

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 13, 2017



Students names:

Isnaldo Francisco de Melo jr
Franck Brazier
Stephane Fagnon

Date of the submission: April 13, 2017

Contents

1	TP1 - Étude et analyse du Ring	4
1.1	Introduction	5
1.2	Ring	5
1.3	Context	5
1.4	Statistical Analysis	6
1.5	Dynamic Analysis	7
1.6	Ring architecture	7
1.7	4+1 Model	7
1.7.1	Development view	8
1.7.2	Logical view	8
1.7.3	Process view	11
1.7.4	Physical view	13
1.7.5	Scenarios	14
1.7.6	Performance View	18
1.8	Design Patterns	18
1.9	Conclusion	19
2	TP2 - Réusinage Architectural du code Ring	21
2.1	Introduction	22
2.2	Ring	22
2.3	Code Quality Analysis	22
2.3.1	Code Scene	22
2.3.2	Better Code Hub	25
2.4	Ptidej	29
2.5	Design Patterns	29
2.5.1	Factory Method	29
2.5.2	Singleton	31
2.5.3	Model View Controller	32
2.5.4	Template Pattern	32
2.6	Anti Pattern	33
2.6.1	Dead Code	33
2.6.2	Complex Class	34
2.6.3	Blob	34
2.7	Problem Exposed	35
2.7.1	Problem I: Over-dependency Problem	35

2.7.2	Problem II: Long Method	36
2.8	Conclusion	37
3	TP3 - Implémentation et contribution au projet Ring	38
3.1	Introduction	39
3.1.1	Ring Project	39
3.1.2	Architecture	39
3.1.3	GTK+	39
3.2	Anti patterns and Bad Smells	39
3.3	Description of the problem	39
3.3.1	Problem I: Over-dependency Problem	39
3.3.2	Problem II: Long Method	41
3.4	Analysis	41
3.4.1	Testing	41
3.4.2	Code Metrics	42
3.4.3	Documentation	42
3.4.4	Code Improvements	42
3.5	Pull Request	43
3.6	Conclusion	43
A	TP1 - Appendices	44
B	TP2 - Appendices	45
C	TP3 - Appendices	46

Chapter 1

TP1 - Étude et analyse du Ring

1.1 Introduction

This document summarizes Ring's Architectural Views using the model of Philippe Kruchten [blueprints], 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 [perry-wolf], 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 Ring

According to the Ring web site , Ring [ring] 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 [doxygen] was developed in Doxygen and the license of the system is GPLv3.

1.3 Context

The Figure 1.1 summarizes part of the context that Ring is inserted in. To report bugs it uses Tuleap [tuleap], this collaborative 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 [gerrit]. Finally, translations can be sent by Transifex [transifex] software program. GTK+ is used to create the user interface, it is a toolkit and API.

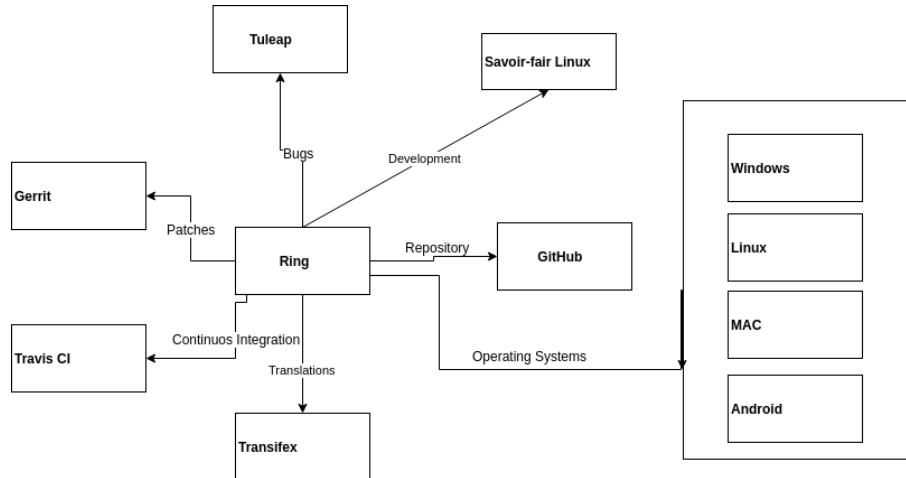


Figure 1.1: Context Diagram

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.

1.4 Statistical Analysis

To explore the Ring System [**ring**], we used the software Understand [**understand**]. 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.

1.5 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 [gnu'compiler]. 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 [ltnng] and profilers (uftrace) [uftrace].

1.6 Ring architecture

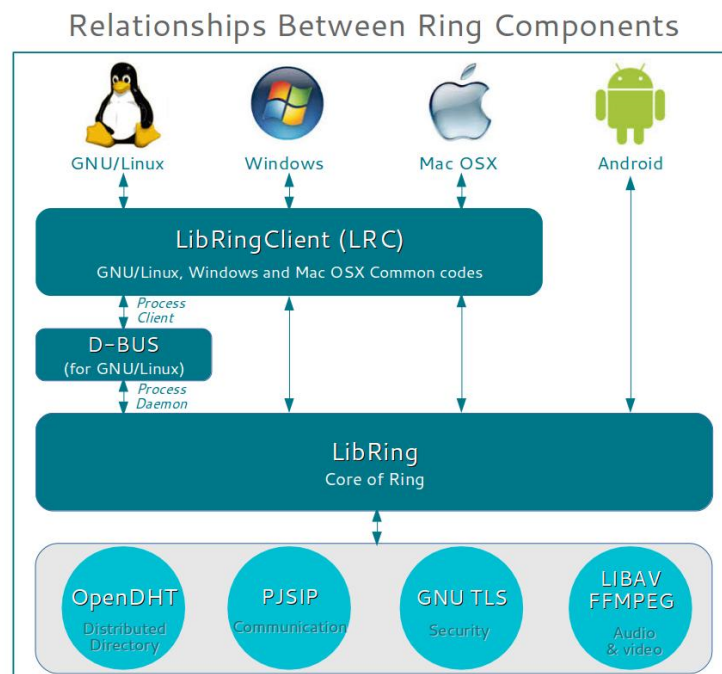


Figure 1.2: Ring Components

1.7 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 [blueprints]. 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.

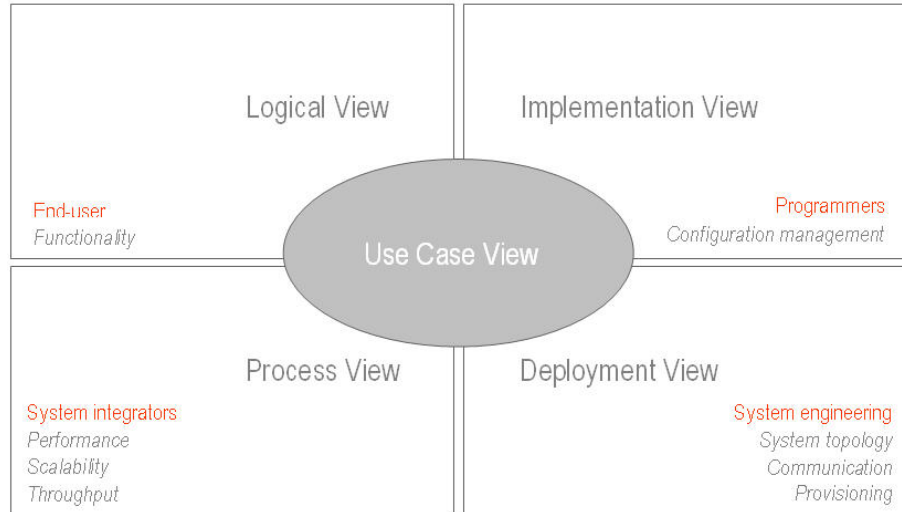


Figure 1.3: 4+1 architectural view model [modeling]

1.7.1 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 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

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*.

1.7.2 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

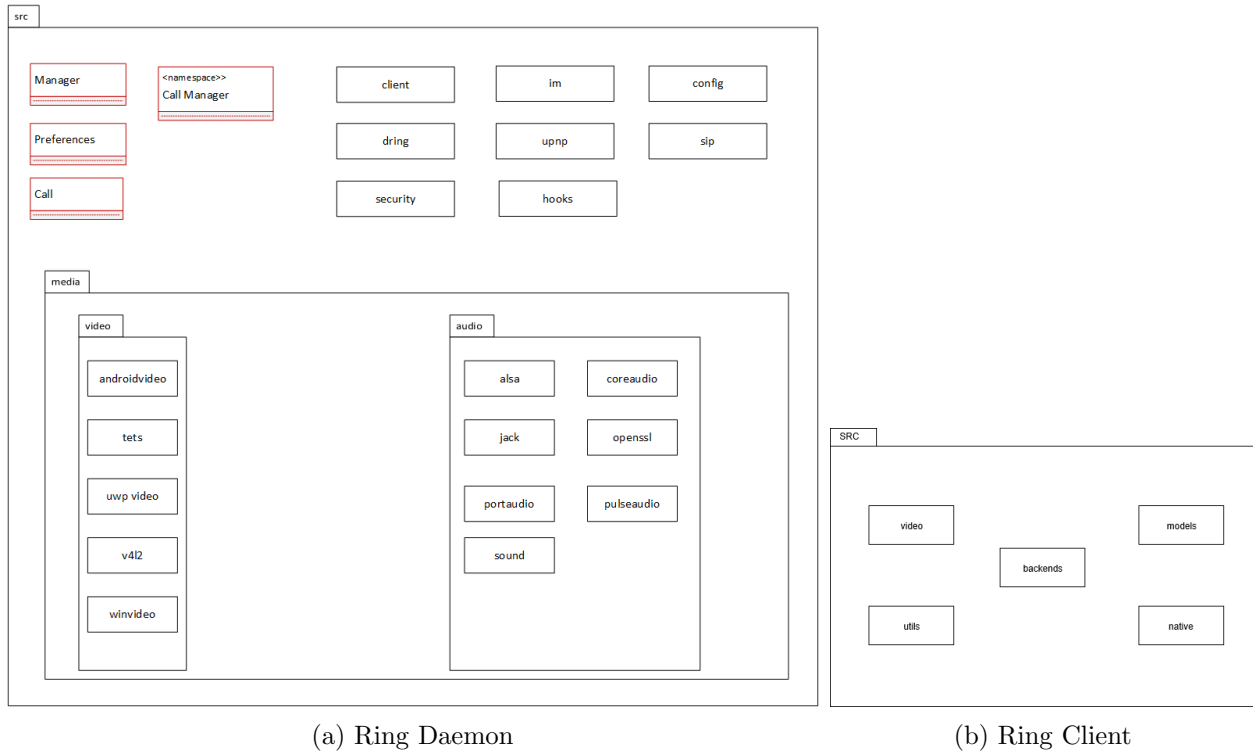


Figure 1.4: Package diagrams

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:

- 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.

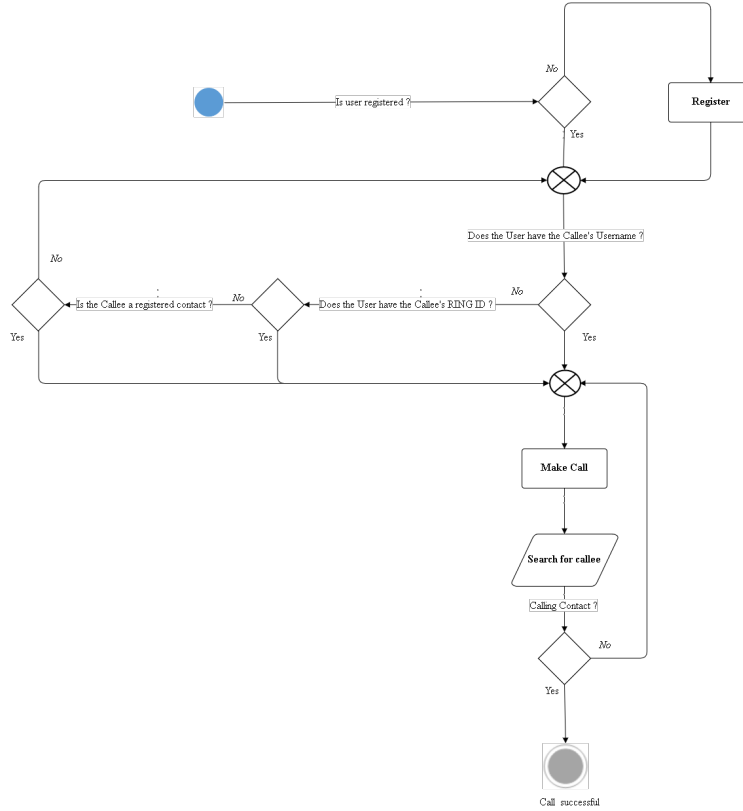


Figure 1.5: Activity diagram for Making a Call

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.

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

In Ring's architecture, the Logical View is represented in Figure 1.8. This figure shows specifically the classes involved in Calling: callmanager, manager, and call.

Since Ring's architecture works using states, calls have a state property, represented in Figure 1.8: Call States.

The complete diagram can be found in Appendix ??.

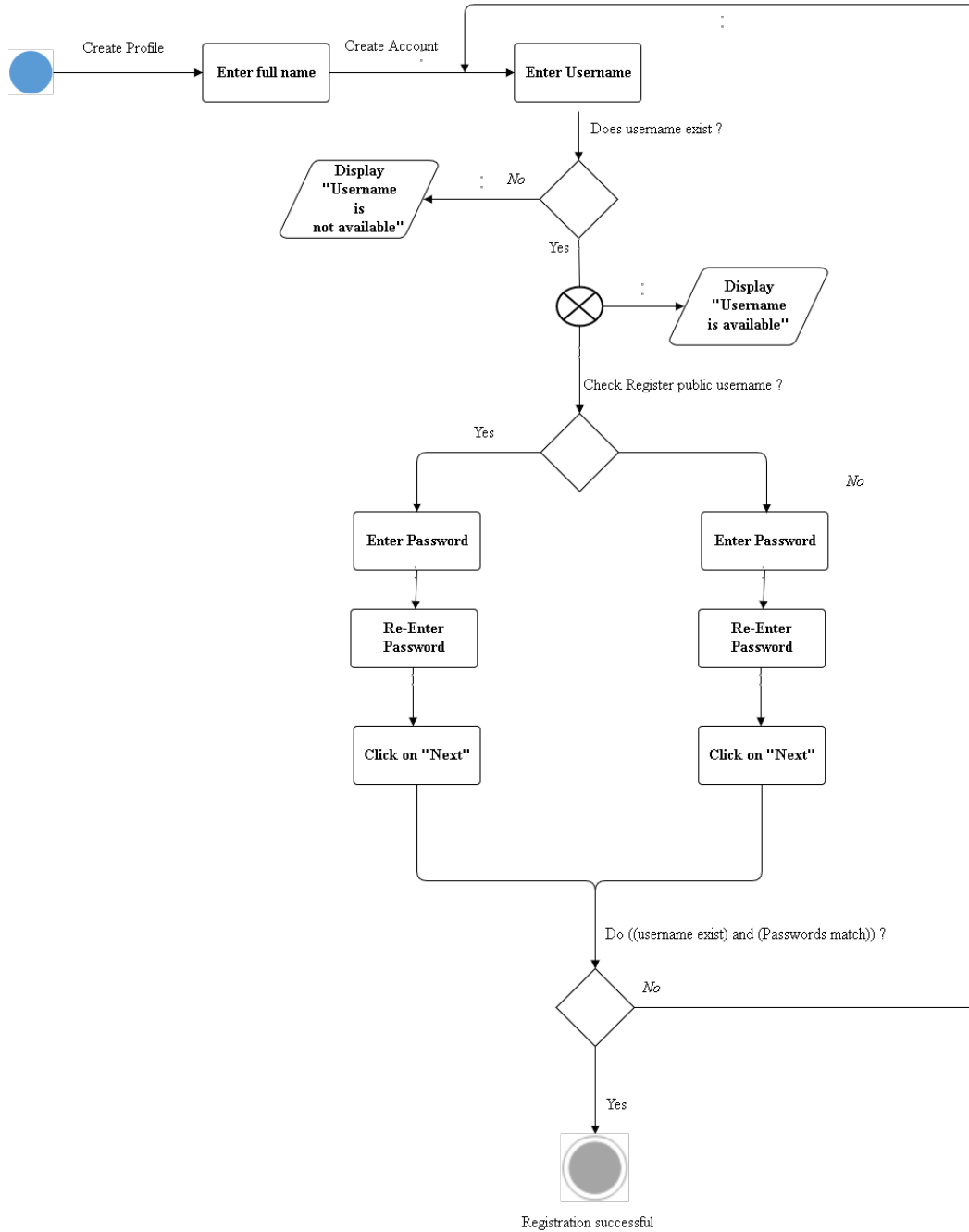


Figure 1.6: Activity diagram for complete registration

1.7.3 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.

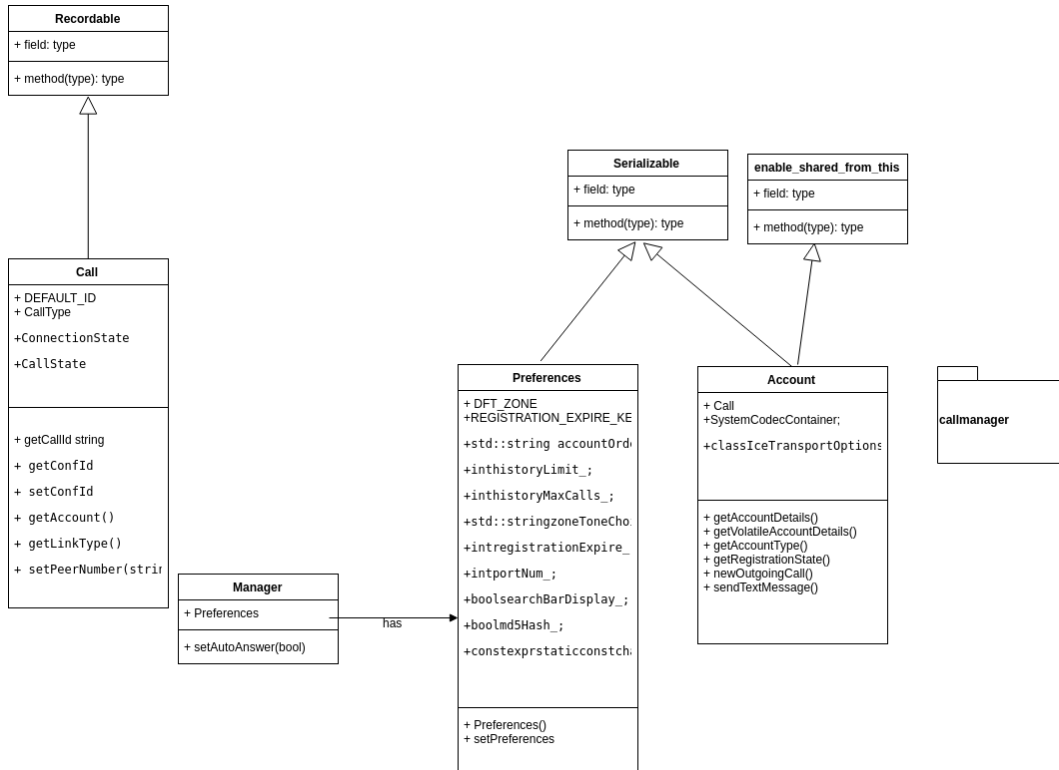


Figure 1.7: Representation of the Logical View

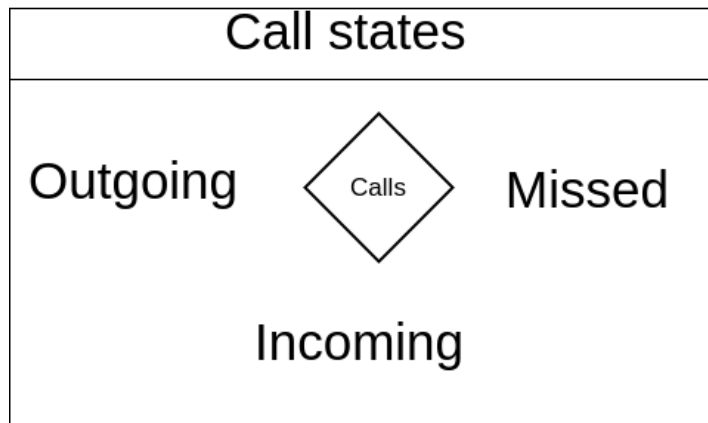


Figure 1.8: Call States

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 1.9

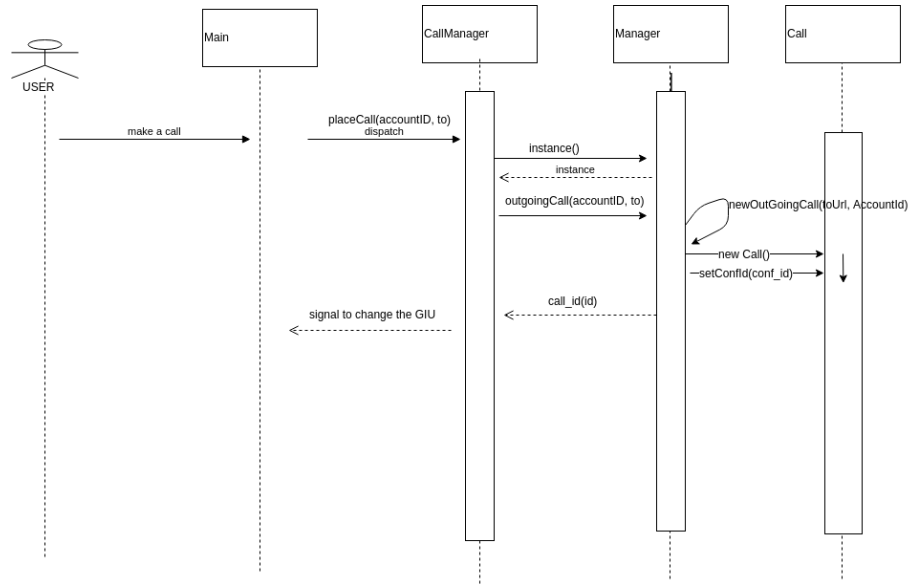


Figure 1.9: Sequence Diagram

Overview

In Ring, the Process View 1.9 shows the sequence specifically for a call, with the operations among the classes involved.

1.7.4 Physical view

Usage

The Physical View is used to represent non-functional requirements of the system.

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.

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

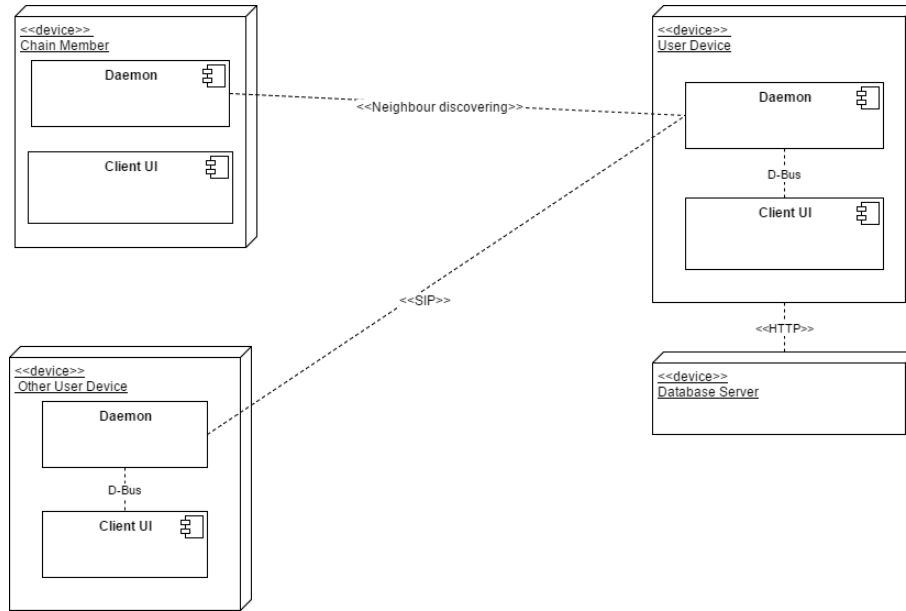


Figure 1.10: Deployment Diagram for Ring system

every device, two components are present, a client responsible for the user interface and a daemon for all the communications with the outside world.

1.7.5 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.

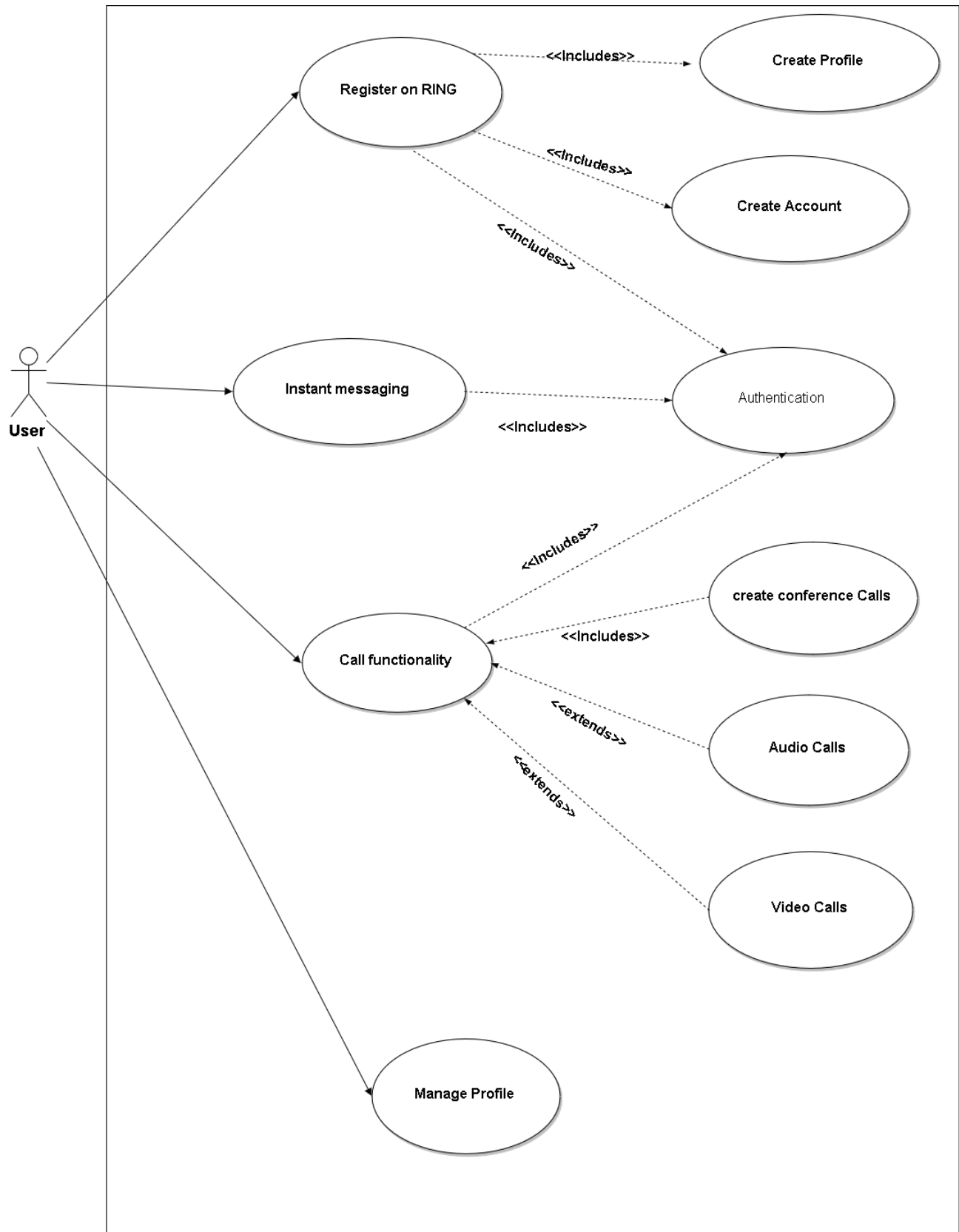


Figure 1.11: User use case

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	<ol style="list-style-type: none"> 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 	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 1. User is registered 2. User launches ring 3. User types a contact's name or username or Ring ID in the search bar 	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 1. Search by name <i>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.</i> 2. Search by public username <i>The User puts in the search bar the username of the other user he is looking for.</i> 3. Search by Ring ID <i>The User puts in the search bar the Ring ID of the other user he is looking for.</i> 	<ol style="list-style-type: none"> 1. Audio Calls <i>The User can make audio calls</i> 2. Create Conference calls <i>The User can add another user to a current call, or make the call a conference</i> 3. Video Calls <i>The User can make audio calls</i>

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	<ol style="list-style-type: none"> 1. User is registered 2. User launches ring 3. User receives a call 4. User chooses to accept or reject the call 	<ol style="list-style-type: none"> 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

1.7.6 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 [ibm] and other profiling tools available in the market, as ufttrace [ufttrace]. 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.

1.8 Design Patterns

The following patterns were found in this project:

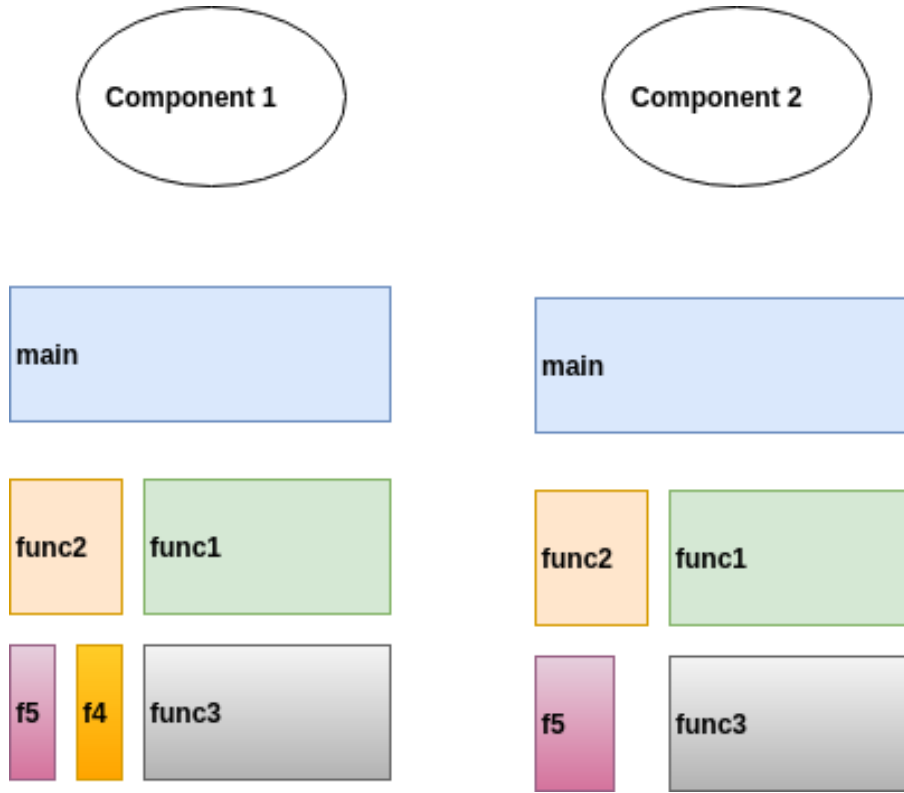


Figure 1.12: Suggested Performance Diagram

Factory Factory Method a design pattern used to creating objects as in template method[pat1]. In the project, the calls, described in the diagrams above are created as a factory.

MVC The Ring's architecture overall presents the Model View Controller model[pat2]. 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.

1.9 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, some Design patterns used on the code are presented with examples and diagrams. This part is followed by the anti-patterns of the code. All these 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 Ring

According to the Ring web site , Ring [**ring**] 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 make Ring usable as a building block for IoT projects.

The code documentation [**doxygen**] was developed in Doxygen and the license of the system is GPLv3.

2.3 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.

2.3.1 Code Scene

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

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

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

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.

2.3.2 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 .

2.4 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. (1) SAD is a tool for the detection and correction of software architecture defects and (2) EPI is a tool for pattern identification.

2.5 Design Patterns

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

2.5.1 Factory Method

Definition

Factory Method a design pattern used to create objects as in template method[**pat1**]. In the project, the calls, described in the diagrams above are created as a factory.

Motivation

The Factory Design Pattern is probably the most used design pattern in modern Orientated Object Programming languages like Java and CSharp. It comes in different variants and implementations. This pattern is introduced and further explained in the Gang of Four (GoF) patterns: Factory Method and Abstract Factory.

Intent

This pattern's aim is to create an object without exposing the instantiation logic to the client. It's kind of a *blackbox* which the client ask for an item (here an instance of a class or a derivated one)and it gives it the specified one. 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 facilitate the creation of instances of a descendant from an specific class. In Ring code they used it to create several calls or accounts for example. This will make the call creating uniform and avoid problems in a data structure to collect them. This pattern is useful when there are several inherited classes, in this case the factory returns the good instance of the class, furthermore the developer can easily add a inherited class into the design and all your work is to implement this class and modify the factory to include this one. So 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

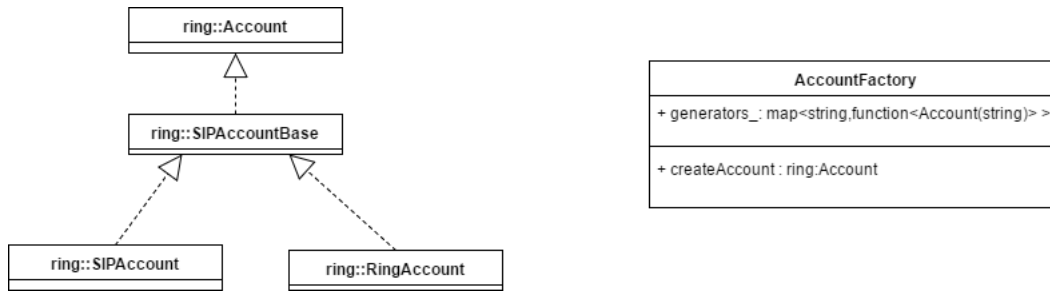


Figure 2.1: Factory Method illustration

one instance for an Account given the parameters by 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 this type. This map is called *generators_* in the code as shown in the Figure 2.1.

2.5.2 Singleton

Motivation

In the Ring project there are several classes which must have only one instance that's why the Singleton pattern is present there. There are two ways for implements the single instantiation either the singleton pattern or a static class, the designers must have chose the former because you can fully exploit the inheritance and polymorphism with this one and are able to create an interface and implements it. In this way it assures a simple way to add some features with implementing an descendant class. This gave an opportunity to further works.

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 reading the code, in the process we saw a call for *getInstance()* which lead us to a singleton class assuming that this class can not be instantiated more than once.

Example of Code

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

```

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

```

2.5.3 Model View Controller

The Ring's architecture overall presents the Model View Controller model[pat2]. This pattern specifies that an application consists of a data model, presentation information, and control information, and Ring's follow this pattern clearly.

Motivation

This split of role lets the ring system to be easily multi platform, the view depends a lot of the platform where the system is installed on whereas the daemon (i.e. the model) is the same for all the platform. Even if each of them has some specific characteristic, this pattern allows the designers and the developers to design a common code for a C++ project and make the development shorter otherwise they should have to create a version for each one of the targeted platform. This pattern lets the developers to change the view without touching the model if they want, by example in the case of an update for the user interface.

Identification

In Ring architecture is divided in daemon and gnome-client is an example.

Diagram

The Figure 2.2 summarizes 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.

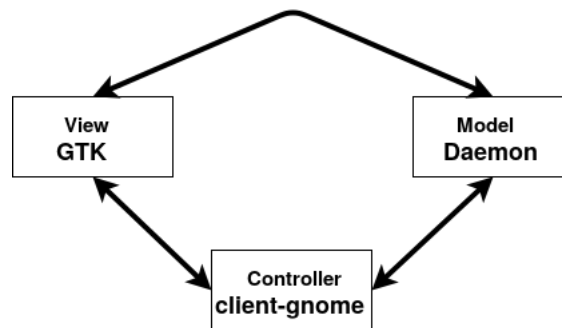


Figure 2.2: Model View Control Diagram

2.5.4 Template Pattern

In Template pattern, 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. This pattern comes under behavior pattern category.

Intent

The intention to use Template is to reduce time and lines of code.

Example of Code

Example of code, taken from Shared pointer header that is used in several classes, it uses the template Call.

Analysis

The motivation to the template is to reduce the time related creating new methods and properties that can be used for several classes.

Identification

To identify the use of Template in Ring was straight from the source code.

2.6 Anti Pattern

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

2.6.1 Dead Code

Explanation

Throughout the software development and versions some codes can became obsolete. As consequence part 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, 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 to understand the code.

Solution

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

2.6.2 Complex Class

Explanation

Some classes and methods are unnecessarily complex. On our case, chained if that can difficult the reading of the code.

Code example

The code with more than 50 lines straight, including complex functions executions, and several chained ifs, and mutex call. An example of if on the Class Call, between lines 408 and 460, so 52 lines.

Another example comes from "*sipvoiplink*" in which *try_respond_stateless* includes a lot of if-return statements all along more than 120 lines.

Analysis

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

Solution

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

2.6.3 Blob

Explanation

The Blob is an anti-pattern that do not precisely define the role of class. The class allocates several responsibilities that may not be related with the class.

Code example

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

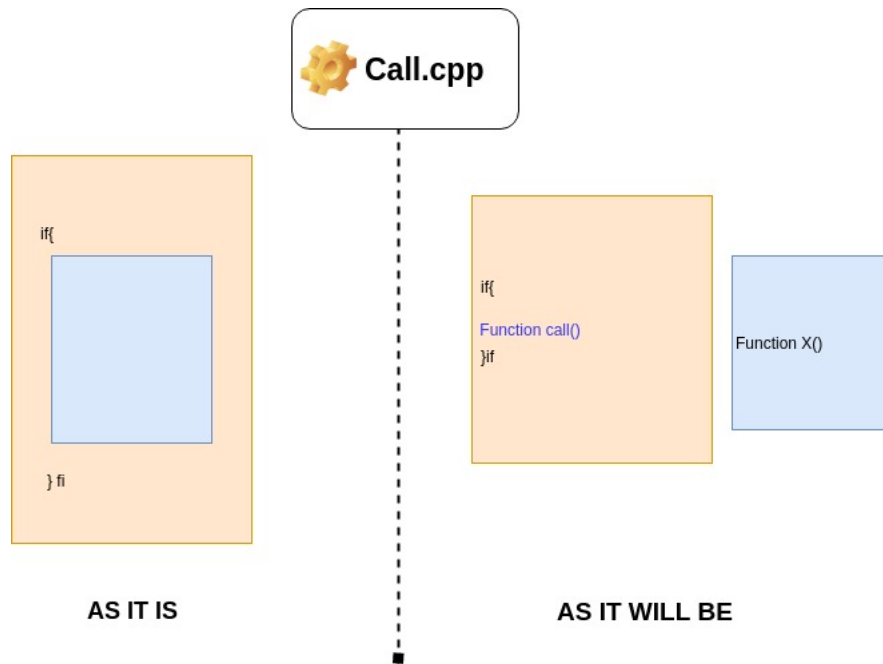


Figure 2.3: Sketch illustrating a way of reducing the complexity of call.cpp

Analysis

The problem related with this code is similar with the previous one, if the code works, there is no need to change it.

Solution

A proper solution would be to refactor the two files: `manager.cpp` and `account.cpp` properly defining its roles.

2.7 Problem Exposed

2.7.1 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.

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[qt] IDE, which also enables the use of Gtest for unit tests. Figure 3.2 represents the class diagram of the solution.

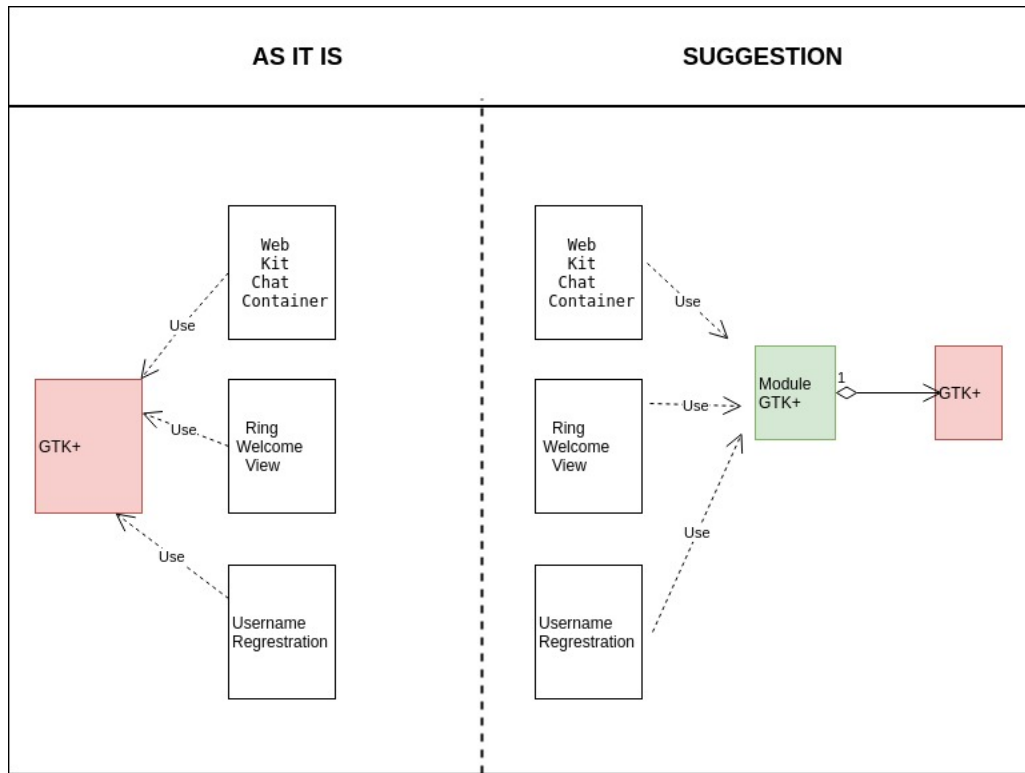


Figure 2.4: Suggestion for GTK+ dependency

2.7.2 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 with

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 simple

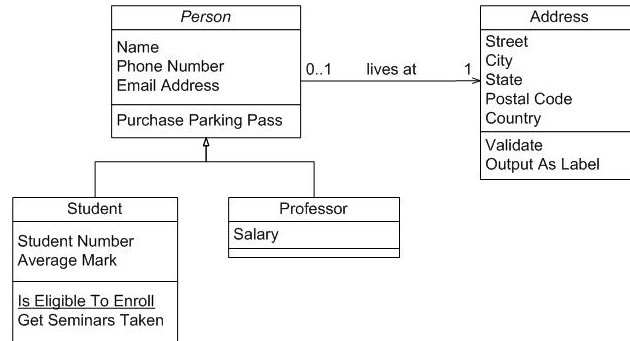


Figure 2.5: Simplified Class Diagram of the solution

2.8 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 invoke GTK+ directly. For example: ringwelcomeview.cpp, Username Registration Box Private.cpp, webview chat context menu.cpp. Considering the View of Ring, which several classes invoke 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

3.1.1 Ring Project

Ring project is a innovative and interesting project that aims to help integrates people around the world. Ring is part of GNU, this mile stone marks our involvement in the Free Software philosophy [official'blog].

3.1.2 Architecture

The current paradigm of Ring project requires a direct knowledge of GTK+ and other libraries [?? Which ones / why ??] .

3.1.3 GTK+

This is a multi-platform toolkit for the graphical user interfaces and can be used from small to large projects. It is an example of direct dependency of the Ring Project.

3.2 Anti patterns and Bad Smells

Ring project has interesting characteristics but also has some drawback where we can highlight Blob and Spaghetti code. For the Blob code a major refactoring would need to be done. Some code smells that can be found on the code are the following:

1. Long Method
2. Duplicated Code
3. Long Parameter List
4. Dead Code

3.3 Description of the problem

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

3.3.1 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.

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[qt] IDE, which also enables the use of Gtest for unit tests. Figure 3.2 represents the class diagram of the solution.

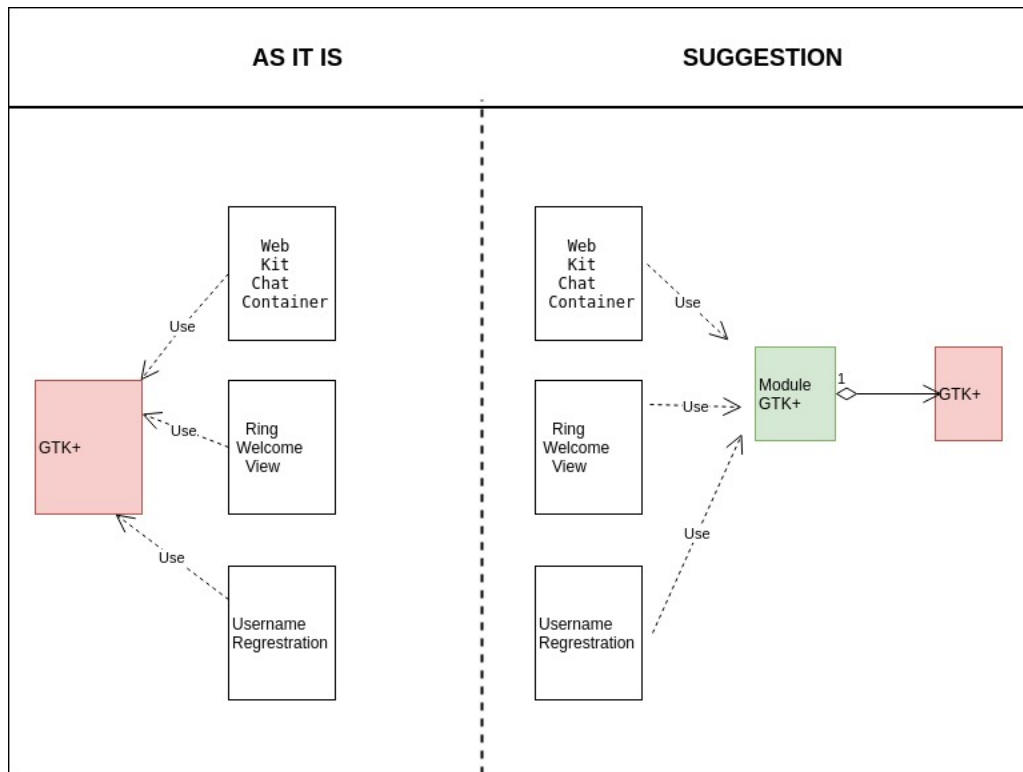


Figure 3.1: Suggestion for GTK+ dependency

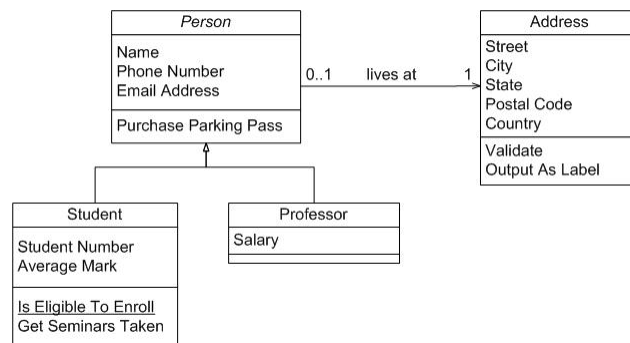


Figure 3.2: Simplified Class Diagram of the solution

3.3.2 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 with

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 simple

3.4 Analysis

3.4.1 Testing

Testing the code is a very important task which can be done in two ways: static and dynamic testing.

Static Testing: It can test and find defects without executing code. Static Testing is done during verification process. This testing includes reviewing of the documents (including source code) and static analysis. This is 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 validation process. For example: unit testing, integration testing, system testing, etc.

Static test: Walk-through

As static testing technique, we use a walk-through process to improve the quality of the code. Although not a form of formal testing, the walk-through process gives the possibility to evaluate the code and then a better quality code.

Dynamic test: Unit Testing

The unit test can be defined as the lower level of testing on the software development, 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, even more in early stages of software development.

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

Mock Testing

The reference [unit'test'ref] gives an insight of the mock testing done.

Framework

The framework used was C++ Testing Framework [c++].

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

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 [accu]. To test the new class, we created a test script to test several properties of it.

3.4.2 Code Metrics

From the metrics point of view, the introduction of this changed the metrics on the point of view of

3.4.3 Documentation

The documentation of the class in the section Appendix.

3.4.4 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 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

3.6 Conclusion



In summary, Ring project is a innovative and interesting project that aims to help integrates people around the world. One of the main problems relative to its architecture is the over dependency and association of the project with other libraries, for example its intrinsic dependency on GTK+.

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.

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.

Appendix A

TP1 - Appendices






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

Appendix B






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 

Appendix C

TP3 - Appendices

```
/** Example of Documentation sphere.cpp Purpose: Calculates the area of a circle and the
volume of a sphere.
@author Unknown @version 1.0 3/17/17 */
/** Returns the area of a circle with the specified radius.
@param radius The radius of the circle. @return The area of the circle. */
double area(double radius);
/** Returns the volume of a sphere with the specified radius.
@param radius The radius of the circle. @return The volume of the sphere. */
double volume(double radius);
// Returns the area of a circle with the specified radius. double area(double radius)
// Returns the volume of a sphere with the specified radius. double volume(double radius)
All References over Testing are inaccurate !!
```