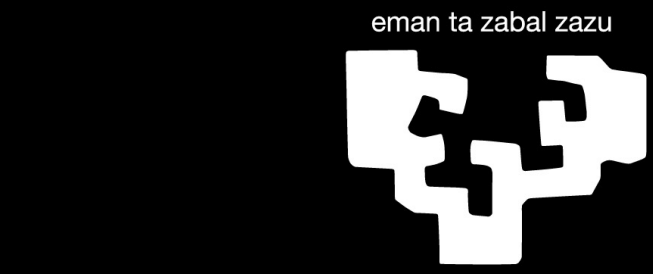


Universidad del País Vasco

Euskal Herriko Unibertsitatea

Kharon

A self-hosted, open source and lightweight directory synchronization application.

Mikel Alejo Barcina Ribera
Student in Computer Science degree
mabarcina001@ikasle.ehu.eus

To Felix Ángel and Manuela.

Acknowledgements

To family, friends and directors, the gratitude I feel towards you is too great to be expressed in such few words. I hope that in these few lines I could show you my appreciation and indebtedness.

Thank you, family and friends —who I also consider as family—, for always being supportive and encouraging me to keep going, even in the hardest of times.

Thank you, dear directors, for making my journey in the university worth it, and in this last year, for believing in me and my project.

And last but not the least, my deepest thanks, to my beloved parents, for absolutely everything that I am and all that I have. You have always been there for me. I will be eternally grateful.

Abstract

The objective of this project is to develop an open source application, which allows synchronizing a directory in real time, between a client and a server. *Kharon* has been built using cutting edge libraries and tools, such as *Qt* for the graphical user interface, *Boost* libraries for networking, concurrency and function composition, *Inotify* for file monitoring or *Librsync* for comparing and updating files remotely, among others. The result of combining these libraries is *Kharon*, an application that not only meets the aforementioned objective, but also runs on systems with very low resources.

Contents

List of Figures

1 Project charter

1.1 Introduction

This document is the report of the Undergraduate Project of name "Kharon", which has been carried out by Mikel Alejo Barcina Ribera, a student of the Faculty of Computer Science of the University of the Basque Country. Directed by Javier Dolado Cosín, and co-directed by Iñaki Morlán Santa Catalina.

1.2 The why

Initially, this was not meant to be a project. In fact, there was only just a simple question to be answered: *"How do I synchronize a directory and all its contents with my Raspberry Pi?®"*. The idea was to have a client running all the time caching every event in the monitored directory, and transmitting those events to the server. Although many solutions do exist[1], none of them seemed to suit my needs. Moreover, more questions arose:

1. Can the application be lightweight?
2. Can I have an application with a simple and intuitive Graphical User Interface (GUI)?
3. Can the application's options be tweaked in order to have it the way I like it?

After those questions, inevitably, some others came up:

1. Can the application have an advanced hidden menu, for more advanced users?
2. Can the application have a simple user system that allows them share files among themselves, with a permission system?
3. Can I run the application securely without laying on protocols like *SSH*?

Et cetera. Thus, instead of wondering about all this, a more deep research was carried out to find an application that would suit all the above requirements, and some more. Unfortunately, nothing convincing was found. The poor Raspberry Pi®—*1, Model B*—could not handle some of the solutions, and even less combine them with other services that were already running in its tiny but fierce processor.

Therefore, I thought about using this premises as my motivation to realize a software project. That way, I would be able not only to finish my degree doing something that inspires me, but also to contribute to the open source community, by releasing a software that may result useful for some others too.

[1] `https://en.wikipedia.org/wiki/Comparison_of_file_synchronization_software`

2 Project planning

2.1 Information system's definition

The information system for the project will be digitally held. The project will be saved in three main places: a personal computer, an external hard drive disk, and GitHub®—this last one will be used for the code—.

2.1.1 Directory structure

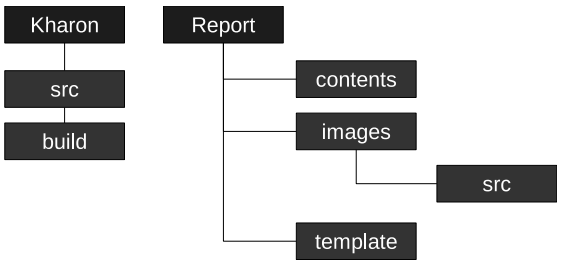


Figure 2.1: Directory structure of the information system

As shown in the Figure 2.1, the first level will contain two main directories:

- ***Kharon*** directory: the directory that contains the application's source code and binaries.
- ***Report*** directory: the directory containing the report of the project.

Inside the ***Kharon*** directory:

- ***src*** directory: it will contain all the source files of the application. It might be subdivided in many other directories, or even be substituted if rearranging the directory structure helps generating a cleaner source code.
- ***build*** directory: it will contain all the executable files.

Inside the ***Report*** directory:

- ***contents*** directory: it will contain the source *.tex* files and, their corresponding subdirectories —if any —, will be stored here.
- ***images*** directory: it will contain all the images used in the report.
- ***template*** directory: it will contain all the source files that are somehow related to the LaTeX template used for the report.

Inside the ***images*** directory, a ***src*** sub-directory will be used to store all the source files for the diagrams and images created in other programs.

2.1.2 Tools to be used in the documentation

The project's report will be done, mainly, using the following tools:

- LaTeX for the texts and the report.
- LibreOffice[1] Draw for the diagrams.
- GanttProject[2] for the Gantt charts.
- Umbrello[3] for the class diagrams.

2.1.3 Naming conventions for the report

The files will be named all in lower case, using underscores if there is any information that would help either organize the directory or add meaningful information about the file contents.

[1] `https://www.libreoffice.org/`
[2] `https://www.ganttproject.biz/`
[3] `https://umbrello.kde.org/`

2.2 Project objectives

The objective of the project is to:

- Develop a simple, open source application which allows synchronizing a directory between a client and a server.

The idea is to be able to create an application that allows synchronizing a directory and all its contents with a remote directory. Ideally, they would end up being a copy of each other. In the figure 2.2, there is a basic diagram of how the application is intended to work.

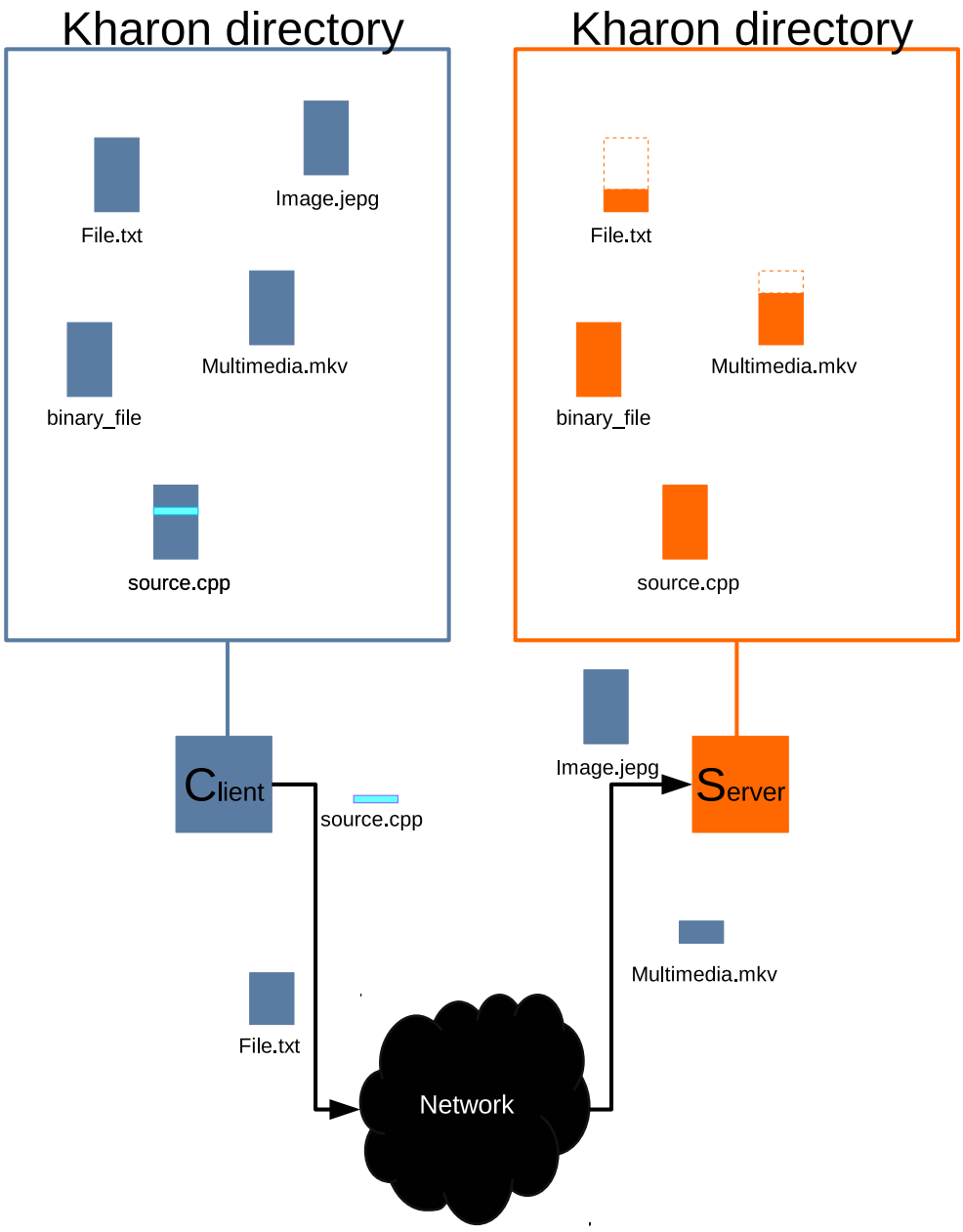


Figure 2.2: The application's general idea

In the diagram, we can observe that in the left side there is a little blue box which stands for the client application. That client application monitors everything that is going on inside the directory. In this case, there are only files represented, but there could be directories with subdirectories as well. Once an action is taken over a file or a directory —create, modify, rename, move, delete, etc. —those actions would be sent to the server in order to be replicated in the remote directory. In the case of having little changes, instead of submitting the whole file, those changes would be sent instead, in order to patch the file. Obviously if there is a whole file missing in the server side, the entire file would be transmitted.

The medium which would be used for transmitting the data is the network, either a local area network or a wide area network. If there were enough time to develop some other features, the link would be secured using symmetric key cryptography, and a two-way synchronization would be implemented —from the client to the server and vice versa —.

Summarizing, and as stated before, the main objective is: to develop a client-server application, with an easy to use GUI, which allows synchronizing in real time a specified directory.

2.3 Scope definition

The scope of this project is to meet the *Project objectives* —section 2.2—and reach to the minimum quality specified in the *Quality plan* —section 2.7—.

2.4 Work breakdown structure

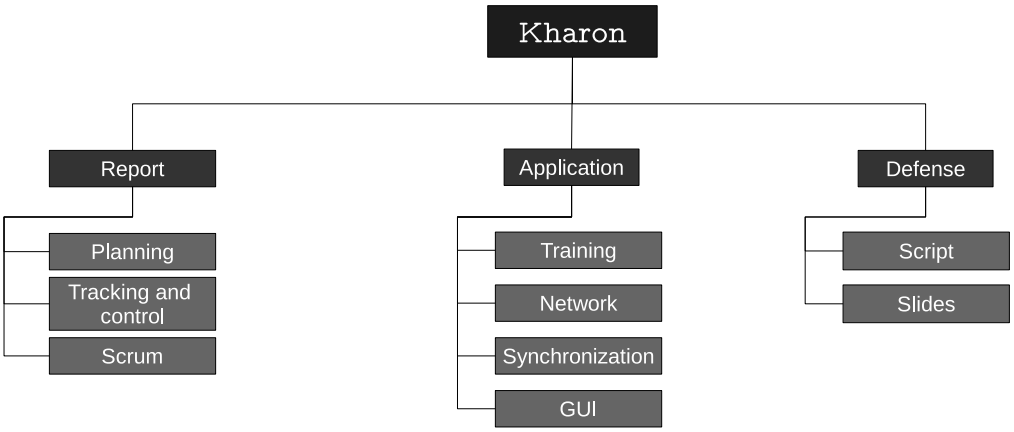
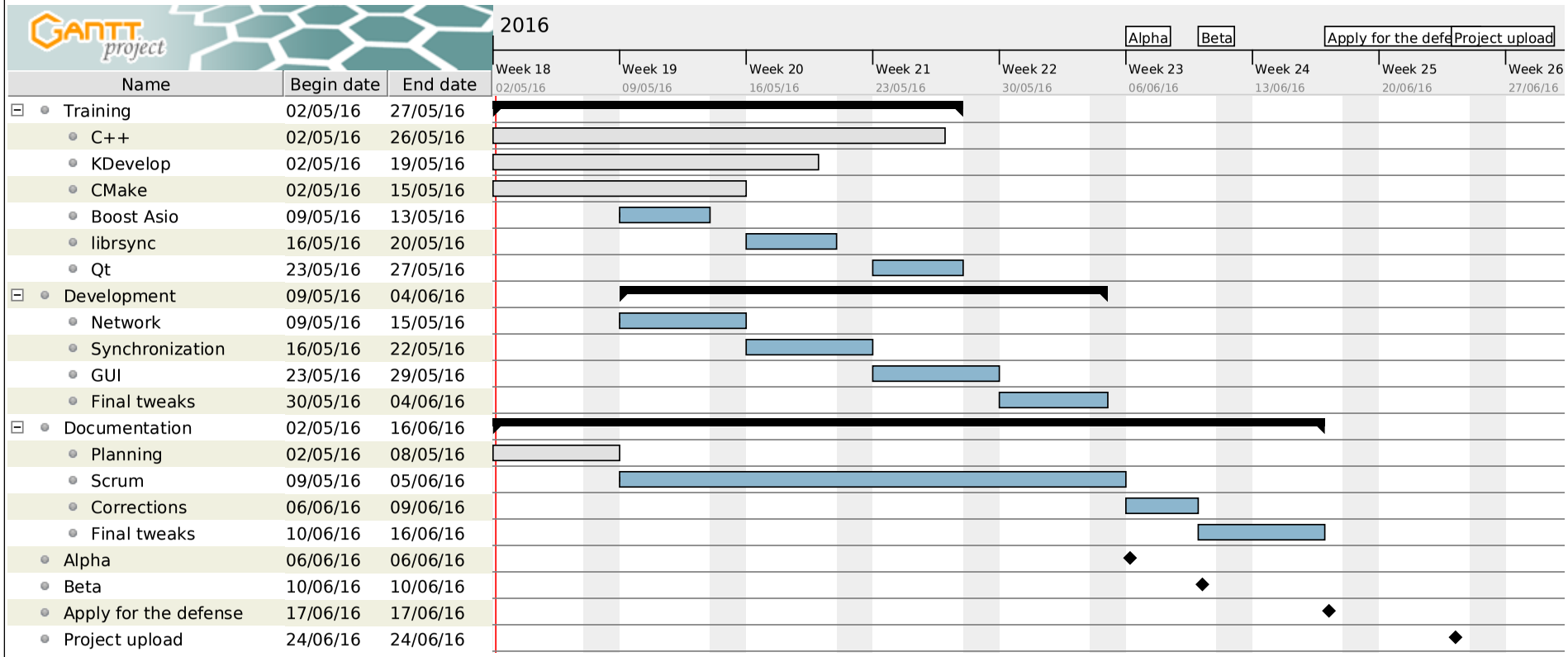


Figure 2.3: Project's work breakdown structure

2.5 Gantt chart

The planning is as follows: one week to have a first glance at the tools, libraries and technologies to be used in the project development, as well as to finish the planning of the project. Four weeks of development of the project, which will be combined with the formation on the different technologies. Two weeks to polish and complete the report. One final week to review the things done and deliver the project.



Name	Begin date	End date
Training	02/05/16	27/05/16
C++	02/05/16	26/05/16
KDevelop	02/05/16	19/05/16
CMake	02/05/16	15/05/16
Boost Asio	09/05/16	13/05/16
librsync	16/05/16	20/05/16
Qt	23/05/16	27/05/16
Development	09/05/16	04/06/16
Network	09/05/16	15/05/16
Synchronization	16/05/16	22/05/16
GUI	23/05/16	29/05/16
Final tweaks	30/05/16	04/06/16
Documentation	02/05/16	16/06/16
Planning	02/05/16	08/05/16
Scrum	09/05/16	05/06/16
Corrections	06/06/16	09/06/16
Final tweaks	10/06/16	16/06/16
Alpha	06/06/16	06/06/16
Beta	10/06/16	10/06/16
Apply for the defense	17/06/16	17/06/16
Project upload	24/06/16	24/06/16

2.6 Milestones

- Alpha version of the report: 2016-06-06.
- Beta version of the report: 2016-06-10.
- Apply for the defense of the project: 2016-06-17.
- Project upload or delivery: 2016-06-24.

2.7 Quality plan

2.7.1 Minimum quality

- **Shared directory:** The application must be able to monitor and synchronize in real time a directory between two systems, including all the files and subdirectories placed inside.
- **Intuitive GUI:** The application must provide an intuitive GUI that helps using the application, even for the more novice users.

2.7.2 Added value

- **End to end encryption:** The server and the client should be able to communicate securely over the network using end to end encryption, in order to provide confidentiality.
- **Multi-user system:** A user system could be implemented in order to allow more than one user utilize the application, and have his own personal shared directory.
- **Sharing system:** A sharing system could be implemented so as to allow users share files among them.
- **Customization:** The user could tweak the application's parameters using the GUI, and he could be given the option of choosing between a basic set of options and a more advanced one.
- **Traceability:** The end user should be able to keep traceability of every action done by the application.

2.8 Risk management plan

There are some risks that should be taken into account, and these are enlisted below. As well as the risks, a contingency plan is proposed in order deal with the identified risks, in case they end up happening.

2.8.1 Risk identification

- Data loss.

It is likely to happen some kind of data loss during the project life. An unconscious deletion or a hardware error can trigger this problem and make the project disappear in seconds. Therefore, this is considered a high risk.

- Not reaching minimum quality.

Even though the minimum requirements do not seem very demanding, there is a possibility of not reaching to them. The libraries to be used need to be studied and correctly integrated in order to assure the correct functioning of the application. The main risk is to end up changing the structure of the application on a regular basis, and not advancing into meeting the deadlines and project objectives.

This is considered a high risk.

2.8.2 Contingency plans

- Data loss.

In order to deal with this risk a simple yet effective solution is proposed: perform periodic backups of the project. The project itself will be saved in three different places —a personal computer, an external hard drive disk and an on line repository for the code —, but apart from those two dedicated USB keys will be used to store the backups of the project.

The backups must be made, at least, weekly, and at most, daily.

- Not reaching minimum quality.

So as to deal with this an agile methodology will be used in order to provide continuous stable releases of the application. In theory, this would help in readjusting the application's development objectives, while the development is taking place. Therefore, if something is taking too much time to be developed or improving it is impossible due to deadlines, it will be left like that, in order to work towards meeting the main objective of the project.

3 Technologies

3.1 Boost libraries

Boost is a set of libraries made by some of the greatest experts of *C++*, who at the same time are part of the *ISO C++ Committee*, which takes care of the standard releases of the programming language. These libraries include some useful resources that in the vast majority of the cases are not platform specific; this is, the library allows us to program once, as the library will take care of adapting our code to the operating system's specific system calls. Therefore, it makes it easier to develop multi platform applications.

Among those useful libraries we can find one that takes care of the file system's basic operations[1], another one that copes with threading [2], or even another library that deals with input-output operations of any kind[3]. Of course, there are many, many more[4].

3.1.1 Asio

Boost Asio takes care of the input-output operations of an operating system. It does not matter whether it is a file descriptor, a socket or any other type of stream or descriptor that we want to manage, *Asio* takes care of them all. However, platform-independent code is just the tip of the iceberg. The actual striking feature of this library is asynchrony. In a nutshell: it allows launching several asynchronous operations, and retrieving the results when the operations have been completed, without blocking the program. The following example is borrowed from the library's overview page [5]

[1] `http://www.boost.org/doc/libs/release/libs/filesystem/`

[2] `http://www.boost.org/doc/libs/release/libs/thread/`

[3] `http://www.boost.org/doc/libs/1_61_0/doc/html/boost_asio.html`

[4] `http://www.boost.org/doc/libs/`

[5] `http://www.boost.org/doc/libs/1_61_0/doc/html/boost_asio/overview/core/basics.html`

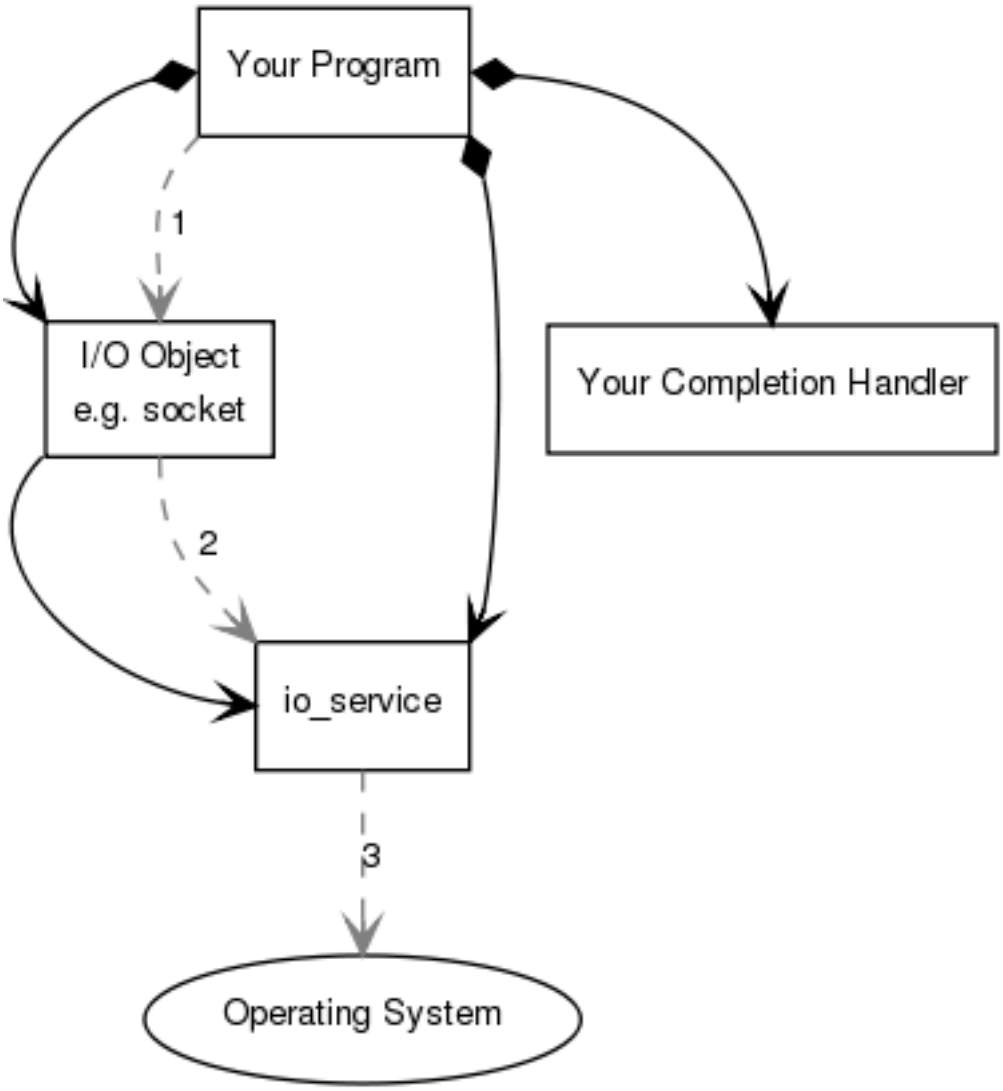


Figure 3.1: Diagram of the first steps of an asynchronous operation

The thing that makes different this asynchronous call from any other synchronous call is the **io_service**, and this is how it works —note: the *io_service* is nothing but an event processing loop—:

1. After creating the IO object, we call the asynchronous operation, which in this particular case is the *connect* operation.

```
socket.async_connect(endpoint, handler);
```

2. The IO call, will forward the request to the *io_service*.

3. The *io_service* immediately tells the operating system that it has to perform a connect operation.

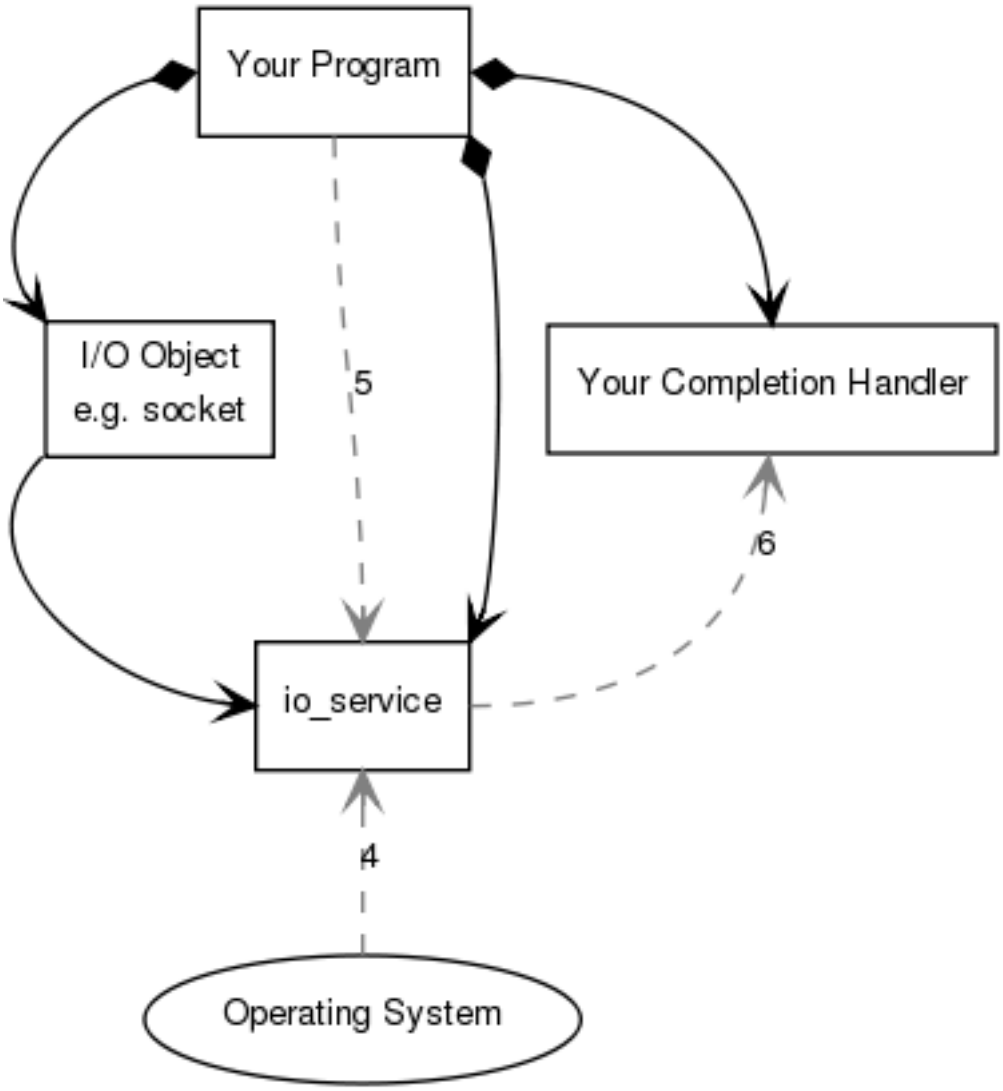


Figure 3.2: Diagram of the last steps of an asynchronous operation

4. The operating system places the results on a *io_service* queue.

5. The programmer calls the function *run* of the *io_service* in order to retrieve the results. Here, the main program flow will block until the *io_service* runs out of work. Once the handlers have been executed and return, the *io_service* *run* statement unblocks.

6. The *io_service* encodes the result in an *error_code* and dispatches the results to the corresponding handler. The handler, is just a function that is called when the asynchronous operation is over.

However, the implementation of this flow is platform specific. In the case of *GNU/Linux* systems, the asynchronous call is not made until the programmer calls *io_service*'s *run* function. Although the main functionality is still there: the thread

is not blocked when calling to the asynchronous functions. This is due to the underlying system calls —like *epoll*[6]—to provide this asynchrony, which have their limitations.

```
#include <iostream>
#include <boost/asio.hpp>

void connect_handler(const boost::system::error_code& error)
{
        if(error)
        {
                std::cout << error.message() << std::endl;
                exit(EXIT_FAILURE);
        }
        else
        {
                std::cout << "Successfully connected!" << std::
                    endl;
        }
}

int main()
{
        boost::asio::io_service io_service;
        boost::asio::ip::tcp::socket socket(io_service);
        boost::asio::ip::tcp::endpoint endpoint(
                boost::asio::ip::address::from_string("127.0.0.1
                    "), 9000);

        socket.async_connect(endpoint, connect_handler);

        io_service.run();
        exit(EXIT_SUCCESS);
}
```

The code above shows a simple program that attempts to connect to *localhost* or "*127.0.0.1*". It creates an *io_service* object, then, the actual *IO* object —the *socket*—is created, it defines the *endpoint* to connect to, and the *async_connect* receives as parameters the endpoint, and the handler or function that will take care of the following actions when the operation is complete. Finally, calling the *io_service.run()* function will launch the asynchronous connect operation and the main flow of the program will be blocked there. Once the connect operation completes, the *io_service* will execute the handler, and once the execution has finished and the handler returns, the *io_service run* statement unblocks and the main flow continues.

One last thing to take into account about the *io_service* is that once the pro-

[6]https://en.wikipedia.org/wiki/Epoll

grammer makes a call to the *run* function and the service finishes up all the work, if subsequent calls want to be made in order to give the service more work, it must be reseted with the *reset* function. One of the ways of avoiding this is to wrap the *io_service* with an *Work* object, which will make the service not return, and therefore that allows giving the service more and more work until the service is unwrapped from the *Work* class.

3.1.2 Thread

Boost Thread gives a platform independent way of implementing multi-threaded applications. It makes it easy for the programmer to create, manage and join threads. This is extremely useful to perform several tasks at the same time. Below there is a code snippet which tries to shed light on the concept:

```
#include <iostream>
#include <boost/asio.hpp>
#include <boost/bind.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

#include <boost/thread.hpp>

/** A mutex for blocking the standard output, because it is not
    thread safe.
 * This means that if all the threads try to use the standard
    output at the
 * same time some unexpected behaviour may occur.
 */
boost::mutex global_lock;

int sum(int a, int b, boost::asio::io_service* io_service)
{
        /** Lock the standard output and print the ID of the
            caller thread */
        global_lock.lock();
        std::cout << "Thread [" << boost::this_thread::get_id()
           << "]" << std::endl;
        global_lock.unlock();

        /** Create a deadline timer and wait five seconds */
        boost::asio::deadline_timer
                timer(*io_service, boost::posix_time::seconds(5))
                   ;

        timer.wait();

        /** Print the results */
        global_lock.lock();
```

```
        std::cout << "The result for [" << a << " + " << b << "]
            is: [" <<  (a + b)
                << "]" <<std::endl;
        global_lock.unlock();

        return (a + b);
}

int main(int argc, char* argv[])
{
        boost::asio::io_service io_service;

        /** Create threads one by one and assign them work */
        boost::thread thread_one(boost::bind(&sum, 2, 1, &
            io_service));
        boost::thread thread_two(boost::bind(&sum, 5, 5, &
            io_service));
        boost::thread thread_three(boost::bind(&sum, 21, 55, &
            io_service));

        io_service.run();

        /** Wait until all the threads to terminate */
        thread_one.join();
        thread_two.join();
        thread_three.join();

        io_service.stop();
        return(EXIT_SUCCESS);
}
```

In this example, three threads are manually created and given the work of performing a simple calculation. For the sake of example, those calculations last five seconds each. As the three created threads are concurrent, the program is expected to last around five seconds.

What if, this program used only the main thread instead of the concurrent threads? Well, there would only be one thread making those calculations, and therefore the same thread would be waiting five seconds first, another five seconds for the second call, and another five seconds for the third one. Therefore the program would be expected to last around fifteen seconds.

Note that in this example there might be racing conditions when accessing the standard output. The fastest thread locking the standard output will be the one getting the access to it, and even though the threads are fired in a certain order, this does not mean that the thread which was launched first will be the one getting the access first to every use of the standard output. Nonetheless *Boost Thread* provides ways of synchronizing that access, so as to set an order in which the threads should

access those functions[7]. However, this topic is out of the scope of the project, as it is not used in the application.

Finally, the example shows how useful the threads can be, and how easy it is to implement them in a platform independent way. *Boost Thread* takes care of translating the abstract code into operating system specific Application Programming Interface (API) calls.

3.1.3 Bind

Boost Bind states the following[1] as its purpose: "*boost::bind is a generalization of the standard functions std::bind1st and std::bind2nd. It supports arbitrary function objects, functions, function pointers, and member function pointers, and is able to bind any argument to a specific value or route input arguments into arbitrary positions.*"

This is too complicated to understand. Even more if you are told that this is used for partial function application[8]. Thus, below I attach a code snippet that might help understanding what this library is about:

```
#include <iostream>

#include <boost/bind.hpp>

int sum(int a, int b)
{
        return a + b;
}

int substract(int a, int b)
{
        return a - b;
}

void print_result(int result)
{
        std::cout << "The result is: " << result << std::endl;
}

int main(int argc, char* argv[])
{
        int number = 2;

        /** Call print_result(sum(number, 1)) */
        boost::bind(print_result, boost::bind(
                sum, number, _1
```

[7] http://www.boost.org/doc/libs/1_59_0_b1/doc/html/thread/synchronization.html
[8] https://en.wikipedia.org/wiki/Partial_application

```
        ))(1);

        /** Call print_result(substract(5, number)) */
        boost::bind(print_result, boost::bind(
                substract, _1, _2
        ))(5, number);

        /** Call print_result(sum(2,substract(2,1))) */
        boost::bind(print_result, boost::bind(
                sum, _2, boost::bind(
                        substract, _1, _2
                )
        ))(1, 2);

        return(EXIT_SUCCESS);
}
```

This is one of the uses of this library. As shown in the example, it allows us nesting calls to functions, which in certain situations might turn useful. Furthermore, *placholders* come in place —in the example the arguments of the functions are specified with _1 and _2 —, which allow us to bind the arguments in whichever order we want to, or even repeat the same argument if we specify the same placeholder.

So as to make it easier for the reader to understand the concept underlying, the values bound to the functions are simple values. Nonetheless bind allows us to make much more complex things with it.

In this particular application, bind essentially does "tell the *io_service* which handler has to execute". This is: after launching the asynchronous operations and having the *io_service* ready to return those values, we have to tell it which is going to be the function or the handler that will take care of that. A really simple example of how handlers work is included in section 3.1.1.

3.2 librsync

librsync is a library that allows calculating and patching the differences between two files over the network. This library contains the algorithms that the *rsync*[9] application uses. The good thing is, the two files do not need to be in the same file system in order to make comparisons and apply the differences between them. Basically, it works the following way:

[9] `https://en.wikipedia.org/wiki/Rsync`

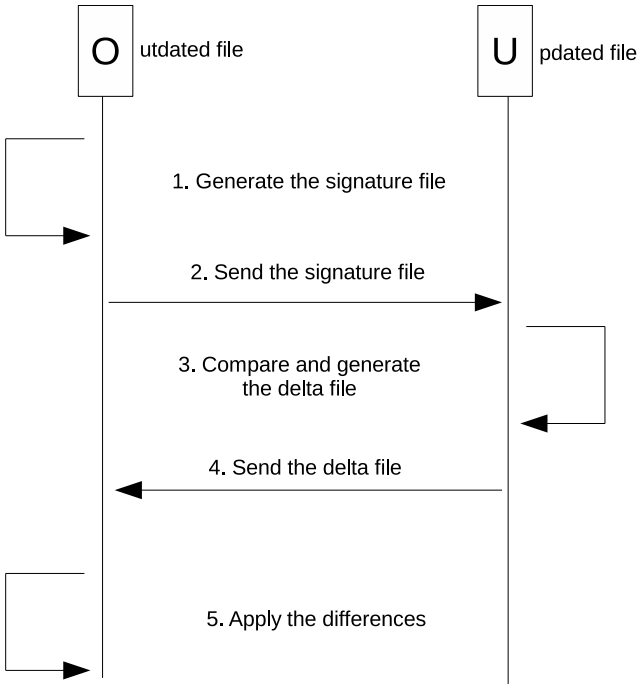


Figure 3.3: Flow diagram of the way *librsync* works

librsync generates a *signature file*—a set of checksums—from the outdated file. The *signature file* is sent afterwards to the other end, which makes use of it to make comparisons with the updated version of the file, and generate a delta file as an output —a file which contains the actual differences between the outdated and the updated file—. That *delta file* is transferred back to where the outdated file is, and with both files the differences are *"patched"*.

3.3 Scrum methodology

This methodology is considered one of the agile methodologies, and its aim is to release stable releases of the software in an enclosed period of time, called sprints. This way, it is possible to adapt to the clients needs and opinions, as the client may want to include or remove some of the functionalities the application provides.

There are three main roles in the *Scrum* methodology:

- **Product owner:** A person who acts as a bridge between the client and the *Scrum* team. This person will be responsible for including or removing user stories, and arranging them with priorities.
- **Scrum master:** this person is the one that will act as a supporter for the team, not as the project manager. The task of this person is to act as a coach, and help the team meeting the goals set. It is there to facilitate communication among the members of the team, and to ensure that the *Scrum* processes are followed correctly.

- **Development team:** the group of people responsible of carrying out all the planning, development, testing, *etc.*

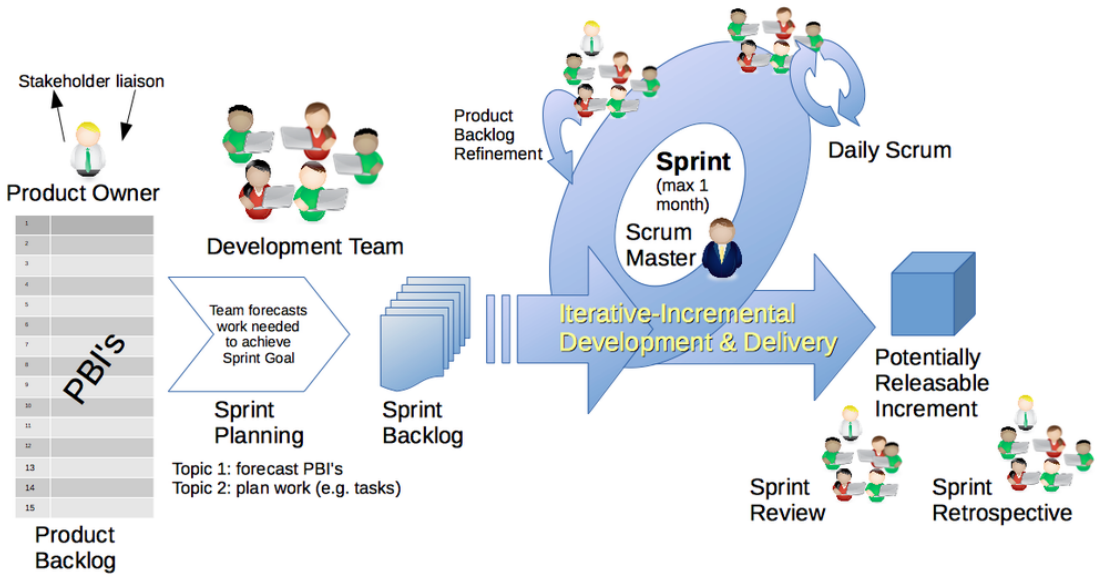


Figure 3.4: An schematic of the Scrum Framework process. *By Dr ian mitchell, CC BY-SA 4.0*

For a development to happen, first the product stakeholders or the product owners must specify the features to have in the application. Those features, are placed in a list which is called the *product backlog*. Then, the development team *plans* the sprint, taking some user stories from the *product backlog*, and making them part of the *sprint backlog*. The *development team* will have meetings everyday, called *daily scrum meetings*, and they will not last more than fifteen minutes. These meetings will help readjusting the sprint backlog in case that some user stories are taking longer to develop, and each team member must answer the following questions:

- What did I do yesterday that helped the development team meet the sprint goal?
- What will I do today to help the development team meet the sprint goal?
- Do I see any impediment that prevents me or the development team from meeting the sprint goal?

At the end of the sprint, the team must make a *sprint review*, in which the met —and unmet—goals are commented. Plus, the work done until that time is presented to the team members and, of course, the product owner. After this review, a *sprint retrospective* must be done, in which two main questions need to be answered:

- What went well during the sprint?
- What could be improved in the next sprint?

4 Kharon, the application

4.1 The client

4.1.1 A general idea of how it works

The client part is divided in three main classes —there are many more but they are secondary—:

- The **Kharon** class: dedicated to the user interface.
- The **Client** class: used for networking.
- The **Inotify** class: used for the file system events monitoring and processing.

A thread is created for each module, with the intention of making each class's work to be assigned to their corresponding thread. However, there is a little problem with this approach. When a class calls to the other class's member functions, the caller's thread takes care of executing all the statements of that function, and consequently, it ends up carrying out all the work. The figure 4.1 illustrates the problem.

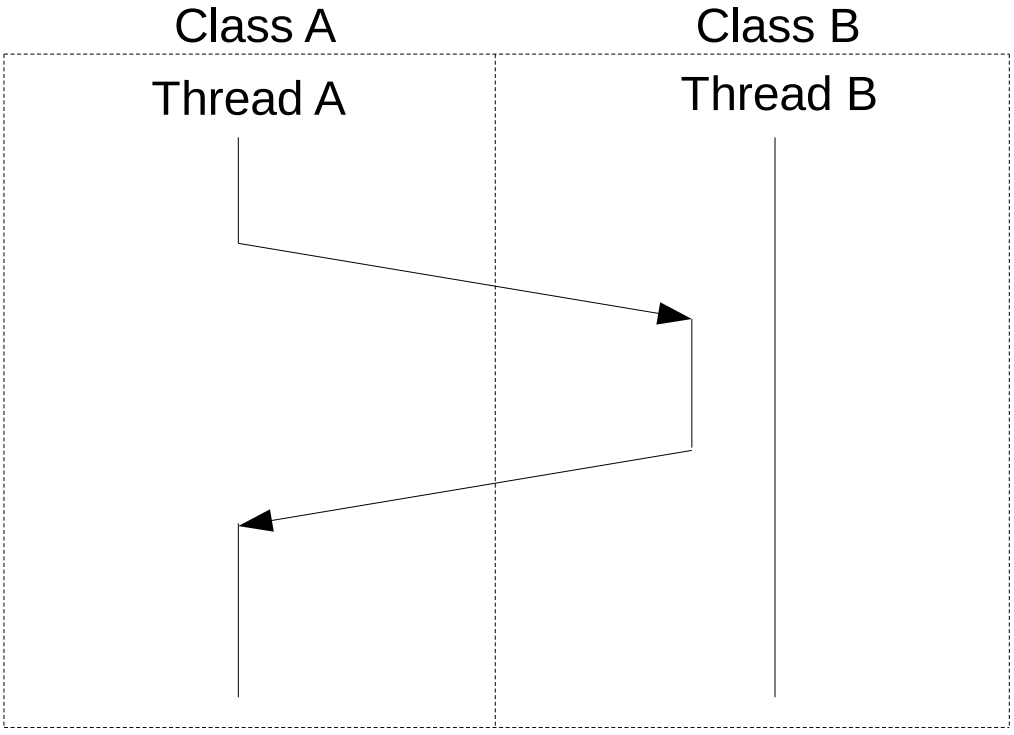


Figure 4.1: Illustration of the threading problem

The solution which has been found in order to overcome this problem, is to create little functions in the receiver class that assign work to the receiver's thread, and the consequence is that the caller's thread has only to execute an statement that performs that assignment. The figure 4.2 illustrates the solution.

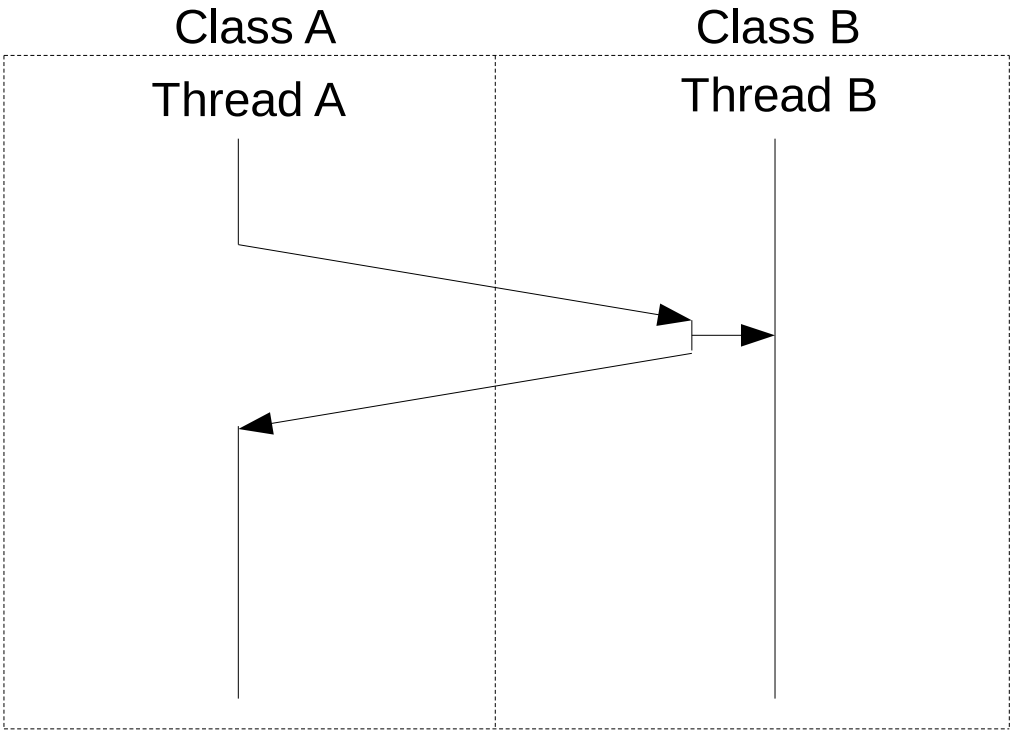


Figure 4.2: Illustration of the threading solution

The threads are of two kinds: there is one thread of the Qt kind, and the other two are of the *Boost* kind. This is important, because there is an exception in which one class's thread runs code from the other class: when the *Client* attempts to connect, it calls a member function from *Kharon* with the result of that attempt, in order to update some graphic elements of it. Here, instead of giving the task of updating the graphic elements according to the result to *Kharon*'s thread, the code is executed by the *Client*'s thread, breaking the "each thread stays in its scope" rule.

Finally, the client and the server communicate using a very simple protocol. That protocol makes use of a *Message* class —see 4.9 for the class diagram —, which separates each packet into two parts: the header and the body. On the one hand, the header includes the action to be taken and the size of the body. On the other hand, the body is the data. Basically, it is like a buffer that helps managing the packets. Here is the file containing the commands:

```
/*
 * This file is part of Kharon.
 *
```

```
 * Kharon is free software; you can redistribute it and/or
    modify
 * it under the terms of the GNU General Public License as
    published by
 * the Free Software Foundation; either version 2 of the License
    , or
 * (at your option) any later version.
 *
 * Kharon is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty
    of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
    License along
 * with Kharon; if not, write to the Free Software Foundation,
    Inc.,
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 */

/**
 * @file commands.hpp
 * @author MikelAlejoBR
 * @brief File containing command declarations.
 *
 * This file contains the definition of the commands used by
    both the client
 * and the server.
 */

#ifndef COMMANDS_HPP
#define COMMANDS_HPP

#define MAX_HEADER_SIZE 7

#define NOTGOOD 20
#define GOOD 21

#define ROOT_DIR 30
#define CREATE_DIR 31
#define CREATE_FILE 32
#define MODIFY_FILE 33
#define MOVE_FILE 34
#define DELETE_DIR 35
#define DELETE_FILE 36
```

```
#define END 50

#endif // COMMANDS_HPP
```

4.1.2 The graphical user interface

The GUI is one of the important parts of the application. If the user does not like the user interface, or thinks it is too complicated to use, he or she might end up not using it. Although this application does not have many functionalities, the design is clean and simple, allowing almost every person to use it. Below, there is a set of screenshots showing the graphical user interface's behaviour.

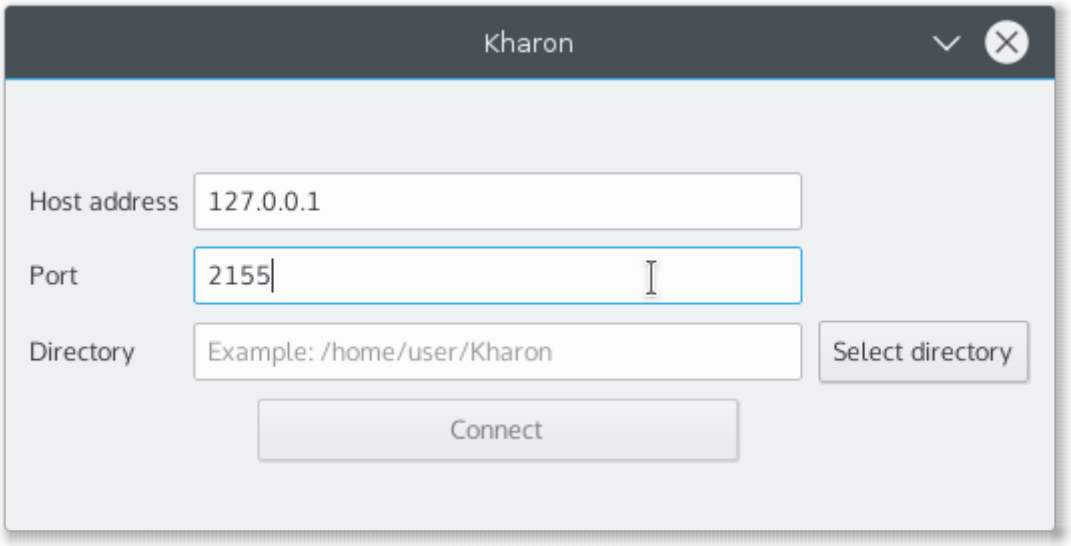


Figure 4.3: Screenshot of the graphical user interface

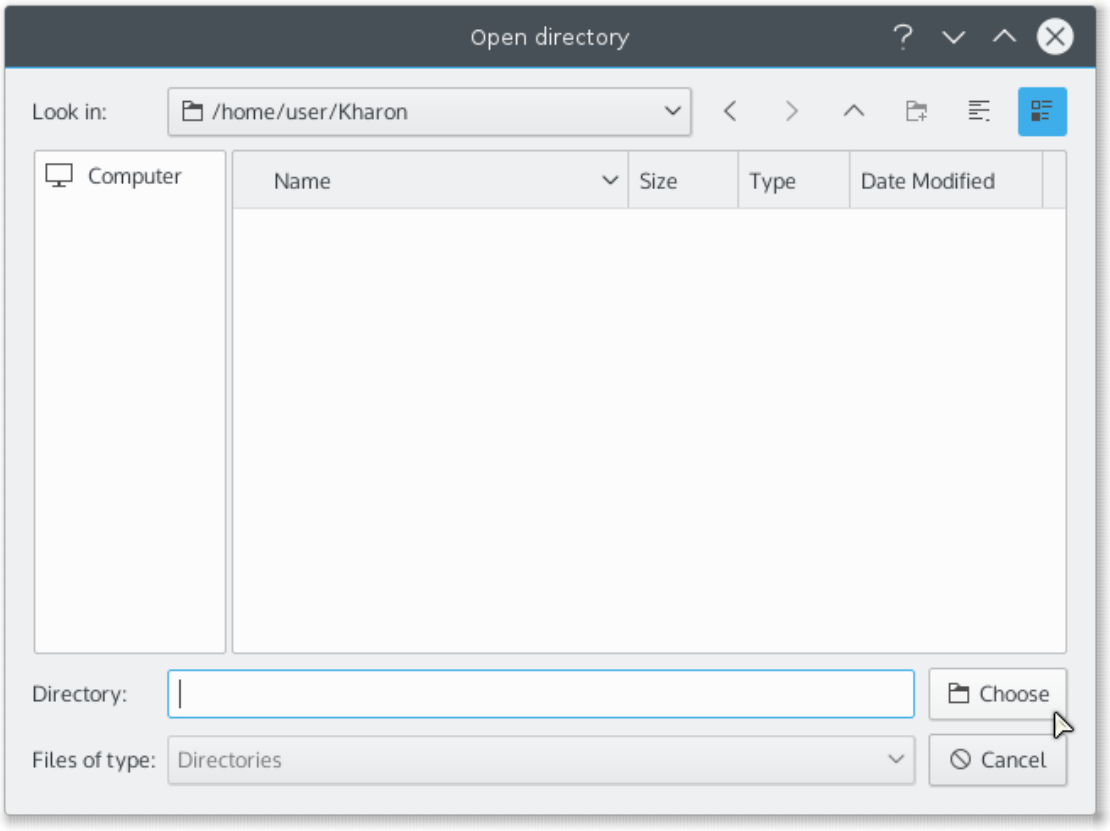


Figure 4.4: Screenshot of the directory selection dialog

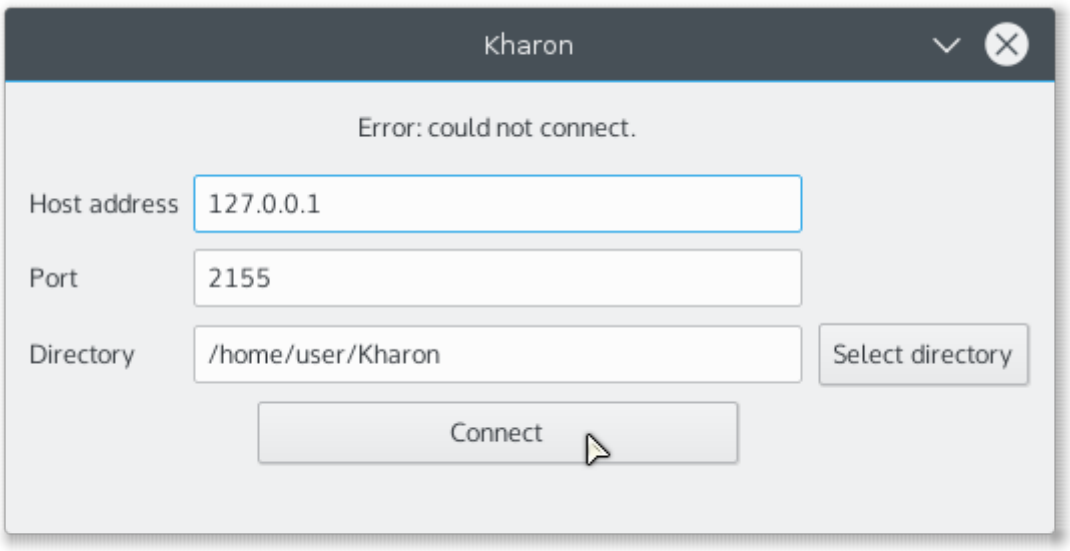


Figure 4.5: Screenshot of the graphical user interface when the connection attempt fails

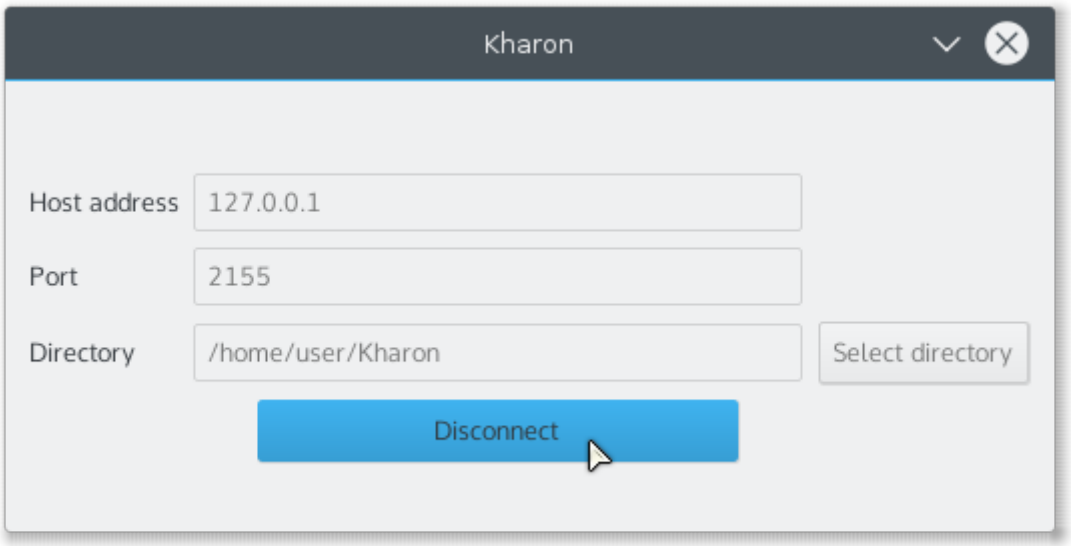


Figure 4.6: Screenshot of the graphical user interface when the connection is successful

4.1.2.1 Class diagram

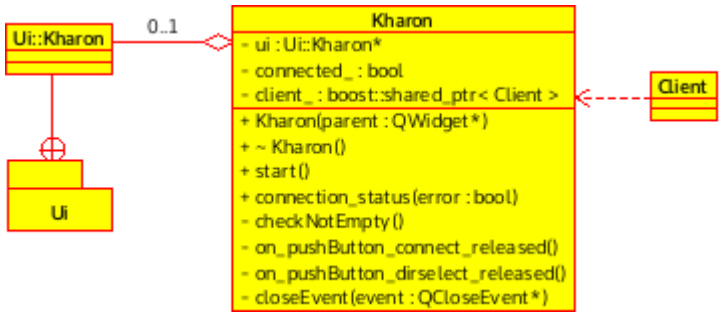


Figure 4.7: Class diagram of the graphical user interface

The figure 4.7 shows the basic class diagram of the graphical user interface, which it is pretty straightforward. The main class is **Kharon**, which has a **Ui** class that is generated from an *.ui* file. This file, is generated using the graphical user interface generator software known by the name Qt Creator. This is extremely useful, as it allows us as programmers to access every element of our user interface, using pointers. Here is an actual example of that:

```
ui->pushButton_connect->setEnabled(true);
```

Finally, the **Kharon** class depends on the **Client** class and vice versa, as they share some member function calls of each other. In fact, they needed to be forward declared in order to make this work.

The functions worth commenting are the *connection_status* and *start* functions. The former one is used when a connection attempt has been made, and it is called from the *Client* object. It activates or deactivates the inputs and buttons depending on the successfulness of the operation. The latter function passes the shared pointer of the *Kharon* object to the *Client* object. This function is called after the *Kharon* object has been completely constructed, as otherwise the shared pointer would not be ready to be used[2].

Finally, the *closeEvent* function closes the application nicely when the user clicks the close button of the window. It stops the Inotify service, makes the client send a *terminate session* message to the server, shuts down the client, and finally closes the GUI.

4.1.3 The client module

Another big module of the client side is the client module. Its the glue that sticks together the three main classes mentioned in section 4.1.1. It processes the actions sent by the GUI, it communicates with the server, and it also takes the actions from the monitoring module and performs the corresponding network operations with them.

4.1.3.1 Class diagram

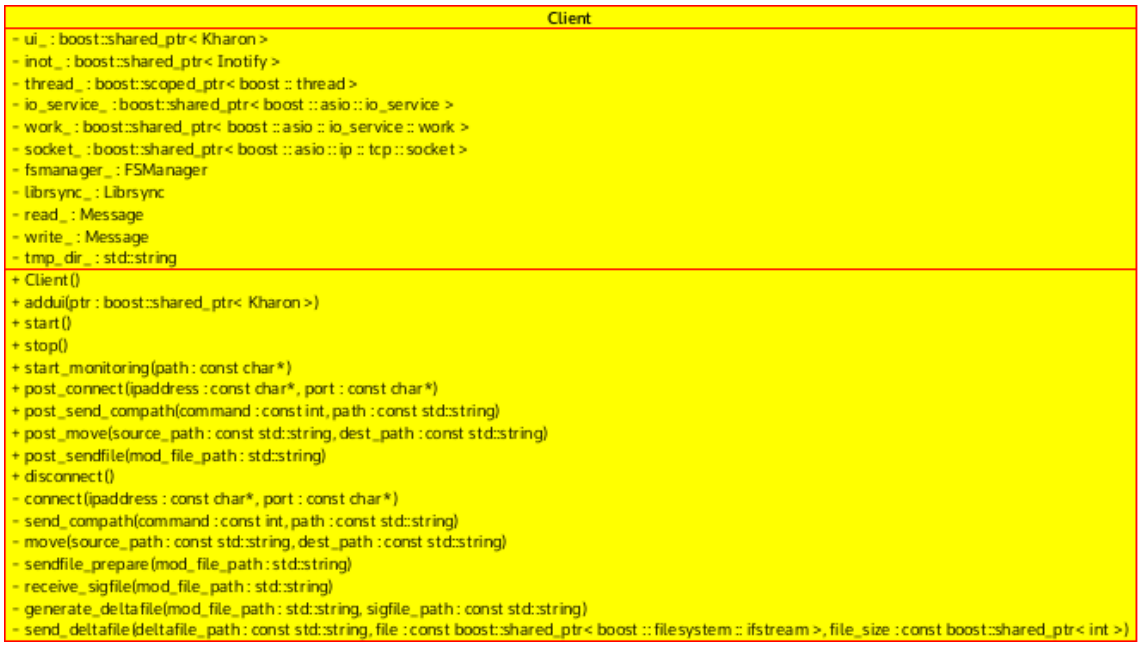


Figure 4.8: Class diagram of the client class without the rest of the elements

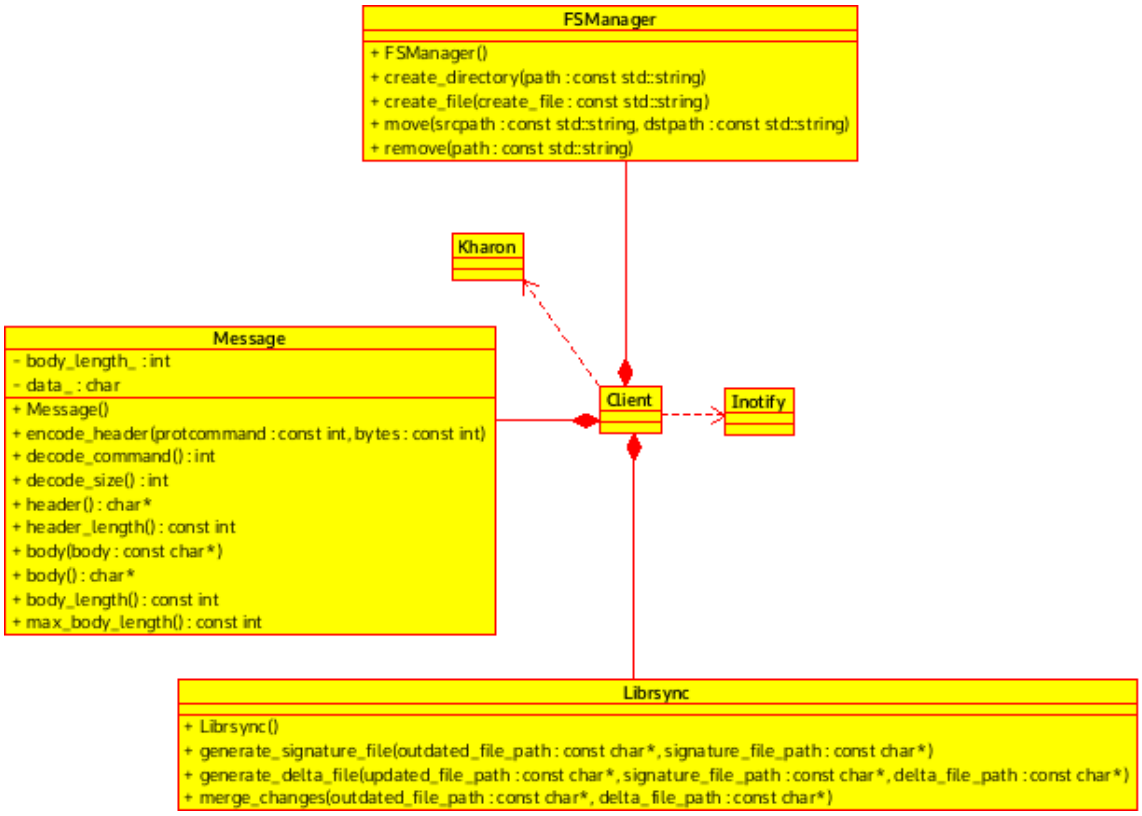


Figure 4.9: Class diagram of the client module

The *Client* class is very simple to understand. It has a few public members that are just the way of giving the *Client*'s thread work. There are some functions that need to be run externally, like the constructor, or the ones that do not have the *post* prefix in their name. However, those functions are usually called when the class is initialized or destroyed, and they do not perform any heavy operations. The public *post* functions just call the non-*post* private functions of the similar name, using *Boost io_service's post function*, which in this case assigns the work to the running thread —see section 3.1.1 for more information about this —.

The client also has access to the *FSManager* and *Librsync* classes: the former helps dealing with some basic file system operations, and the latter helps, in this case, to generate the *delta file* from the *signature file* and the updated file, once the *signature file* has been received from the server —see section 3.2 for the complete explanation —.

4.1.4 The Inotify module

The last class of the three main classes that make the client is the Inotify class. Built on top of the Inotify program written in C, and using borrowed code from "*pkrnjevic*"[1] —he gave me the express permission to use it—I modified the code a

[1] `https://github.com/pkrnjevic`

bit in order to suit the needs of the application. Basically, it monitors the events on a given directory, and if new sub directories are created, those are monitored too. The events are processed using a *polling system* —an asynchronous way of knowing if a descriptor has any updates —and a loop. Inside the loop, depending on the event, there are some calls to the *Client* class which make it interact with the server.

4.1.4.1 The class diagram

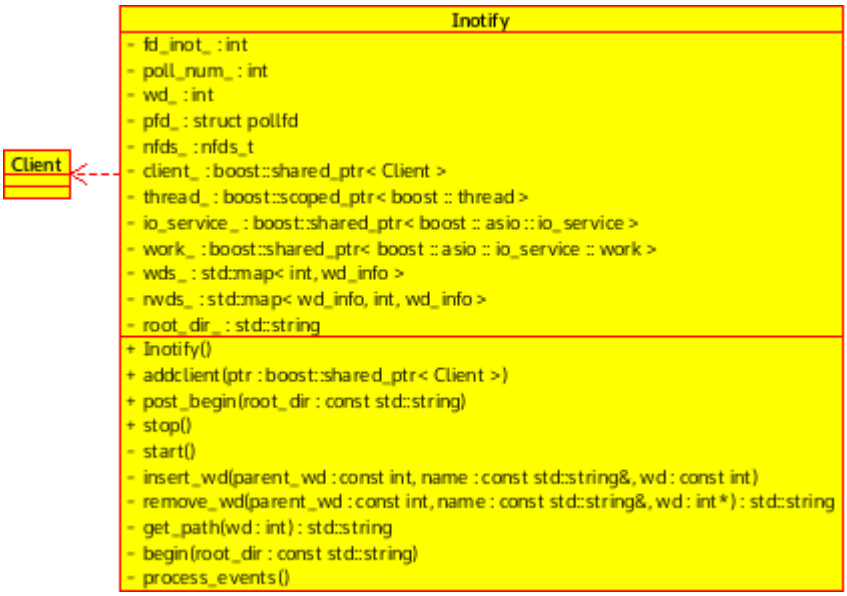


Figure 4.10: Class diagram of the *Inotify* module

This class also creates a new thread when starting, as otherwise its nested loop could block any other thread calling this class' object member functions —check section 5.2.3 for a more detailed explanation of the problem —. The general idea of how it works is shown in the following *pseudo code* snippet:

```
/** ... */
int inotify_instance_fd;
struct wait_for_activity wfa;
EVENTSLIST = {CREATE, MODIFY, MOVE, DELETE};
FILEVENTS = {MODIFY};

/** ... */
inotify_instance_fd = initialize_inotify();
wfa.file_descriptor_to_watch = inotify_instance_fd;

/** ... */
inotify_add_watch(inotify_instance_fd, "directory/", EVENTSLIST)
    ;
```

```
inotify_add_watch(inotify_instance_fd, "other_dir/", EVENTSLIST)
    ;
inotify_add_watch(inotify_instance_fd, "my_file.txt", FILEVENTS)
    ;

while(true)
{
        int there_is_activity = wait_for_activity(wfa);
        if (there_is_activity)
        {
                buffer = read_events(inotify_instance_fd);
                for(int i=0; i<buffer.size; i++)
                {
                        eventType event = buffer[i];

                        if(event.isCreateDir)
                                /** Do things */
                        if(event.isCreateFile)
                                /** Do things */
                        if(event.is...)
                                /** ... */
                }
        }
}
```

Firstly the Inotify instance is created and represented with a file descriptor. That descriptor is polled using a function specially designed for such purposes, which blocks until activity is detected in the file descriptor. Internally, the Inotify instance manages a list of what are called *watch descriptors*, which are bound to a specific path —let that path be a directory or a file—. The *Kernel* is the one who tells Inotify instance about the new events, and the instance takes care of grouping those events and getting them ready to be read. Once they are read, they can be used to perform specific tasks related to that event.

The problem with the Inotify instance is that it only reports which file descriptor generated the event. Imagine the directory "*A*" is being monitored for *create file* events, and that it has the file descriptor "0" assigned to it. If a file "*F*" is created inside, the event generated will tell us that descriptor "0" has reported a *create file* event, with the name of "*F*". Therefore, if we want to know to which directory corresponds that descriptor, we have to somehow store that relation externally. And here it is when the clever solution from "*pkrnjevic*" comes in place.

He figured out a way of storing that data in *C++* maps. Maps, are structures that can bind a key value to data. In the code snippet below a little example of the usage of this tool is shown:

```
        std::map <int, std::string> string_map;
        string_map[2155] = "Hello World!";
```

```
/** Print "Hello World!" */
std::cout << string_map[2155] << std::endl;
```

Here, a simple map is created and the *"Hello World!"* string is mapped to the key value "*2155*", and printed afterwards.

"*pkrnjevic*", has a little bit more complex map. He creates a little structure containing the parent watch descriptor and the name of the directory, in the following format:

```
struct wd_info {
        int parent_wd;
        std::string name;
}

std::map <int, wd_info> wds_list;

/** Assume the root folder's watch descriptor is 1 */

/** The value assigned to watch descriptor is 2, for
   example */
int watch_descriptor = inotify_add_watch(...);

/** Inserting a watch descriptor in the map list */
wd_info wd_struct;

wd_struct.parent_wd = 1;
wd_struct.name = "my_directory";
wds_list[watch_descriptor] = wd_struct;
```

When Inotify detects an event in the watch descriptor number "2" —the one corresponding to the directory "*my_directory*"—, for example a *file creation* event, the following can be done to obtain the full path:

```
/** Assume the root folder's parent watch descriptor is
   set to zero */
std::string get_full_path(int wd)
{
        if(wds_list[wd].parent_wd == 0)
                return wds_list[wd].name;
        else
                return (get_full_path(wds_list[wd].
                   parent_wd) + "/" + wds_list[wd].name)
                   ;
}

/** ... */
eventType event = event_buffer[i];

int watch_descriptor = event->getwd;
```

```
        std::string full_path = get_full_path(watch_descriptor);

        /** ... */
```

This way, a watch descriptor is associated to a path and its parent descriptor, and the full path can be obtained recursively.

4.2 The server

4.2.1 A general idea of how it works

The server is just an application that listens at a given port. It makes heavy use of the asynchrony that *Boost Asio* provides, and therefore it focuses on efficiency and responsiveness, with the aim of being able to rapidly process every event that comes from the client. A session is created for each client, which makes the connection independent to the rest of the connected users.

4.2.1.1 The class diagram

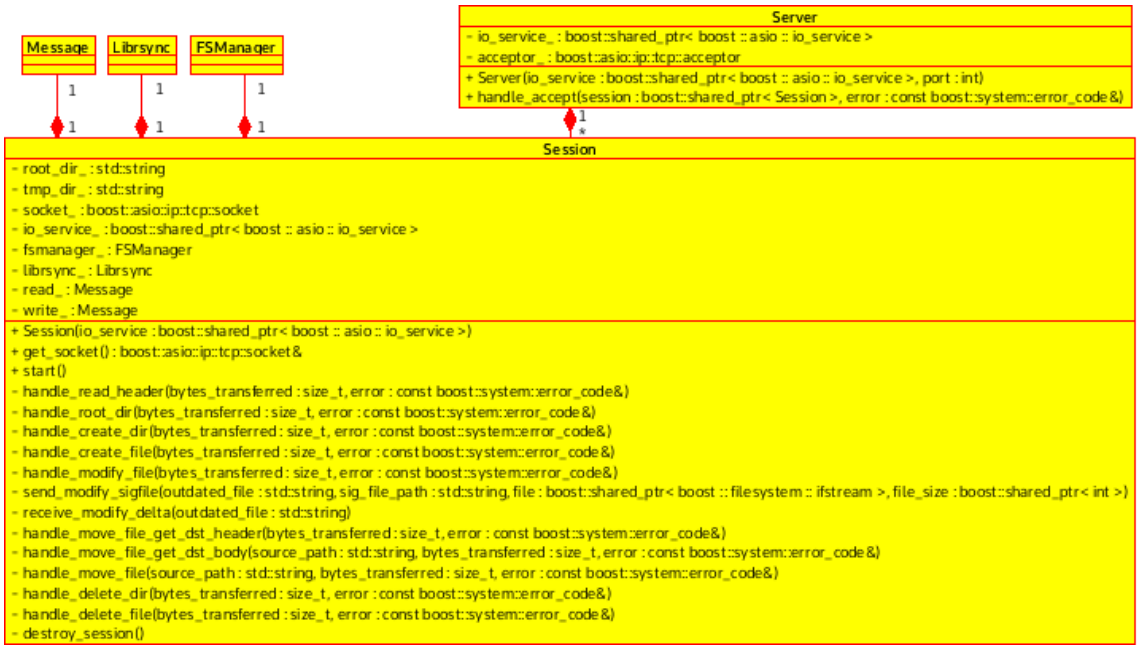


Figure 4.11: Class diagram of the server module

The *Server* class just accepts connections and creates sessions for them. The *Session* class is the one that will keep the session alive, by chaining operations, and depending on which command is received from the client the corresponding handler will be executed.

The following example shows how that chaining works, and the logic behind it:

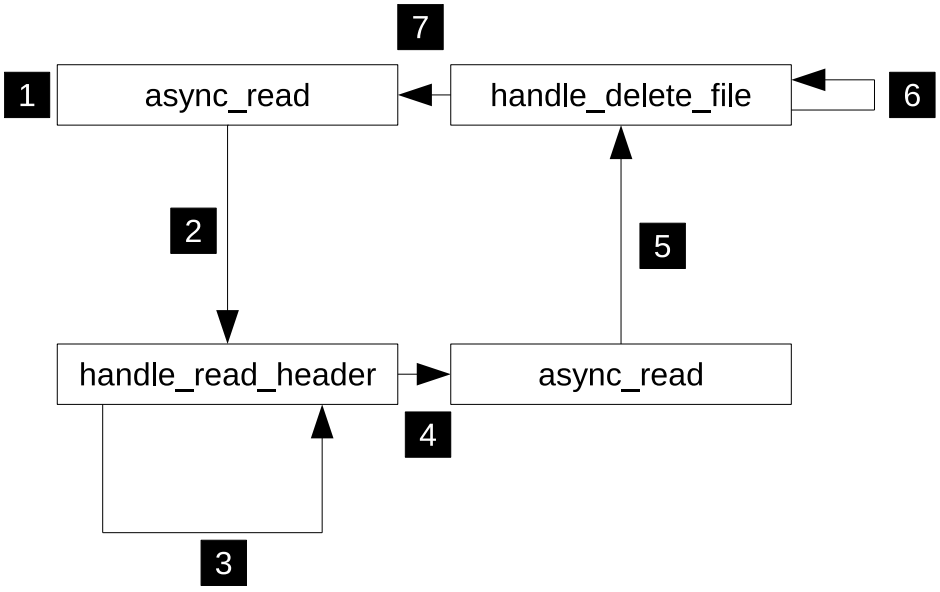


Figure 4.12: Diagram of how the *Session* class works

1. The *Session* class is created, and an *asynchronous read* is performed, in order to read the message header.
2. The header is read, and the *io_service* forwards the results to the handler.
3. The handler decodes header, and extracts the following information: the command which has been sent from the client, and the size of the body of the incoming message —in this case with the path of the file being deleted —.
4. The handler performs another asynchronous read, expecting a message of the size which was decoded from the header.
5. The body is read, and the *io_service* forwards the results to the handler.
6. The handler attempts to delete the file.
7. The handler performs an asynchronous read for a new header message, closing the loop.

Playing with different handlers depending on the command received, it is possible to build a fully functional *Session* class that can handle a wide variety of commands.

4.3 The challenges

4.3.1 Starting from scratch

One of the biggest challenges of this project has been to learn how the tools and libraries are used. Starting from the documentation itself, LaTeX has sometimes

become in a painful tool to use. Even for the minimum and supposedly easiest thing to do LaTeX turns it into a nightmare. Once one figures out how everything works on LaTeX, everything becomes a little bit more easier, but still. Proof of this is the uncountable resources that have been consulted in order to fix common LaTeX problems. [3–24]

After struggling with the documentation, *C++* comes into place. If learning the language was not challenging enough, the planning said that it had to be learned along with *Boost*, which is a library intended for intermediate *C++* programmers. That made the first weeks of development slow, as the consultations were of the following kind: knowing what was a virtual destructor in *C++* [25], compiling projects built with *Boost* [26], learning about *C++* constructors [27], facing threading errors [28], consulting CMake examples to make more complex projects [29], linking the *Boost* libraries to the targets [30], fixing errors on host name resolution [31], following complete guides which teach how to use *Boost* [32], learning about *Boost Bind* [1], consulting specific data types and functions [33–44], consulting on *Boost File system* library and facing errors [45, 46], learning how to run *io_service* in a different thread [47], learning about which signals package should be used in *C++* [48] and finally using *Boost Signals* for it [49], propagating exceptions between threads [50], figuring out which way is the best in order to transfer files using *Boost* [51], and many, many secondary resources that have not been listed.

4.3.2 Compiling Qt projects with CMake

Qt Creator and KDevelop use different ways to organize, arrange and compile the projects. Qt Creator uses an integrated compiled, and an special syntax to indicate the files and modules that need to be compiled. It automates the generation of makefiles interpreting the syntax of that special, *.pro* suffixed file. On the other hand, KDevelop uses CMake, which turns compiling Qt projects into a little problem. Qt applications make use of different modules —such as the *Qt Widgets* or *Qt Network* used in this application —in order to add functionality to our graphical user interface. Nonetheless, those modules must be specified —either in the *.pro* file or in the CMakeLists—when compiling the application as otherwise the compiler will complain about their absence. Essentially, this problem can be solved checking the on line manual of Qt [52]. Here is a code snippet that shows the proper usage of CMake to compile projects that contain Qt modules. The *AUTOMOC* [53] [54] and *AUTOUIC* [55] [56]files generate the Meta-Object Compiler. *See: Meta-object system* (MOC) and User Interface Compiler. *See: UIC* (UIC) files automatically, whereas *CMAKE_INCLUDE_CURRENT_DIR* includes the current source directory and the current binary directory to the targets [57].

```
cmake_minimum_required(VERSION 3.0.0 FATAL_ERROR)
project(Example)

set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTOUIC ON)
```

```
set(CMAKE_INCLUDE_CURRENT_DIR ON)

find_package(Qt5Widgets REQUIRED)
find_package(Qt5Network REQUIRED)

include_directories(${Qt5Widgets_INCLUDE_DIRS})
add_definitions(${QT_DEFINITIONS})

add_executable(main main.cpp)
target_link_libraries(main Qt5::Widgets Qt5::Network)
```

4.3.3 How to make forward declarations in C++

One of the biggest problems I faced when developing the client part of the application was how to properly include one class in another and vice versa. I thought that adding a simple *#include* in each of the class headers would make it, but this little problem made the compiler complain a lot. After a lot of research and many hours spent visiting websites, tutorials and much more, I learned that this way of "including" is called a forward declaration. Therefore, as I already knew what was the name of the technique I had to use, I finally stumbled upon some resources [58] [59] [60] that helped me compile the application without any kind of error.

The trick is to consists on declaring a class without its structure or implementation, and later in the header file declare a pointer to an object of that class. Finally, in the source code file —the *.cpp* file—, is where the header of that class is included, which makes the pointer have all the functionality that the class may have. Here is a little example of that:

```
/**************
 * classa.hpp *
 * ************/
#include "classb.hpp"

class ClassA
{
public:
        ClassA();
        ~ClassA();

        /** ... */

private:
        ClassB b_object_;
};

/**************
 * classa.hpp *
```

```
 * ************/

/** Implementation of ClassA */

/**************
 * classb.hpp *
 * ************/
class ClassA;

class ClassB
{
public:
        ClassB();
        ~ClassB();

        void add_ref_aobject(ClassA* ptr);

        /** ... */

        private:
        ClassA* a_object_;
};

/**************
 * classb.cpp *
 * ************/
#include "classa.hpp"

/* ... */

void ClassB::add_ref_aobject(ClassA* ptr)
{
        a_object_ = ptr;
}

/* ... */
```

4.3.4 Understanding *Boost* libraries

Maybe this has been the most challenging part of the project. I was not used to program so differently, with things like *Boost Asio*, *Boost Bind* and *Boost Thread*. It has supposed a massive challenge, and a big amount of problems and misconceptions I had were altruistically solved by the people of the official *Boost* Internet Relay Chat (IRC) channel.

They helped me by not giving me the direct answer. Instead, they helped by giving me tips and making me figure out the answer to my question.

Probably you are asking yourself why this section is so short, and why I do not include any specific problem here. This is because of two reasons:

1. Many problems were much more advanced than the explanations shown in this report. Because of this reason, trying to explain the problems I had would extend too much the report, and probably I would not achieve making the reader understand what the problem is, and what was the solution to it.

2. The clearest representation of the problems and difficulties I had with the libraries are spread throughout the report, in the form of concept explanations and examples.

4.3.5 Minor issues

Some other minor issues have involved learning about some specific topics:

- Using simple labels [61].
- Defining events for visual elements in order to learn the syntax and implement them manually afterwards [62].
- Converting from the Qt type *QString* to *const char** [63] [64].
- Learning about how to properly use signals and slots in Qt [65] [66].
- And learning how to implement a directory browse dialog [67].

5 Conclusions

5.1 Tracking and control

It has been tried to follow the initial planning —the one shown in the *Gantt* chart—, but as shown in the *Gantt charts* of the section 5.1.1, some delays have been suffered throughout the project. Initially, the project was thought to be rather simple, just like the technologies to be learned. Nonetheless, appearances can be deceiving. The task of developing an application while learning a new language, new tools and new libraries is a massive challenge.

Although the *Scrum* methodology helped a bit in focusing on small tasks, and setting concrete goals, the constant learning which had to be done led to many structural changes and therefore, kept delaying more and more the finish date of the application development.

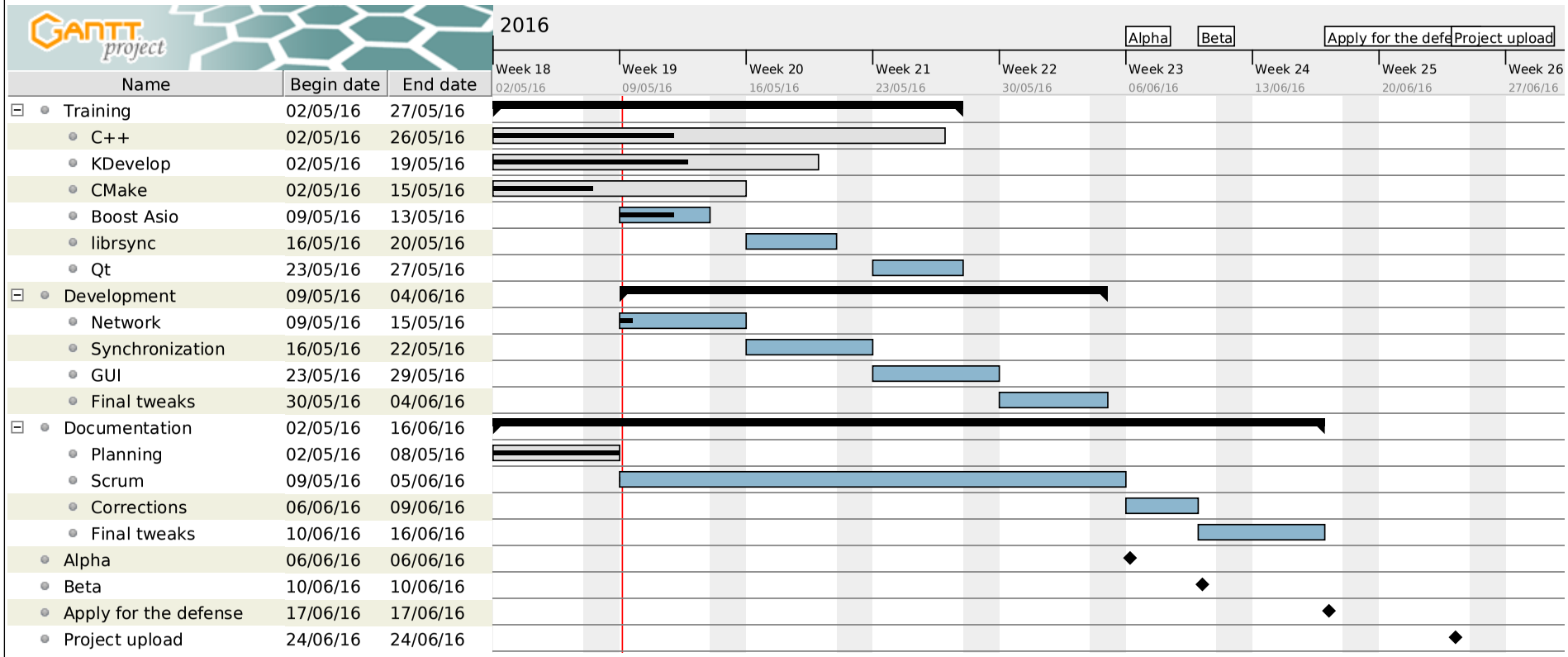
For example, the part which has been kept untouched for a long time has been the GUI part, which it did not change much since it was developed. Instead, the client module, the synchronization module and the server module have been constantly changing, as new concepts, ways and tricks were being learned.

The table below represents the work invested in the project, and how it has been in constant growth since the project started —except for a sprint in which it had to be shortened due to academic obligations—. The granularity was set in fifteen minutes, in order to make tracking and control easier.

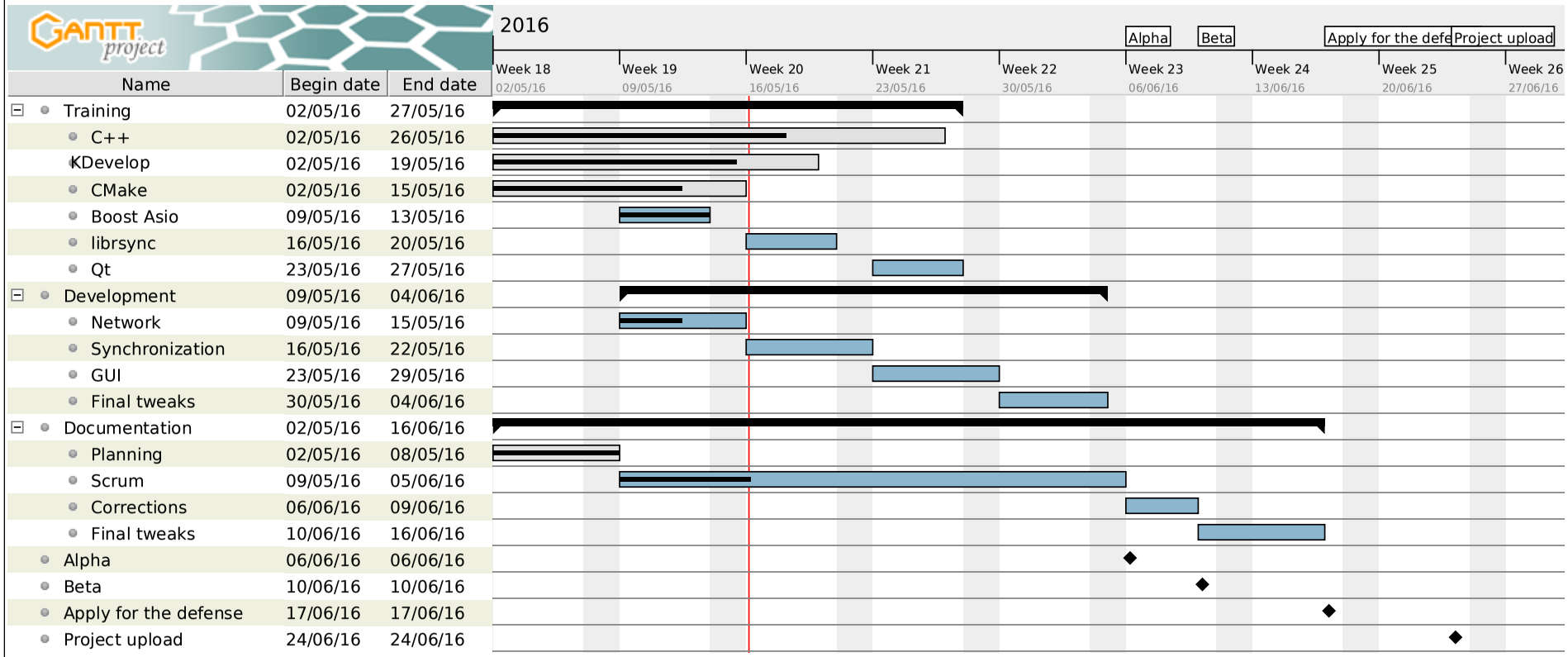
Task	Time invested
Develop the first draft of the report	23h 30’
Correct the first draft	2h 15’
Learning the tools	35h 45’
First sprint *Network*	44h 30’
Second sprint *Synchronization*	43h 15’
Third sprint *GUI*	46h 15’
Fourth sprint *Final tweaks*	39h 45’
Fifth sprint *Bonus: network, sync and gui*	46h 30’
Sixth sprint *Bonus: network, sync and gui*	45h 15’
Alpha version of the report	6h 15’
Beta version of the report	9h
Final version of the report	12h 45’

5.1.1 Gantt chart’s snapshots

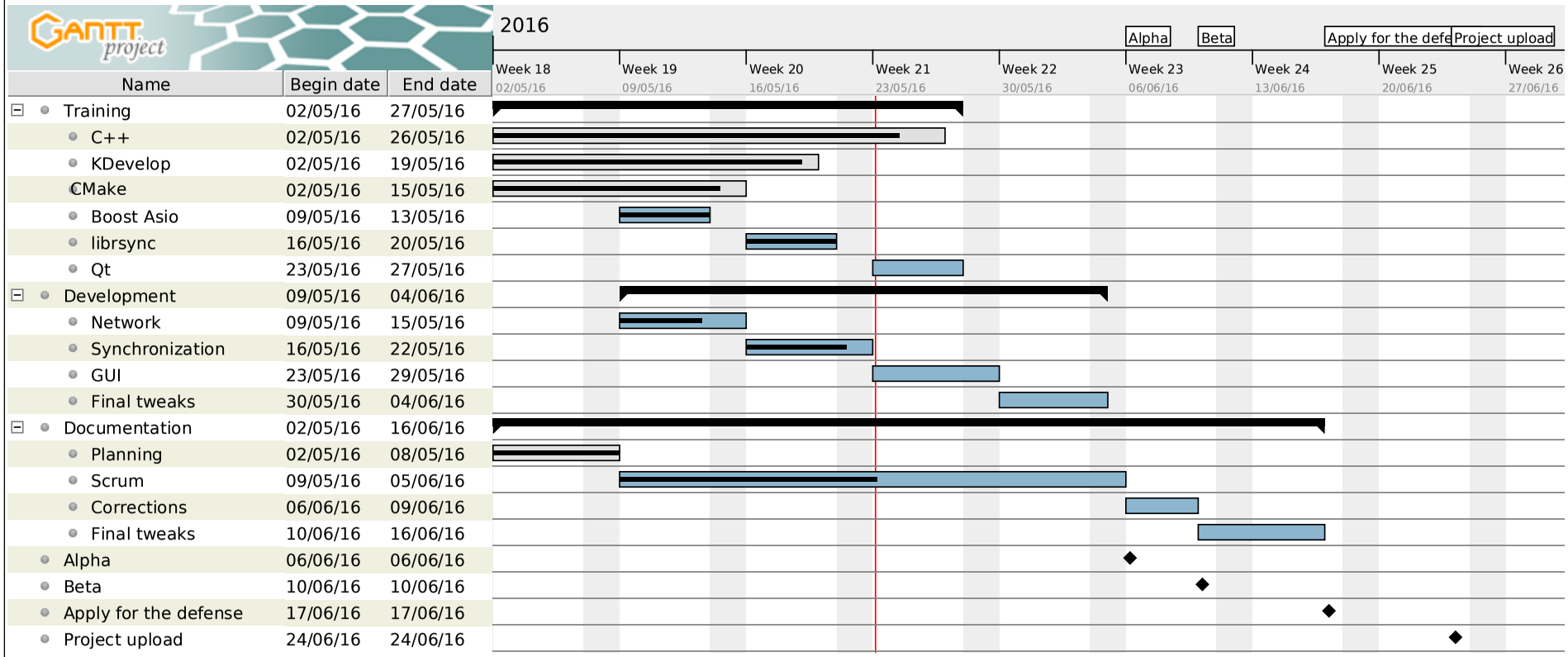
The following pages include snapshots of the *Gantt chart* at different stages of the project. The snapshots were taken every Monday.



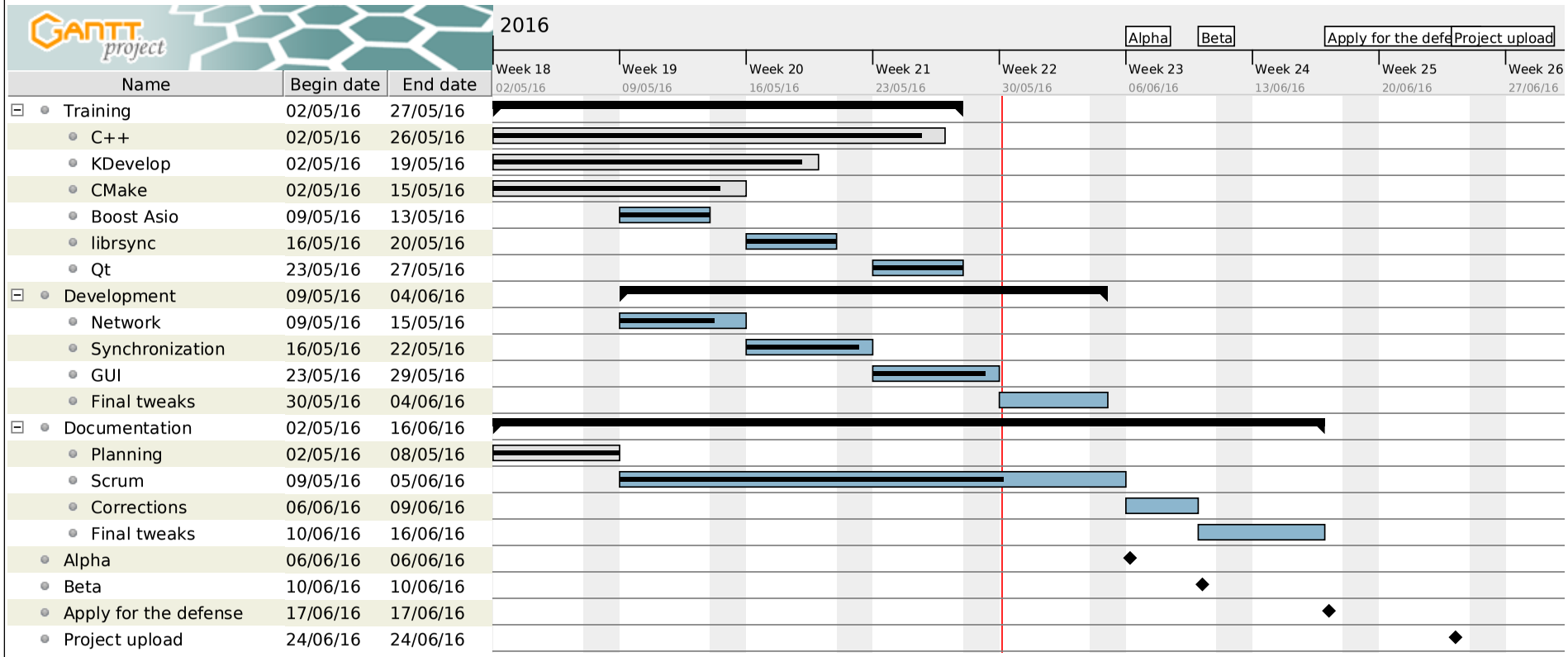
Name	Begin date	End date
Training	02/05/16	27/05/16
C++	02/05/16	26/05/16
KDevelop	02/05/16	19/05/16
CMake	02/05/16	15/05/16
Boost Asio	09/05/16	13/05/16
librsync	16/05/16	20/05/16
Qt	23/05/16	27/05/16
Development	09/05/16	04/06/16
Network	09/05/16	15/05/16
Synchronization	16/05/16	22/05/16
GUI	23/05/16	29/05/16
Final tweaks	30/05/16	04/06/16
Documentation	02/05/16	16/06/16
Planning	02/05/16	08/05/16
Scrum	09/05/16	05/06/16
Corrections	06/06/16	09/06/16
Final tweaks	10/06/16	16/06/16
Alpha	06/06/16	06/06/16
Beta	10/06/16	10/06/16
Apply for the defense	17/06/16	17/06/16
Project upload	24/06/16	24/06/16



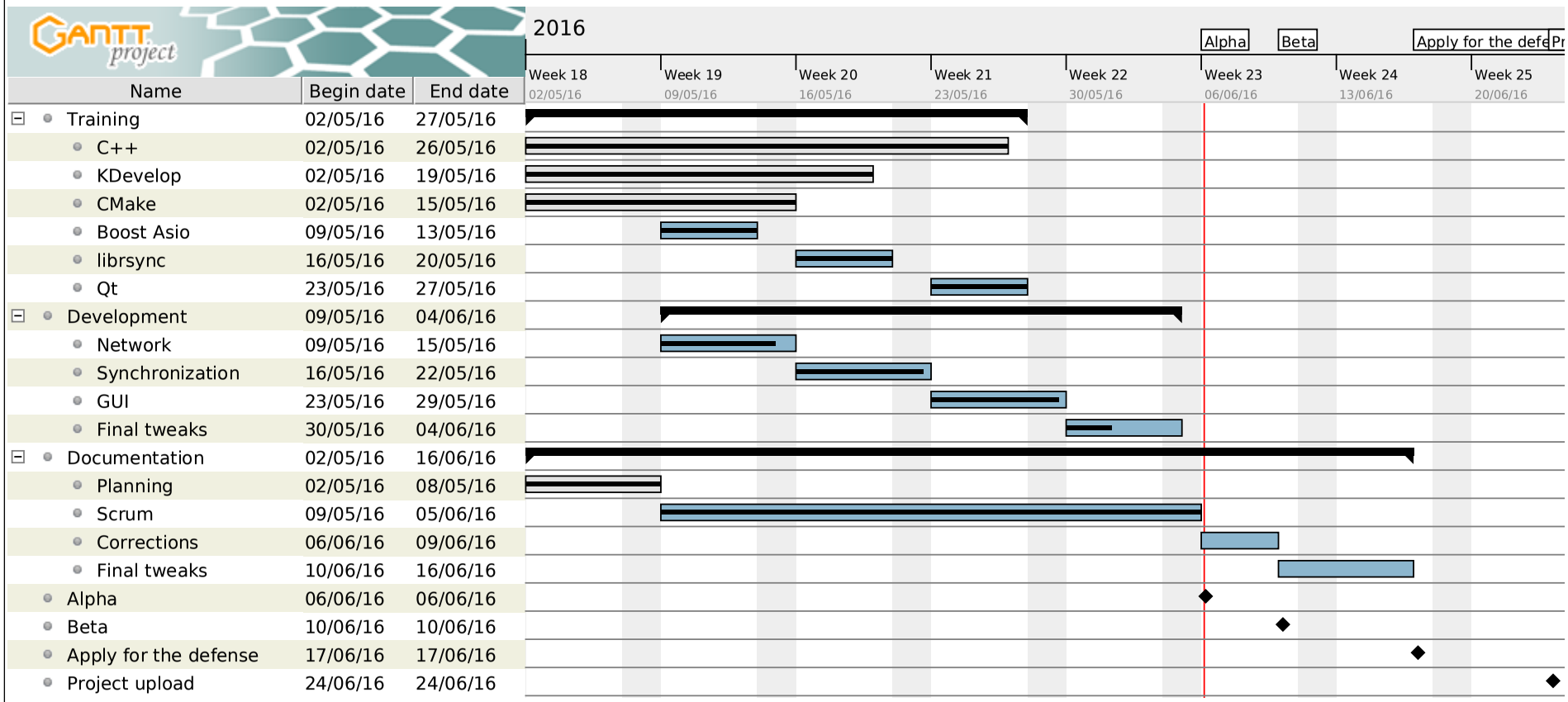
Name	Begin date	End date
Training	02/05/16	27/05/16
C++	02/05/16	26/05/16
KDevelop	02/05/16	19/05/16
CMake	02/05/16	15/05/16
Boost Asio	09/05/16	13/05/16
librsync	16/05/16	20/05/16
Qt	23/05/16	27/05/16
Development	09/05/16	04/06/16
Network	09/05/16	15/05/16
Synchronization	16/05/16	22/05/16
GUI	23/05/16	29/05/16
Final tweaks	30/05/16	04/06/16
Documentation	02/05/16	16/06/16
Planning	02/05/16	08/05/16
Scrum	09/05/16	05/06/16
Corrections	06/06/16	09/06/16
Final tweaks	10/06/16	16/06/16
Alpha	06/06/16	06/06/16
Beta	10/06/16	10/06/16
Apply for the defense	17/06/16	17/06/16
Project upload	24/06/16	24/06/16



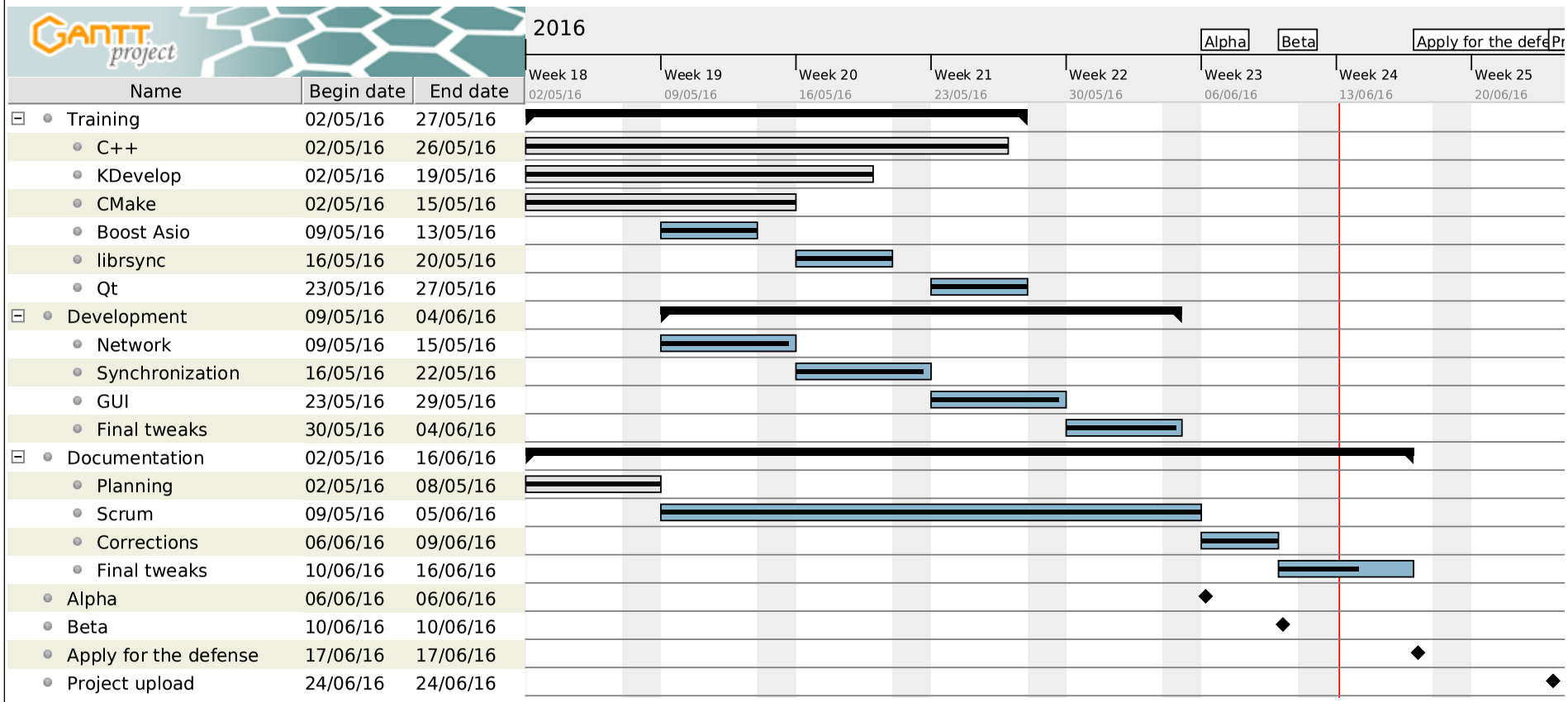
Name	Begin date	End date
Training	02/05/16	27/05/16
C++	02/05/16	26/05/16
KDevelop	02/05/16	19/05/16
CMake	02/05/16	15/05/16
Boost Asio	09/05/16	13/05/16
librsync	16/05/16	20/05/16
Qt	23/05/16	27/05/16
Development	09/05/16	04/06/16
Network	09/05/16	15/05/16
Synchronization	16/05/16	22/05/16
GUI	23/05/16	29/05/16
Final tweaks	30/05/16	04/06/16
Documentation	02/05/16	16/06/16
Planning	02/05/16	08/05/16
Scrum	09/05/16	05/06/16
Corrections	06/06/16	09/06/16
Final tweaks	10/06/16	16/06/16
Alpha	06/06/16	06/06/16
Beta	10/06/16	10/06/16
Apply for the defense	17/06/16	17/06/16
Project upload	24/06/16	24/06/16



Name	Begin date	End date
Training	02/05/16	27/05/16
C++	02/05/16	26/05/16
KDevelop	02/05/16	19/05/16
CMake	02/05/16	15/05/16
Boost Asio	09/05/16	13/05/16
librsync	16/05/16	20/05/16
Qt	23/05/16	27/05/16
Development	09/05/16	04/06/16
Network	09/05/16	15/05/16
Synchronization	16/05/16	22/05/16
GUI	23/05/16	29/05/16
Final tweaks	30/05/16	04/06/16
Documentation	02/05/16	16/06/16
Planning	02/05/16	08/05/16
Scrum	09/05/16	05/06/16
Corrections	06/06/16	09/06/16
Final tweaks	10/06/16	16/06/16
Alpha	06/06/16	06/06/16
Beta	10/06/16	10/06/16
Apply for the defense	17/06/16	17/06/16
Project upload	24/06/16	24/06/16



Name	Begin date	End date
Training	02/05/16	27/05/16
C++	02/05/16	26/05/16
KDevelop	02/05/16	19/05/16
CMake	02/05/16	15/05/16
Boost Asio	09/05/16	13/05/16
librsync	16/05/16	20/05/16
Qt	23/05/16	27/05/16
Development	09/05/16	04/06/16
Network	09/05/16	15/05/16
Synchronization	16/05/16	22/05/16
GUI	23/05/16	29/05/16
Final tweaks	30/05/16	04/06/16
Documentation	02/05/16	16/06/16
Planning	02/05/16	08/05/16
Scrum	09/05/16	05/06/16
Corrections	06/06/16	09/06/16
Final tweaks	10/06/16	16/06/16
Alpha	06/06/16	06/06/16
Beta	10/06/16	10/06/16
Apply for the defense	17/06/16	17/06/16
Project upload	24/06/16	24/06/16



Name	Begin date	End date
Training	02/05/16	27/05/16
C++	02/05/16	26/05/16
KDevelop	02/05/16	19/05/16
CMake	02/05/16	15/05/16
Boost Asio	09/05/16	13/05/16
librsync	16/05/16	20/05/16
Qt	23/05/16	27/05/16
Development	09/05/16	04/06/16
Network	09/05/16	15/05/16
Synchronization	16/05/16	22/05/16
GUI	23/05/16	29/05/16
Final tweaks	30/05/16	04/06/16
Documentation	02/05/16	16/06/16
Planning	02/05/16	08/05/16
Scrum	09/05/16	05/06/16
Corrections	06/06/16	09/06/16
Final tweaks	10/06/16	16/06/16
Alpha	06/06/16	06/06/16
Beta	10/06/16	10/06/16
Apply for the defense	17/06/16	17/06/16
Project upload	24/06/16	24/06/16

5.2 Limitations

Although the minimum quality planned for the application has been achieved, the software presents some limitations, mainly because of the strict deadlines and planning restrictions. The general learning curve of this project has been steep, and consequently the speed at which the development was taking place was considerably slow. Some of the limitations are explained in the following sections.

5.2.1 Lack of testing

Due to the delays caused by the learning process, and the demanding planning in terms of meeting the deadlines, I decided to reduce the effort put in this regard, in order to be able to release an application as complete as possible. Consequently, the application may misbehave in certain situations.

5.2.2 Networking

Transmission Control Protocol (TCP) is presented to the programmer as an continuous stream of bytes, whereas User Datagram Protocol (UDP) is presented as a message oriented protocol. Due to an initial misunderstanding of how TCP worked, the application was developed with the "*packets idea*" in mind. Although the examples, guides and tutorials followed used TCP as the network protocol, this misconception may have led to an less efficient network code.

5.2.3 Threading

Threads are tend to be used in pools. These pools hold an amount of threads that when they are idle, take some work and do it. However, in this project's application, the threads have not been grouped in a pool. The GUI launches a thread, the client another one and the directory monitoring module the third one.

This is due to how the *io_service* works, and due to the demanding deadlines set in the planning which made me decided leave the investigation of turning the synchronous, loop based Inotify module into an asynchronous module. The *io_service* executes the specified handlers in the asynchronous operations once these have been finished, and until those handlers return, no more handlers are executed. Therefore, if the *io_service* is told to execute a function or a handler which has an infinite loop within, the handler will never return and no more handlers will be executed.

5.3 Future work lines

The functionalities mentioned in the 2.7.2 are a good example of how this application could be improved in the future. However, before adding any new functionalities the actual limitations should be overcome first. By order of relevance, the essential core aspects of the application should be reviewed:

1. Whether the network code could be simplified or not, by checking if the TCP concept has been efficiently applied.

2. Find a way of getting rid of the nested loops that the current implementation of the synchronization module presents, by making it asynchronous.

3. Continue and go in depth about error handling and testing, which would improve the application's stability.

5.4 Project conclusions

The objective of building a client-server application which synchronizes a directory between the two ends has been achieved, although with some limitations. From the beginning, it was thought to use libraries that already were developed, as otherwise the full development of all the three main modules of the application would have been independent projects themselves.

Therefore, the outcome of this project is the result of the integration of different libraries, being some of them written in a different programming language and having being wrapped into the main programming language that has been used to develop the application. This focus on using already developed libraries has made the project be similar to an non-academic project, as usually when working in a company, the usage of frameworks and already developed and tested libraries is crucial in order to be able to meet the deadlines.

Not only the application has been important, as the amount of knowledge gained into advanced topics like sending file differences over the network, or using asynchrony in order to enhance scalability and efficiency are widely used by professionals nowadays in a wide range of applications.

Another important aspect of this application is that it has been entirely built using open source tools and libraries. Besides, these assets are free to use —except for Qt if it is used for a commercial project —and surprising though it may seem, they provide support for those developers that may have some sort of problem using the aforementioned assets.

In general, it has been a successful project, not only because of the final application, but also because of the knowledge gained in cutting edge technologies.

Acronyms

API Application Programming Interface. 16, 49

GUI Graphical User Interface. 1, 6, 8, 25, 28, 39, 46

IDE Integrated Development Environment. 49

IRC Internet Relay Chat. 37

MOC Meta-Object Compiler. *See: Meta-object system.* 35

SDK Software development kit. 50

TCP Transmission Control Protocol. 46, 47

UDP User Datagram Protocol. 46

UIC User Interface Compiler. *See: UIC.* 35

Glossary

Agile software development Agile software development is a set of principles for software development in which requirements and solutions evolve through collaboration between self-organizing cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change. *https://en.wikipedia.org/wiki/Agile_software_development*. 9, 18

CMake is cross-platform free and open-source software for managing the build process of software using a compiler-independent method. It is designed to support directory hierarchies and applications that depend on multiple libraries. It is used in conjunction with native build environments such as make, Apple's Xcode, and Microsoft Visual Studio. It has minimal dependencies, requiring only a C++ compiler on its own build system. *Source: https://en.wikipedia.org/wiki/CMake*. 35, 49

CMakeLists is the file that CMake uses. 35

Forward declaration is a declaration of an identifier for which the programmer has not given yet a complete definition. In this project, it is used when two classes need to include each other. 27, 36

Inotify is an API which provides a mechanism for monitoring filesystem events. Inotify can be used to monitor individual files, or to monitor directories. When a directory is monitored, Inotify will return events for the directory itself, and for files inside the directory.. 28, 29, 31, 32, 46, 49

KDevelop is a free software integrated development environment Integrated Development Environment (IDE) for the KDE Platform on Unix-like computer operating systems. KDevelop includes no compiler; instead, it uses an external compiler such as GCC or Clang to produce executable binaries. *Source: https://en.wikipedia.org/wiki/KDevelop*. 35

make is a build automation tool that automatically builds executable programs and libraries from source code by reading files —called makefiles —which specify how to derive the target program. *Source: http://www.boost.org/doc/libs/1_58_0/libs/smart_ptr/shared_ptr.htm*. 49

Makefile is a file containing a set of directives used with the make build automation tool. *Source: https://en.wikipedia.org/wiki/Makefile*. 35, 49

Meta-object system is a part of Qt framework core provided to support Qt extensions to C++ like signals/slots for inter-object communication. *Source: https://en.wikipedia.org/wiki/Meta-object_System*. 35, 48

Qt is a cross-platform application framework that is widely used for developing application software that can be run on various software and hardware platforms with little or no change in the underlying codebase, while still being a native application with the capabilities and speed thereof. *Source: https://en.wikipedia.org/wiki/Qt_%28software%29*. 22, 35, 38, 47, 50

Qt Creator is a cross-platform C++, JavaScript and QML integrated development environment which is part of the Software development kit (SDK) for the Qt GUI Application development framework. It includes a visual debugger and an integrated GUI layout and forms designer. *Source: https://en.wikipedia.org/wiki/Qt_Creator*. 27, 35, 50

Shared pointer is a container for an standard raw pointer pointing to a dynamically allocated object. The object will only be destroyed when all the shared pointers referencing to it are deleted or reseted. *Source: http://www.boost.org/doc/libs/1_58_0/libs/smart_ptr/shared_ptr.htm*. 28, 50

UIC reads an XML format user interface definition (.ui) file as generated by Qt Designer —or Qt Creator—and creates a corresponding C++ header file. *Source: http://doc.qt.io/qt-4.8/uic.html*. 35, 48

Bibliography

[1] Boost. *Boost.Bind*. unknown. URL: http://www.boost.org/doc/libs/1_61_0/libs/bind/doc/html/bind.html (visited on 05/09/2016).

[2] Barry. *Getting 'bad_weak_ptr' error*. June 2016. URL: http://stackoverflow.com/questions/37630593/getting-bad-weak-ptr-error (visited on 06/04/2016).

[3] egreg. *Comments in BibTex*. June 2015. URL: http://tex.stackexchange.com/questions/21709/comments-in-bibtex (visited on 01/28/2016).

[4] Heiko Oberdiek. *Line breaks of long URLs in bibliography?* Sept. 2013. URL: http://tex.stackexchange.com/questions/134191/line-breaks-of-long-urls-in-bibliography (visited on 01/28/2016).

[5] herohuyongtao. *How to center the table in Latex*. Feb. 2014. URL: http://tex.stackexchange.com/questions/162462/how-to-center-the-table-in-latex (visited on 01/28/2016).

[6] Wikibooks. *LaTeX/Macros*. Dec. 2015. URL: https://en.wikibooks.org/wiki/LaTeX/Macros (visited on 01/28/2016).

[7] Stefan Kottwitz. *Color changes cell height in tabular*. Oct. 2011. URL: http://tex.stackexchange.com/questions/31547/color-changes-cell-height-in-tabular (visited on 01/28/2016).

[8] Wikibooks. *LaTeX/Colors*. Jan. 2016. URL: https://en.wikibooks.org/wiki/LaTeX/Colors (visited on 01/28/2016).

[9] Wikibooks. *LaTeX/Tables*. Jan. 2016. URL: https://en.wikibooks.org/wiki/LaTeX/Tables (visited on 01/14/2016).

[10] Ignasi. *How to make pgfgantt scale to specific widths in the page? (ex. textwidth)*. June 2012. URL: http://tex.stackexchange.com/questions/59001/how-to-make-pgfgantt-scale-to-specific-widths-in-the-page-ex-textwidth (visited on 01/31/2016).

[11] domwass. *What's the quickest way to write "2nd" "3rd" etc in LaTeX?* Oct. 2014. URL: http://tex.stackexchange.com/questions/4118/whats-the-quickest-way-to-write-2nd-3rd-etc-in-latex (visited on 02/04/2016).

[12] Yiannis Lazarides. *How to get "LaTeX" symbol in document*. Dec. 2010. URL: http://tex.stackexchange.com/questions/7546/how-to-get-latex-symbol-in-document (visited on 05/07/2016).

[13] Rasmus. *Command line tool to crop PDF files*. Aug. 2012. URL: http://askubuntu.com/questions/124692/command-line-tool-to-crop-pdf-files (visited on 05/07/2016).

[14] phi. *Citation overfull hbox error*. Nov. 2008. URL: http://latex-community.org/forum/viewtopic.php?f=5&t=3239 (visited on 05/30/2016).

[15] Joseph Wright. *Bibliography error: "Use of blx@bbl@verbadd@i doesn't match its definition. verb"*. May 2016. URL: http://tex.stackexchange.com/questions/311426/bibliography-error-use-of-blxbblverbaddi-doesnt-match-its-definition-ve (visited on 06/08/2016).

[16] Wikibooks. *LaTeX/Source Code Listings*. Jan. 2016. URL: https://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings (visited on 06/09/2016).

[17] ShreevatsaR. *How can I make an enumerate list start at something other than 1?* July 2010. URL: http://tex.stackexchange.com/questions/142/how-can-i-make-an-enumerate-list-start-at-something-other-than-1 (visited on 06/09/2016).

[18] Ulrike Fischer. *pdfTeX warning (ext4): destination with the same identifier (nam e{page.1}) has been already used, duplicate ignored*. May 2011. URL: http://tex.stackexchange.com/questions/142/how-can-i-make-an-enumerate-list-start-at-something-other-than-1 (visited on 06/10/2016).

[19] myrtille. *Bad formatting of Bibliography*. Jan. 2013. URL: http://tex.stackexchange.com/questions/89751/bad-formatting-of-bibliography (visited on 06/10/2016).

[20] ShareLatex. *Page numbering*. Unknown. URL: https://www.sharelatex.com/learn/Page_numbering#Numbering_styles (visited on 06/10/2016).

[21] Werner. *Use textwidth for Image width Only when it outgrows the page [duplicate]*. Jan. 2012. URL: http://tex.stackexchange.com/questions/41787/use-textwidth-for-image-width-only-when-it-outgrows-the-page (visited on 06/12/2016).

[22] Wikibooks. *LaTeX/Glossary*. June 2016. URL: https://en.wikibooks.org/wiki/LaTeX/Glossary (visited on 06/12/2016).

[23] Werner.

printglossaries is not generating anything for me. Mar. 2014. URL: http://tex.stackexchange.com/questions/43759/printglossaries-is-not-generating-anything-for-me (visited on 06/12/2016).

[24] Stefan_K. *Bibliography does not appear in TOC*. July 2010. URL: http://latex-community.org/forum/viewtopic.php?p=36233#p36233 (visited on 06/12/2016).

[25] ProgrammerInterview. *C++: What is a Virtual Destructor*. Jan. 2016. URL: http://www.programmerinterview.com/index.php/c-cplusplus/virtual-destructors/ (visited on 02/04/2016).

[26] Devin Watson. *Using Boost.asio with cmake?* 2013. URL: http://stackoverflow.com/questions/15290386/using-boost-asio-with-cmake (visited on 04/13/2016).

[27] Ranjit Bhatta. *C++ Constructors*. Unknown. URL: http://www.programiz.com/cpp-programming/constructors (visited on 05/02/2016).

[28] AlaskaJoslin. *Boost thread error: undefined reference*. Feb. 2011. URL: http://stackoverflow.com/questions/3584365/boost-thread-error-undefined-reference/35345376#35345376 (visited on 05/02/2016).

[29] Daviddoria. *CMake/Examples*. Nov. 2010. URL: https://cmake.org/Wiki/CMake/Examples (visited on 05/02/2016).

[30] Mat. *specifying link flags for only one static lib while linking executable*. Apr. 2011. URL: http://stackoverflow.com/questions/5693405/specifying-link-flags-for-only-one-static-lib-while-linking-executable/5693465#5693465 (visited on 05/02/2016).

[31] Neeraj Adhikari. *boost::asio failed to connect to localhost without WLAN*. July 2014. URL: http://stackoverflow.com/questions/22322506/boostasio-failed-to-connect-to-localhost-without-wlan/24830462#24830462 (visited on 05/03/2016).

[32] Drew_Benton. *A guide to getting started with boost::asio*. Jan. 2011. URL: http://www.gamedev.net/blog/950/entry-2249317-a-guide-to-getting-started-with-boostasio (visited on 05/09/2016).

[33] TutorialsPoint. *C++ Strings*. Unknown. URL: http://www.tutorialspoint.com/cplusplus/cpp_strings.htm (visited on 05/12/2016).

[34] cppreference. *std::strncpy*. Unknown. URL: http://en.cppreference.com/w/cpp/string/byte/strncpy (visited on 05/14/2016).

[35] cppreference. *memcpy*. Unknown. URL: http://www.cplusplus.com/reference/cstring/memcpy/ (visited on 05/15/2016).

[36] Unknown. *The GNU C Library: Deleting Files*. unknown. URL: http://www.gnu.org/software/libc/manual/html_node/Deleting-Files.html (visited on 05/19/2016).

[37] C++ Reference. *strcat*. Unknown. URL: http://www.cplusplus.com/reference/cstring/strcat/ (visited on 05/19/2016).

[38] *fopen - Linux programmer's manual*. (Visited on 05/19/2016).

[39] *remove - Linux programmer's manual*. (Visited on 05/19/2016).

[40] cppreference. *std::ifstream::ifstream*. unknown. URL: http://www.cplusplus.com/reference/fstream/ifstream/ifstream/ (visited on 06/04/2016).

[41] cppreference. *std::istream::gcount*. unknown. URL: http://www.cplusplus.com/reference/istream/istream/gcount/ (visited on 06/04/2016).

[42] cppreference. *std::ios_base::openmode*. unknown. URL: http://www.cplusplus.com/reference/ios/ios_base/openmode/ (visited on 06/04/2016).

[43] cppreference. *std::istream::read*. unknown. URL: http://www.cplusplus.com/reference/istream/istream/read/ (visited on 06/04/2016).

[44] Oliver Charlesworth. *How to store a const char* in std :: string?* June 2011. URL: http://stackoverflow.com/questions/6214160/how-to-store-a-const-char-in-std-string (visited on 06/06/2016).

[45] Boost. *Filesystem Home*. unknown. URL: http://www.boost.org/doc/libs/1_61_0/libs/filesystem/doc/index.htm (visited on 05/22/2016).

[46] Kerrek SB. *c++ boost::filesystem undefined reference to 'boost::filesystem3::path::root_name() const'*. Nov. 2011. URL: http://stackoverflow.com/questions/7972314/c-boostfilesystem-undefined-reference-to-boostfilesystem3pathroot-nam (visited on 05/22/2016).

[47] Nick. *Using boost::asio::io_service in another thread*. June 2013. URL: http://codelever.com/blog/2013/06/18/using-boost-asio-io-service-in-another-thread/ (visited on 05/24/2016).

[48] Klaim. *Which C++ signals/slots library should I choose?* Dec. 2008. URL: http://stackoverflow.com/questions/359928/which-c-signals-slots-library-should-i-choose (visited on 06/02/2016).

[49] MattyT. *Complete example using Boost::Signals for C++ Eventing*. Apr. 2009. URL: http://stackoverflow.com/questions/768351/complete-example-using-boostsignals-for-c-eventing (visited on 06/02/2016).

[50] Gerardo Hernandez. *How can I propagate exceptions between threads?* Sept. 2015. URL: http://stackoverflow.com/questions/233127/how-can-i-propagate-exceptions-between-threads (visited on 06/02/2016).

[51] Jonathan Potter. *Writing simple file-transfer program using boost::asio. Have major send
receive desync*. Nov. 2014. URL: http://stackoverflow.com/questions/26765701/writing-simple-file-transfer-program-using-boostasio-have-major-send-receive (visited on 06/04/2016).

[52] The Qt Company. *CMake Manual*. unknown. URL: http://doc.qt.io/qt-5/cmake-manual.html (visited on 05/23/2016).

[53] The Qt Company. *Using the Meta-Object Compiler (moc)*. unknown. URL: http://doc.qt.io/qt-4.8/moc.html (visited on 05/23/2016).

[54] Wikipedia. *Meta-object System*. unknown. URL: https://en.wikipedia.org/wiki/Meta-object_System (visited on 05/23/2016).

[55] Kitware Inc. *AUTOUIC*. unknown. URL: https://cmake.org/cmake/help/v3.0/prop_tgt/AUTOUIC.html (visited on 05/23/2016).

[56] The Qt Company. *User Interface Compiler (uic)*. unknown. URL: http://doc.qt.io/qt-4.8/uic.html (visited on 05/23/2016).

[57] Filipe Sousa. *[CMake] CMAKE_INCLUDE_CURRENT_DIR?* Mar. 2015. URL: https://cmake.org/pipermail/cmake/2007-March/013216.html (visited on 05/23/2016).

[58] GManNickG. *"does not name a type" error*. Jan. 2010. URL: http://stackoverflow.com/questions/2133250/does-not-name-a-type-error (visited on 06/03/2016).

[59] pnezis. *QT/C++ yet another issue about accessing UI files*. Jan. 2012. URL: http://stackoverflow.com/questions/8868916/qt-c-yet-another-issue-about-accessing-ui-files (visited on 06/03/2016).

[60] Macmade. *"Field has incomplete type" error*. Sept. 2012. URL: http://stackoverflow.com/questions/12466055/field-has-incomplete-type-error (visited on 06/03/2016).

[61] The Qt Company. *QLineEdit Class*. unknown. URL: http://doc.qt.io/qt-4.8/qlineedit.html (visited on 05/23/2016).

[62] user362638. *How to define an OnClick event handler for a button from within Qt Creator?* Aug. 2013. URL: http://stackoverflow.com/questions/18041876/how-to-define-an-onclick-event-handler-for-a-button-from-within-qt-creator (visited on 05/23/2016).

[63] Eli Bendersky. *QString to char* conversion*. Mar. 2010. URL: http://stackoverflow.com/questions/2523765/qstring-to-char-conversion (visited on 05/25/2016).

[64] The Qt Company. *QString Class*. unknown. URL: http://doc.qt.io/qt-5/qstring.html (visited on 06/01/2016).

[65] The Qt Company. *Signals & Slots*. unknown. URL: http://doc.qt.io/qt-4.8/signalsandslots.html (visited on 06/02/2016).

[66] ProgrammingKnowledge. *QT C++ GUI Tutorial 3- Qt Signal and slots (QSlider and QProgressBar)*. July 2013. URL: https://www.youtube.com/watch?v=iwe-uW0P7ys (visited on 06/02/2016).

[67] The Qt Company. *QFileDialog Class*. unknown. URL: http://doc.qt.io/qt-5/qfiledialog.html#getExistingDirectory (visited on 06/01/2016).

[68] ProgrammerInterview. *How to use GNU licenses for your own software*. Apr. 2014. URL: http://www.gnu.org/licenses/gpl-howto.html (visited on 02/04/2016).

[69] FreeMemory. *return statement vs exit() in main()*. Jan. 2009. URL: http://stackoverflow.com/questions/461449/return-statement-vs-exit-in-main (visited on 02/04/2016).

[70] librsync. *librsync*. Unknown. URL: https://librsync.sourcefrog.net/index.html (visited on 05/18/2016).

[71] williamkun. *williamkun/librsync-example*. Apr. 2013. URL: https://github.com/williamkun/librsync-example (visited on 05/18/2016).

[72] Jason. *When should I use perror("...") and fprintf(stderr, "...")?* Aug. 2012. URL: http://stackoverflow.com/questions/12102332/when-should-i-use-perror-and-fprintfstderr (visited on 05/19/2016).

[73] David Abrahams. *Error and Exception Handling*. unknown. URL: http://www.boost.org/community/error_handling.html (visited on 05/30/2016).