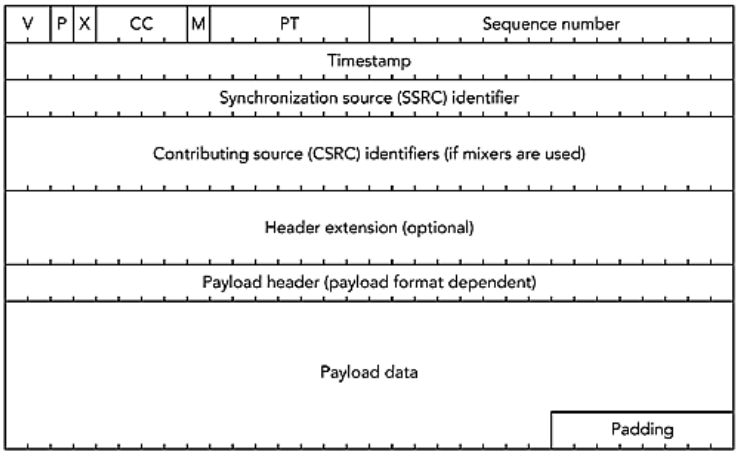


Figure 4.2: RTP packet [33]

[1] works only in local networks
[2] Service Level Agreement

Voice codecs		
Codec name	**Bitrate (kb/s)**	**MOS**
G.711	64	4.1
G.723.1	6.3	3.9
G.726	32	3.85
G.729	8	3.92
Speex	8/16/32	unknown

Table 4.1: Voice codecs [10]

4.3 VoIP

Realization of Voice over IP deals with many problems. Factors effecting voice quality are choice of voice codec, echo cancellation, packetization, packet loss, delay, jitter (delay variation). First the voice needs to be digitized. That is done by a microphone, which consists of analog-to-digital converter and a mechanism for generating current based on sound in it's proximity. It is important to note that human ear can hear sounds of frequency between 16 to 16000Hz, but only fraction of this spectrum is used when speaking. It is 400 - 3500Hz to be concrete. By recording only a part of the spectrum lower sampling frequency might be used resulting in lower bitrate. The sampling frequency is determined based on Nyquist–Shannon sampling theorem. The AD conversion in digital telephone networks is for example in digital telephone networks done by PCM[3], which takes 8000 samples - discrete values on the AD converter. Each sample has 8 bits so the resulting bitrate is 64kbit/s. PCM is for it's high bandwidth consumption and low level of captured information not suitable for VoIP [25]. There are alternatives specialized for representing speech, which has certain characteristics whereas PCM is used for any type of sound.

The next step is compressing the recorded voice. There are several compression algorithms. First difference is in length of voice chunks they compress. G.729 takes 10ms as opposed to G.723.1 that takes 30ms. The longer the chunks of voice are the less overhead (IP and Ethernet headers) is sent over the transport media. On the other hand the shorter the length the smaller impact on the conversation should the packet get lost. The compression algorithms used by the voice codecs should ideally lower the data size significantly, while causing short algorithmic delay and taking up insignificant amount of CPU time. Unfortunately there is no codec that would excel in all of the above. Good choice is thus a suitable compromise of the qualities. Since codecs are very hard to compare based on those qualities MOS[4] as means of comparison attribute. The MOS is a subjective evaluation of quality of voice encoded/decoded by the particular codec on a scale 1-5. The table 4.3 shows the most popular voice codecs with bitrates and MOS.

Once the speech is digitized and encoded it needs to be sent over the network. Since VoIP is an interactive service the latency should be less than 150ms. If the delay is too long the users experience the two conversations effect. The types of delay that need to be taken into account at all times are processing delay, packetization delay serialization delay. The processing delay represents time needed for voice digitization and compression and is codec specific. Packetization delay occurres when the encoded voice is being loaded into packets and depends on how long chunks of speech the particular codec uses. Serialization means

[3]Pulse-Code Modulation
[4]Mean Opinion Score

sending off the data through the network link and the delay is dependent on the available bandwidth on that link. For links with low bandwidth it is recommended to use codecs with low bitrates. G.729 for example uses voice activity detection and silence suppression. The caller on the other side then is sent carefully generated comfort noise so that he does not think the call went down. The comfort noise however has much lower bitrate requiring lower bandwidth.

Last and largest portion of the total delay is time of network delivery. We need to account for switching and routing delays, link transmission delays and also for jitter buffer delays. The routing protocols work based on destination address which may, and often does, result in packets of one conversation taking different routes to the caller. Different routes mean different delays - jitter. Jitter buffer is used to accumulate arriving packets and presenting them to the user in the correct order. Since RTP uses sequence numbers it is an easy task to reorder packets. The problem is when packets are delivered too long apart. In that case the missing packet is assumed to be lost for the resulting delay would have a worse effect than the missing packet. Network administrators may influence the network transmission delay using QoS mechanisms such as DiffServ[5]. DiffServ allows the administrators to give the interactive data like voice and video a higher priority than traffic like emails or HTTP. In order to do so the traffic must be classified and then marked. IPv4 provides means for marking traffic and thus allows using QoS throughout the internet. The edge router through which the traffic enters the autonomous system either trusts the marking of data from a neighboring network or (re)marks the traffic itself. Although most of the networks utilize QoS there are still some that either don't or have incorrect setup.

Overcoming NAT and firewalls

First what is Network Address Translation (NAT) and what is it good for. The internet is based on Internet Protocol version 4. The IPv4 was designed in 1981, when internet was just a pack of machines mostly owned by universities and nobody could imagine IPv4 address space[6] could ever be used up. The enormous expansion of the internet cause that the IPv4 addresses were all used up in February of 2011[15]. This is where IPv6 comes in offering using 128 bit address and thus offering $3.4 \cdot 10^{38}$ cardinality of address space allowing anyone to have public IP address. The transition from IPv4 to IPv6 is however costly and complex due to the decentralized nature of the internet.

To connect new devices to the internet without having to wait for/transition to IPv6 NAT is used. It basically "hides" an entire network behind one IP address. This solves the problem problem with insufficient amount of addresses, but creates another problem. The devices behind NAT obtain a private address and can not be connected to directly from the internet. The only device that is directly visible from the internet is one with public IP address. Figure 4.3 shows the typical home network setup.

If a user on the home computer wants look at a website, the home computer sends a request to the router, which performs translation of the home private address to a ISP's network's private address. And the same happens on edge of provider's network. The routers performing NAT create a mapping of source IP and source port to translated address and port. The web server sends a reply back to the ISP, with it's address and the translated port number. The edge router looks at mapping and translates back the address and port

[5] Differentiated Services

[6] IPv4 address is a 32 bit number - 4,294,967,296 (2^{32}) addresses

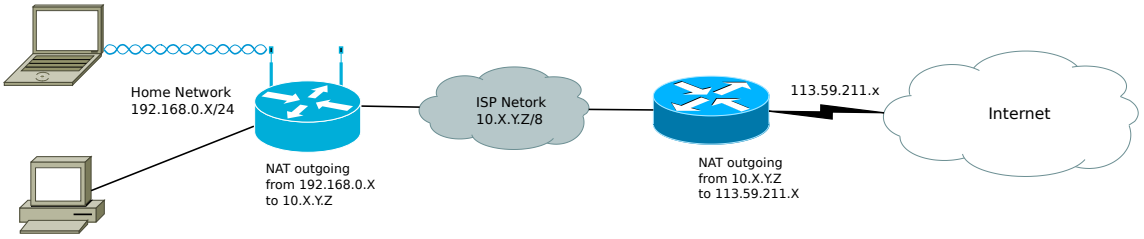


Figure 4.3: Typical connection to the internet

number and sends it the home router. Same action is performed by the home router and the packets are sent the home computer, which made the request.

Accessing the home computer from the outside can not be done for two reasons. First private network is not present in the internet's routing tables. And second even if we knew the address of the ISP's edge router, without the dynamic mappings the router would not know which device in it's network to froward the packets to. To summarize it is only possible to send data to a device in a private network if it initiates the communication. More so that is possible only if the peer has a public address. Two computers behind NAT can not communicate directly[7].

There are several solutions for overcoming NAT like STUN or TURN[8] (RFC 5766). STUN is a client-server protocol where the server has a public address and client is behind NAT. Client contacts the server with a request and the server then tries acquire port and IP address, which translate to the client's address and a certain port number. STUN however is not always successful as it can't work with all types of NAT. Next there is TURN, which acts as a relay server and relays traffic from one client to another in case no other mechanism works. It is clear that this is a last resort. Maintaining such servers costs money for the have be connected through links with high bandwidth and the traffic first goes in and then out.

Both of the above mentioned protocols are not universal enough for all kinds address translation systems. Interactive Connectivity Establishment (ICE) by IETF defined in RFC 5245 describes a methodology of taking the best of each NAT overcoming protocol to allow the clients with private addresses to communicate with each other either directly of through third party relay server.

[7]It is possible through port forwarding, which is intentionally omitted
[8]Traversal Using Relay NAT

Chapter 5

Port to Telepathy

Telepathy communication framework described in detail in chapter 3 is a logical choice of communication architecture if one wishes to create a powerful and easily maintainable IM application. A proof that porting to telepathy is a step in the right direction is for example a new client that is being developed by the KDE community and is supposed to become a default KDE IM application. The first release is expected any time now. At that point both main Linux window managers KDE and Gnome (Empathy) will have their default IM program based on telepathy. That fact alone promises great and lasting support of this architecture. This chapter talks about the porting of Makneto from iris library to telepathy and explains the design of the application and the reasoning behind it.

The first important change in the architecture of makneto is separation of user interface from the communication backend. There were several reasons supporting this decision. First is that since Telepathy is quite complicated and it takes time to get familiar with it would be best if user interface used a simplified abstraction. And that is where the communication backend comes in. Moreover the backend is the only piece of code that will have to be adjusted should Telepathy undergo any design changes or adjustments. The plan is to make the backend as portable as possible so it could run on most platforms possible. The user interface will be created specifically for the targeted platform. With that in mind it makes sense to separate backend and frontend. For instead of adjusting user interfaces for all the platforms to the changes the backend will absorb it and frontends will stay unchanged.

One of the aims of this work is to strip the current application of the iris library and implement the backend based on Telepathy. Qt4 bindings for Telepathy were chosen to be used simply because the rest of the application was implemented using this framework and it is overall very convenient to use with high level of abstraction. The following figure 5.1 shows a diagram of the new Makneto architecture.

5.1 Connection

As mentioned earlier Telepathy communicates over D-Bus, which means that if the application has to have a D-Bus well-known name in order for Telepathy to be able to address it. Namely the backend needs to implement `Client Handler` interface by subclassing `AbstractClientHandler` and register with D-Bus. Along with well-known name (in this case Makneto) the application lets telepathy know what channels it is capable of handling. Makneto backend is a Telepathy handler and at this point is able to handle text chat, multi-

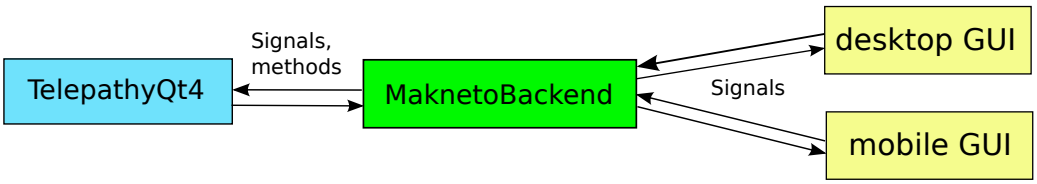


Figure 5.1: Simplified schema of Makneto components

user chat, audio and file transfer. It is also prepared for video calls, which is functionality outside of the scope of this thesis. More on that in the following chapter.

Client handler needs to be initialized and that is where the `TelepathyInitializer` class comes in. It merely needs a name of the client and sets up features for contact, connection, account and channel factories. The factories ensure that if the managed object like connection for example is ready it supports all the features specified upon the creation of the connection factory. That way the factories hold the desired features and once the object emits `ready()` signal there is no need to query the available features. The factories are supplied to constructor of Account Manager. Account Manager contains `Account` objects representing accounts of the user. To be able to use Account Manager `becomeReady()` method must be called and then wait for `finished()` signal. Then the accounts may be accessed. The accounts we get from the Account Manager must also be made ready the same way as the manager. Once everything is ready to use the `Initializer` emits `finished()` signal.

At this point nothing stands in the way of having the accounts go online. This is handled through a context menu of a contact list, that is described later in this chapter, on the frontend and by `TelepathyClient`. When an account is brought online the Connection Manager creates a Connection with features specified in the `ConnectionFactory`. If there is another telepathy application already running with the account online the Connection for that account will be shared to prevent wasting resources.

Figure 5.2 shows classes of the Makneto backend that participate on the communication with Telepathy and through it with anyone else. `TelepathyClient` handles both incoming and outgoing Channels of types asked for upon registration with D-Bus. Once a Channel arrives the `TelepathyClient` checks if a session with the contact already exists and if it does the Channel is handed over to it. If not than new session is created and signal is emitted to inform the user interface. `Session` class encapsulates all of the communication of the user with the outside world. This is the main difference between iris and telepathy. Iris uses a single point of entry for all communication whereas with telepathy it is done via Channels. The session represents all kinds of interaction the user might have with the outside world through Makneto.

It is important to note that outgoing channels even though requested via `Session` invoke `handleChannels()` method of the `TelepathyClient`. There is a flag present with the channel indication whether it has been requested or is coming from the outside.

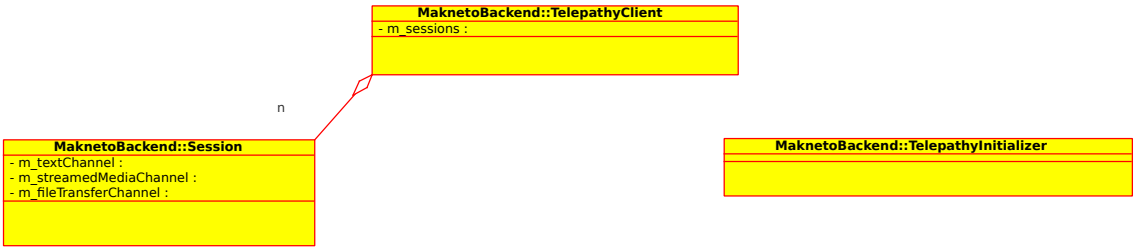


Figure 5.2: Makneto backend class diagram

5.2 Whiteboarding

Whiteboarding at this point is not supported by Gabble, connection manager for XMPP. Makneto itself the feature supports for it is Makneto's main feature. Until a plugin for whiteboard support is written for Gabble, the whiteboard messages will be tunneled through text messages. The main problem was sending SVG data to clients that would simply display the data to the user. Luckily XMPP offers resource parameter, which is used to check if the contact on the other side is using Makneto and thus can interpret the data correctly. Unfortunately this meant loosing the ability to whiteboard with clients implementing SVGWB by Joonas Govenius, though the JEP has not yet been approved. Whiteboard classes belong to the backend and once the functionality is added to Gabble it will be a simple task to make use of it and restore the lost compatibility.

The main whiteboarding classes remained unchanged except for the SVG data delivery. The previous implementation based on the iris library supplied conveniently a XML element containing the whiteboard information. Since telepathy does all the XML parsing and interpretation for us, the only option was reconstruct the `wb` element from string.

5.3 Contact list

Makneto's contact list has been ported to telepathy and utilizes Qt model/view architecture. The model is a part of the backend and the view is a part of the user interface. The contact list model is based on a model from Telepathy-Qt4-Yell project [6]. The model is connected to all the possible signals stating the item's properties have changed. Once the item has changed, the model's slot receives the signal and emits `changed()` signal, letting the view know it should update information showing to the user.

One of the greatest benefits of telepathy - multi protocol support - is now present in Makneto. The new contact list model has two types of nodes (items) as shown in the class diagram in figure 5.3. Since it is a tree model we will start from the nodes. Nodes are individual accounts registered with telepathy. These accounts may used in other applications built on telepathy. Leaves are items representing contacts under each account. Although whiteboarding is supported only through XMPP, the other of the features like text chatting or file transfer are available over the other protocols. It must be of course supported by the client of the peer.

As you can see from the class diagram the `TelepathyClient` has an instance of the contact list model. This is useful for requesting sessions with contacts from the frontend -

`RosterWidget`. It simply specifies the contact or account by sending index of currently selected item of the model. Any other way would require implementing special Makneto classes for contact and account in order to keep the frontend independent on telepathy

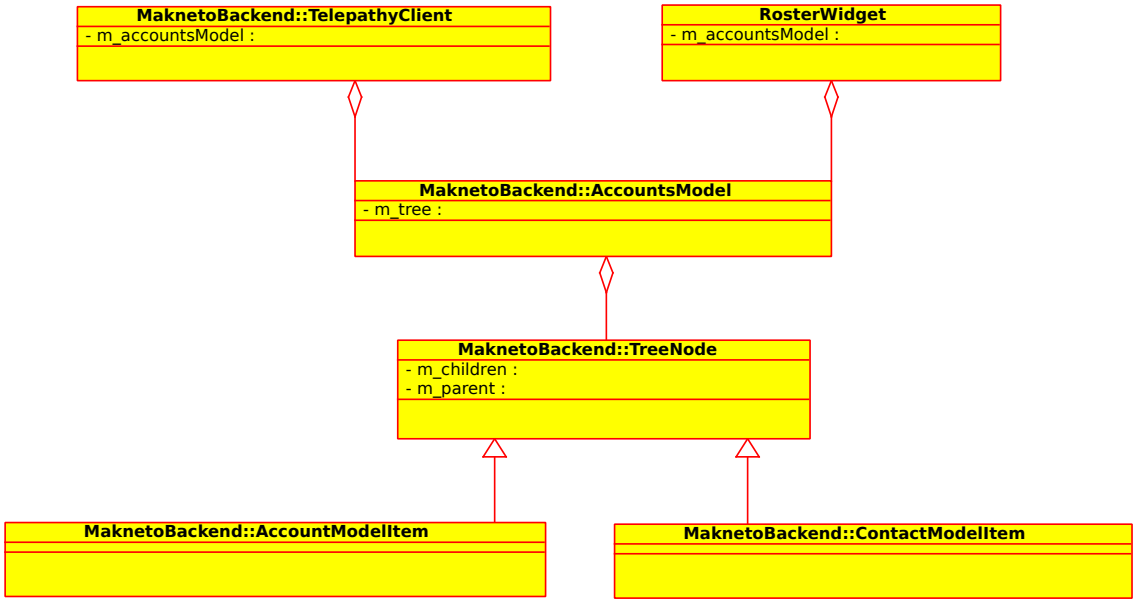


Figure 5.3: Makneto contact list class diagram

The user interface was kept almost the same even though both of the communication libraries have very different usage. Moreover this work does not aim at the frontend. One major change of the GUI is worth noting. The main class `Makneto` was an attribute in most of the classes and was passed along through all those classes. The singleton design pattern was utilized, for only one instance of the class exists. Makneto's `Instance()` method is static making it possible to get instance of the class anywhere within the application. `TelepathyClient` is also implemented as a singleton.

Chapter 6

VoIP implementation

The final chapter of this work contains details about implementation of VoIP into the application Makneto. Frameworks GStreamer [8] and Farsight2 [4] used for handling multimedia are described. And the last section talks about the implementation itself. Information in this chapter were found at [8, 30, 4].

Implementing an application handling multimedia means working with devices like sound card, camera, etc. There are several ways of doing so. First would be low-level access, which is platform and device specific. In GNU/Linux it requires using either Open Sound System (OSS) or Advanced Linux Sound Architecture (ALSA). Both of those offer abstraction of hardware beneath, but very low and work only on UNIX like systems and GNU/Linux respectively. Since we are aiming at portability they are not a good choice. Next is using a multimedia framework with higher level of abstraction supported on multiple platforms such as GStreamer [8] or Phonon [30].

6.1 GStreamer

GStreamer is a multimedia framework for writing streaming multimedia applications. It is based on GLib 2.0 and offers API for writing plugins and thus extending it's functionality. Programmers use API for creating applications handling audio, video or both. Recently a stable bindings for Qt have been released making the development even more convenient. Have a look at figure 6.1 to get an idea of what the framework consists of. GStreamer abstracts media sources (file, HTTP, ALSA, UDP, etc.), file formats (avi, ogg, mp4, etc.) and various filters, codecs and outputs. Moreover it has VoIP and video conferencing support. By using GStreamer the implementation does not have to deal with writing codecs or access to the system media sources. All of this is done by the framework.

Building an application using GStreamer is done by chaining the basic objects - elements. Each element performs a specific function like encoding audio stream, displaying video on the screen on reading from a network stream. It has data inputs and outputs called pads. Depending on the element it might have just an output pad called source[1]. That might be for example one representing an audio file. If it contains just an input pad - sink, then it might be a sound card or screen. Elements containing two pads are filters, encoders, decoders, etc. and more than two would be muxers, demuxers, etc.

Chaining elements to perform certain function means connecting their sources and sinks. The chain is called pipeline and there is an example of one in figure 6.2. There are several

[1] Source because it is a source from the next element's point of view

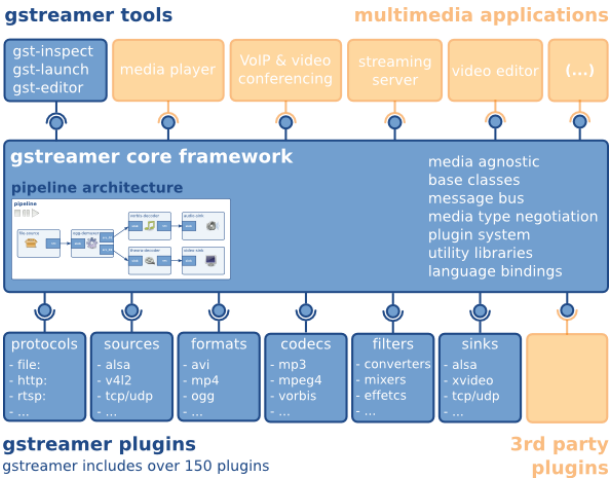


Figure 6.1: GStreamer architecture from [8]

elements linked together starting with the file-source. The file contains both audio and video and thus a demuxer is used with two sources, one for each stream. Next there is decoding and presenting the result to the user. With a large number of plugins supporting most of known codecs and devices this framework becomes a very powerful tool.

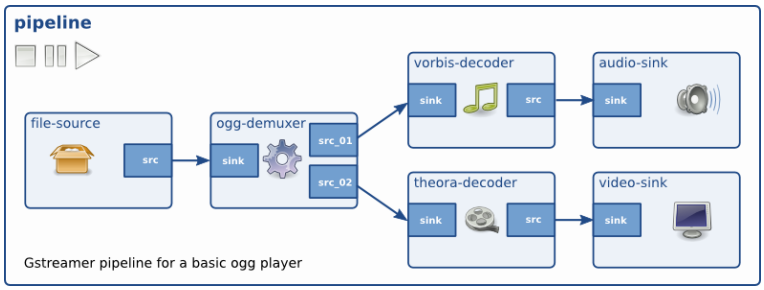


Figure 6.2: Example of a GStreamer pipeline from [8]

Pipeline is a special case of bin, which is a collection of elements. Since bin is also an element the programmer can create a pipeline consisting of several elements and performing certain function and use it as element in a larger and more complex pipeline. This design allows abstraction on a very high level. When linking elements together the media type that one produces and the other consumes must be compatible. That is determined through elements' capabilities in a process called "caps negotiation". GStreamer provides a way of automatic linking of elements called autoplugging based on capabilities.

When an element is created and linked there is no data flowing until it's state changed to play. The default state in which no resources are allocated is null. Next state is ready, when the element has all of it's resources allocated, but the stream is not opened yet. It is during this phase when the user finds out that the resource is unavailable or incompatible. From ready the elements go to paused by opening the stream a buffering it's contents. At this point the elements should be ready to switch state play instantly. Finally the play state is same as paused, but the data is actually flowing.

GStreamer is designed for high performance. Data between elements are copied only when absolutely necessary, otherwise pointer dereference is used. More so the stream processing is running in separate threads.

6.2 Farsight

Farsight2 [4] is library implemented in C GLib based on GStreamer architecture. It eases transporting multimedia over the internet. It is meant to be used instant messenger application for audio and video calls. Farsight is used by Telepathy via telepathy-farsight library for which there are even Qt bindings. Empathy utilizes farsight for it's audio and video calls and so does for example Nokia Internet Tablets.

This library provides GStreamer elements for multimedia streaming protocols. The supported protocols are UDP, RTP and MSN. One of the farsight's features is NAT and firewall overcoming ability. It implements ICE for reaching into private networks and HTTP tunneling for breaching restrictive filters. Moreover farsight supports dynamic codec negotiation and switching. It also makes it possible to bridge incompatible protocols and conference with for example a GTalk and Yahoo user.

Using farsight is consistent with using GStreamer, because it implements GInterface. `FsConference` represents a conference that can contain one or more sessions. `FsSession` corresponds to one RTP session and offers a sink and source pad. The sink pad should be connected to either microphone or video camera, generally an audio or video input device. The source pad would be linked to audio or video output device. One session has one media type and one sink pad, but can have more source pads in case of more participants [2]. For audio calls with video there will be two `FsSession` objects each for one type of multimedia. Each session has one or more transmitters. `FsTransmitter` abstracts the network layer. One transmitter serves for one transport method (UDP with ICE, UDP, HTTP tunneling, etc.). Finally there is `FsStream` object that represents a connection to one participant. The streams correspond with source pads in the session. Relationship among farsight objects are best seen in figure 6.3.

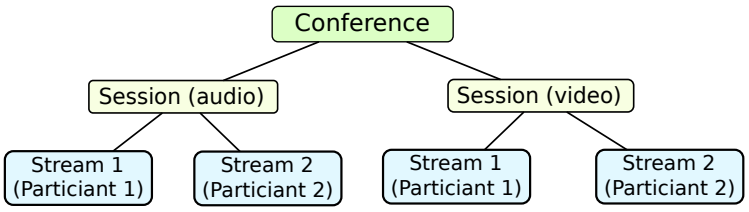


Figure 6.3: Farsight architecture

If used with telepathy, farsight utilizes `StreamedMediaChannel` for signaling. The protocol used for signaling depends on the used connection manager. In case of Gabble Jingle is used and SIP in case of Sofiasip. Farsight first creates a session and it's type is established. If there are more than just media types then sessions within this conference are created. Streams are built next. If the client allows user to specify his or her preferred

[2] one source for each participant

codecs it should be supplied to farsight when it emits `stream-get-codec-config` signal. Once farsight has the preferences it creates the stream. In case of VoIP the streams are bidirectional. At this point farsight negotiates the codecs. It goes through gstreamer plugins and looks for encoding and decoding elements. List of available codecs is sent to the remote machine. With codec negotiation finished the only thing left to is arranging a peer to peer transfer of the data through the stream. This is accomplished by using ICE. First remote candidates followed by local candidates are acquired. Candidate is an IP address and port where the participant might be able to receive data. Local and remote candidates are put in candidate pairs. Thanks to ICE farsight is able to determine whether the peers are on network and transfer data locally. When a working and suitable candidate pair is established the streaming begins. Farsight lets the application know by emitting signal `src-pad-added` of the stream object. That means the src pad of the stream is ready to supply data to a prepared GStreamer pipeline. Until this moment all the data transferred went through the signaling protocol.

6.3 VoIP in Makneto

The implementation of VoIP in Makneto is based on KDE Telepathy Call UI [24] by George Kiagiadakis. It uses telepathy farsight Qt4 bindings to create a `TfChannel`. The application communicates with the farsight channel via GLib signals. Events the application checks for are creation of a session followed by creation of a stream, channel closure and codec configuration request. Once stream is created more signals need to be connected. These include stream source pad ready, stream closed and resource requested. By emitting `resource-requested` signal farsight lets the application know that the GStreamer pipelines for processing incoming and outgoing data should be created and put into playing state.

The GStreamer pipeline used for audio data transmission is in figure 6.4. The chain starts with audio input representing either Pulse, ALSA or OSS input devices depending on what is available. The volume element is used for adjusting volume of the raw audio input. There is a volume widget available within the backend. Volume is linked to an audio converting element, which converts the raw audio between different formats. The following element handles audio resampling. And the final element represents the RTP stream encapsulated by farsight.

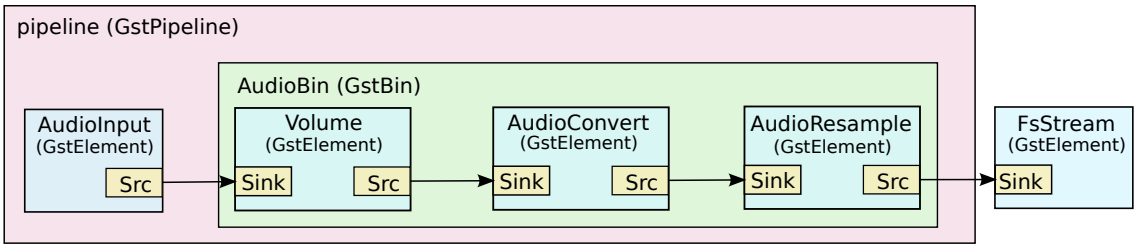


Figure 6.4: GStreamer pipeline used for VoIP in Makneto

The receiving pipeline is very similar. The `AudioBin` is used unchanged. The data however flow in the opposite direction. Moreover instead of audio input an audio output must be used. The process of acquiring it is the same.

Makneto is prepared for video calls as well. However there is a known problem when starting a second stream with farsight. Namely racing occurs when the new stream is created. Sometimes the packets of the new stream arrive before the `src-pad-added` signal of the stream is connected. This means the stream is not linked and thus no data get to the GStreamer pipeline.

Testing

Firstly we were unable acquire many test subjects, which makes the results less representable. The main problem was the need for up-to-date software. TelepathyQt4 for example is required to be in a less than three weeks old version. That greatly reduces the number of possible test subjects to the ones capable of compiling various libraries from source and resolving dependencies. This is usually done by the package manager. In this case however the package managers offer versions that are too old for compilation. And even when successfully compiled outdated farsight will prevent successful cal establishment. The computers used for testing ran Fedora 15, Ubuntu and Kubuntu. All of them used the Farsight2 0.0.28, GStreamer at least 0.10.33, Telepathy-Farsight at least 0.0.17 and TelepathyQt4 0.5.16.

The tests of VoIP functionality of Makneto have revealed that successful call establishment depends on updated Farsight2 and telepathy-farsight. We were able to make a call from Makneto on one laptop to Makneto on another laptop with no problem. Another attempt however resulted in stream working in one direction only. Sometimes the call is not successful at all for no apparent reason. All the necessary steps finish with no error and yet there is no sound. Due to low number of test subjects we were unable to insolate the problem causing the call establishment to fail.

Next set of tests was done using Makneto on one side and Empathy on other. In configurations where we were previously able to establish data flow in just one direction Empathy work in both. However not even Empathy work on hundred percent and calls did not succeed.

Conclusion

The world of today can be described as one enormous network, to which everyone is or soon will be connected. One of the Internet's greatest features is bringing people together. People who hundred of years ago would have to travel sometimes even months to see each other or wait for a letter to get an opinion on an idea from a colleague. Today it is just a few clicks away through collaborative applications where Makneto undoubtedly has it rightful place.

In this work the application Makneto was taken and ported the communication module from iris library to telepathy. Due to differences in the connection scheme of both libraries - centralized iris as opposed to distributed telepathy - the architecture of Makneto had to be adjusted. The removal of iris also simplifies the application deployment in the long run. Iris is not available via package managers whereas telepathy is.

Makneto now thanks to telepathy supports most of the known instant messaging protocols. Depending on the protocol the application is able to provide file transfer and chatrooms. The whiteboarding is due to non existing plugin for Gabble now incompatible with clients other than Makneto. This is only temporary until the Gabble plugin is made. XEP describing the whiteboarding for XMPP is however not approved and Gabble developer might implement it only when approved. At this point the SVG data for whiteboard are transferred through text messages.

VoIP functionality was added to Makneto using GStreamer and Farsight2. Adding a video support for the calls partially depends on the development of farsight as it tends to have problems with adding video stream. The current version supports one to one audio calls. Makneto is able to establish calls not only with itself but also with other applications such as Empathy.

The application Makneto was described in detail in chapter 1 including the underlying architecture and the technologies it builds on. Makneto's architecture is shown and it's strengths and weaknesses pointed out. Makneto uses XMPP protocol for communication. Chapter 2 explains how it works and what it features.

Existing whiteboarding and VoIP programs are listed and analyzed in chapter 4. Numerous applications exist featuring whiteboard and VoIP and some of them even video conferencing. All of the programs unfortunately used proprietory protocols and do not provide any information about it at all. Then this chapter goes over some protocols used for streaming media. Lastly it mentions issues that must be overcome when implementing VoIP and how to deal with them correctly.

Chapter 5 goes over the changes made in Makneto regarding the port to telepathy. The changes in Makneto's architecture and the reasoning behind them are discussed. Telepathy is very powerful as well as complex framework. It's architecture and qualities are described in chapter 3.

Finally the chapter 6 explains how was the VoIP functionality implemented into Makneto.

It describes the used libraries including the way the entire process works. The application is included on CD enclosed in this work along with a manual containing install instructions.

All the goals of this thesis were achieved.

Makneto is a great application and there are many possible improvements that can be made. Firstly the GUI needs to be improved for not all the implemented functionality is well connected with it. Next it could offer conference calls since both the whiteboard and the text chat allow more participants. Video support for audio calls might added as well. The whiteboard could support various plugins such as chess or checkers that could be written in QML and thus sent over the application on the fly. Lastly there is no frontend for increasingly popular and common mobile devices, which would take Makneto to the next level.

Bibliography

[1] Terry Brock. Skype outage today. http://blogs.skype.com/business/2010/12/skype_outage_today.html. [online; accessed 27 December 2010].

[2] Brosix. Brosix - seruce corporate instatnt messaging for companies. http://www.brosix.com. [online; accessed 30 December 2010].

[3] Collabora. D-bus. http://www.freedesktop.org/wiki/Software/dbus. [online; accessed 19 November 2010].

[4] Collabora. Farsight2 wiki. http://farsight.freedesktop.org/wiki/. [online; accessed 4 April 2011].

[5] Collabora. Telepathy. http://telepathy.freedesktop.org. [online; accessed 12 November 2010].

[6] Collabora. Telepathy qt4 yell. http://cgit.collabora.com/git/freedesktop.org-mirror/telepathy/telepathy-qt4-yell.g [online; accessed 6 March 2010].

[7] Collabora. Telepathy wiki. http://telepathy.freedesktop.org/doc/book/index.html. [online; accessed 18 November 2010].

[8] Collabora. Gstreamer 0.10 core reference manual. http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html/, 2008. [online; accessed 10 March 2011].

[9] XMPP Standard Foundation. Xmpp standards foundation. http://xmpp.org/. [online; accessed 11 December 2010].

[10] Bur Goode. Voice over internet protocol, 2002.

[11] Google. Google talk. http://www.google.com/talk/. [online; accessed 12 January 2011].

[12] Joonas Govenius. Svg whiteboarding. http://xmpp.org/extensions/inbox/wb.html, 2006-06-19. [online; accessed 11 December 2010].

[13] H. Schulzrinne et. al. Rtp: A transport protocol for real-time applications. http://www.ietf.org/rfc/rfc1889.txt, 1996. [online; accessed 7 January 2011].

[14] H. Schulzrinne et. al. Rtp: A transport protocol for real-time applications. http://www.ietf.org/rfc/rfc3550.txt, 2003. [online; accessed 7 January 2011].

[15] IANA. Iana address space report. http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml/. [online; accessed 23 February 2011].

[16] ICQ. Icq. http://www.icq.com. [online; accessed 29 December 2010].

[17] J. Rosenberg et. al. Stun - simple traversal of user datagram protocol (udp) through network address translators (nats). http://tools.ietf.org/html/rfc3489, 2003. [online; accessed 6 January 2011].

[18] J. Rosenberg et. al. Session traversal utilities for nat (stun). http://tools.ietf.org/html/rfc5389, 2008. [online; accessed 6 January 2011].

[19] J. Rosenberg et. al. Sip: Session initiation protocol. http://www.ietf.org/rfc/rfc3261.txt, June 2002. [online; accessed 9 January 2011].

[20] Joe Hildebrand, Peter Millard, Ryan Eatmon, Peter Saint-Andre. Xep-0045: Multi-user chat. http://xmpp.org/extensions/xep-0030.html. [online; accessed 21 January 2011].

[21] Joe Hildebrand, Peter Saint-Andre, Remko Tronçon, Jacek Konieczny. Xep-0115: Capabilities advertisement. http://xmpp.org/extensions/xep-0115.html. [online; accessed 23 January 2011].

[22] KaZaA. Download music - music downloads and mp3 downloads from kazaa.com. http://www.kazaa.com. [online; accessed 26 December 2010].

[23] KDE. Kde. http://www.kde.org/. [online; accessed 3 December 2010].

[24] George Kiagiadakis. Real-time communication and collaboration/components/call ui. http://community.kde.org/Real-Time_Communication_and_Collaboration/Components/Call_UI. [online; accessed 6 April 2010].

[25] A. M. Kondoz. *Digital Speech*. John Wiley & Sons Inc., 2004.

[26] M. Handley et. al. Sdp: Session description protocol. http://www.ietf.org/rfc/rfc4566.txt, July 2006. [online; accessed 9 January 2011].

[27] M. Handley et. al. Sip: Session initiation protocol. http://www.ietf.org/rfc/rfc2543.txt, March 1999. [online; accessed 9 January 2011].

[28] Microsoft. Windows live messenger 2011. http://explore.live.com/windows-live-messenger. [online; accessed 4 January 2011].

[29] Daniel Molkentin. *The Book of Qt 4*. O'Reilly, July 2007.

[30] Nokia. Phonon module. http://doc.qt.nokia.com/4.7/phonon-module.html. [online; accessed 7 April 2011].

[31] Nokia. Qt - a cross-platform application and ui framework. http://qt.nokia.com/. [online; accessed 15 November 2010].

[32] America OnLine. Aol instant messenger. http://www.aim.com. [online; accessed 2 January 2011].

[33] Colin Perkins. *RTP: Audio and Video for the Internet*. Addison-Wesley, June 2008.

[34] The GNOME Project. Empathy - gnome live! http://live.gnome.org/Empathy. [online; accessed 1 Decemeber 2010].

[35] Psi. Iris library. http://psi-im.org/wiki/Iris_Library. [online; accessed 17 November 2010].

[36] Peter Saint-Andre. Xep-0045: Multi-user chat. http://xmpp.org/extensions/xep-0045.html. [online; accessed 21 January 2011].

[37] Peter Saint-Andre. XMPP Core. http://www.ietf.org/rfc/rfc3920.txt. [online; accessed 27 December 2010].

[38] Peter Saint-Andre. XMPP IM. http://www.ietf.org/rfc/rfc3921.txt. [online; accessed 27 December 2010].

[39] Peter Saint-Andre, Kevin Smith, and Remko Troncon. *XMPP: The Definitive Guide*. O'Reilly, April 2009.

[40] Henning Schulzrinne Salman A. Baset. An analysis of the skype peer-to-peer internet telephony protocol. http://www.cs.columbia.edu/ library/TR-repository/reports/reports-2004/cucs-039-04.pdf, 2004. [online, accessed 26 December 2010.

[41] Scott Ludwig, Joe Beda, Peter Saint-Andre, Robert McQueen, Sean Egan, Joe Hildebrand. Xep-0166: Jingle. http://xmpp.org/extensions/xep-0166.html. [online; accessed 25 January 2011].

[42] Skype.com. Skype. http://www.skype.com. [online; accessed 26 December 2010].

[43] W3C. Scalable vector graphics(svg). http://www.w3.org/Graphics/SVG/. [online; accessed 18 December 2010].

[44] W3C. Svg tiny 1.2 specification. http://www.w3.org/TR/SVGTiny12/. [online; accessed 18 December 2010].

[45] Wikipedia. Wikipedia, skype protocol. http://en.wikipedia.org/wiki/Skype_Protocol. [online; accessed 26 December 2010].

[46] Wikipedia. Wikipedia, yahoo! messenger. http://en.wikipedia.org/wiki/Yahoo!_Messenger. [online; accessed 27 December 2010].

[47] Yahoo! Yahoo! messenger. `http://messenger.yahoo.com`. [online; accessed 30 December 2010].

[48] Jaroslav řezník. Sdílená tabule. Master thesis, 2008.

Appendix A

CD Content

/Makneto/* - revision of program Makneto from May 23rd 2011
/Makneto/README - manual for compiling Makneto
/thesis/* - source code of this thesis
/projekt.pdf - text of this work