UNIVERSITY OF TARTU
Faculty of Mathematics and Computer Science
Institute of Computer Science
Speciality of Computer Science

Madis Kapsi

Usability Improvements for the Friend-to-Friend Computing Pidgin Plugin

Bachelor Thesis

Supervisor:	Ulrich Norbisrath
Co-supervisor:	Artjom Lind

Author: .... ”......” June 2010
Supervisor: …... “......” June 2010
Co-suprevisor: .... “......” June 2010

Allow to Defense:
Professor: .... “......” June 2010

Tartu 2010

Contents

Chapter 1

Introduction

The aim of this thesis is to analyze the existing user interface of the F2F Pidgin plug-in, suggest ideas that could improve user-friendliness and implement several of them. The more general target is making more computing power available for everyday use through Friend-to-Friend (F2F) Computing[4]. Most people today use some kind of an instant messenger like Pidgin. The F2F Computing Pidgin plug-in enables Pidgin users to share computing power, run shared applications and play games with friends and colleagues. There are several games and applications available for F2F Computing like “Hangman”, “Bub’s brothers”, Monte Carlo computations, or distributed Blender.

F2F Computing is a concept that combines ideas from peer-to-peer networking, Cloud Computing and social networks. It offers a framework on top of multi-protocol instant messengers like Pidgin[11] or SIP Communicator[12]. F2F Computing can be used by individual researchers, small research groups or companies for combining their computational power with friends and colleagues.

Regarding the sharing of computational power, F2F Computing has several advantages compared to other Grid systems. First, current computing Grids are fairly complicated and always need an IT specialist to set up and run them. Furthermore, with F2F, users do not need to make any extra expenses to start a new Grid. Just gather a group of your contacts and start sharing computational tasks, running applications or playing games. This means that more computing power is available for users who need it, but cannot afford to purchase or rent expensive systems. Also research groups that are not computer science related, might not need an IT specialist at all to run their own Grid with F2F Computing.

The basic Pidgin plug-in has no specific user interface elements and is built for any libpurple based instant messenger. This paper concentrates on a user interface optimized for Pidgin.

Chapter 2

Preliminaries

This chapter will give a short overview about papers and theses about Friend-to-Friend Computing, a few examples of instant messenger clients and technologies used in Friend-to-Friend Computing and in its Pidgin plug-in. The papers about F2F Computing will give a good idea about its underlying principles.

2.1 Friend-to-Friend Computing framework

Friend-to-Friend Computing framework is built, so it could be used on different platforms with the help of using instant messaging systems. The basic concepts and the first prototype of this framework, realized as a plug-in to SIP Communicator[12], are described in Keio Kraaner's master thesis "Friend-to-Friend Computing"[19] written in 2008 and in paper by Ulrich Norbisrath, Keio Kraaner, Eero Vainikko, Oleg Batrašev "Friend-to-Friend Computing - Instant Messaging Based Spontaneous Desktop Grid"[20]. The most relevant part is the chapter about the framework, as it describes general concepts of F2F Computing. The API and programmers guide is not relevant anymore with current system, because F2F Computing core has been re-written in C programming language since then.

2.2 Friend-to-Friend Computing mobile

Mobile phones of today have already more computational power than PC-s from late 1990-s. Mobile users also use their phones for other purposes than making calls.

Sven Kirsimäe reviews the mobile industry in the context on software development and creates F2F Mobile Computing framework in his master thesis "F2F Mobile Computing"[18]. The review of the mobile industry part is not so relevant to F2F Computing, as it is to mobile application developing in general. He reached conclusion that the most reasonable platform choice for mobile framework was Symbian S60v3. The following part about the F2F Mobile Computing framework describes the process of creating the framework. The thesis also contains a F2F Mobile Computing demo – installing and running the application on a mobile phone.

2.3 Instant messengers

Instant messaging is a form of real-time direct text-based communication between two or more people. Instant messenger is a client side program that is used for the messaging process – sending and receiving text and other data. These programs consists of two major components - core and user interface. Core provides communication between clients and support for different protocols. User interface is front end part visible to user.

There are several different types of messengers, but the main purpose is still communication with other users. Some messengers use peer-to-peer protocol others require peers to be connected to central server. Different messaging systems have created their own protocols over time that are referred as IM protocols. The most popular IM services according to [6] are Skype, Windows Live Messengers, Facebook chat and Tencent QQ (majority of users are in China).

Programs to use Instant Messenger systems are called IM clients and the choice of them is quite large at the moment[7]. Messenger clients can be divided to single- and multiprotocol by the number of messaging systems supported. Some client side applications are also platform dependent, because messenger client uses some system libraries that are not available on other operating systems. Table 2.1 shows different popular messenger clients. Pidgin is a good example of multi protocol and multi platform client, that is available for Windows and Linux.

Next I give a short overview of different examples for instant messenger protocols and clients.

2.3.1 Libpurple

`Libpurple`[8] is a core library for an IM program. Instant messenger programs built on `libpurple` use it to connect to IM networks, manage user accounts and preferences. At the moment `libpurple` without any plug-ins or modifications supports 15 different messaging protocols, most common of which are MSN, XMPP, Yahoo, and IRC. A few examples of `libpurple` based messenger clients are Adium for OS X, Pidgin for Gtk (Windows, Linux/Unix), Meebo web-based and Finch text-based for Linux/Unix.

Plug-ins written for `libpurple`, without using any platform specific packages, in principle can be used with any `libpurple` based instant messenger client. There are three different types of `libpurple` plug-ins that do not contain UI code: core, loader and protocol. First of those is for extending core functionality. Loader plug-ins are invisible and do not appear in plug-in list (accessible from Pidgins buddy list "Tools/-Plugins" menu). Their purpose is to enable loading plug-ins written in other languages, for example Perl and Tcl support are provided via loader plug-ins. Protocol plug-ins are also invisible and enable `libpurple` to connect to so many different networks.

2.3.2 Telepathy

Telepathy[15] advertises itself as the Flexible Communications Framework. Telepathy is, like `libpurple`, unified for several different instant messaging servers and protocols. The aim of Telepathy is to provide a way to integrate chat or VoIP into desktop applications (word processors, CAD programs and others). Its only current drawback is that Telepathy is available for GNOME only, so Windows and OS X users can not use it in their native operating systems and need dual boot systems or virtual machines for using Telepathy.

At the moment most prominent Client for Telepathy is Empathy[2], as it is shipped with Ubuntu 10.04 as a default chat client instead of the Pidgin. Empathy looks quite similar to Pidgin in first glance, but there are differences as well. One, that is most noticeable in everyday use, is support for chat themes via WebKitGtk, which is something similar to Adiums "WebKit Message View"[17].

2.3.3 Skype

Skype[13] is a good example of a one protocol instant messenger that is available on almost all operating systems, including mobile phones. However, the Linux version is much more basic and does not support several features, in comparison to the Windows version, for example playing a game with a contact. As Skype is entirely built for its own protocol it works very good and the VoIP part, which made Skype so popular, works perfect on all the platforms.

In comparison to libpurple and Pidgin, Skype is much better documented. There is a well structured API[14] with example codes for creating a 3-rd party extension, where `libpurple` and Pidgin API-s are lists of file references.

2.3.4 Adium

Adium[1] is a Pidgin reimplementation for Mac OS X. One significant user interface difference to Pidgin is that Adium allows a user to modify the way messages are displayed via the "WebKit Message View"[17]. From the user friendliness point of view Pidgin has fairly better choice of user interface languages, as Pidgin "speaks" more than 80 different languages, where Adium can only manage 27. This is mainly because Pidgin has a significantly larger development group.

2.3.5 Other messenger clients

As mentioned before there are many more different messenger clients available. Some of the most popular single protocol messengers are Windows Live Messenger, Yahoo! Messenger and ICQ. First of those is also only available for Windows operating systems. Yahoo! Messenger is supported on Windows and Mac OS X, it was also available for Linux, but last release was 6 years ago. ICQ does not have Linux version as well, but it has a version for mobile phones .

A few of the popular multi-protocol messenger clients are Trillian, Kopete and eBuddy. Trillian, like several others, does not have Linux support, but it has a web-based version, that is accessible via browser in any operating system. It is also available on iPhone and BlackBerry. However, Kopete is one of few IM clients that is not available for Windows and has versions for Unix-like systems and Mac OS X. Last is the eBuddy that is a web-based IM client that is accessible on any computer that has an Internet connection and a browser. eBuddy has also applications for mobile phones, iPhone, iPod Touch and Android.

See also Table 2.1.

Messenger client	Type	Supported platforms
Adium	Multi-protocol	Mac OS X
AOL	Single protocol	Windows, Mac OS X, Linux
eBuddy	Multi-protocol	Web based/Mobile devices
ICQ	Single protocol	Windows, Mac OS X
Pidgin	Multi-protocol	Windows, Linux
Skype	Single protocol	Windows, Mac OS X, Linux
Trillian	Multi-protocol	Windows, Mac OS X, iPhone
Windows Live Messenger	Dual protocol	Windows
Yahoo! Messenger	Dual protocol	Windows, Mac OS X, Unix/Linux

Table 2.1: Instant messenger clients

2.3.6 Roundup

There is no perfect Instant Messenger client, that suits every-ones needs and requirements. However, there are plenty to choose from and most of these clients can be also extended using plug-ins and extensions, so the user can “build” the perfect messenger.

2.4 Peer-to-peer

Peer-to-peer[10] (P2P) is a distributed network architecture, that makes portion of users resources (processing power, disk storage or network bandwidth) directly available to other network participants. Peer-to-peer is special because it does not need central coordination(servers or stable hosts). Network participants are hosts and clients at the same time. In comparison to traditional server-client solutions, where servers are suppliers and clients consumers, the peers in P2P are both consumers and suppliers.

There are three main types of peer-to-peer networks. Pure P2P networks, that consist solely of peers. Hybrid peer-to-peer has some special preferred nodes for running infrastructure functions. Centralized peer-to-peer networks have a central server that is used for indexing functions and to bootstrap the system.

P2P was popularized by file sharing systems like Napster[9], which is a sample of centralized model of peer-to-peer network. Sharing files and real time data is common for P2P, for example Skype calls are passed using P2P technology.

Peer-to-peer networks also divide to unstructured and structured networks, classification is based on how the nodes are connected. Nodes are peers in overlay network and there is a link between two nodes if a peer knows the location of another peer in P2P network.

Structured P2P networks have globally consistent protocol, to guarantee most efficient routing to a peer that has desired resource. Maintaining the mapping is responsibility of nodes and is distributed in such way, that it will cause minimal disruption in the case of change in the set of participants.

In unstructured network new peer that wants to join the network can copy a set of existing links from another node and later form its own links over time. In unstructured P2P networks the query, for finding desired data, has to be flooded through network to as many nodes as possible. Disadvantages with this are, that queries might not be resolved and flooding creates high amount of signaling traffic. Queries not being solved is not so big problem with popular content and search for them is more likely to be successful, where for rare items it is real possibility that search does not provide any results. Typically unstructured networks have very poor search efficiency.

2.5 GTK+

GTK+[5] is a toolkit for creating graphical user interfaces. It has cross platform compatibility and an easy to use API. Initially GTK+ was created for and used by GIMP. Today it is used by many applications, including GNOME desktop project.

GTK+ itself is written in C, but it supports C/C++, Perl and Python. It is based on four libraries that are also developed by GTK+ team, these libraries are `GLib`, `Pango`, `Cairo` and `ATK`. `GLib` is a low-level core library for structure handling and portability wrappers. `Pango` is a layout and text rendering library. `Cairo` is a 2D graphics library, supporting multiple output devices, that can use hardware graphics acceleration when it is available. Last `ATK` is a library that provides accessibility support, so the application can be used with screen readers, magnifiers, and alternative input devices.

2.6 Technologies in this thesis

In this thesis the main purpose is improving usability, so the most used is the GTK+ library. Also `libpurple` and F2F Computing core libraries are used for several features implemented.

Chapter 3

Plug-in usability analysis

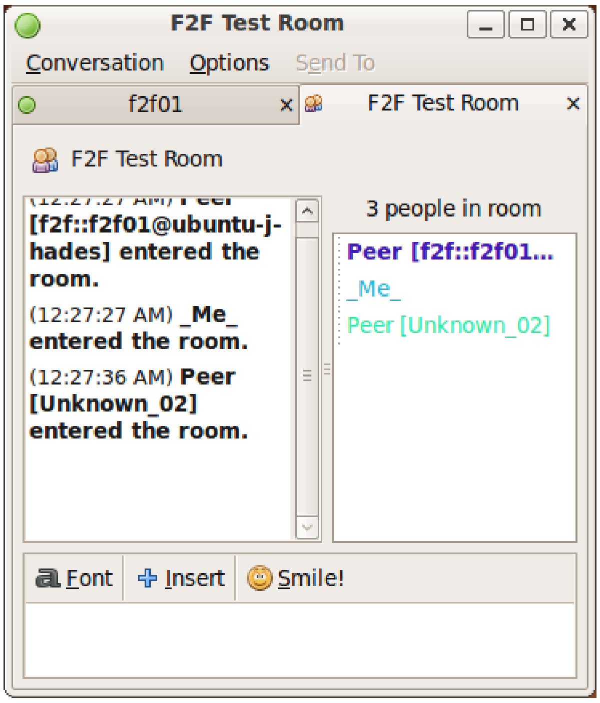
This chapter describes existing F2F Computing plug-in for Pidgin[11]. As it wasn't specifically designed for Pidgin the user interface had almost no elements for using it, as shown in Figure 3.1. That is why I would suggest some changes to chat window, configuration and plug-in monitoring interfaces.

The ideas for chat window would be a F2F menu on chat window toolbar(Figure 3.5) and a progress bar on info pane(Figure 3.6). For configuration – plug-in and group preferences dialogs. A debugger to show each groups log and finally user information modifications regarding to computer information.

3.1 Using existing system

Steps to use F2F Computing Pidgin plug-in after it is installed:

1. Log in existing messaging account(guide at [16])
2. Create new F2F protocol account
   (a) Open "Accounts" menu from Buddy List and click on "Manage Accounts"
   (b) List of existing accounts will open (Figure 3.2). Click "Add..".
   (c) User name should be the same as the actual accounts user name
   (d) Set Local Alias to "_Me_" (see Figure 3.3).
   (e) Click "Add" and F2F account is ready.
3. Add chat to Buddy list
   (a) From buddy list menu "Buddies" select "Add chat.."
   (b) For "Account" choose F2F account
   (c) Fill in other fields (example in Figure3.4)
   (d) Click "Add" to finish.
4. Open created chat by double-clicking on the name.
5. Invite contacts to chat – only F2F contacts can be added. They are in F2F section in buddy list and names start with "Peer"



(12:27:27 AM) Peer [f2f::f2f01@ubuntu-j-hades] entered the room.

(12:27:27 AM) _Me_ entered the room.

(12:27:36 AM) Peer [Unknown_02] entered the room.

Peer [f2f::f2f01...

_Me_

Peer [Unknown_02]

Figure 3.1: Chat before

   (a) Drag and drop them from Buddy List
   (b) Or "Conversation / Invite..."

6. Chat window can be used for instant messaging

7. To submit service for current group

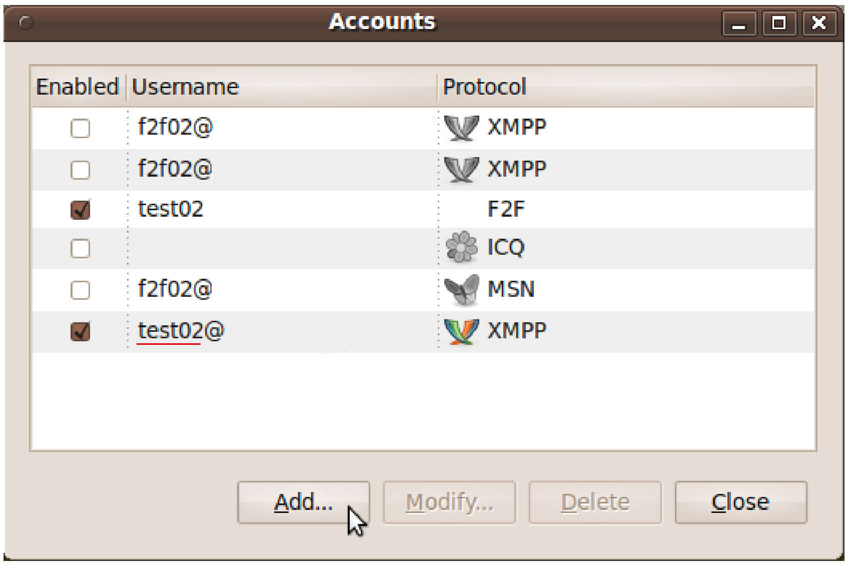
   (a) In F2F chat room: "Conversation / More / Submit Service"
   (b) Browse for compatible task
   (c) Submit

8. Task is completed in the background, no indication in the chat window when it's completed

As shown the steps for submitting service to chat group aren't complicated, but required menu item is in "More" sub menu, which doesn't indicate to F2F in any way. Also after submitting the service user interface does not give any feedback about progress made. Only way of receiving any information is to open Pidgin built in Debug Window.

3.2 Ideas for improving chat window

As described in last section there are almost no user interface elements in the chat window and its menus. The only indication of using an F2F Computing chat are the contacts in the buddy list and in chat participants list, as their names start with "Peer". A few ideas for the chat window are:



Enabled	Username	Protocol
☐	f2f02@	XMPP
☐	f2f02@	XMPP
☑	test02	F2F
☐		ICQ
☐	f2f02@	MSN
☑	test02@	XMPP

Figure 3.2: Account list

1. The toolbar shown in Figure 3.5 could have an F2F Computing menu. From that menu a user can access plug-in specific commands like submitting a service or configuring the plug-in. The location is chosen like this, because it is easily noticeable and uses empty space on the toolbar.

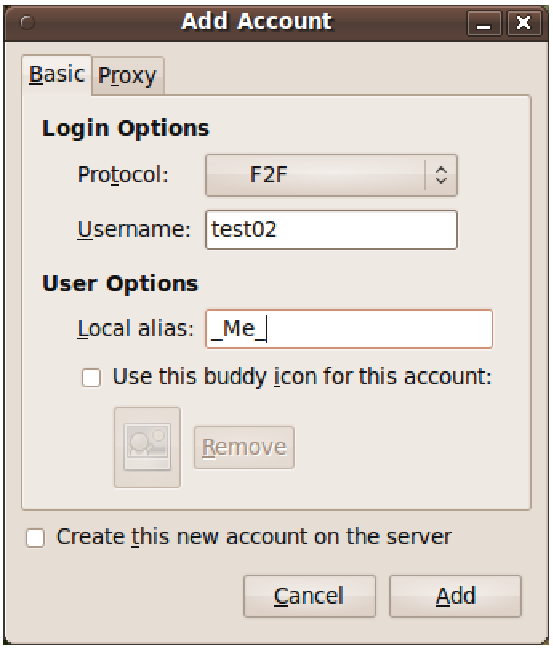
2. An optional progress bar should exist for observing progress of the submitted task. The location for it could be in the top right corner of the chat window. Right from the chat name and on top of the list of contacts in chat - located on the Infopane of Pidgin chat window structure, shown in Figure 3.6.

Proposed changes will not change the look of the chat window much, but will make using the plug-in easier and a more pleasant experience. The chat window will look like shown in Figure 3.7.

3.3 Configuration

The Pidgin plug-in was in no way configurable, but as one of the uses is sharing computing resources it should be. While browsing the web, working on documents or performing any other tasks which do not require much of computers resources, user should be able to allow a higher utilization of the computer by the F2F Computing plug-in. However, while using more of computers resources, but not wanting to shut down the plug-in all together, it would be useful to be able to reduce the shared computing power or priority of the F2F Computing task. For example a user has a quad core processor and 4GB of RAM, but is using the computer to edit text documents and send e-mails. He could let the plug-in use 3 of 4 cores and about 2GB RAM. And later while editing videos, photos and music, turn it down to 1 core and 512 - 1024 MB of RAM.

User interface itself should be configurable as well – for example if user does not wish to have a progress bar in the chat window, it could be disabled.



Login Options

Protocol: F2F

Username: test02

User Options

Local alias: _Me_

☐ Use this buddy icon for this account:

☐ Create this new account on the server

Figure 3.3: Add account dialog

3.4 User info

Before submitting a service, that uses a lot of computing resources, it would be good to know, on what kind of hardware F2F Computing is ran on the contact side. For example F2F Computing is also available for Symbian S60 smart phones and sending taxing computational tasks there would not make sense, while playing games with that contact is reasonable. For that purpose there should be a way for the user to set their computer information and share it with their F2F Computing contacts.

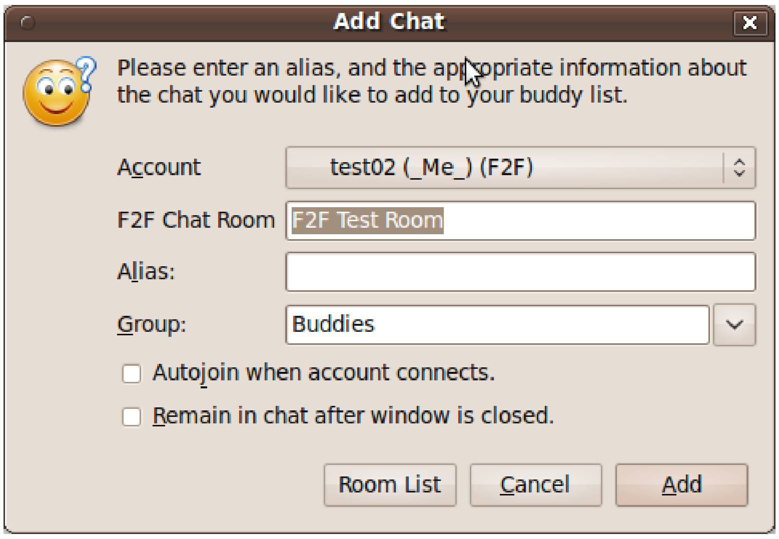
If the user is able to change how much computing power is shared, then this information could also be shared via user information that is visible to contacts. Accessing this info would be just like getting any users info – right click on contacts name and select "Get Info". Modifying information about user is in preferences dialog, which is available from the chat window toolbar "F2F" menu or "Conversation / More" menu. As not all the F2F Computing is done via Pidgin plug-in, this information should be accessible from the F2F core.

However, user might participate in different groups at the same time and change shared performance per group. This is why there should be so-called global preferences for plug-in where user can set the maximum amount shared performance figures at any given moment. And for every group a separate preferences where user can set different, lower values for certain group.

Finally, there is only one alias which is the same in every group. But there could be a system similar to IRC where user can set different nicknames in every chat, so the alias is not constant in all groups.

3.5 Computing information

For more difficult computational tasks, groups might get quite large and viewing every contact's computer information separately is not too user friendly. For that purpose


Add Chat

Please enter an alias, and the appropriate information about the chat you would like to add to your buddy list.

Account: test02 (_Me_) (F2F)

F2F Chat Room: F2F Test Room

Alias:

Group: Buddies

☐ Autojoin when account connects.

☐ Remain in chat after window is closed.

Room List | Cancel | Add

Figure 3.4: Add chat dialog

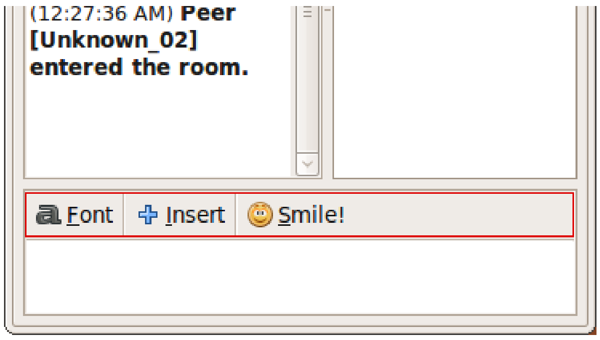

(12:27:36 AM) **Peer [Unknown_02] entered the room.**

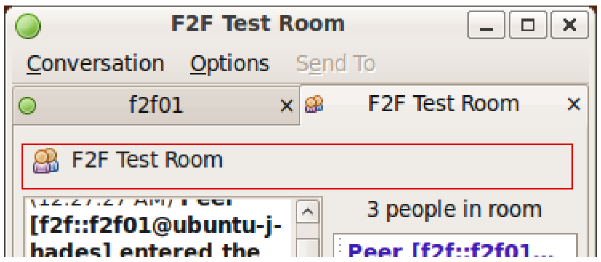
Figure 3.5: Chat toolbar

there could be a table, that shows entire groups shared computer resources. The table should show computers processor and RAM information and shared cores and memory amounts. For example one row might contain data like: cpu - Amd x2 250 3,6GHz, shared cores - 1, RAM - 6GB ddr2 800MHz, shared memory - 2GB.

For some tasks Internet connection type might be also important as completing a computational task by sending large files over slow connection might take longer than sending them to a bit slower computer, but over a faster connection.

3.6 Plug-in activities information

Only information about ongoing task at the moment is in Pidgin Debug window with the rest of the Pidgins log. That means besides suggested progress bar there is no easy access to view information about the plug-in progress or activities. That is why F2F Computing should have an information/debug window of its own. The information flow should be adjustable so that a user could select how technical info is shown - network activity, core and/or plug-in information.



[f2f::f2f01@ubuntu-j-hades] entered the

Peer [f2f::f2f01...

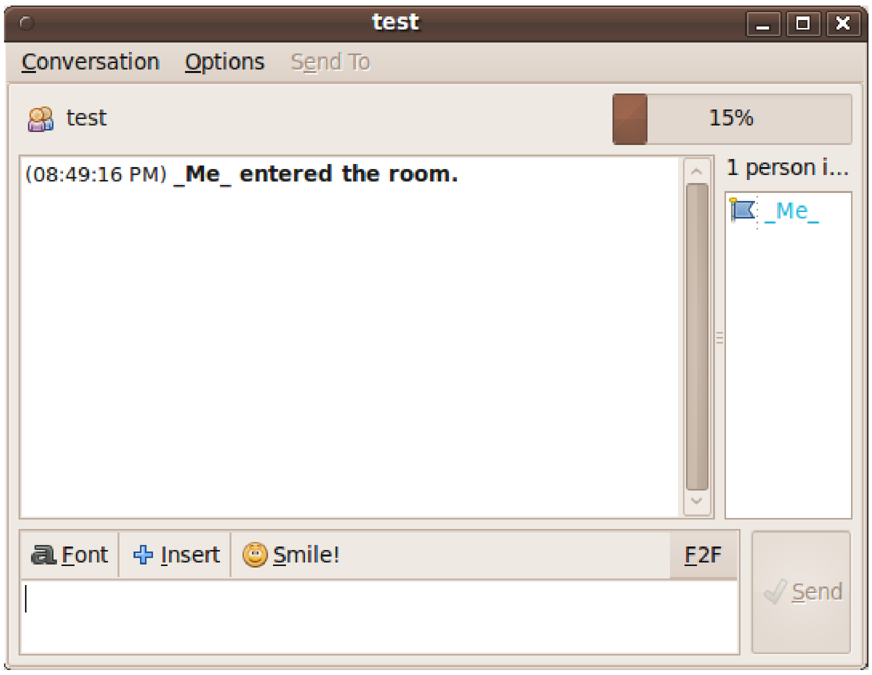
Figure 3.6: Infopane

Proposed change	Description	Type of change
F2F Computing menu	Small menu on the chat window toolbar for accessing F2F Computing features	User interface
Progress bar	On the information pane showing current tasks progress if it can be represented in numeric values	User interface
Configuration interface	User can configure plug-in and group level settings for F2F computing	Configuration
User information	F2F Computing specific information for “Get User info”	User information
Group information	Computer information about all the F2F chat members	User information/user interface
Plug in activities information	A simple log window that shows F2F information - network activity and core log	User interface

Table 3.1: Suggested changes

3.7 Roundup

Overall suggested changes would make using plug-in more user friendly. Program that has really good user interface is often preferred to another that might have much better functionality, but has not as good user interface. Nowadays, users value their time and do not like searching through menus for desired features and this is the motivation for these improvements.



(08:49:16 PM) **_Me_ entered the room.**

Figure 3.7: Chat window with proposed changes

Chapter 4

Chat window and user information improvements

In this chapter, I describe actual improvements made to the chat window, which is the main user interface for the plug-in, and changes to user information and its representation.

4.1 F2F menu on chat toolbar

On top of the F2F Chat is a menu (see Figure 4.1). Locating the "Submit Service"-option is not obvious, as it is located in the "Conversation / More" sub menu. Therefore, the F2F menu on the chat window toolbar (see Figure 3.5) would simplify access to plug-in features.

The menu contains the following elements:

- Submit Service – this menu item is for submitting a new task to the current group. Only the group owner is allowed to submit new tasks.
- Plug-in Preferences – for opening the preferences window, the content of the preferences dialog is described in Section 5.1.1.
- Group Preferences – opens the dialog where the user can adjust group specific settings, described in Section 5.1.2.
- Get Group Info – opens table where all contacts are shown with their user information, shared via F2F Computing. More details in 4.3.

The most complicated part of implementing these menus was making "Submit Service" not available when the user is not allowed to use it and making Submit Service work in toolbar "F2F" menu.

The user has to fulfill several requirements to be able to submit an F2F Service. User has to be owner of the group and no previous Service can be active, which are easiest to check after the user has clicked on the menu item, to be able to submit a service. Owner and participants of the chat can not be determined while loading the user interface, therefore limiting the usage of "Submit Service" was done with checks after clicking on the menu item. If the user has not fulfilled requirements a notification is shown and submission will be canceled.

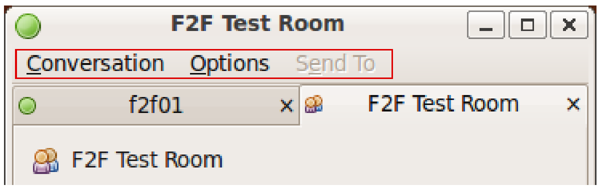


Figure 4.1: Chat window menu

Submit Service from "Conversation / More.. / Submit Service" adds the current PurpleConversation as a parameter to the target function by default, as from the " F2F" menu on toolbar "Submit Service" did not. To pass this argument via data pointer, the `f2f_job_submit_blist_menu_item`, that handles submit action, had to be modified slightly. When submitting is done from the "Conversation / More.." menu, this pointer is NULL and when using "F2F" menu, it is a pointer to the required data. An "if" condition checks whether this pointer is NULL, and if not, extracts the required data.

4.2 Progress bar

As there was no way to track the task progress and the easiest to understand option is a progress bar, I chose it for the basic feedback. In Pidgin, the chat info is placed on the Info pane(Figure 3.6). This is the part of the chat window where the chat name is located and i.e. where in the MSN chat contact photos occur.

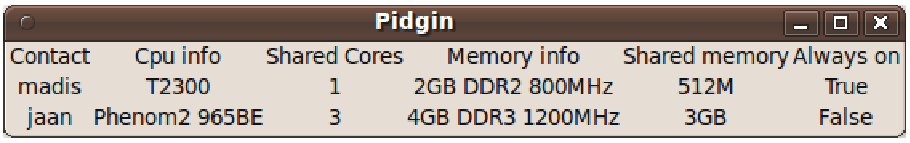
For the progress bar I used a `GtkWidget` Progress Bar. This widget takes numeric values from 0 to 1 and represents completion level, where 0 corresponds to 0% and 1 to 100%. There is also the possibility to animate the progress bar using its pulse function, that moves a small part, about 20% of the progress bar size, back and forth on it.

Each chat has its own progress bar, that is identified by the chat id. I created a new structure, that consists of the progress bar widget, chat id, completion level numeric value and text shown on progress bar. These variables are required for the case that chat window is closed while the task is in progress and they store progress bar values in memory until chat is reopened or Pidgin is closed. A new instance of the structure is created when chat is opened for the first time and default values are given to variables – 0.0 and "No Task". While a job is in progress text value shows percents between 1 to 99 and when it hits 100%, text will show "Complete!". The progress bar can be updates even if it is disabled in the preferences, so if the service is started while the progress bar was disabled and is later enabled from preferences, it will show the progress from the beginning of the task.

There are two functions to modify progress bar value, one is to increase it by a value and the other sets it to the value that is passed via an argument. A guide for using progress bar is in Chapter 6.

4.3 Table of computer information

This table is created for accessing computer information of all the group members at once. The table contains the same info that a user can modify in the preferences dialog, shown in Figure 5.2 and described in the next chapter. To open this window choose "Get Group Info" from "F2F" menu.



Contact	Cpu info	Shared Cores	Memory info	Shared memory	Always on
madis	T2300	1	2GB DDR2 800MHz	512M	True
jaan	Phenom2 965BE	3	4GB DDR3 1200MHz	3GB	False

Figure 4.2: Contact computer information table

The information table opens in a new window that is a Gtk Window. The table in it is a Gtk Table, that contains 6 columns: contact, cpu info, shared cores, memory info, shared memory and always on (Figure 4.2).

When "Get Group Info" is pressed `group_info_button_callback` is called, this function will call `create_info_table` and pass the group information as a list (structure of it is described in 6.4). `create_info_table` first creates the window and a new table in it, with 6 columns and number of group members plus one rows. After creating an empty table the header of the table is attached, this is static and can be modified in the `create_info_table` function. Finally for each group member a data row is inserted. If the header structure is modified, adding user data should be modified as well.

4.4 Roundup

Progress bar and F2F menu are the biggest visual changes to the chat window. From an average user's point of view, they are probably also the most important as the menu grants easy access to plug-in features and the progress bar gives easy to understand feedback. For implementing these changes the most important was understanding the structure of the chat window and usage of a few `GtkWidgets`.

Chapter 5

Configuration and debug interfaces

Before starting this thesis, the F2F Computing Pidgin plug-in had no configuration interface or support for it. Also all the log that was generated was sent to the Pidgin Debug window. If a user wanted to receive it, it could be done using a filter ("f2fprpl:"). However if several F2F groups were open at the same time, they all will correspond to the same filter, so there was no way to view the log only for one group.

As it was suggested in 3 the preferences user interface and debug window for each chat were implemented. The following chapter will describe the process.

5.1 Preferences

Configurability is a very important feature for any plug-in. F2F Computing is no exception, because it has features that some users might need and some will never use. Furthermore, it might use the user's computer for computational tasks and this should be configurable. The user should be able to choose the maximum resources available for use.

Preferences were divided into two groups – plug-in and group. Plug-in preferences are global or default values. For example, the amount of memory to be used – global preferences allow to use 1GB of memory, so the sum of memory used by all the groups is limited to 1GB.

5.1.1 Plug-in preferences

Plug-in level preferences are divided into three groups: user interface settings, computer info and shared resources like shown in Figure 5.1.

The user interface settings contain check box-type elements for enabling user interface elements: F2F Computing menu on the chat toolbar, progress bar on the info pane and debug (log) window for each chat. They can be disabled all at once by disabling "Show user interface elements". Each check box has its own `boolean` value, that is checked each time, that chat window is re-drawn. Re-draw of a chat window is automatic, when it is closed and reopened, or it can be called using `redraw_ui_elements` in "`f2f-ui.h`".

Gtk check buttons are used for the check boxes. They are put in a group, with main check box "Show plug-in ui elements", that disables all of the others. The `booleans` are stored in Pidgin "*prefs.xml*" in the "*plugins/f2f/ui/*" section, so they can be restored after restarting the program. All the check boxes have their preferences entry: "*show_all_elements*", "*show_toolbar_menu*", "*show_progressbar*" and "*show_debug_window*".

Computer info elements are CPU and RAM information and always on check box (whether the computer is always running or not, for example servers are always on) - these are global default values. Gtk entry elements are used for text input areas and gtk check button for check box. Each of these has its own entry in preferences file in "*plugins/f2f/info*" section: "*cpu*", "*ram*" and "*alwayson*". The first two of them are `string` values and always on is a `boolean`.

In the shared resources section there are two sliders that can be adjusted to desired levels. The amount of maximum cores and memory are automatically detected from the system. The minimum number of processor cores is set to 1 and is the default value as well. Minimum memory to be shared is 16 megabytes, the default value is 256 megabytes. The sliders are Gtk adjustment objects, they require several arguments like minimum, maximum, current and step increment values. For cores the increment value is 1 and for memory 16 megabytes - these values are set in function `computer_preferences_frame`. The current values are read from the preferences file.

The most difficult thing about the preferences was storing the settings and detecting the number of cores and amount of memory.

Like briefly described before, Pidgin stores its own and plug-ins preferences in the "*prefs.xml*" file. This xml file has three main subsections "*purple*", "*plugins*" and "*pidgin*" and each has its own sub elements. For example "*plugins*" has subsections for "*core*", "*prpl*" and "*gtk*" plug-ins. Accessing these preferences and modifying them is described in Chapter 6.

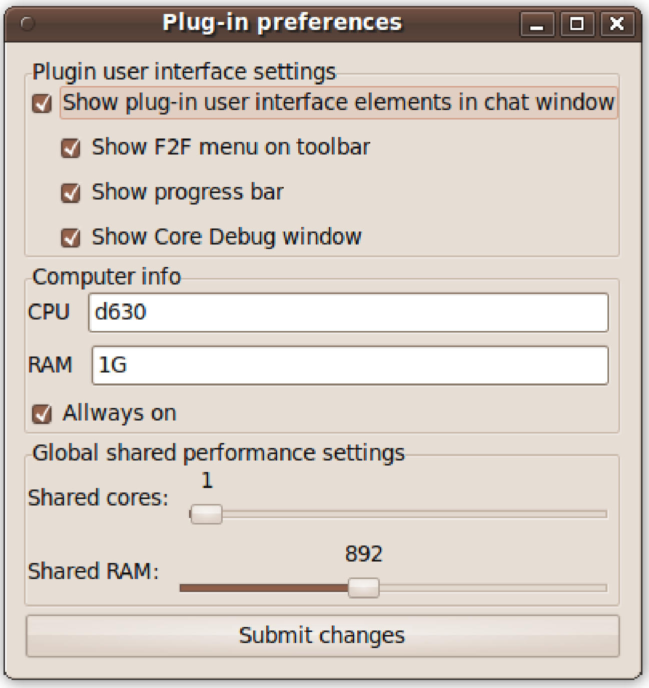
Detecting the number of cores and amount of memory is a more complicated matter, as each operating system has its own structure and access to system information might be limited. At the moment automatic detection is implemented only for Linux operating systems. This is done by using `sysconf`, that returns configurable system variables. To use `sysconf` "`unistd.h`" is required and the syntax to use it is `sysconf(int` ***name***`)` and it returns value as a `long`. To get processor count `int` ***name*** is "*_SC_NPROCESSORS_ONLN*". Detecting amount of memory is done by requesting number of physical memory pages ("*_SC_PHYS_PAGES*") and size of one memory page ("*_SC_PAGESIZE*"). Multiplying these values gives the amount of system memory in bytes, that can be calculated to megabytes that are shown in preferences.

5.1.2 Group preferences

Group preferences are similar to plug-in preferences however they only apply to the group from where a user selected "Group preferences".

Group preferences are divided into four sections: nickname, computer info, group shared resources and user interface settings. As suggested in the analysis there is a change of the nickname possible for each group, but the core does not support it at the moment. The computer info is duplicated because a user might want to set more detailed information for some groups.

Shared resources contain two sliders that get their maximum value from the plug-in preferences value of shared cores and shared memory. For example, let us assume that the computer of a specific user has 4 cores and 4 gigabytes of memory. He sets the plug-in preferences to 2 cores and 1 gigabyte of memory to be shared. Now the


Plug-in preferences

Plugin user interface settings

- [✓] Show plug-in user interface elements in chat window
  - [✓] Show F2F menu on toolbar
  - [✓] Show progress bar
  - [✓] Show Core Debug window

Computer info

CPU: d630

RAM: 1G

- [✓] Allways on

Global shared performance settings

Shared cores: 1

Shared RAM: 892

Submit changes

Figure 5.1: Plug-in preferences dialog

maximum values for each group is 2 cores and 1 gigabyte of RAM. However if several groups are active the combined total value can not exceed the maximum values set in plug-in preferences. Default values are set to half of the available resources.

User interface settings perform the same way as the plug-in level settings and they inherit their default values from the global settings.

Just like the plug-in level preferences, the most difficult part was storing the settings. However the solution was more complex, as the group ceases to exist after participants leave it, so the preferences should not be stored on the system. That is why I created a new structure where I store the group settings while Pidgin is running and the group is active. The instance of this structure is created while the chat is opened for the first time. It is then associated to this group by its ID, so it can be identified later.

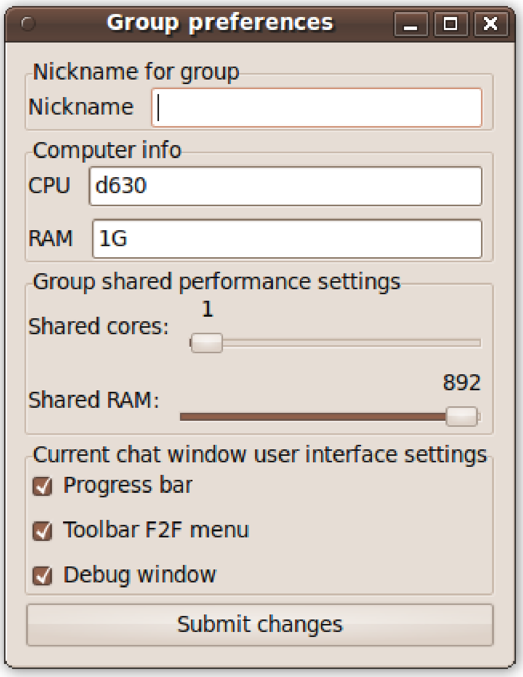
The group preferences structure consists of `char` values where text values of preferences are stored, `booleans` for check box preferences, and `doubles` for slider values.

5.2 Core debug

For more advanced users just a progress bar might not be enough to track the progress of a task, the F2F core, or plug-in activity and Pidgin Debug is a bit inconvenient for plug-in log. For that purpose, I created a new window which displays only F2F Computing specific information, for example when the task file is transmitted between clients.

This window is completely stand-alone and is not dependent on chat window or its parts. It is created using scrolled window and text view GTK widgets. The user can enable or disable this window in the preferences dialog described in the last section. Closing the window will also disable it for that group.

The core debugger is created at the first loading of the chat, just like group prefer-


Group preferences

Nickname for group

Nickname

Computer info

CPU: d630

RAM: 1G

Group shared performance settings

Shared cores: 1

Shared RAM: 892

Current chat window user interface settings

[✓] Progress bar

[✓] Toolbar F2F menu

[✓] Debug window

Submit changes

Figure 5.2: Group preferences dialog

ences. It starts logging automatically to a buffer even if the debug window is disabled from the preferences. A separate debugger is created for each group and can be disabled for one group and enabled for another.

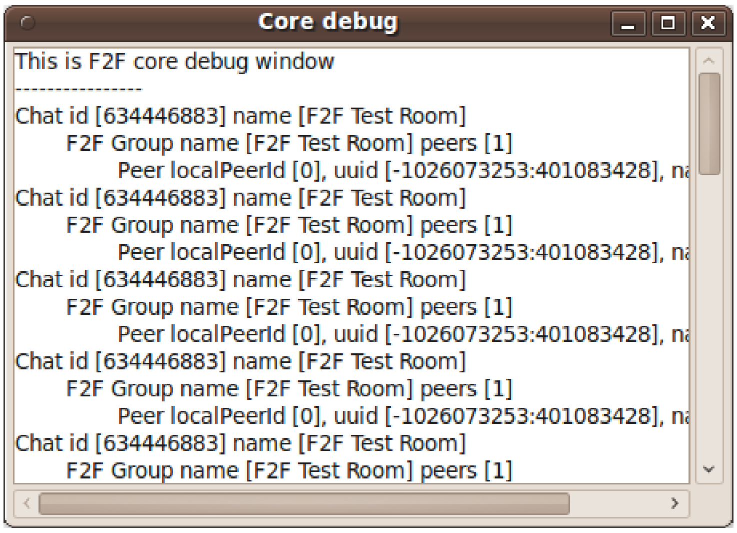
Core debug has its own structure that has the required graphic elements to show the window to the user and text buffer where all the log is appended. Also it has a `boolean` value to set its window visible status. Information about usage of the debug window is in Chapter 6.

The debug window is shown in Figure 5.3

5.3 Roundup

Although all features implemented in this chapter do not yet have support by core, like adjusting usage of cores and memory, they were created for the future use. However, the Core Debug window is fully functioning and can be used to output detailed information about plug-in actions. The only required argument to do so, is the local chat id.

Implementation of these features was a bit more complex as to those in the previous chapter. Group preferences and debugger buffer had to be stored in the memory while a group is active, and plug-in preferences are kept in the preferences file of Pidgin.



```
This is F2F core debug window
----------------
Chat id [634446883] name [F2F Test Room]
        F2F Group name [F2F Test Room] peers [1]
                Peer localPeerId [0], uuid [-1026073253:401083428], na
Chat id [634446883] name [F2F Test Room]
        F2F Group name [F2F Test Room] peers [1]
                Peer localPeerId [0], uuid [-1026073253:401083428], na
Chat id [634446883] name [F2F Test Room]
        F2F Group name [F2F Test Room] peers [1]
                Peer localPeerId [0], uuid [-1026073253:401083428], na
Chat id [634446883] name [F2F Test Room]
        F2F Group name [F2F Test Room] peers [1]
                Peer localPeerId [0], uuid [-1026073253:401083428], na
Chat id [634446883] name [F2F Test Room]
        F2F Group name [F2F Test Room] peers [1]
```

Figure 5.3: Core Debug dialog (contains only core synchronization messages).

Chapter 6

Usage of implemented functionality

The user interface elements need several libraries installed and included in build scripts before it can be ran. The list of the required libraries is: `gtk-2.0`, `atk-1.0`, `cairo`, `pango-1.0`, `pixman-1`, `freetype2`, `directfb`, `libpng12`, `pidgin`, `glib-2.0` and `libpurple`. The last two are required by base system as well. Most libraries are included because of the dependencies in `gtk-2.0`.

The source code of the implemented functionality is available at F2F Computing Git repository[3].

6.1 Progress bar

`void f2f_progress_bar_set_value(int` ***`chat_id`***`, gdouble` ***`value`***`)` - this function sets the progress bar completion level between 0 and 1. Argument ***`chat_id`*** is local chat id and value is `double` ***`value`*** in range [0..1]. In case ***`value`*** is greater than 1 it will be set to 1 and progress bar will show message "Complete!". Example of usage: `f2f_progress_bar_set_value`(634446883, 0.37); this sets progress bar value for chat 634446883 to 37%.

`void f2f_progress_bar_increase_value(int` ***`chat_id`***`, gdouble` ***`increase`***`)` - this function increases completion level of a chat by a value given in argument ***`increase`*** . If ***`increase`*** or previous completion level with ***`increase`*** exceeds 1.0, then progress will be set to 1.0 and text will show "Complete!". Example of usage(previous completion level was 58% (0.58)): `f2f_progress_bar_increase_value`(634446883, 0.02); this will increase progress bar value from 58% to 60%.

6.2 Chat Debugger

`void debugger_write(int` ***`chat_id`***`, char*` ***`text`***`)` - this function appends ***`text`*** to text buffer of the chat First argument is chats local id, second is the text to write to the buffer of the debugger. Example of usage: DEBUGGER_WRITE(634446883, "Friend-to-Friend Computing is cool!"); it will write text given text to chats debug buffer and if debug window is enabled will appear.

6.3 Configuration

6.3.1 Global preferences

Global preferences are accessible anywhere in code, as communication to preferences file is built into libpurple. There are three types of data used by global preferences in "*prefs.xml*" - `booleans`, `integers` and `strings`.

`purple_prefs_get_[string/bool/int]` (`const char*` ***name***) - this command will return the value of preferences entry.

`purple_prefs_set_[string/bool/int]` (`const char*` ***name***, `const char*/int/bool` ***value***) - sets preference entry value.

Here follows the list of names of preferences and explanations what they represent:

"*/plugins/f2f/ui/show_all_elements*" - `boolean` to enable/disable all user interface elements in chat window. Default: True;

"*/plugins/f2f/ui/show_toolbar_menu*" - `boolean`, enables/disables F2F menu on toolbar. Default: True;

"*/plugins/f2f/ui/show_progressbar*" - `boolean`, enables/disables progress bar on infopane. Default: True;

"*/plugins/f2f/ui/show_debug_window*" - `boolean`, enables/disables core debug window. Default: False;

"*/plugins/f2f/prefs/cores*" - `integer`, number of shared cores. Default: 1;

"*/plugins/f2f/prefs/ram*" - `integer`, amount of shared memory in megabytes. Default: 256;

"*/plugins/f2f/info/cpu*" - `string`, processor information in text form - speed, model. Default: -;

"*/plugins/f2f/info/ram*" - `string`, memory information - total amount, speed, type. Default: -;

"*/plugins/f2f/info/always_on*" - `boolean`, if computer is never turned of, then true. Default: False;

6.3.2 Group preferences

Group preferences consist of two custom structures - `ComputerSettings` and `ComputerInfo`. `ComputerInfo` is used in `ComputerSetting` structure.

ComputerInfo - variables that contain important information are `gdoubles` ***cores*** and ***memory***, `const chars` ****cpu_text*** and ****memory_text***. ***cores*** is the number of cores allowed to use by the group, ***memory*** is the amount of memory shared to group. ***cpu_text*** is the text form information about processor and ***memory_text*** is the memory information.

ComputerSettings - contains `const char*` *`nickname`*, gbooleans *`progress_bar`*, *`toolbar_menu`*, *`debug_window`*, `ComputerInfo*` *`info_for_group`*. *`nickname`* is the alias for the group (currently not supported), `booleans` are for user interface elements as their names suggest.

To get these preferences for group the function to call is:

`GroupSettings* get_settings_by_chat_id(int` *`chat_id`*`)` - returns preferences for group with given id.

6.4 Table of computer information

`void create_info_table(GList` *`*data`*`)` - displays a new window that contains a table containing data in `GList`. Each `GList` element is a `ComputerSettings` instance that holds users information for certain group.

Summary

Friend-to-Friend Computing is created with a principle to be easy to set up and use. F2F Computing Pidgin plug-in is one of the possibilities to use Friend-to-Friend Computing. In this thesis I analyzed the plug-in that already existed, discussed some new features, that would make using the plug-in even more user friendly., and implemented several of those.

The changes to the main user interface, the chat window, are quite simple, but in my opinion make a big difference in terms of usability. The F2F menu on the toolbar, makes access to other plug-in features easier. Implementation of this menu requires understanding of of Gtk[5] widgets and Pidgin chat windows structure. This menu can also be extended with ease if plug-in gets new features. The second user interface change to chat window was the adding of the progress bar. The purpose of this element is to give feedback about completion while running computational tasks in terms of a visual progress which can be represented as a percentage. While running other types of services, for example games, it can be disabled, as it has no purpose then.

Another implemented element is the configuration interface of the plug-in. The configuration is divided into two sections – plug-in and group configuration. In plug-in configuration a user can set so called global values to limit the total resources used by plug-in. Group preferences does the same for each group, only the values must stay in the limits of global values. However these limitations are one of the things that are not supported by the current F2F Computing core. Configuration interfaces also contain user information to be shared with other contacts and controls to toggle user interface elements. The implementation of the preferences was challenging, as the settings have to be stored, so they can be used at request. Some of these preferences are stored in the file system, so that they are accessible even after restarting Pidgin, others are kept in the memory while program is running.

A feature useful for all users is the "Core Debug window", that is separate for each group and contains only plug-in specific data. It was created to replace the Pidgins debug interface, where all the log is shown, but if several groups are active, it is quite difficult to follow. The created debugger can display any data as long as it is representable as text.

Last, I implemented the table of computer information shared by all the contacts in a F2F group. This has the most use for computation tasks in a sense, that group owner can check hardware used by group members before submitting service.

All these changes make a significant difference in the user experience even at the moment, when not all of the features are in working order and require further development of the core. Development of the interface improvements was an interesting experience, as Pidgin is an open source software and anyone can create plug-ins for it. However, it is not very well documented and there is a lot of testing and guessing, what does what. Pidgin documentation aside, it was a great hands on experience of

C/C++ programming and the result was a more user friendly version of the Friend to Friend Computing Pidgin plug-in.

The source code produced during implementation is integrated into the Friend-to-Friend Computing Git repository[3].

F2F Computing Pidgini laienduse kasutamismugavuse parandamine

Bakalaureusetöö (6EAP)

Madis Kapsi

Resümee

F2F Computing ehk sõprusraalimine on loodud eesmärgiga, et hõlbustada võrkraalimist (Grid computing). Sõprusraalimine ühendab võrkraalimise kiirsuhtlus süsteemidega. Üks selline lahendus on sõprusraalimise Pidgini laiendus, mille puhul on võrkraalimine ühendatud erinevaid kiirsuhtlus süsteeme (MSN, ICQ, IRC ja mitmed teised) toetava Pidginiga.

Minu bakalaureusetöö eesmärk oli antud laienduse kasutajaliidese analüüs ja edasiarendus. Analüüsi ja edasiarendamise juures pidasin silmas, et tulemust saaks kasutada igaüks, kes on oskab Pidginit kasutada ja sinna laiendusi lisada, misläbi hõlbustada veelgi võrkraalimist kasutaja jaoks.

Esialgne rakendus oli loodud kasutades Pidgini enda sisseehitatud võimalusi kasutajaliidese kuvamiseks ja eraldiseisev osa kasutajaliidesel puudus. Analüüsi käigus pakkusin välja erinevad võimalused arendada kasutajaliidest selliselt, et see võimaldaks ka häälestamist ja parandaks kasutajale antavat tagasisidet toimuva kohta.

Suurimad visuaalsed muudatused, mis on kohe nähtavad vestlusakna avamisel on lisandunud F2F menüü vestluse tööriistade juures (kirjastiili ja suuruse muutmine, faili saatmine) ja edenemisriba akna ülaservas, kus MSN-i vestluse korral asub kontakti pilt. Loodud menüü võimaldab ligipääsu sõprusraalimise põhisele funktsionaalsusele. Edenemisriba annab kasutajale tagasisidet käimasoleva protsessi edenemisest, kui see on esitletav arvulisel kujul.

Kasutajaliidese uus osa on iga F2F grupi enda logi aken, mis on loodud asendamaks Pidgini enda logi, kuhu saadetakse kogu Pidgini ja tema laienduste logi, mille tõttu on sealt vajaliku info jälgimine raskendatud. Samuti on täiesti uus grupi teabe vaatamise aken, kus on arvuti ja arvutusjõu jagamise põhine informatsioon grupi liikmete kohta. Eelkõige on see loodud silmas pidades võrkraalimise arvutusjõu jagamise osa.

Kasutamismugavust silmas pidades on äärmiselt oluline ka antud laienduse häälestamise võimalus, mille kasutajaliidese poolne tugi sai ka realiseeritud. Häälestusliidesest on võimalik piirata ressursside jagamist ja kontaktidele näidatavat informatsiooni muuta. See on kõige olulisem osa kasutajaliidese arendustest, millel puudub sõprusraalimise tuuma poolne tugi ja sai loodud edasiarenduste toetuse eesmärgil.

Oma töös arendasin kasutamismugavust silmas pidades Pidgini laienduse kasutajaliidest sõprusraalimiseks. Kirjeldatud on erinevate komponentide loomist ja nende kasutamisvõimalusi. Edasise arengu käigus vajab kindlasti Pidgini laiendus taas uusi muudatusi, kuid praeguse rakenduse kasutajaliidese parandused on sellisel juhul heaks alguspunktiks, millele tuginedes järjekordseid täiustusi läbi viia.

Bibliography

[1] Adium. http://www.adium.im/.

[2] Empathy. http://live.gnome.org/Empathy.

[3] F2f computing git repository. http://git.ulno.net/cgi-bin/gitweb.cgi?p=f2f.git.

[4] Friend-to-friend (f2f) computing. http://ulno.net/f2f/.

[5] Gtk. http://www.gtk.org/.

[6] Im user base. http://en.wikipedia.org/wiki/Instant_messaging#User_base.

[7] Instant messenger clients. http://en.wikipedia.org/wiki/Comparison_of_instant_messaging_cl

[8] Libpurple. http://developer.pidgin.im/wiki/WhatIsLibpurple.

[9] Napster. http://en.wikipedia.org/wiki/Napster.

[10] Peer-to-peer. http://en.wikipedia.org/wiki/Peertopeer.

[11] Pidgin. http://www.pidgin.im.

[12] Sip communicator. http://sip-communicator.org/.

[13] Skype. http://www.skype.com.

[14] Skype api. https://developer.skype.com/Docs.

[15] Telepathy. http://telepathy.freedesktop.org/.

[16] Using pidgin. http://developer.pidgin.im/wiki/Using

[17] Webkit. http://trac.adium.im/wiki/WebKit.

[18] Sven Kirsimäe. F2f mobile computing. Master's thesis, University Of Tartu, 2009.

[19] Keio Kraaner. Friend-to-friend computing. Master's thesis, University Of Tartu, 2008.

[20] Eero Vainikko Oleg Batrašev Ulrich Norbisrath, Keio Kraaner. Friend-to-friend computing - instant messaging based spontaneous desktop grid. *The Third International Conference on Internet and Web Applications and Services (ICIW 2008)*, pages pp. 245–256, June 2008. Athens/Greece, IEEE Computer Society Press.