LOG8430
Architecture logicielle et conception avancée

Yann-Gaël Guéhéneuc
Fabio Petrillo
Département Génie Informatique et Génie Logiciel
École Polytechnique de Montréal, Québec, Canada
`yann-gael.gueheneuc[at]polymtl.ca`
`Fabio[at]petrillo.com`

April 18, 2017

Students names:

Isnaldo Francisco de Melo jr
Franck Brazier
Stephane Fagnon

Date of the submission: April 18, 2017

Contents

Introduction

ezf

Ring

According to the Ring web site , Ring [24] is a free software that allows its users to communicate in multiple ways. It can be used as a telephone, a messenger, for teleconferencing and media sharing. Its communication technology and portable library also makes Ring usable as a building block for IoT projects. The main goals directing Ring's development are:

- Making it simple for everyone to use complex technologies

- Propose ways to protect privacy and personal information of the user

- Use industry standards (well defined protocols, methods and portable languages recognized by industry experts)

- Prioritize connectivity (by using protocols such as ICE, STUN/TURN, UPnP and NAT-PMP which allow the user to join his peers even in difficult network configurations such as multiple firewalls and NAT)

- Comply with the system's user interface : Ring supports multiple platforms, does not limit the user to one interface and ensures that the user's choice of platform is respected.

- Stay free and improve technology and expertise.

- Use industry standards (well defined protocols, methods and portable languages recognized by industry experts)

The code documentation [23] was developed in Doxygen and the license of the system is GPLv3. Ring is involved in the Free Software philosophy; that is proved by the fact that it is a GNU[4] package [16].

Chapter 1

TP1 - Étude et analyse du Ring

1.1   Introduction

This document summarizes Ring's Architectural Views using the model of Philippe Kruchten [15], this model describes 4 Views and some scenarios used in the application.

Although this is not the main focus of this report, we also developed a new View, which summarizes the information specifically for complex software that performance is a key fact and several components are dependent.

Software architecture [22], as defined by Perry and Wolf, is a union of elements, form and rationale (also constraints). Form is the different constraints between the data, processing or connecting elements in comparison to the rationale, which can be defined as the system constraints itself.

1.2   Context

The Figure 1.1 summarizes part of the context that Ring is inserted in. To report bugs it uses
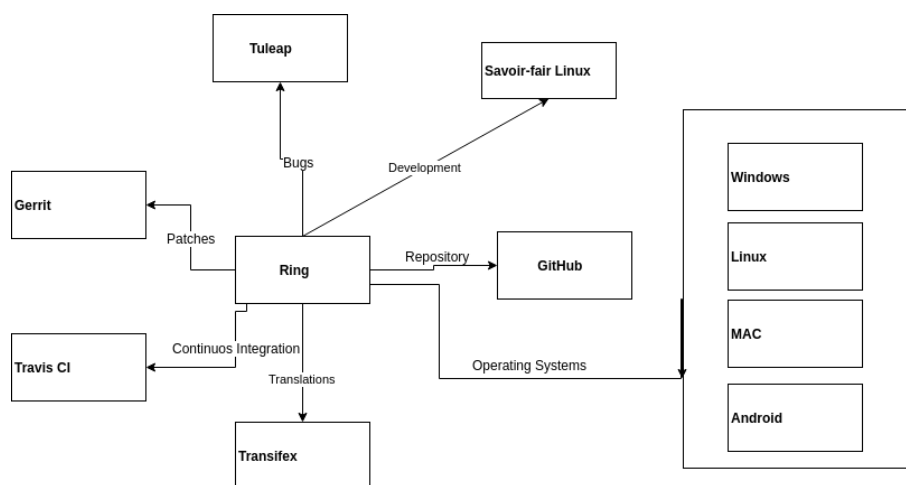
Figure 1.1: Context Diagram

Tuleap [29], this colaborative tool Tuleap lets tracking the issues on the system. So to contribute on the source code it is necessary to have an account on this tool. Patches can be sent on Gerrit [12]. Finally, translations can be sent by Transifex [28] software program. GTK+ is used to create the user interface, it is a toolkit and API.

Ring is meant to be a project on which everyone can work and participate, this point is clearly reflected in the context view and has some effects in the project architectural point of view. Each developer must take into account that everyone may read, try to understand the code, improve or change the code that's why, for example, each class, method, attribute, namespace must have an understandable name in English. Furthermore the "open" aspect pushes the developers to have a splitter architecture than usual for letting other not be submerged by too large implementation of a method or a class.

1.3   Statistical Analysis

To explore the Ring System [24], we used the software Understand [25]. This tool was developed by SciTools and has the purpose to do static codes analysis, in order to help programmers who work on large and complex legacy codes basis. It provides a good knowledge of the codes that are analyzed by giving information on functions, classes, variables, how they are used, called, modified, and interacted with. It also collects metrics about the code and provides different ways for the user to view it. Understand produces graphs, makes standard testing, dependency analysis, has an editor and a search functionality.

Understand produces graphs that show how the analyzed code connects (dependencies), how it flows (control flow graphs), what functions call other functions (call graphs), and also the possibility of having a customized display (a graph in which only the elements that the programmer is interested will be presented).

The testing functionality of Understand helps the programmer to check his code using either published coding standards, or a proper customized standards. The dependency analysis feature of Understand to see all the dependencies in the code and how they connect, through Understand's interactive graph or its textual Dependency Browser. Understand also integrates a powerful editor and a search functionality with multiple options.

1.4   Dynamic Analysis

To explore the software better we used several dynamic analysis approaches. First, we changed the source code to complement the debug console logs. Later, we compiled the linux part of the project *client-gnome* and *daemon* using the compiler option -finstruments-functions. This options generates automatically instrumentation in the begin and end of each function [5].
This instrumentation gives the possibility to dynamic analyze the code, in other words, to analyze the code during execution time. We made this by analyzing the code with tracing tools [1] and profilers (uftrace) [14].

1.5   Ring architecture

1.6   4+1 Model

The "4+1" model is a describing model of systems called software-intensives. The model is based on concurrent views that are able to describe in general [15]. Figure 1.3 illustrates the 4+1 model. The model is composed of 4 Views: Process View, Physical, Logical and Deployment. But also, it includes some scenarios. The 4+1 Ring's Model is presented below.

1.6.1   Development view

Usage

The Development View focus on representing the system as modules as well as their overall organization. Layers are used to represent the sub-systems. We present the package diagram for
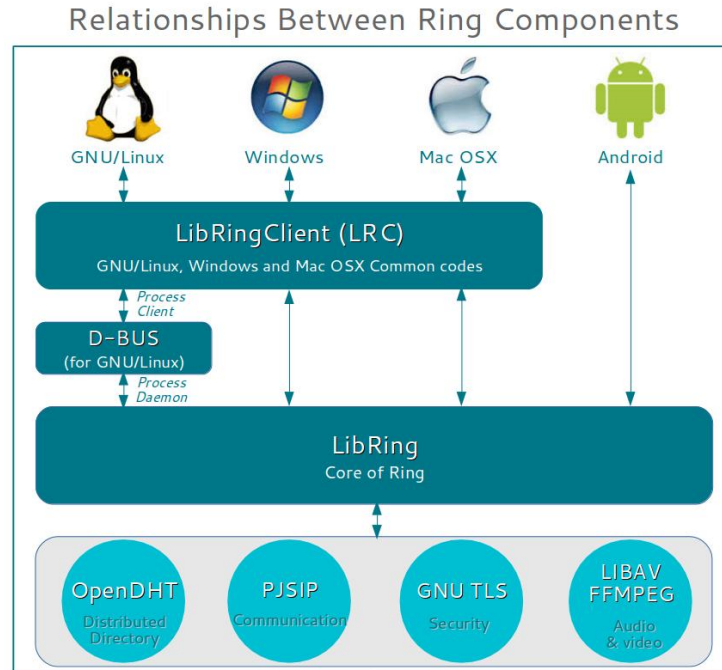
Figure 1.2: Ring Components
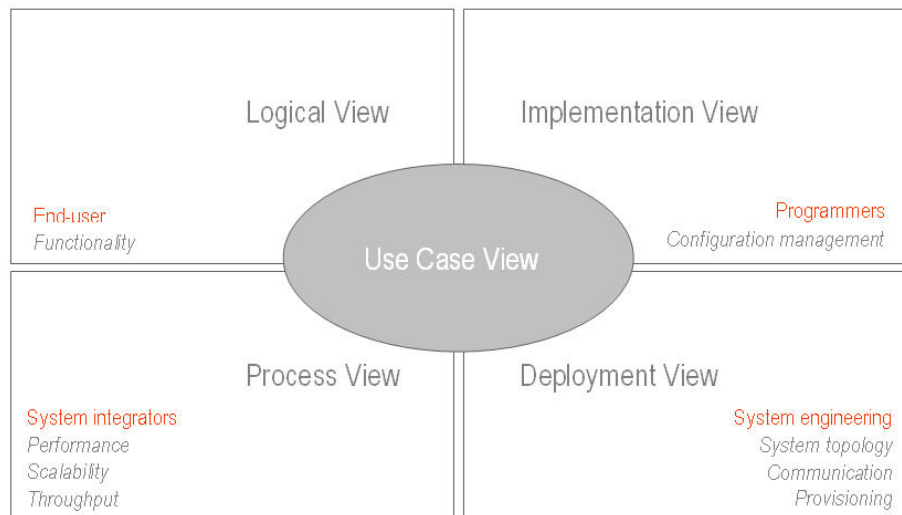
Figure 1.3: 4+1 architectural view model [21]

illustrating the Development view.

Diagram

The diagram is based on components and connector. The components represent the modules, subsystems and layers. The Connectors represent the relations among the components e.g.

dependency or compilation requirements. The Booch notation is used on this diagram.

Overview

(a) Ring Daemon

(b) Ring Client
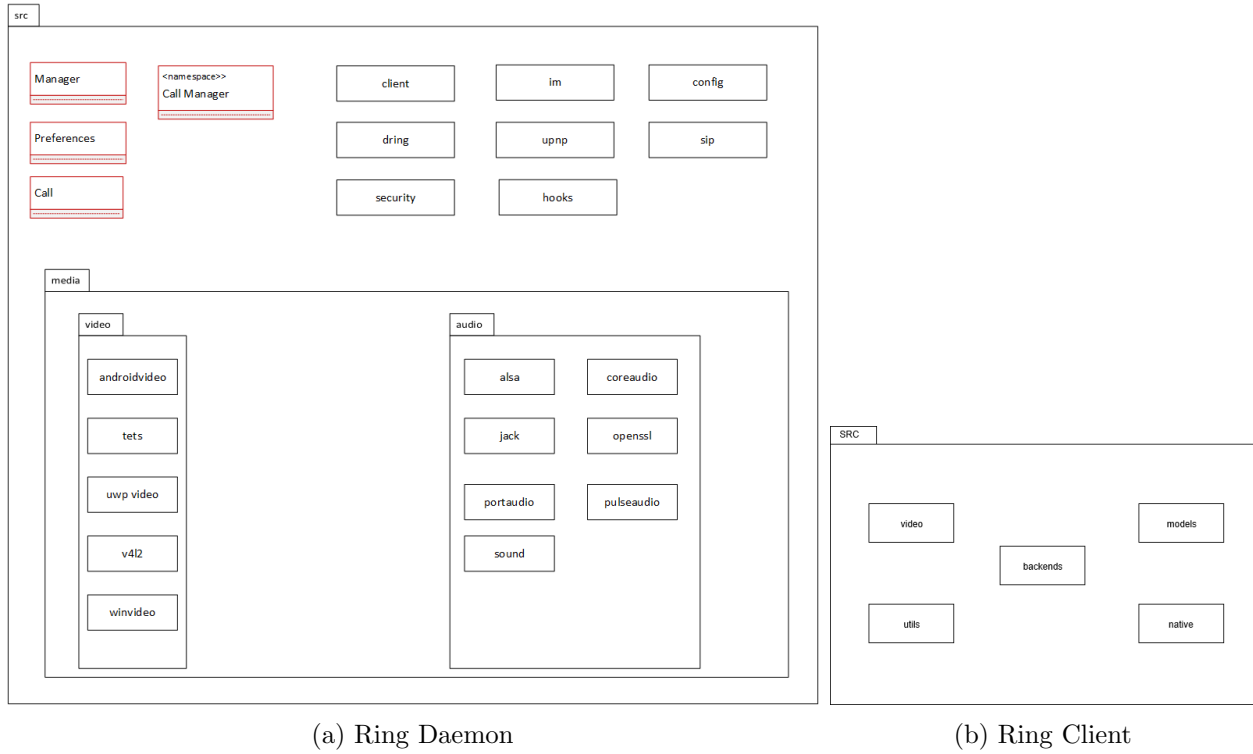
Figure 1.4: Package diagrams

Ring's Development View is shown in Figure 1.4. It has a main package, called ring-project then each Operating system has its own package. In linux, two components work together: the *daemon* and *linux-gnome*.

1.6.2   Logical view

Usage

The functional requirement are described on the logical architecture. In other words, the services for the final user. We chose to present different activity diagrams and one class diagram for the Logical View

Diagram

For the diagrams, it is possible to use an Object Oriented Approach or an data-driven approach using Entity Relationship diagrams. Developed by Grady Booch, the Booch notation (or Booch method), represent the classes as cloud shapes.

The Figure 1.5 presents the necessary steps for a User to take before making a Call:
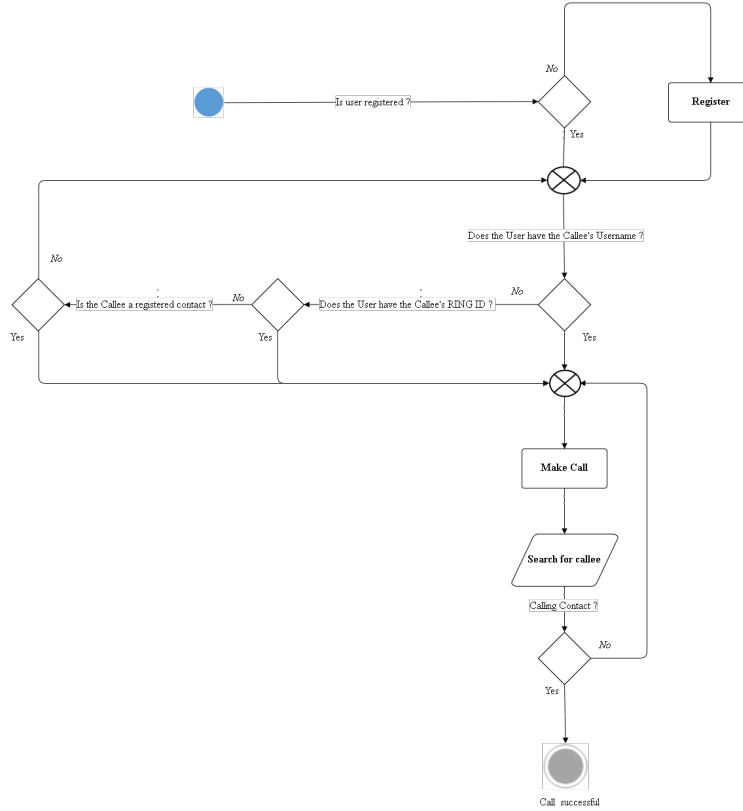
Figure 1.5: Activity diagram for Making a Call

- First, the User needs to be registered(already have a Ring Account); if he is not, he has to create a ring account.

- When the user is registered and launches Ring, the next step for him is to search for the contact to be called; there are 3 possible main Cases.

Case 1 : The User has the Callee's Username.

In this case, the user enters the callee's username in the search bar, if a user is found, the user makes the call (by clicking on a blue icon on the right hand side of callee's id) , then Ring searches for the user to contact him. If Ring finds it, the call is successfully made.

Case 2 : The User has the Callee's Ring ID.

In this case, the user enters the callee's Ring ID in the search bar, if a matching user is found, the user makes the call (by clicking on a blue icon on the right hand side of the callee's id) , then Ring searches for the user to contact him. If Ring finds it, the call is successfully made.

Case 3 : The User has the Callee in his/her contact list.

In this case, the user makes the call (by clicking on a blue icon on the right hand side of the callee's id) , then Ring searches for the user to contact him. If Ring finds it, the call is successfully made.
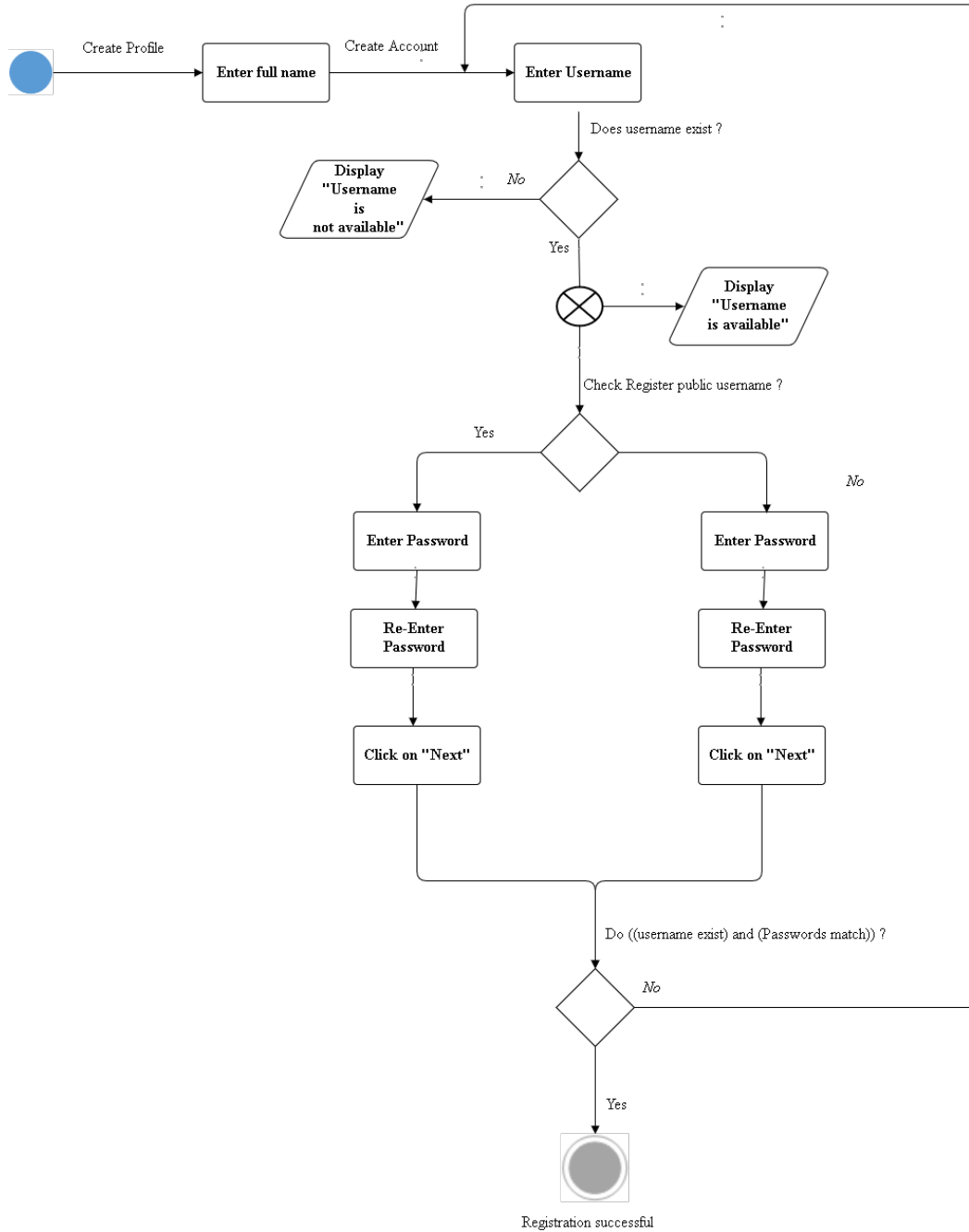
Figure 1.6: Activity diagram for complete registration

The Figure 1.6 presents the steps a User to be registered on Ring:

Step 1: The user must create a profile, which means that he needs to enter his full name.

Step 2: The user must create an account. First, The User puts in a Username (then Ring checks if the username already exists and displays the username's status at the right hand side of the username's textbox) Then the user enters a password, and re-enters the password again in another

textbox. Finally the user clicks on "Next", if the entered username does not exist on Ring and the entered passwords match the registration is successful.

Overview
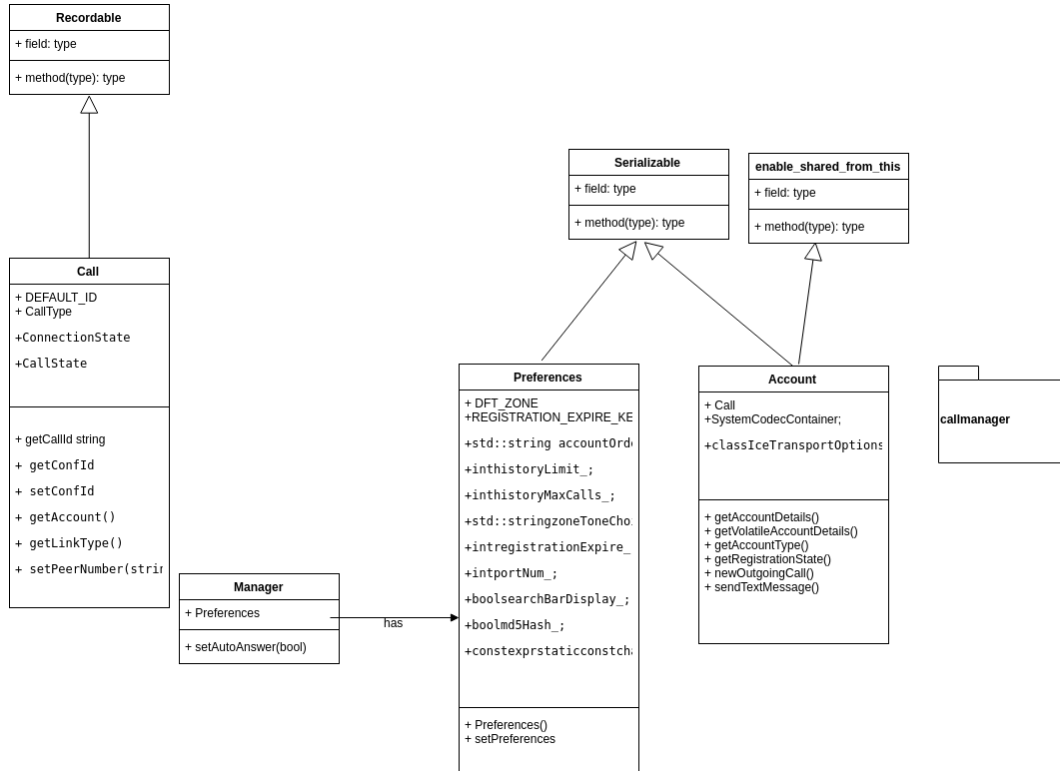
Figure 1.7: Representation of the Logical View

In Ring's architecture, the Logical View is represented in Figure 3.4. This figure shows specifically the classes involved in Calling: callmanager, manager, and call. The complete diagram for all the ring daemon project can be found in Appendix ??.

Since Ring's architecture works using states, calls have a state property, represented in Figure 1.8: Call States. All call start in the Inactive state and finish in the Over state. Wh those call states the developers add a connection state for specifying the different status, these states are : DISCONNECTED, TRYING, PROGRESSING, RINGING and CONNECTED. They also differentiate if the call is INCOMING, OUTGOING or MISSED in their implementation.
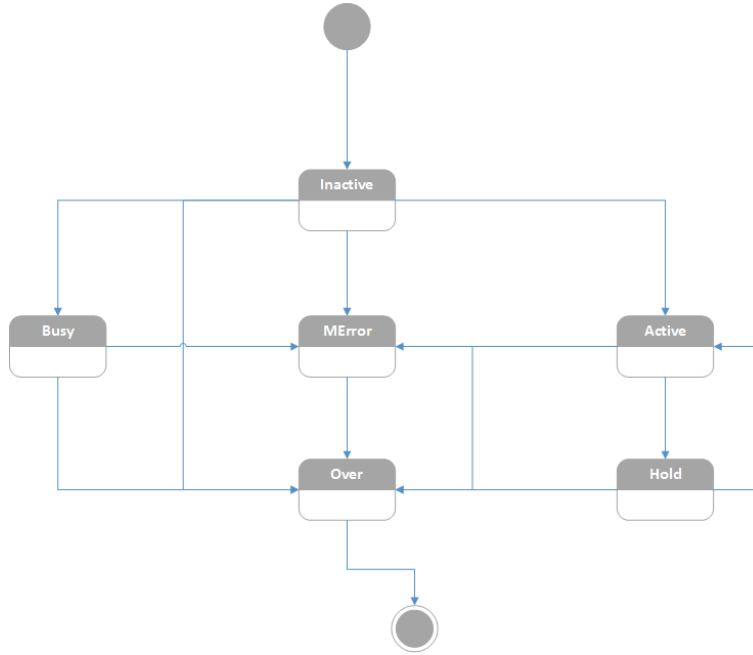
Figure 1.8: Call States

1.6.3 Process view

Usage

The non-functional requirements are described in this view, which varies according to the level of abstraction. As an illustration of the process view we extract a sequence diagram.

Diagram

According to the original paper, the diagrams represent components and connectors. The diagram describes processes, simplified processes and periodic processes. The connectors represent messages, events and connections among the processes.
The notation used is based on the Booch notation specifically for the Ada programming language.

The sequence diagram is presented in Figure 3.2

Overview

In Ring, the Process View 3.2 shows the sequence specifically for a call, with the operations among the classes involved.

1.6.4 Physical view

Usage

The Physical View is used to represent non-functional requirements of the system.
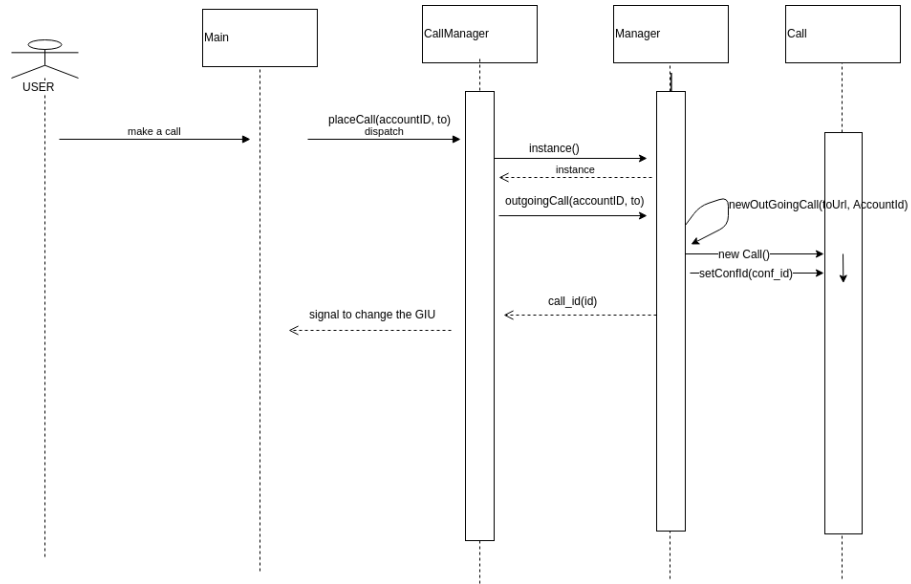
Figure 1.9: Sequence Diagram

Diagram

This diagram represents components and connectors. The components are physical parts, including processors and other devices. The connectors represent communication lines among the components.
The UNAS from TRW is a data-driven notation used to map those components.
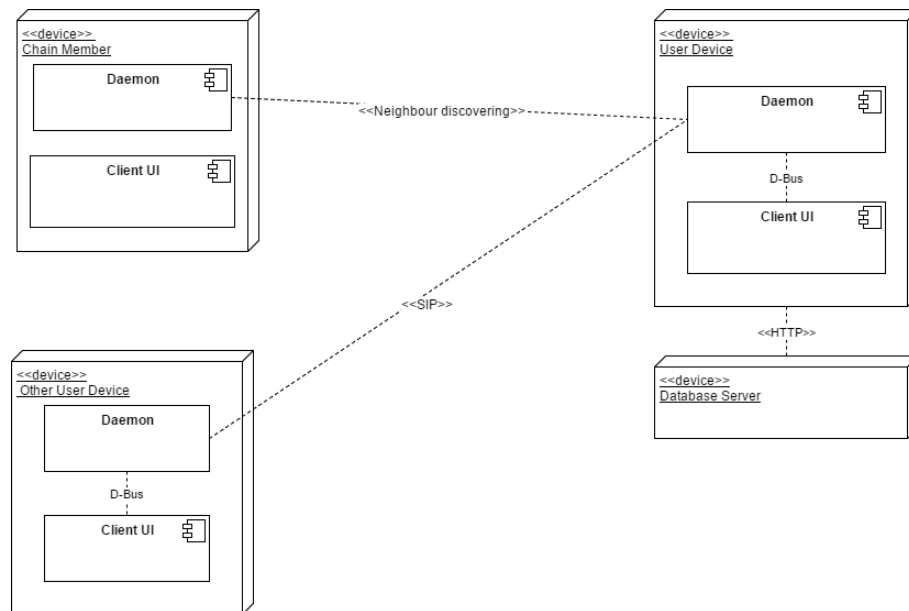
Figure 1.10: Deployment Diagram for Ring system

15

Overview

In Ring, the Physical View is represented in Figure 1.10. This figure shows the deployment of Ring on the different devices. Since Ring is a multi-distributed system, basically all the devices implied are the Ring users device except for the public username registration that needs a central database server.

We chose to present three times the device for illustrate its three different role as an member of a block-chain and as participant of a chat. The devices use SIP (Session Initial Protocol) to communicate with each other and make a call (or messaging). However the most important feature of the system is every single device is a chain link for a distributed hash tree (DHT). In every device, two components are present, a client responsible for the user interface and a daemon for all the communications with the outside world.

1.6.5 Scenarios

Usage

The usage of the scenarios is to describe interactions among objects and among processes of the system.

Diagram

The diagrams show interactions among each of the elements of the application and its different users

The Table 2.5 describes the user account creation and user login. The Table 2.4 describes the user contact search and calling. The Table 2.3 describes the user editing his profile.

Figure 1.11: User use case

Title	User Account Creation	Authentication
Description	User registers into the system	User logs in
Primary Actor	User	User
Preconditions	Must create profile and create account	Must be registered on Ring
Post conditions	None	Enters the system
Main Success Scenario	1. User enters his fullname 2. User enters a username (that is not used on ring) 3. User enters a password 4. User re-enters a password 5. The password entered by the user matches 6. User clicks on Next 7. Ring registers the User	1. User is registered 2. User launches Ring
Extensions		

Table 1.1: Table Scenarios 1

Title	User contact search	Call Functionality
Description	User searches for another user	User calls other users
Primary Actor	User	User
Preconditions	Must be registered on Ring and logged in	Must be registered on Ring and logged in
Post conditions	None	None
Main Success Scenario	1. User is registered 2. User launches ring 3. User types a contact's name or username or Ring ID in the search bar	1. User is registered 2. User launches ring 3. User types a contact's name or username or Ring ID in the search bar 4. User clicks on the blue icon on the right hand side of the contact
Extensions	1. Search by name *The User puts in the search bar the name of the other user he is looking for if that user is already part of his contacts.* 2. Search by public username *The User puts in the search bar the username of the other user he is looking for.* 3. Search by Ring ID *The User puts in the search bar the Ring ID of the other user he is looking for.*	1. Audio Calls *The User can make audio calls* 2. Create Conference calls *The User can add another user to a current call, or make the call a conference* 3. Video Calls *The User can make audio calls*

Table 1.2: Table Scenarios 2

Title	Call Functionality	Manage profile
Description	User receives a call	User edits his profile information
Primary Actor	User	User
Preconditions	Must be registered on Ring and logged in	Must be registered on Ring and logged in
Post conditions	None	None
Main Success Scenario	1. User is registered 2. User launches ring 3. User receives a call 4. User chooses to accept or reject the call	1. User is registered 2. User launches ring 3. User goes to his settings 4. User has the choices among these operations: update profile picture, manage history, call recording settings, enable notifications, checking update settings
Extensions		

Table 1.3: Table Scenarios 3

1.6.6 Performance View

The original scope of the 4+1 model is complete and consistent, it covers all the parts in a software.

However, as an addition we are proposing a new view of the system, the Performance View. This View is a summary for software that the performance has an important role. It contains several parts:

- Bottlenecks — Displays a bar chart featuring the five components of the pipeline: Log Reader, Source Engine, Communication, Target Engine, and Target Database. The results of the bar chart will help you isolate which components are causing bottlenecks.

- Latency — Displays a graphical summary of latency for the selected subscription.

- Statistics — Displays a graph of the performance of the selected metrics with a statistics count area at the bottom.

This view is partially based on IBM's InfoSphere [11] and other profiling tools available in the market, as uftrace [14]. A draft of the Performance diagram can be summarized in the Figure 1.12. This Figure shows several components and their respectively profiling information. Consequently, in this example, Component 1 has a main function, which calls two other functions and so on. This can show specifically how much time a function is spent in a specific function of the system.

1.7 Design Patterns

The following patterns were found in this project:
Factory Factory Method a design pattern used to creating objects as in template method[9]. In the project, the calls, described in the diagrams above are created as a factory.
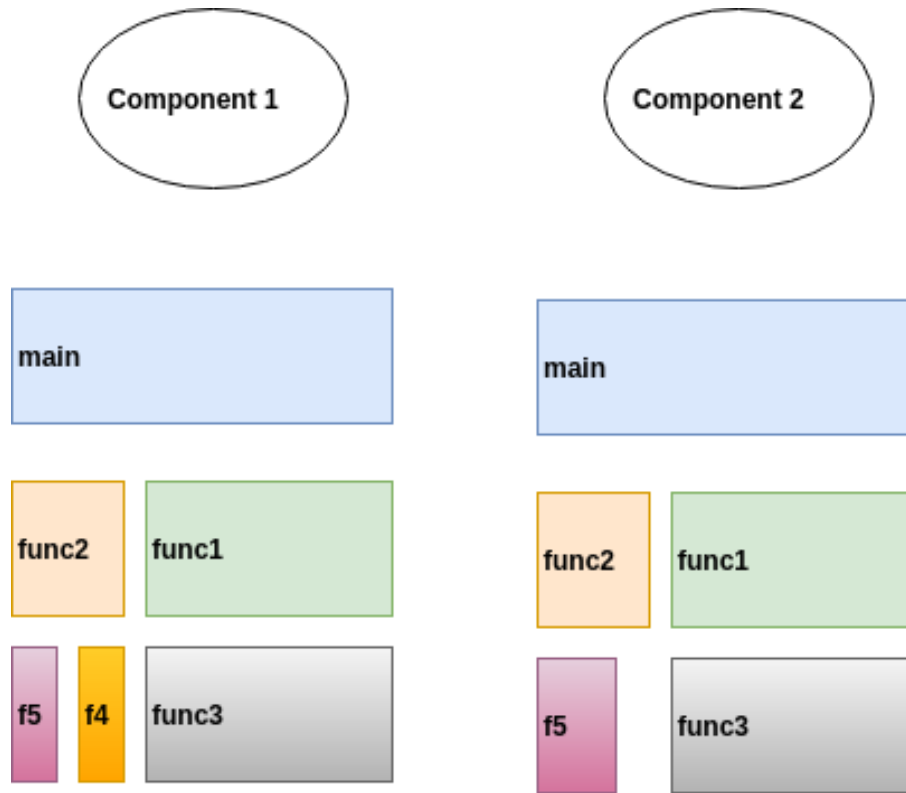
Figure 1.12: Suggested Performance Diagram

MVC The Ring's architecture overall presents the Model View Controller model[10]. This pattern specifies that an application consist of a data model, presentation information, and control information, and Ring's follow this pattern clearly. The daemon is the model whereas the gnome-client is the view, more precisely the GTK+ library implements the view and the code in the gnome-client implements the model.

1.8 Conclusion

The positive aspects of the Ring project are: a free project for communication and the respect for the privacy of the users. The system is supported by Savoir-faire Linux.

However, some drawbacks of Ring can be described:

First of all, as final user highlight, the interface for Windows do not seem to be user-friendly.

Secondly, the documentation is not clear enough for a better understanding of the system. As an example, in the code, it difficult to define in Object Oriented terms the concept of **enable shared from this**. The question, which still remains for further investigation, is whether is it a Objected Oriented design or a implementation need.

Finally, as a small-revolution, as the developers call it, Ring does not bring enough features to really make such a great difference for the common people in confrontation to a competitor, such as Skype.

Chapter 2

TP2 - Réusinage Architectural du code Ring

2.1   Introduction

This work summarizes the analysis of Ring source and presents some suggestions considering several code analysis tools and metrics. First, some metrics extracted from the code are presented, followed by an analysis of their meaning for the project.

Then, several Design patterns used on the code are presented with examples and diagrams. This part is followed by the anti-patterns of the code. The patterns and anti-patterns are described, illustrated by an example found in the code and discussed over their advantages (for patterns) or inconveniences (for anti-patterns).

Finally, a summary is presented including suggestions along with code refactoring and diagrams to explain the suggestions.

2.2   Code Quality Analysis

The analysis of the code's quality will be performed with two tools : Code Scene and Better Code Hub. The following is a report exposing the results obtained from each of the metrics.

2.2.1   Code Scene

Code Scene provides informations about the project that is being analyzed and the code of that project.

Language	Files	Code	Comment	Blank
C++	137	38309	5705	7828
C	162	12699	7924	4446
XML	8	2851	18	293
Python	13	2063	409	865
XSL	2	1186	39	132
HTML	7	671	0	49
DOS Batch	22	471	0	165
Text	5	227	0	93
Objective C++	2	224	53	47
YAML	1	207	0	0
CSS	1	193	0	44
Shell Script	5	127	33	20
Makefile	1	39	0	9
.NET Solution Files	1	39	0	0
Ignored	258	0	0	0

Table 2.1: Code Scene Metrics of Ring Daemon Project

The Table 2.1 is a table resulting of Code Scene's scope analysis of the project. It shows us the following:

- **LOC: Ring daemon project has 59,306 lines of codes**

- **There are 38,309 lines of codes written in C++**

Main Suspects
ring-daemon/src/sip/sipvoiplink.cpp
ring-daemon/src/sip/sipaccount.cpp
ring-daemon/src/ringdht/ringaccount.cpp
ring-daemon/src/sip/sipcall.cpp
ring-daemon/src/manager.cpp

Table 2.2: Code Scene Hotspots detected from Ring Daemon Project

The Table 2.2 shows the different Hotspots (the files of the project that present potential risks for the project).

Coupled Functions	Degree of coupling(%)	Average Revisions	Similarity(%)
invite_session_state_changed_cb transaction_state_changed_cb	61	21	23
SIPVoIPLink::requestKeyframe SIPVoIPLink	55	20	21
invite_session_state_changed_cb sdp_media_update_cb	53	23	0
sdp_media_update_cb transaction_state_changed_cb	53	21	21
SIPVoIPLink::SIPVoIPLink transaction_state_changed_cb	52	21	0
SIPVoIPLink::handleEvents transaction_state_changed_cb	51	22	17
SIPVoIPLink::handleEvents SIPVoIPLink	49	27	17
SIPVoIPLink::SIPVoIPLink invite_session_state_changed_cb	47	23	0
SIPVoIPLink::SIPVoIPLink SIPVoIPLink	46	26	0
SIPVoIPLink::handleEvents invite_session_state_changed_cb	46	24	0
SIPVoIPLink::SIPVoIPLink SIPVoIPLink::guessAccount	46	24	0
SIPVoIPLink::SIPVoIPLink sdp_media_update_cb	44	23	0
SIPVoIPLink::handleEvents sdp_media_update_cb	43	23	13
SIPVoIPLink::guessAccount invite_session_state_changed_cb	42	24	17
SIPVoIPLink::SIPVoIPLink SIPVoIPLink::handleEvents	42	24	0
sdp_media_update_cb SIPVoIPLink	39	26	15
invite_session_state_changed_cb SIPVoIPLink	38	26	0
transaction_request_cb SIPVoIPLink	36	64	0
sdp_media_update_cb transaction_request_cb	31	60	0
SIPVoIPLink::handleEvents transaction_request_cb	26	61	0
SIPVoIPLink::SIPVoIPLink transaction_request_cb	26	61	0
invite_session_state_changed_cb transaction_request_cb	24	61	0
transaction_request_cb transaction_state_changed_cb	23	59	0
SIPVoIPLink::guessAccount transaction_request_cb	22	61	0
sdp_create_offer_cb transaction_request_cb	22	57	0
SIPVoIPLink::requestKeyframe transaction_request_cb	20	55	0

Table 2.3: Internal Temporal Coupling for sipvoiplink.cpp

Coupled Functions	Degree of coupling(%)	Average Revisions	Similarity(%)
SIPAccount::setAccountDetails SIPAccount::unserialize	96	14	22
SIPAccount::getAccountDetails SIPAccount::serialize	91	12	19
SIPAccount::serialize SIPAccount::unserialize	88	13	23
SIPAccount::serialize SIPAccount::setAccountDetails	83	12	20
SIPAccount::getAccountDetails SIPAccount::unserialize	81	14	20
SIPAccount::getAccountDetails SIPAccount::setAccountDetails	76	13	0
SIPAccount::SIPStartCall SIPAccount::newOutgoingCall	40	33	19

Table 2.4: Internal Temporal Coupling for sipaccount.cpp

Analysis

The results of the metrics provided by Code Scene show that the project is mainly written in C++, which was expected. The other noticeable languages present in descending order of importance in terms of lines of codes are: C, XML, Python, XSL. Python scripts were also expected to install the tool. Moreover, there are also other languages but they represent a little portion of the code: HTML, DOS Batch, Objective C++,YAML,CSS.

The comparison of the complexity trend graphs of the hotspots shows that ringaccount.cpp is the file that gained the most in complexity within the four past months (December-March).

The results of the Internal Temporal Coupling analysis show that, except the manager.cpp hotspot, the other hotspots have a lot of internally coupled functions. The highly coupled files are frequently a cpp file and its corresponding header file; which makes sense since most of the Function and Method declarations of these cpp files are found in their corresponding header files. The main focus was on the Internal Temporal Coupling analysis of each hotspot, since coupling was an important factor to take in consideration in a code's quality assessment.

2.2.2 Better Code Hub

Better Code Hub is a tool that makes the analysis of projects' code to asses their quality. That analysis is based on 10 main guidelines that we believe are inspired from Joost Visser's book: Building Maintainable Software.The guidelines are criteria used by the tool (Better Code Hub) to evaluate the quality of the code from 0-10 points scale.
The analysis of the Ring Daemon project by Better Code Hub gives a compliance mark of 5/10. There are no specific criteria to appreciate the overall quality of the code from the compliance marks. Below is presented the guidelines list:

Coupled Functions	Degree of coupling(%)	Average Revisions	Similarity(%)
RingAccount::setAccountDetails RingAccount::unserialize	86	19	28
RingAccount::loadArchive RingAccount::makeArchive	83	16	27
RingAccount::getContactHeader RingAccount::getToUri	82	17	0
RingAccount::acceptTrustRequest RingAccount::sendTrustRequest	81	14	22
RingAccount::getAccountDetails RingAccount::serialize	77	18	18
RingAccount::getAccountDetails RingAccount::setAccountDetails	76	20	19
RingAccount::createRingDevice RingAccount::loadAccount	72	17	0
RingAccount::getAccountDetails RingAccount::unserialize	71	21	17
RingAccount::loadAccount RingAccount::loadArchive	71	20	17
RingAccount::doUnregister RingAccount::serialize	71	14	0
RingAccount::loadAccount RingAccount::makeArchive	70	17	20
RingAccount::doUnregister RingAccount::setAccountDetails	70	16	0
RingAccount::serialize RingAccount::setAccountDetails	70	16	24
RingAccount::serialize RingAccount::unserialize	64	17	30
RingAccount::doUnregister RingAccount::unserialize	64	17	0
RingAccount::getAccountDetails RingAccount::handlePendingCall	63	19	14
RingAccount::createRingDevice RingAccount::getAccountDetails	58	17	0
RingAccount::loadAccount RingAccount::loadIdentity	56	23	0
RingAccount::doUnregister RingAccount::getAccountDetails	55	18	0
RingAccount::SIPStartCall RingAccount::getAccountDetails	54	19	14
RingAccount::doRegister_ RingAccount::sendTextMessage	52	44	0
RingAccount::incomingCall RingAccount::newOutgoingCall	52	34	32

Table 2.5: Internal Temporal Coupling for ringaccount.cpp

Coupled Functions	Degree of coupling(%)	Average Revisions	Similarity(%)
SIPCall::offhold dtmfSend	93	15	0
SIPCall::updateSDPFromSTUN dtmfSend	90	16	0
SIPCall::offhold SIPCall::updateSDPFromSTUN	84	17	0
SIPCall::transfer dtmfSend	84	13	0
SIPCall::transfer SIPCall::updateSDPFromSTUN	82	15	0
SIPCall::offhold SIPCall::transfer	78	14	24
SIPCall::hangup SIPCall::peerHungup	75	28	0
SIPCall::sendSIPInfo SIPCall::transfer	75	15	0
SIPCall::onAnswered SIPCall::onMediaUpdate	72	17	33
SIPCall::sendSIPInfo SIPCall::updateSDPFromSTUN	70	17	0
SIPCall::sendSIPInfo dtmfSend	70	16	12
SIPCall::offhold SIPCall::onhold	69	22	21
SIPCall::onhold dtmfSend	68	21	0
SIPCall::offhold SIPCall::sendSIPInfo	66	17	0
SIPCall::onhold SIPCall::updateSDPFromSTUN	63	22	0
SIPCall::sendSIPInfo SIPCall	61	20	0
SIPCall::setCallMediaLocal SIPCall	60	17	0
SIPCall::transfer SIPCall	58	17	0
SIPCall::hangup SIPCall::sendSIPInfo	57	25	21
SIPCall::onhold SIPCall::setCallMediaLocal	57	19	20
SIPCall::onhold SIPCall::transfer	56	20	20
SIPCall::updateSDPFromSTUN SIPCall	56	20	0
SIPCall::hangup SIPCall	55	27	0
dtmfSend SIPCall	55	18	0

Table 2.6: Internal Temporal Coupling for sipcall.cpp

No significant temporal coupling between functions. Keep up the good work!

Table 2.7: Internal Temporal Coupling for manager.cpp

Results

From the previous analysis, we can now understand better the criteria that led to a compliance mark of 5/10.

Analysis

The results, presented on the last part, lead to the following conclusions: The compliance mark of 5/10 can be justified by the violation of some guidelines BetterCodeHub's. Regarding that, writing short units of codes, writing simple units of code, and automating tests would enhance the quality of the project and therefore increase the compliance mark.

Within most of the guidelines, some refactoring candidates were recurrent; notably Ring. RingAccount, account.cpp and videomanager.cpp .

2.3 Ptidej

Ptidej[**ptidej**] is a tool suite dedicated to the analysis and maintenance of object-oriented architectures. The main resources used were: (1) SAD and (2) EPI.

2.4 Design Patterns

Software design patterns are basically general reusable solutions to several common problems found in software engineering problems. The following patterns were found in this project:

2.4.1 Factory Method

Definition

Factory Method is a design pattern in which objects are created through an interface and class instantiation is deferred to subclasses (subclasses decide which class to instantiate) [9]. The figure 2.1 illustrates the use of the factory method in ring project.

Motivation

The Factory Design Pattern is one of the most used design patterns in modern Orientated Object Programming languages (for example Java and CSharp). It comes in different variants and implementations. This pattern is introduced and further explained in the Gang of Four (GoF) book named Design Patterns: Elements of Reusable Object-Oriented Software patterns [2]

Intent

This pattern's aim is to create an object without exposing the instantiation logic to the client. It's kind of a *blackbox* from which the client asks for an item (in this case an instance of a class or a derivated one) and it gives the item back. The given item refers to the newly created object through a common interface for the use of the client.

Identification

The identification of this design pattern was simple since the factories are explicit on the source code.

Example of Code

```
std::shared_ptr<Account>
AccountFactory::createAccount(const char* const accountType,
                             const std::string& id)
{
    if (hasAccount(id)) {
        RING_ERR("Existing account %s", id.c_str());
        return nullptr;
    }

    std::shared_ptr<Account> account;
    {
        const auto& it = generators_.find(accountType);
        if (it != generators_.cend())
            account = it->second(id);
    }

    {
        std::lock_guard<std::recursive_mutex> lock(mutex_);
        accountMaps_[accountType].insert(std::make_pair(id, account));
    }

    return account;
}
```

In *"src/account_factory.cpp"*.

Analysis

The Factory Pattern is used to create instances of a specific class' descendant in a more easy way. In Ring code the pattern is used to create several calls or accounts; it makes the call creating uniform and helps to avoid having troubles collecting them from a data structure. This pattern is useful when there are several inherited classes.When it is the case, the factory returns the good instance of the class, furthermore the developer can easily add an inherited class to the design. Then, the only work to do is to implement the class and modify the factory to include it. Therefore, the factory improves the maintainability of the system.

Diagram

Figure 2.1 illustrates the Factory used for instantiate a *ring::Account* class. There are two inherited classes in this case *ring::ringAccount* and *ring::SIPAccount*, the factory returns the good
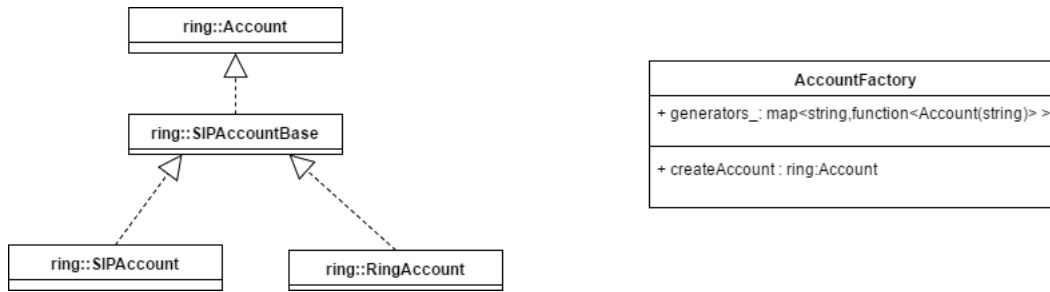
Figure 2.1: Factory Method illustration

one instance for an Account given the parameters of the method *createAccount*. Precisely, this is done by getting the value of a ¡key,value¿ map where the key is the type of needed account and the value is a function returning an instance of that type. The map is called *generators_* in the code as shown in the Figure 2.1.

2.4.2 Singleton

Motivation

The Singleton pattern is used in Ring project because there are several classes that must have only one instance.In order to implement the single instantiation, two methods could be used; either the singleton pattern or a static class. The designers must have chosen the former because the inheritance can be fully exploited along with polymorphism and an interface can be created as well as implemented. It assures simplicity in adding features while implementing a descendant class.

Intent

This pattern intents to ensure that only one instance of a class has been created.

Identification

We have found an example of this pattern while examining the code, a call for *getInstance()* lead us to a singleton class (we assume that that class can not be instantiated more than once).

Example of Code

Here is an implementation of a Meyers-Singleton from the file *"src/smartools.cpp"*

```
Smartools& Smartools::getInstance()
{
    // Meyers-Singleton
    static Smartools instance_;
    return instance_;
}
```

32

2.4.3   Model View Controller

The overall Ring architecture is designed based on the Model View Controller pattern (MVC)[10]. In that pattern, an application must be made of a data model, an information presenter (view), and an information controller (controller).

Motivation

This split of roles allows the ring system to easily be multi platform; the view depends a lot on the platform on which the system is installed. Even if each of the platforms has some specific characteristic, this pattern allows the designers and the developers to design a common code for a C++ project instead of creating a version for each one of the platforms on which the software is supposed to be deployed. Therefore,the MVC pattern makes the development shorter; it also lets the developers change the view without touching the model if they want to. For example if an update of an application's User Interface needs to be performed, it can be done without affecting the model and the controller.

Identification

Ring's architecture is divided into three parts: the GTK (that represents the view), the Daemon, (that represents the model), and Gnome-client (that represents the controller).

Diagram

The Figure 2.2 illustrates the MVC in Ring project. The daemon is the model whereas the gnome-client is the view, more precisely the GTK+ library implements the view and the code in the gnome-client implements the model.
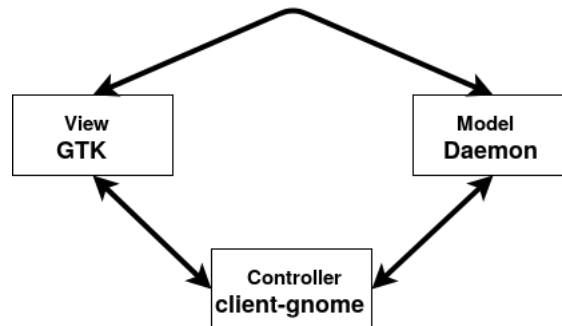

Figure 2.2: Model View Control Diagram

2.4.4   Template Pattern

Template pattern is a design pattern in which an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class.[30]

Intent

The goal of using a template pattern is to reduce time and lines of code.

Example of Code

As an illustrative example, we have a Shared pointer header that is used in several classes through a template Call.

Analysis

The motivation behind the implementation of the template is to reduce the time spent to create new methods and properties that can be used for several classes.

Identification

The Template pattern in Ring was identified in the source code.

2.5   Anti Pattern

This section describes several anti-patterns found on the code. A reference for them was the book Antipatterns [19].

2.5.1   Dead Code

Explanation

Throughout the software development process and the build of newer versions, some codes can become obsolete. The consequence is that some parts of the code can become useless or unnecessary.

Identification

Part of the identification phase of obsolete code is facilitated by the use of IDE's and other tools. In our case, the use of QTCreator framework facilitated the finding. However, the framework also gave false positives specifically related with mutex variables.

Code example

```
Line 426 Method: addSubCall Class: Call
std::lock_guard<std::recursive_mutex> lk (callMutex_);
Line 426 Method: addSubCall Class: Call
std::lock_guard<std::recursive_mutex> lk (this_.callMutex_);
Line 441
std::lock_guard<std::recursive_mutex> lk (this_.callMutex_);
Line 406
(Call::CallState new_state, Call::ConnectionState new_cstate, UNUSED int code)
```

Analysis

The problem with this anti-pattern is the unnecessary time spent to find. Also unnecessary variables can increase the complexity of the code and make it difficult to understand.

Solution

To solve the Dead code it is necessary to remove the unused variables from the source code.

2.5.2   Complex Class

Explanation

Some classes and methods are unnecessarily complex. In our case, chained if statements can reduce the readability of the code.

Code example

The portion of code with more than 50 lines, including complex functions executions, and several chained ifs, and mutex call. An example of chained ifs is present in the body of the Class Call, between lines 408 and 460 (a total of 52 lines).

Another example is the *"sipvoiplink"* file in which *try_respond_stateless* includes a lot of if-return statements all along representing more than 120 lines.

Analysis

Ring's call architecture is based on a finite state machine with different states and the class call is responsible of changing those states. However, the verification of current and new states in chained ifs does not seem to be the best approach, because it makes the code lowly readable. This specific example is related to the fact that only few people worked on the code and since the current version works, there was no need to change the code later so the code stayed as it was.

Solution

A better way to solve this issue would be to refactor the portion of code related with this big sequence of chained ifs. Figure 2.3 is a sketch that illustrates a way of reducing the complexity of the file call.cpp.

2.5.3   Blob

Explanation

The Blob is an anti-pattern in which the role of a class is note precisely defined. The class endorses several responsibilities that may not be related with one another.

Code example

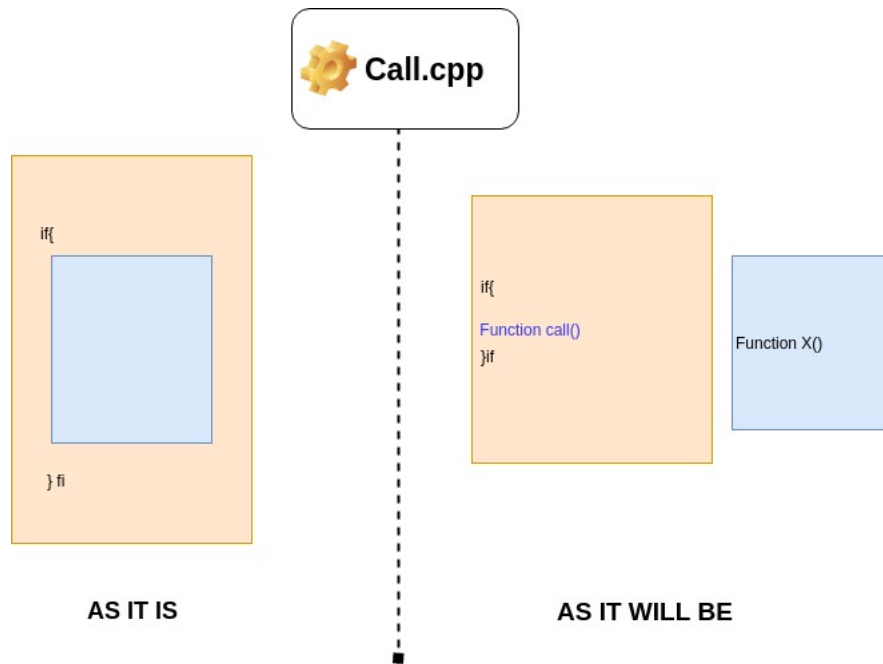The classes manager.cpp and account.cpp are examples of Blob codes.

Figure 2.3: Sketch illustrating a way of reducing the complexity of call.cpp

Analysis

The problem of this code is similar with the previous one, we can easily guess that, ring developers thought this: "if the code works, there is no need to change it" because throughout the new releases, the Blobs identified were neither changed, nor improved. At their early days, the anti-patterns were probably normal Classes with methods, but with the time and the growing needs of Ring new methods must have been implemented; which gave birth to the actual anti-patterns.

Solution

A proper solution would be to refactor the two files: manager.cpp and account.cpp in a way that properly defines each of the files' roles.

2.6   Problem Exposed

2.6.1   Problem I: Over-dependency Problem

Description

Considering the context on which Ring is inserted, as a open source project, a direct dependency on external libraries as GTK is a limitation. This limitation can be overcome by just a creation of an intermediary class which will serve as a interface for request using GTK. Using the static code analysis we were able to find the direct dependencies: ringwelcomeview.cpp, Username Registration Box Private.cpp, webview chat context menu.cpp.

Proposal of Solution

For the problem discussed above we are proposing a class which will do an interface with the classes that use GTK+ currently.Figure 3.1 provides a high-level illustration of our suggestion for the GTK+ dependency. We implement our proposal with the QtCreator[27] IDE, which also enables the use of Gtest for unit tests.

Figure 3.4 represents the class diagram of the solution. By adding a level of indirection and encapsulating the GTK+ calls would reduce the dependency of GTK+ and further external changes on this library will not affect the application.

Figure 3.2 represents a simplified version of the sequence diagram of the proposed solution, whereas the Figure 3.3, represents a simplified activity diagram of the solution.
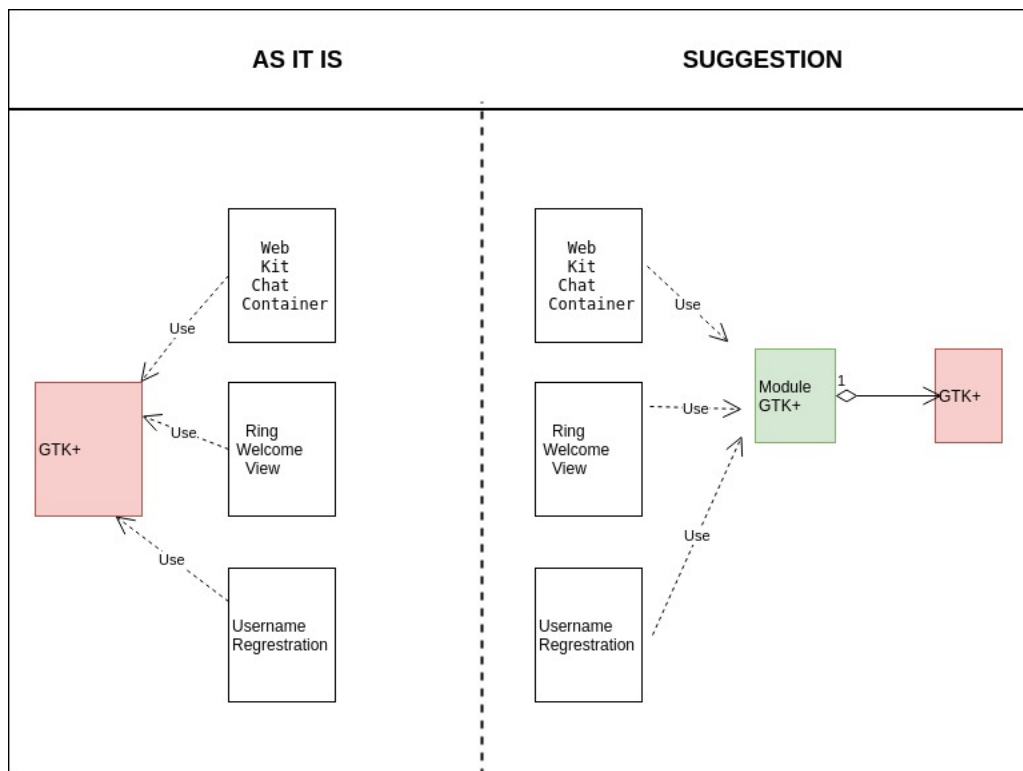

Figure 2.4: Suggestion for GTK+ dependency

2.6.2   Problem II: Long Method

Description

By the analysis of the static we also find another bad smell, which is the long method. This bad smell is related to the static code analysis tools - namely Code Scene that were used before. From this part several guides can be followed to improve the quality of the code as well as to make it more maintainable.
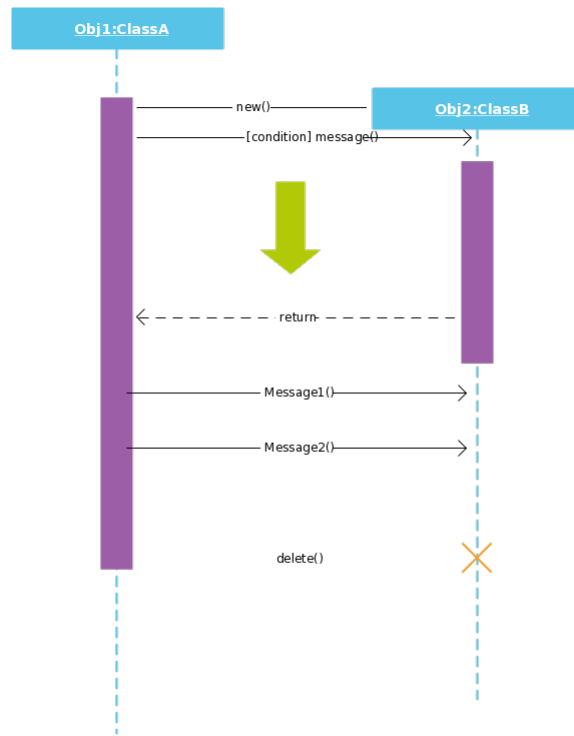
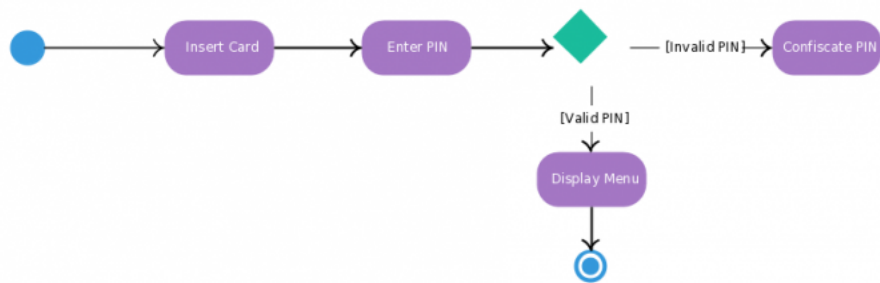Figure 2.5: Simplified Sequence Diagram of the solution

Figure 2.6: Simplified Activity Diagram of the solution
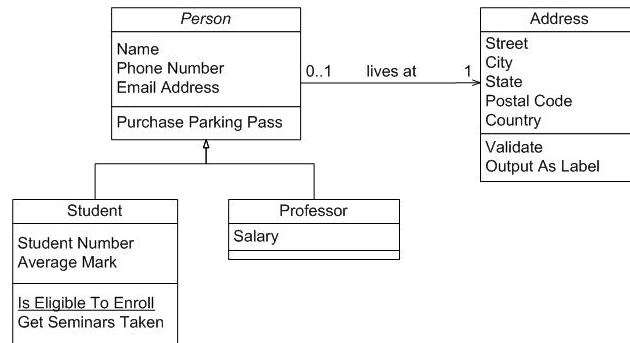
Figure 2.7: Simplified Class Diagram of the solution

Proposal of Solution

Considering the problem presented above, the proposed solution was to refactor the methods x and y and split the method using a simpler method implementation.

2.7   Conclusion

Code Analysis Tools

From the code analysis tools, we can abstract the following information: the code is mainly written in C++; some files of the project have become complex over the last months and in general, the project is not bad but some bad coding practices were found and need to be corrected in order to enhance the quality of the program. From this part several guides can be followed to improve the quality of the code as well as to make it more maintainable:

1. Write short units of codes

2. Write simple units of code and automate tests

3. Write more comments and doc of the codes

Refactoring

The results of the analysis suggested to focus on the refactoring of the classes: manager, account and call. However, for specific demands of the this work, we can reduce specifically for the Code Complex function in Call.cpp.

Unit Test

The absence of unit tests can really impact on the system quality and performance. However, we do not have time to implement those tests.

Dependency evaluation

Several classes invocate GTK+ directly. For example: ringwelcomeview.cpp, Username Registration Box Private.cpp, webview chat context menu.cpp. Considering the View of Ring, which several classes invocate GTK+, an interesting suggestion would be to add one level of indirection by encapsulating the GTK+. Therefore, this suggestion will reduce the dependency of GTK+ and further external changes on this library will not affect the application.

In summary there are many improvements that can be done in Ring-project from the point of view of Architecture: as anti-patterns improvements, adding a level of indirection or improving the overall architecture of the system.

Chapter 3

TP3 - Implémentation et contribution au projet Ring

3.1   Introduction

This work summarizes the analysis of Ring source, presents approaches to overcome issues detected and includes possible changes on the source code.

First, the patterns, anti-patterns and bad smells are presented. Then, some issues concerning the Ring project architecture and source code are highlighted and explained, for each issue a possible solution is proposed.

Finally, a summary is presented including suggestions along with code refactoring and diagrams to explain the suggestions.

3.1.1   Ring Project

Ring project is an innovative and interesting project that aims to help people to communicate all over the world. The software can be used as a telephone, a messenger, for teleconferencing and media sharing. Ring is involved in the Free Software philosophy; that is proved by the fact that it is a GNU[4] package [16]. The code documentation [23] was developed in Doxygen and the license of the system is GPLv3.

3.1.2   Architecture

The current paradigm of Ring project directs knowledge of GTK+ and other libraries that are detailed below:

1. OpenDHT — Decentralizing Communications

2. PJSIP — A Connection With Traditional Telephony

3. GnuTLS — A Reinforce Security Manager

4. FFMPEG and LIBAV — Audio and Video Administrators

3.1.3   GTK+

GTK+, or GIMP Toolkit, as it is also known, is a multi-platform toolkit for creating graphical user interfaces that can be used in different projects of variable scale. In Ring project, GTK+ is intrinsically related to the View part of the Model View Controller design model. GTK+ was originally developed for C language and to use it for C++ it is necessary to wrap the methods. The implementation on Ring project was done using a structures which, in a restricted way, encapsulate the information. GTK+ also gives the possibility to use RefTests for simulating cases of bugs.

3.2   Patterns

The following section is a brief summary of the patterns related with the issues found.

3.2.1 MVC

The Model-View-Controller pattern is a software architectural pattern used in user interfaces development. The principle of this pattern is that a given application is divided into three interconnected - but independent - parts which are the user space interface, the generated model and an intermediary layer. MVC is intrinsically related to the issue and the solution, since the GTK+ was used for the construction of the View part of the model.

3.3 Anti patterns and Bad Smells

Although the Ring project has interesting characteristics there are also flaws that must be exposed in order to improve the quality of the project. Among these flaws, the ones that drew our attention are Blob and Spaghetti codes.

The following flaws (code smells) can be found in the Project's code :

1. Long Method

2. Duplicated Code

3. Long Parameter List

4. Dead Code

3.4 Description of the problem

All this section is a reminder of the work done in the previous TP.

3.4.1 Problem I: Over-dependency Problem

Description

Considering the context in which Ring developed, as a open source project, a direct dependency on external libraries as GTK is a limitation. This limitation can be overcome by the creation of an intermediary class that will serve as an interface for requests involving the usage of GTK. The following direct dependencies have been identify through static code analysis : ringwelcomeview.cpp, Username Registration Box Private.cpp, webview chat context menu.cpp.

Proposal of Solution

In order to solve the problem discussed above, we are proposing a class that will serve as an interface for the classes that currently use GTK+.Figure 3.1 provides a high-level illustration of our suggestion for the GTK+ dependency. We implement our proposal with the QtCreator[27] IDE, which also enables the use of Gtest for unit tests.

Implementation

The implementation was made by cresting a new class, with properties and methods, which is called by the main Class: Ring Client. This class wraps the behavior of the original GTK code. Another possibility was to use the GTKmm toolkit, which has already object oriented code that wraper the original GTK+ code *GTKMM*[7] which wraps the GTK+ library for a C++ use. They assume their interface provides typesafe callbacks, and a comprehensive set of widgets that are easily extensible via inheritance. This wrapper lets the developer the use of C++ features and his strength, as for example, the inheritance or the polymorphism. We worked with GTKMM even if QT5 also contains a binding of GTK+ to C++ programming. Figure 3.4 represents the class diagram of the solution. Adding a level of indirection and encapsulating the GTK+ calls would reduce the dependency of GTK+ and if further external changes were performed on this library, the application would not be affected .

Figure 3.2 represents a simplified version of the sequence diagram of the proposed solution, whereas the Figure 3.3, represents a simplified activity diagram of the solution.
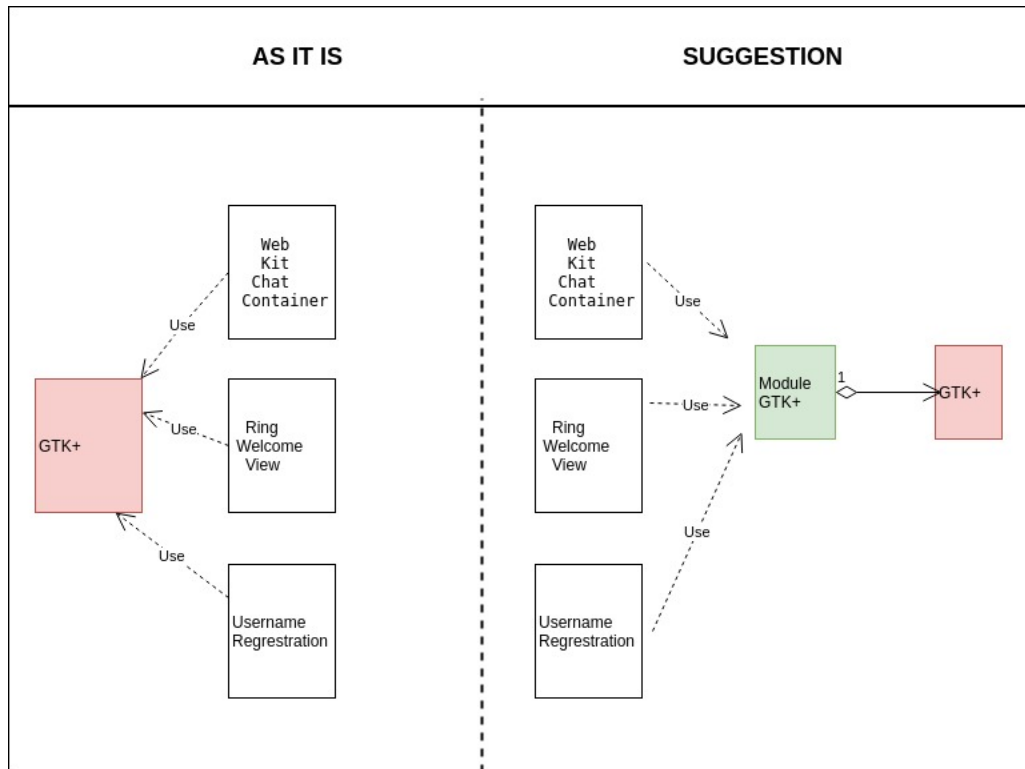

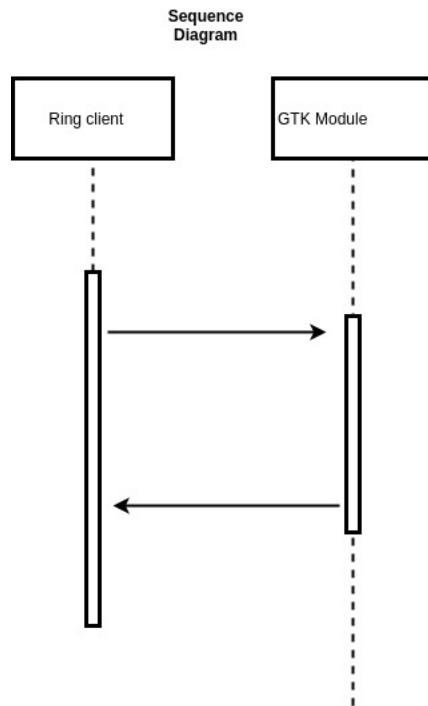Figure 3.1: Suggestion for GTK+ dependency

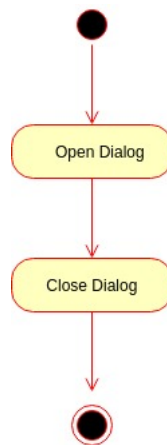Figure 3.2: Simplified Sequence Diagram of the solution

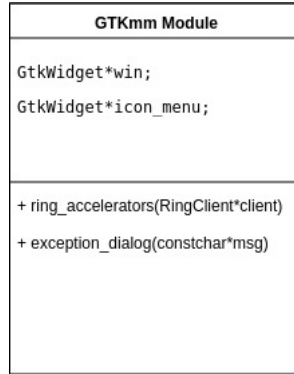Figure 3.3: Simplified Activity Diagram of the solution

```
                  ┌─────────────────────────────────┐
                  │          GTKmm Module           │
                  ├─────────────────────────────────┤
                  │ GtkWidget*win;                  │
                  │                                 │
                  │ GtkWidget*icon_menu;            │
                  │                                 │
                  │                                 │
                  ├─────────────────────────────────┤
                  │ + ring_accelerators(RingClient*client) │
                  │                                 │
                  │ + exception_dialog(constchar*msg) │
                  │                                 │
                  │                                 │
                  │                                 │
                  └─────────────────────────────────┘
```

Figure 3.4: Simplified Class Diagram of the solution

3.4.2 Second Problem: Long Method

Description

The static analysis of the code allowed us to find another bad smell, which is the long method. That bad smell has been identified through the static code analysis tools - namely Code Scene -. Several guidelines can be followed to improve the quality of the code as well as making it more maintainable.

Proposal of Solution

Considering the problem presented above, the proposed solution was to refactor the methods `Call::addSubCall(const std::shared_ptr<Call>& call)` and `RingAccount::handlePendingCall(Pending` split the method using a simpler method implementation.

3.5 Analysis

3.5.1 Testing

Testing the code is a crucial task for the current projects of software engineering. The testing activity can be performed in two ways : static and dynamic testing [8].

The Figure 3.5, from [26], summarizes the tests possibilities.

Static Testing: The code is tested and defects are found without executing it. Static Testing is done during the verification process. This type of testing is performed along with the review of the documents (including source code) and static analysis. It is a useful and cost effective way of testing. For example: reviewing, walk-through, inspection, etc.

Dynamic Testing: In dynamic testing, the software code is executed to demonstrate the result of running tests. It's done during the validation process. For example: unit testing, integration testing, system testing, etc.
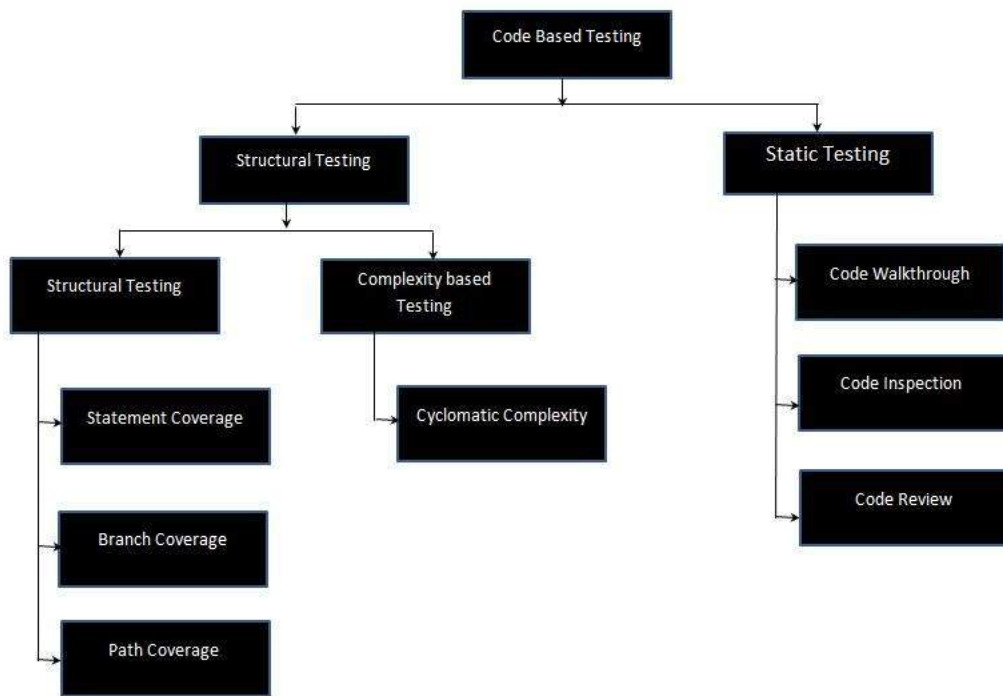
46

Figure 3.5: Code base testing

Static test: Software Walk-through

As one of the static testing techniques, we use a walk-through process to improve the quality of the code. Although it is not a formal way of testing, the walk-through process gives the opportunity to evaluate the code as well as enhancing its quality. This test is described as more detailed than a informal verification of the code, but less formal form a technical review and a code inspection [26].

Dynamic test: Unit Testing

The unit test can be defined as the lowest level of testing in software development. That test is performed by isolating individual units of software. It aims to achieve a high level of decision coverage on the code. This kind of test can detect many problems at the unit that is being tested and even more in early stages of software development.

In a conventional structured programming language, such as C, the unit to be tested is traditionally a function or sub-routine. To test such a unit in isolation, the external program units called by the unit under test as well as the external data used by the unit under test have to be simulated.

Test Driven Development is a current trend in terms of established techniques for delivering better software faster [**TDD**].

Integration testing

This testing technique evaluates the interactions among classes (inter-class testing) related through dependencies, I.e., associations, aggregations, specialization [20]. Considering the core role that GTK has on Ring Project, an integration test would be necessary.

Mock Testing

The mock testing is a unit test in which objects are created. Those objects that mimic the behavior of real objects. A mock-based approach focuses on fully specifying what the correct interaction is and detecting when it goes wrong [3]. This type of testing allows the tester to set up test scenarios without being forced to mobilize, large and heavy resources such as databases. For example, during the unit test process,instead of calling a real database for testing, a database can be simulated by using a mock object. The reference [18] gives an insight of the mock testing done.

Testing Framework

The framework used was C++ Google Testing Framework [6]. That framework's goal is to help developers to write better tests in C++ regardless of the platform used (Windows, Linux, Mac). It does so through its functionalities that, according to them [6], fit in to satisfy the characteristics of good tests (independence, repeatability, organization, portability, Re-usability, speed and intensive log keeping).

Basic unit testing

The basic unit testing can be defined as the test of a class' method during a class intra-method testing.

Test case : class GTK module

Using a unit test configuration, we were able to test some properties of this class the reference of the construction of the tests were [13]. To test the new class, we created a test script to test several properties of it.

Test cases for long Methods]

Table 3.1: Test Cases

Test Scenario	Test Case	Pre Conditions	Test Step	Test Data	Expected Results
XXXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
XXXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
XXXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX

3.5.2 Principles of Modular Design

Cohesion is a measure of how related or focused are the responsibilities of a class, a module, a subsystem.

Coupling or dependency is the degree to which each class, module, subsystem relies on one another.

Considering the concepts explained above, the proposed change is able to increase cohesion and decrease coupling.

3.5.3 SOLID Principles

Defined by Robert C. Martin[17], the proposed change also is related with the Single responsibility principle (SRP), which the every class should have responsibility over a single part of the functionality provided by the system.
– A class should have only one reason to change
– Services should be narrowly aligned with that responsibility
– Cohesion
Responsibility should be entirely encapsulated by the class.

3.5.4 Code Metrics

From the metrics point of view, the proposed change was able to impact by XX percent on the code.

3.5.5   Impact evaluation

Considering the suggestion as interface between the current status of the Ring and a new status, where this class would be an intermediary for the calls, the will an impact of this change.

Arguments Pro change
Arguments for this change are the following:

1. Reduce the dependency

2. Improve the architecture quality

3. Long term enhancement of the code overall

Arguments Against change
Arguments against this change are the following:

1. The paradigm of the Project relies directly on GTK+

2. The necessary change will require investment on refactoring several classes

3. No short term directly advantage

3.5.6   Documentation

The documentation of the class in the section Appendix.

3.5.7   Code Improvements

From the metrics point of view, the introduction of this new class reduces the complexity by adding a new indirection. There were some

3.5.8   Challenges

Building Ring

To begin with, Ring installation does not fully support an installation in a folder which its path contains a space character. When this problem occurred, no hint is given for the source of this problem and lets us spending time on this *tiny* problem. Furthermore, the compiling time of Ring and its dependencies on usual machines (our laptops) forces us to schedule our tries smarter in a way it was more efficiently to build after a large of the improvements already integrated.

Refactoring Long-Methods

In the refactoring of long methods, we faced some issues, a part of these methods were actually several if-statements in a row where one branch leads to a return statement. The inner branches also used a multiples variables declared and instantiate out of the if-statements scope that's why a solution was whether the implementation of short functions with a lot of parameters or to declare and instantiate again those variables. In this context we chose one way over the other case by case for each method.

Creating a new class

For the creation of a new class, specifically the GTK module, it was hard to define some codes that belong specifically for the new class. The codes were sparse and this represented a major difficult for the group to develop the code for the class. This was expected and shows the intrinsic relation - and dependency - of Ring and this library throughout the code.

3.6 Pull Request

On Ring, they use Tuleap to control the pull requests and bug fixes. Below is the list of pull requests:

1. Creation of Class GTK Module

2. Long method fix I

3. Long method fix II

3.7 Conclusion

In summary, Ring project is a innovative open source project that aims to help integrates people around the world for free. One of the drawbacks relative to its architecture is the over dependency of the project with other libraries, for example its intrinsic dependency on GTK+.

Although this is a architectural decision, there are some difficulties related to this over-dependency, which can be difficult to support the code as well as other difficulties.

The proposed solution is to reduce the over dependency by introducing a level of association. This increases the code quality and can reduce maintenance time, since the gnome would not depend on this expertise to be done.

Considering the Five Principles developed by Robert C. Martin, the proposed suggestion to create a GTK interface embraces completely the Single Responsibility Principle, since the class would be responsible specifically and only for the GTK module operations.

However, considering the multiple interfaces of the system, the implementation would need to be carefully done considering the Interface Segregation Principle.

This suggestion would change the current paradigm of the project, which is: gnome version relies on GTK+. However, this would require time and effort that the company may not be willing to invest for a specific platform.

Conclusion

fzen

Appendix A

TPs Changes

From the original reports submitted, the changes made were the following:

1. Diagram changes on TP1 and TP2

2. Spelling corrections

3. Gnone-client addition

4. GTK+ definition

Appendix B

TP1 - Appendices

1. Complete class diagram of Ring daemon
2. Complete class diagram of Ring Gnome Client

Appendix C

TP2 - Appendices

Quality metrics

The following attached files show the results of the Complexity Trends analysis of the hotspots by Code scene:

1. Complexity Trend for sipvoiplink.cpp

2. Complexity Trend for sipaccount.cpp

3. Complexity Trend for ringaccount.cpp

4. Complexity Trend for sipcall.cpp

5. Complexity Trend for Manager.cpp

The following attached files show the results of the Internal Temporal Coupling analysis result of each of the hotspots:

1. Internal Temporal Coupling for sipvoiplink.cpp

2. Internal Temporal Coupling for sipaccount.cpp

3. Internal Temporal Coupling for ringaccount.cpp

4. Internal Temporal Coupling for sipcall.cpp

5. Internal Temporal Coupling for Manager.cpp

The following are definitions of SAD and EPI:

- SAD is a tool for the detection and correction of software architecture defects

- EPI is a tool for pattern identification.

Appendix D

TP3 - Appendices

D.0.1   src/Call.cpp

/** Example of Documentation call.cpp Purpose: method to reduce the size of the function addSubCall

@author GroupOne @version 1.0 3/17/17 */

/** No returns
@param Its own object. */

```
void
Call::addSubCall(const std::shared_ptr<Call>& call)
{
    if (connectionState_ == ConnectionState::CONNECTED
            || callState_ == CallState::ACTIVE
            || callState_ == CallState::OVER) {
        call->removeCall();
    } else {
        std::lock_guard<std::recursive_mutex> lk (callMutex_);
        if (not subcalls.emplace(call).second)
            return;
        call->quiet = true;

        for (auto& pmsg : pendingOutMessages_)
            call->sendTextMessage(pmsg.first, pmsg.second);


        std::weak_ptr<Call> wthis = shared_from_this();
        std::weak_ptr<Call> wcall = call;
        call->addStateListener([wcall, wthis](Call::CallState new_state,
                                              Call::ConnectionState new_cstate,
                                              UNUSED int code) {
            if (auto call = wcall.lock()) {
                if (auto sthis = wthis.lock()) {
                    auto& this_ = *sthis;
                    auto sit = this_.subcalls.find(call);
                    if (sit == this_.subcalls.end())
                        return;
                    RING_WARN("[call:%s] DeviceCall call %s state changed %d %d",
                            this_.getCallId().c_str(), call->getCallId().c_str(),
                            static_cast<int>(new_state), static_cast<int>(new_cstate));
                    if (new_state == CallState::OVER) {
                        std::lock_guard<std::recursive_mutex> lk (this_.callMutex_);
                        this_.subcalls.erase(call);
                    } else if (new_state == CallState::ACTIVE && this_.callState_ == CallState::INACTIVE) {
                        this_.setState(new_state);
                    }
                    if ((unsigned)this_.connectionState_ < (unsigned)new_cstate && (unsigned)new_cstate <= (unsigned)ConnectionState::RINGING) {
                        this_.setState(new_cstate);
                    } else if (new_cstate == ConnectionState::DISCONNECTED && new_state == CallState::ACTIVE) {
                        this->method_disconneted_active_call(&this_);
                    }
                    if (new_state == CallState::ACTIVE && new_cstate == ConnectionState::CONNECTED) {
                        std::lock_guard<std::recursive_mutex> lk (this_.callMutex_);
                        RING_WARN("[call:%s] peer answer", this_.getCallId().c_str());
                        auto subcalls = std::move(this_.subcalls);
                        for (auto& sub : subcalls) {
                            if (sub != call)
                                sub->hangup(0);
                        }
                        this_.merge(call);
                        Manager::instance().peerAnsweredCall(this_);
                    }
                    RING_WARN("[call:%s] Remaining %zu subcalls", this_.getCallId().c_str(),
                            this_.subcalls.size());
                    if (this_.subcalls.empty())
                        this_.pendingOutMessages_.clear();
                } else {
                    RING_WARN("DeviceCall IGNORED call %s state changed %d %d",
                            call->getCallId().c_str(), static_cast<int>(new_state),
                            static_cast<int>(new_cstate));
                }
            }
        });
        setState(ConnectionState::TRYING);
    }
}
```

```
/*New Method to reduce the size of the function above*/
void
Call::method_disconneted_active_call([Object] this_ )
{
    std::lock_guard<std::recursive_mutex> lk (this_.callMutex_);
    RING_WARN("[call:%s] peer hangup", this_.getCallId().c_str());
    auto subcalls = std::move(this_.subcalls);
    for (auto& sub : subcalls) {
        if (sub != call)
            try {
                sub->hangup(0);
            } catch(const std::exception& e) {
                RING_WARN("[call:%s] error hanging up: %s",
                        this_.getCallId().c_str(), e.what());
            }
    }
    this_.peerHungup();
}
```

D.0.2 src/ringdht/ringaccount.cpp

```
tls::TlsParams RingAccount::provide_tlsParams(dht::InfoHash remote_h, std::weak_ptr<SIPCall> call, bool incoming){
    std::weak_ptr<RingAccount> w = std::static_pointer_cast<RingAccount>(shared_from_this());
    std::weak_ptr<SIPCall> wcall = call;
return tls::TlsParams tlsParams {
    /*.ca_list = */"",
    /*.cert = */identity_.second,
    /*.cert_key = */identity_.first,
    /*.dh_params = */dhParams_,
    /*.timeout = */std::chrono::duration_cast<decltype(tls::TlsParams::timeout)>(TLS_TIMEOUT),
    /*.cert_check = */[w,wcall,remote_h,incoming](unsigned status, const gnutls_datum_t* cert_list, unsigned cert_num) -> pj_status_t {
        try {
            if (auto call = wcall.lock()) {
                if (auto sthis = w.lock()) {
                    auto& this_ = *sthis;
                    std::shared_ptr<dht::crypto::Certificate> peer_cert;
                    auto ret = check_peer_certificate(remote_h, status, cert_list, cert_num, peer_cert);
                    if (ret == PJ_SUCCESS and peer_cert) {
                        std::lock_guard<std::mutex> lock(this_.callsMutex_);
                        for (auto& pscall : this_.pendingSipCalls_) {
                            if (auto pcall = pscall.call.lock()) {
                                if (pcall == call and not pscall.from_cert) {
                                    RING_DBG("[call:%s] got peer certificate from TLS negotiation", call->getCallId().c_str());
                                    tls::CertificateStore::instance().pinCertificate(peer_cert);
                                    pscall.from_cert = peer_cert;
                                    break;
                                }
                            }
                        }
                    }
                    return ret;
                }
            }
            return PJ_SSL_CERT_EUNTRUSTED;
        } catch (const std::exception& e) {
            RING_ERR("[peer:%s] TLS certificate check exception: %s",
                    remote_h.toString().c_str(), e.what());
            return PJ_SSL_CERT_EUNKNOWN;
        }
    }
};
}
```

```cpp
bool
RingAccount::handlePendingCall(PendingCall& pc, bool incoming)
{
    auto call = pc.call.lock();
    if (not call)
        return true;

    auto ice = pc.ice_sp.get();
    if (not ice or ice->isFailed()) {
        RING_ERR("[call:%s] Null or failed ICE transport", call->getCallId().c_str());
        call->onFailure();
        return true;
    }

    // Return to pending list if not negotiated yet and not in timeout
    if (not ice->isRunning()) {
        if ((std::chrono::steady_clock::now() - pc.start) >= ICE_NEGOTIATION_TIMEOUT) {
            RING_WARN("[call:%s] Timeout on ICE negotiation", call->getCallId().c_str());
            call->onFailure();
            return true;
        }
        // Cleanup pending call if call is over (cancelled by user or any other reason)
        return call->getState() == Call::CallState::OVER;
    }

    // Securize a SIP transport with TLS (on top of ICE tranport) and assign the call with it
    auto remote_h = pc.from;
    if (not identity_.first or not identity_.second)
        throw std::runtime_error("No identity configured for this account.");


     tls::TlsParams tlsParams = provide_tlsParams(call,remote_h,incoming);
    call->setTransport(link_->sipTransportBroker->getTlsIceTransport(pc.ice_sp, ICE_COMP_SIP_TRANSPORT,
                                                    tlsParams));

    // Notify of fully available connection between peers
    RING_DBG("[call:%s] SIP communication established", call->getCallId().c_str());
    call->setState(Call::ConnectionState::PROGRESSING);

    // Incoming call?
    if (incoming) {
        std::lock_guard<std::mutex> lock(callsMutex_);
        pendingSipCalls_.emplace_back(std::move(pc)); // copy of pc
    } else
        createOutgoingCall(call, remote_h.toString(), ice->getRemoteAddress(ICE_COMP_SIP_TRANSPORT));

    return true;
}
```

Bibliography

[1] EfficiOS. *LTTng Documents*. `http://lttng.org/docs/v2.9/`. [Online; accessed 27-January-2017].

[2] Ralph Johnson Erich Gamma Richard Helm and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. `https://ring.cx/`. [Book; accessed 27-January-2017].

[3] Bert F. *Answer to the question: What is the purpose of mock objects?* `http://stackoverflow.com/questions/3622455/what-is-the-purpose-of-mock-objects`. [Online; accessed 11-April-2017].

[4] Gnu. *Gnu Home*. `https://www.gnu.org/home.en.html`. [Online; accessed 07-April-2017].

[5] Gnu. *Instrumentation Options*. `https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html`. [Online; accessed 27-January-2017].

[6] Google. *C++ Test Framework*. `https://github.com/google/googletest/blob/master/googletest/docs/Primer.md`. [Online; accessed 27-April-2017].

[7] GTKMM. *C++ Interfaces for GTK+ and GNOME*. `http://www.gtkmm.org/en/index.html`. [Online; accessed 12-April-2017].

[8] Software Testing Help. *Static Testing and Dynamic Testing – Difference Between These Two Important Testing Techniques*. `http://www.softwaretestinghelp.com/static-testing-and-dynamic-testing-difference/`. [Online; accessed 27-April-2017].

[9] IBM. *Java Design Patterns 101*. `https://www.ibm.com/developerworks/java/tutorials/j-patterns/j-patterns.html`. [Online; accessed 30-January-2017].

[10] IBM. *Model-View-Controller design pattern*. `https://www.ibm.com/support/knowledgecenter/en/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdmvcdespat.htm`. [Online; accessed 30-January-2017].

[11] IBM. *Performance View*. `http://www.ibm.com/support/knowledgecenter/SSX3HK_6.5.0/com.ibm.cdcdoc.mcadminguide.doc/concepts/performance_view_monitoring_perspective.html/`. [Online; accessed 27-January-2017].

[12] *Introduction to Gerrit*. `https://gerrit-documentation.storage.googleapis.com/Documentation/2.13.7/intro-quick.html`. [Online; accessed 12-April-2017].

[13] IPL Limited -now Civica Digital-. *Unit Testing Of C++ Classes*. `https://accu.org/index.php/journals/1389`. [Online; accessed 27-January-2017]. 1993.

[14] Namhyung Kim. *uftrace in github*. `https://github.com/namhyung/uftrace`. [Online; accessed 27-January-2017].

[15]   Philippe Kruchten. "Architectural Blueprints — The "4+1" View Model of Software Archi-tecture". In: *IEEE Software* 12.6 (1995), pp. 42–50.

[16]   Savoir-faire Linux. *Ring official gnu package.* `https://blog.savoirfairelinux.com/en-ca/2016/ring-official-gnu-package/`. [Online; accessed 10-April-2017].

[17]   Robert Cecil Martin. *Principles of Object Oriented Design.* `http://www.butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod`. [Online; accessed 27-April-2017].

[18]   Mkyong. *Unit Test – What is Mocking? and Why?* `https://www.mkyong.com/unittest/unit-test-what-is-mocking-and-why/`. [Online; accessed 12-April-2017].

[19]   Colin J. Neill and Philip A. Laplante. *Antipatterns: Identification, Refactoring, and Manage-ment.*

[20]   Martyn A Ould and Charles Unwin (ed). *Testing in Software Development, BCS (1986).* Page 71.

[21]   James Parnitzke. *Modeling the MDM Blueprint – Part V.* `https://pragmaticarchitect.wordpress.com/2009/04/16/modeling-the-mdm-blueprint-part-v`. [Online; accessed 30-January-2017].

[22]   Dewayne E. Perry and Alexander L. Wolf. "Foundations for the Study of Software Architec-ture". In: *ACM SIGSOFT SOFTWARE ENGINEERING NOTES* 17.4 (1992), p. 40.

[23]   Ring. *Code documentation.* `https://test.savoirfairelinux.com/job/ring-daemon-doc/doxygen/`. [Online; accessed 27-January-2017].

[24]   Ring. *Initial page.* `https://ring.cx/`. [Online; accessed 27-January-2017].

[25]   Scitools. *Implementing Agile in very large enterprises.* `https://scitools.com/trial-download-3/`. [Online; accessed 27-January-2017].

[26]   Tutorial Spoint. *Code-Based Testing.* `https://www.tutorialspoint.com/software_testing_dictionary/code_based_testing.htm`. [Online; accessed 27-April-2017].

[27]   Qt Team. *Qt Creator.* `https://www.qt.io/ide/`. [Online; accessed 27-April-2017].

[28]   *Transifex.* `https://www.transifex.com/`. [Online; accessed 12-April-2017].

[29]   *Tuleap : All in One.* `https://www.tuleap.org/sites/default/files/tuleap-datasheets.pdf`. [Online; accessed 12-April-2017].

[30]   Tutorialspoint. *Template Pattern.* `https://www.tutorialspoint.com/design_pattern/template_pattern.htm`. [Online; accessed 16-March-2017].