Masters in Informatics Engineering

Internship

Final Report

Implementation of a Mobile Communication Application for BlackBerry 10

Filipe André Maia de Figueiredo

famaia@student.dei.uc.pt

DEI Supervisor:

Prof. Dr. Pedro Furtado

WIT Software supervisor:

Eng. Filipe Figueiredo

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Acknowledgments

To my entire family, in particular to my parents, who made all of this possible.

To both supervisors, Filipe Figueiredo and Pedro Furtado, for their time, patience and support during the whole internship.

To Ana Carolina, my girlfriend and best friend, who always stood by me, helping me and giving me her support to overcome this journey.

To all my friends and colleagues, for their support, friendship and experiences shared during the past six years.

Abstract

Is the smartphone an essential prop for life?

Over the past few years, the number of smartphones' users worldwide is facing an incredibly fast growth. With the smartphones evolution, applications based on Internet access have grown almost exponentially, mostly due to their minimal requirements and cost. *Over The Top* (OTT) applications began to appear and people started to use them to communicate instead of the traditional calls and *Short Message Service* (SMS). It is cheaper – most of these applications are cost free – and have more features, when it comes to comparison with the traditional mechanisms.

To try to cope with the OTT usage growth, *Mobile Network Operators* (MNOs) came together with a solution: the creation of a solution that could compete against the OTT phenomena – the Rich Communication Suite (RCS). RCS specifies a set of communication services, including instant messaging, file transfer, real time video sharing and *Voice over Internet Protocol* (VoIP) calls.

WIT-Software has an RCS Solution called WIT Communications Suite (WCS) that is currently available for the iOS and Android mobile platforms.

With the launch of BlackBerry 10, a mobile phone that runs QNX (a Unix-like system), it is now possible to use the communication stack library developed in C++ by WIT-Software in BlackBerry 10 applications. Therefore, it is easier to bring WCS to BlackBerry 10 platform, and that is the goal of this internship.

And yes! There are new props that can't be dispensed in today's life.

Keywords

"BlackBerry 10", "Communication", "GSMA", "Instant Messaging", "Mobile", "Rich Communications Suite"

Index

Index of Figures

Index of Tables

Glossary

BlackBerry 10	A proprietary mobile operating system
Circuit Switching	Circuit Switching is a type of communication in which a dedicated channel (also called circuit) is established for the duration of the transmission. [1] This technology is commonly used in cellular systems such as GSM.
GSM	GSM (Global System for Mobile communications) is an open, digital cellular technology used for transmitting mobile voice and data services. [2]
OTT application	An over-the-top (OTT) application is any app or service that provides a product over the Internet and bypasses traditional distribution. Services that come over the top are most typically related to media and communication and are generally, if not always, lower in cost than the traditional method of delivery [3].
QNX	QNX is an Unix-like real-time system aimed at the embedded systems market
Redmine	Redmine is a web platform for project management that includes several tools to check the project state, such as charts, task boards, etc.
Third-party application	Programs written (by other companies or individuals) to work within operating systems

Acronyms

AB	Address Book
API	Application Programming Interface
BB10	BlackBerry 10
BBM	BlackBerry Messenger
CS Call	Circuit Switch Call
FT	File Transfer
GSM	Global System for Mobile Communications
GSMA	Global System for Mobile Communications Association
IM	Instant Messaging
IMEI	International Mobile Equipment Identity
IMS	IP Multimedia Subsystem
J2ME	Java 2 Platform, Micro Edition
LTE	Long Term Evolution
MMS	Multimedia Messaging Service
MNO(s)	Mobile Network Operator(s)
MSISDN	Mobile Station International Subscriber Directory Number
MVC	Model View Controller
OMA	Open Mobile Alliance
OS	Operating System
OTP	One Time Password
OTT	Over The Top
P2P	Peer-to-Peer
POC	Proof of Concept

RCS	Rich Communications Suit
SDK	Software Development Kit
SIP	Session Initiation Protocol
SMS	Short Message Service
SOA	Service Oriented Architecture
SVN	Subversion
UI	User Interface
UX	User Experience
VoIP	Voice over IP
WCS	WIT Communications Suite

1. Introduction

This document's aim is to provide a description regarding the work developed during the one-year internship on the context of Thesis/Internship subject, as part of the syllabus of the Master in Informatics Engineering course, accomplished at the Sciences and Technology Faculty – Coimbra's University.

The internship took place at WIT-Software, S.A [4]. The supervisors where: Filipe Figueiredo, Eng. (Software Developer at WIT-Software) and Professor Doctor Pedro Furtado (professor at University of Coimbra).

1.1. The Company

WIT-Software is a company with a high specialization level developing innovative applications and services for Mobile Operators and Mobile Internet Companies. Founded in 2001, as a spin-off from the University of Coimbra, WIT-Software soon established a partnership with Vodafone Portugal, with which it continues to work to this day. Over the years, the company has registered a high growth in the telecommunications sector and this continued performance allows the company to mark an outstanding position in the worldwide marketplace. WIT has customers in 15 countries all over the world, including major operators such as Vodafone, Telefónica, Orange, Deutsche Telekom and TeliaSonera.

From the organizational point of view, the company is divided into three Business Units: 1) Telco; 2) Mobile and 3) TV. The Telco Business Unit provides converged solutions based on the IP Multimedia Subsystem (IMS) for voice (VoIP, Mobile VoIP and Voice over LTE), messaging (SMS, MMS and IM), Rich Communication Suite (RCS-e, RCS2.0 and RCS 5.0/5.1) and Multimedia Telephony Services (MMTel). The solution range includes a complete set of RCS client applications (for iPhone, Android, iPad, tablets, PC and Web browsers) and an RCS Application Server capable to deliver full RCS functionality in IMS and pre-IMS networks. [5] The Mobile Business Unit has been developing mobile services for mobile platforms (Android, iOS, Windows Phone, BlackBerry, J2ME and Symbian) for Telco Operators, Banks, Utility and Media companies, etc. The TV Business Unit it is focused on the development of software applications for Cable and IPTV (Internet Protocol Television) operators.

WIT's success is being achieved by its very high day-by-day performance and a peculiar way of facing the nowadays global challenges, bearing in mind the competitive advantage for a rich set of clients. And, naturally, the external recognition appears by means of *performance awards*. Since 2009, WIT-Software is considered a PME Líder (Small and Medium Enterprise leader) - always with excellence [6]. The results come from the financial stability too.

WIT is also certified in Quality standards (ISO 9001) [7], Environment (EN ISO 14001) [8] and Innovation (NP EN 4457) [9].

Recently, WIT-Software was considered one of the two most innovative SMEs of the year, winning the award PME Inovação COTEC-BPI 2013 (SME Innovation COTEC-BPI 2013).

1.2. Motivation

It is difficult to live without a mobile phone nowadays – this is easy to perceive just by looking around and counting the people that do not have already one, two or even more! If we add to this fact the mobile communications system evolution - greater coverage, faster connections, decrease in costs for data packets, etc. – we will have the recipe for a new arrangement: the smartphone concept.

Smartphone is a mobile devices category providing advanced capabilities beyond a typical mobile phone [10] (for example, email clients, internet browsers, communication over IP applications, etc. are not things that can be seen in every mobile phone).

Over the past few years, the number of smartphones users worldwide is facing an incredibly fast growth. A study conducted in March 23rd, 2010 about the amount of smartphones users worldwide predicts that in 2014 this number would be between 1 and 1.5 billion users (please refer to figure 1) [11].

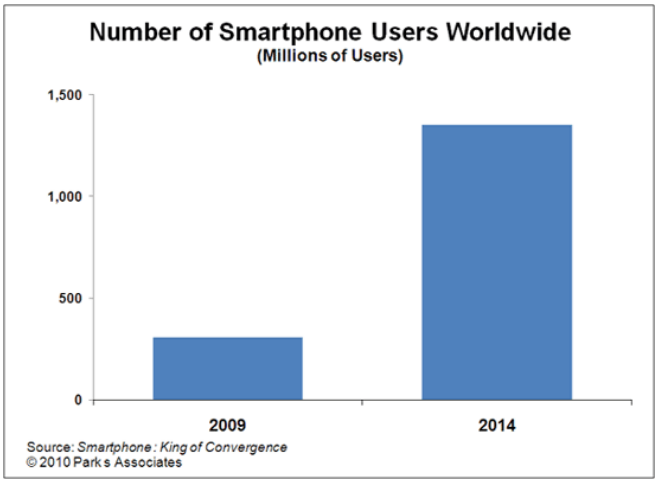

Figure 1 - Expected number of smartphone users in 2014

A more recent study (from January 16th, 2014) suggests that this number will be exceeded and defends it should be around 1.75 billion users, at the end of this year [12].

From these studies, it is not forced to assume that the world is entering in a new technological paradigm: the mobile devices era.

With smartphones evolution, applications based on Internet access have grown almost exponentially, mostly due to their minimal requirements and cost. OTT applications began to appear and people started to use them to communicate instead of the traditional calls and SMS.

It is cheaper for the user (most of these applications are based on the free/freemium[1] system) and has more features than the traditional mechanisms (for example, file share).

The telecom operators consider these applications a serious menace, once they saw their revenues affected with the growth of OTT applications in some mature markets.

A study done in April 29th, 2013 predicted that in the end of 2013 the OTT messages volume would be twice the volume of P2P SMS traffic [13]. The results can be observed in Table 1.

Type	April 2013	December 2013 (prediction)	Number of users	Messages sent by user/day (avg.)
OTT Messages	19.1 Billion	41 Billion	586.3 Million	32.6
SMS	17.6 Billion	19.5 Billion	3.5 Billion	5

Table 1 – OTT usage vs. SMS usage in 2013

This study just confirms that the MNOs have reasons to be anxious with the future of their business.

To try to cope with the OTT usage growth, MNOs come together with a solution: the creation of a solution that could compete against the OTT phenomena. That is how the Rich Communications Suite (RCS) was born.

"*Rich Communications Suite (RCS) is the platform that enables the delivery of communication experiences beyond voice and SMS, providing consumers with instant messaging or chat, live video and file sharing – across devices, on any network*". [14]

To help RCS to stand out in the market, GMSA also created a customer-facing brand: *joyn* [15]. To be accredited by GSMA, RCS applications need to use this brand and its particular design specification.

At the time being, SMS is one of the most ubiquitous communication mechanisms available and users expect them to work for any given cell phone number. MNOs cannot wait to see their huge investment in the IMS infrastructure rewarded with the achievement of this level of ubiquity by their RCS clients, giving them an advantage over OTT IM applications that require download, registration and do not interoperate. It will be difficult to reach this goal, as it requires every handset to have an embedded RCS client. For handsets that do not come with an RCS client, it is possible to download it from the phone's store and install it separately.

As mentioned in the company's description, WIT-Software has a RCS solution called WCS (WIT Communication Suite). This solution is already available for Android and iOS mobile platforms.

With the launch of BlackBerry 10, a mobile phone that runs QNX, it is now possible to use the SIP communication stack library developed in C++ by WIT-Software[2] in BlackBerry 10

[1] **Freemium system –** A system where some of the features are free and others are not free. For example, Skype has free conference calls between clients, but to perform a video conference call, the user has to pay for that particular service.

applications. Therefore, it is easier to create the application: only the UI and the application business layer needs to be developed, since all of the communication features and channels are already implemented.

1.3. Context

To understand the objectives of the internship, it is important to get familiarized with its context, which will be described at the current section.

1.3.1. The Rich Communication Suite

To face the threat constituted by the growth of OTT applications usage, the GSMA (organization composed by over 800 MNOs) launched the **Rich Communication Suite (RCS)** initiative in the year of 2008. RCS specifies a set of communication services including instant messaging, file transfer, real time video sharing and VoIP calls that, like SMS and MMS, do not require account registration and take advantage of the telecommunications network infrastructure, in particular of the IP Multimedia Subsystem (IMS) network and the standard IP protocols such as the SIP (Session Initiation Protocol). The RCS specification details both client and network components and describes all the necessary mechanisms and procedures to make sure that the RCS services are interoperable and work across all RCS clients and networks. As mentioned above, the first version of the RCS specification was released in 2008. Despite of RCS, as a commercially available service, is still a child – very close to the beginning. The first RCS client was released in 2012, while other OTT apps have been available for several years (Skype, for example).

RCS is built upon the IMS architectural framework, however this framework does not ensure the interoperability of services across devices, operators and suppliers. To ensure that, the Open Mobile Alliance (OMA) was formed by MNOs, content and service providers and IT vendors. The purpose of this alliance is to create open technical specifications for services (most of the time called service enablers) that are interoperable, reusable and extensible in order to reduce their implementation and operational costs.

RCS is based on the capability discovery mechanism and the Enhanced Address Book (real time updated address book where the contacts information is built from the RCS features that they support – the capabilities). For example, if contact X is only able to use the chat feature, when contact Y opens contact X profile, only the chat feature will be available for use. The Figure 2 is a screenshot of a contact's details page (from the WCS's *joyn* iOS application). In that particular

[2] **Communications Library (COMLib)** - this library is responsible for the communication between the WCS mobile application and the IMS network. It is implemented in C++, and at the beginning of this internship it was running only on iOS and Android mobile devices. At this moment, COMLib is running on the BlackBerry 10 platform too (ported by the trainee).

case, the contact is available to perform chat, voice calls and video calls (yellow icon) but it cannot receive file transfers (gray icon). The available services of a contact depends on several factors like the RCS version of its client application and the services provided by its MNO.

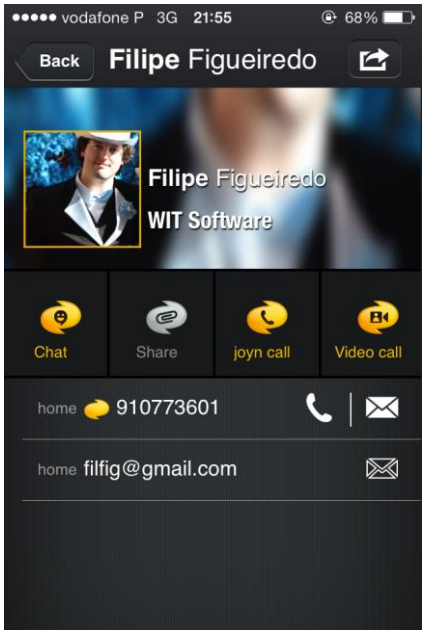

Figure 2 – Contact details page

The RCS specification defines two type of clients: 1) Embeded Clients and 2) Downloadable clients. Embeded clients are part of an handset implementation and they have fully integration with all native applications (like the Galery, the Contacts, etc.). Typically, this type of clients provides a better usage of the application – thre user can, for example, share a contact directly from the Contacts native application. The downloadable applications are typically implemented with the platform's SDK, so they are tied to the supported APIs. These applications can be downloaded from the mobile phone's applications store.

Over the past few years, changes were made to the RCS specification and several versions of it have been released. The work that will be developed during this internship is based on RCS Release 5, more specifically on the RCS Release 5.1 v2.0, an updated version of the RCS 5 release.

1.3.2. The Message+ Application

Message+ is one of the applications that compose the WCS Product and it is based on the RCS spec. However, *Message+* requirements do not match the entire RCS spec requirements (for example, *Message+* does not have calls).

In the company's web site, it is possible to read a good description of this application, which will be quoted next:

"*For those Operators that do not want to launch Voice-over-IP, while CS voice is still the main source of revenue, we have created a lightweight version of RCS, called WIT Message+. This proposition is only IP messaging, tightly integrated with legacy messaging (SMS, MMS). WIT Message+ is as powerful as the iMessage service from Apple, but is available in Android and iOS platforms and is compliant with Telecom Standards"* [16].

WIT Message+ is already available for Android and iOS platforms through the devices' Application Store. The application was bought by Vodafone (so far) and its name is "*Vodafone Message +"*.

The final output of the internship is a working demo of this application's main features (this subject will be explored in the next section).

1.4. Objectives

The internship scope was to extend the already existing RCS solution owned by WIT-Software (WCS) to the BlackBerry 10 platform. WCS is composed by more than one application, among them we have *joyn* and *Message+*. Both of these applications share the same communication library – only the UI and the enable/available features change. At the beginning of this internship, the main goal was to produce a functional demo of some features available in the *joyn* application running on BB10 platform. However, a few weeks after the beginning of the internship, one of the company's clients demonstrated interest on having *Message+* running on BB10 platform. So, the internship changed its scope at the end of the first semester. As mentioned above, this change only affects the UI and the available features of the application. All the work related with the communication library was not affected.

The first step of the internship consisted on the porting of the existing communication library (COMLib) to the BB10 platform. After the conclusion of this step, it was necessary to implement the specific platform services in the COMLib. As an example of these services, the communication library needs to have access to the IMEI of the phone where the application is running. However, the process to get this information is specific to each platform (this will be detailed later in this document). At the end of the implementation of these essential tasks related with COMLib (some others were implemented later), it was time to finally use it. So, the implementation of the UI and all the business logic necessary to communicate between the UI and COMLib began.

Message+ was already implemented for Android and iOS platforms, so they were used as a guideline for the design and the operation principles of the BlackBerry 10 application. The final output of the internship allows its users to use the next features available in *Message+*:

- Exchange IM between 2 people;
- Exchange image files between 2 people;
- Send and receive SMS and MMS;

These were the selected features to be implemented in the context of this internship, because they represent the main features provided by *Message+*. To have the "full version" of *Message+*, some other features should be implemented:

- Exchange IM between more than 2 people;
- Exchange video, audio, vCard or location between 2 or more people;
- Send SMS/MMS broadcast to more than 2 people;

These last features were out of the internship's scope, so they were considered as future work, if the company intends to invest in this project.

1.5. Internship's Main Challenges

During the internship's course the trainee had to deal with multiple challenges. The first one consisted on the porting of COMLib to run on BlackBerry 10 (including its dependencies).

After that, the trainee felt the need to learn about BB10 programming, because he had no experience in this area. Later, the trainee had to design the entire architecture of the application before starting the implementation process. This represented a high-responsibility task, because the entire performance and scalability/flexibility of the application depended on it. During the implementation process some challenges appeared too, as the lack of performance existing on the access to the native contacts. All these challenges and respective solutions will be documented later in this report.

1.6. Document Structure

This document is divided in the following chapters:

- **Chapter 1: Introduction –** This chapter contains the description of the company where the internship took place, the motivation for the project, the description of the company's product and the presentation of the goals to achieve at the end of the internship.
- **Chapter 2: State of the Art** – This chapter describes the analysis of competitors performed and the analysis of existing technologies to create BlackBerry 10 applications.
- **Chapter 3: Requirements** – The requirements elicitation is described in this section
- **Chapter 4: System Architecture –** In this section the architecture of the already existing system and the architecture of the application to be developed will be explained.
- **Chapter 5: Software Development Methodology –** This chapter's scope is to describe the planning of the internship, the methodology used for this project and the risk analysis made.
- **Chapter 6: Implementation and Software Quality** – Details about the implementation process and tests will be explained in this chapter.
- **Chapter 7**: **Conclusions** – This is the final chapter of this document. In it, will be presented a final reflection about the work done during the internship and a brief description of what needs to be implemented if the company decides to continue the development of the project.

Apart from this document, the internship documentation is also comprised with the following appendixes, available in the Confidential CD:

- WIT_Communications_Suit_for_BlackBerry10_ - _Appendix_A_-_State_of_the_Art_
- WIT_Communications_Suit_for_BlackBerry10_ - _Appendix_B_-_ProductBacklog_
- WIT_Communications_Suit_for_BlackBerry10_ - _Appendix_C_-_Software_Quality_

2. State of the Art

The first part of this section consists on current market state analysis, regarding to applications like VoIP clients, IM clients and other applications used to exchange multimedia between people.

It is very important to do a good and complete comparative analysis between all the existing applications with the same functionalities that the company's product supports.

Some of the applications analyzed do not support BlackBerry platform yet, but the trainee decided to analyze them as well on other platforms, taking into account that they might become future competitors.

Since *joyn* is a standard application whose main features are defined by GSMA, there are other companies that develop *joyn* applications like WCS. So, it is important to include such applications in the present comparative analysis.

The second and last part of this section will focus on the technologies that will be used to develop the application (frontend and backend).

This analysis was conducted during the month of October 2013. Therefore, some information will be inevitably outdated in the course of time.

2.1. Competitors

The competition is divided in two main aspects: 1) direct and 2) indirect competition. The companies which develop a product similar to the one owned by WIT-Software, in this case, companies developing *joyn* clients, and looking after to sell their final product to the same costumers that WIT-Software wants to acquire, represent the direct competition. Summit Tech, Jibe Mobile, Nable Communications Inc. and Neusoft Corporation represent these types of competitors in this analysis.

The indirect competition is represented by companies developing products containing features also covered by WCS. Although it is not a product having exactly the same features as the WCS one, it can be a competitor in some features. WhatsApp Messenger, BlackBerry Messenger, WeChat, Line, KakaoTalk Messenger, Viber, Skype, Google Hangouts and Facebook Messenger exemplify the most important indirect competitors in this analysis and they will be described here. However, 29 applications were explored and they are included in the comparative table on the **Appendix A**.

For a more detailed analysis of each application and about the ones not mentioned here, please refer to the appendix named above.

2.1.1. Indirect Competitors

This section contains the detailed analysis on the most important indirect competitors. A description of the products and their main features is given at the beginning of this subtopic. To look for a comparative analysis between all the applications, please refer to the table available at the end of the Competitors chapter.

WhatsApp Messenger

WhatsApp Messenger [17] is a cross-platform mobile messaging application allowing the users to exchange messages without the need to pay for SMS. WhatsApp Messenger was first released in August 2009, when the use of smartphones started to grow exponentially every day and people began to stop using SMS to use communication over IP applications.

Per day, 20 billion messages [18] are exchanged between WhatsApp Messenger users across the supported platforms: iPhone, BlackBerry, Android, Windows Phone and Symbian.

Nowadays, WhatsApp Messenger counts to with about 350 million [19] active users per month and is free for the first year (with a symbolic cost of 0.99 US Dollars – 0.79€ per year after that period).

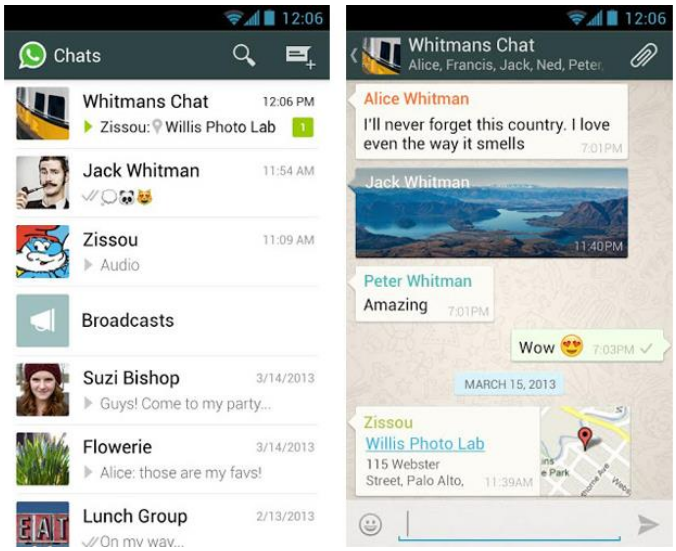

Figure 3 - WhatsApp Messenger interface

Key features:

- IM (1-to-1 and group till 50 friends);
- File Transfer in Chat and Group Chat;
- Location and vCard[3] share;
- Voice and Video messages;
- Presence[4] status.

[3] **vCard –** file format for electronic business (or personal) cards. They can contain name, address information, phone numbers, e-mail and other details about someone.

BlackBerry Messenger

BlackBerry Messenger [20] is a very popular native application of BlackBerry OS. It was first released in 2006, developed by RIM (Research In Motion Limited, the manufacturer of BlackBerry devices) only for BlackBerry phones, but recently, in 21st October 2013, Android and iPhone versions of this application were released too and are acquiring a large number of users (in the first 24 hours after the release of BBM for iPhone and Android, 10 million people downloaded the application).

RIM's application has about 51 million active users per day that exchange about 10 billion messages between them, every day. [21]

BlackBerry Messenger has different functionalities, depending on the platform where it is installed. In BlackBerry platform, considering the latest software version available (BlackBerry 10), there are a lot of features that are not supported by the iPhone and the Android versions. Most important ones are video and voice messages. To use these two features, the user needs to install on its BlackBerry the corresponding add-ons (BBM Video and BBM Voice).

Figure 4 - BlackBerry Messenger interface

Key features:

- IM (1-to-1 and group till 30 friends)
- File transfer (video, images and audio)
- Voice messages
- Screen Sharing
- 3rd party apps integration
- Voice and Video calls over IP (only in BB10 devices)

[4] **Presence –** The application can know the status of the user at any time (Online, Offline, Busy, etc.)

WeChat

WeChat is a mobile text and voice messaging communication service developed by Tencent in China, first released in 2011 [22]. WeChat is supported on Wi-Fi, 2G, 3G and 4G data networks.

With an attractive UI and UX available to its users, this application supports many important features that others don't, for example it supports calls and video calls (WhatsApp, one of the main competitors, does not.) This is leading the application to a fast growth (Nowadays, about 236 million people [23] are using WeChat).

WeChat is available for iOS (iPhone, iPad and iPod), Android, BlackBerry, Windows Phone and Symbian.

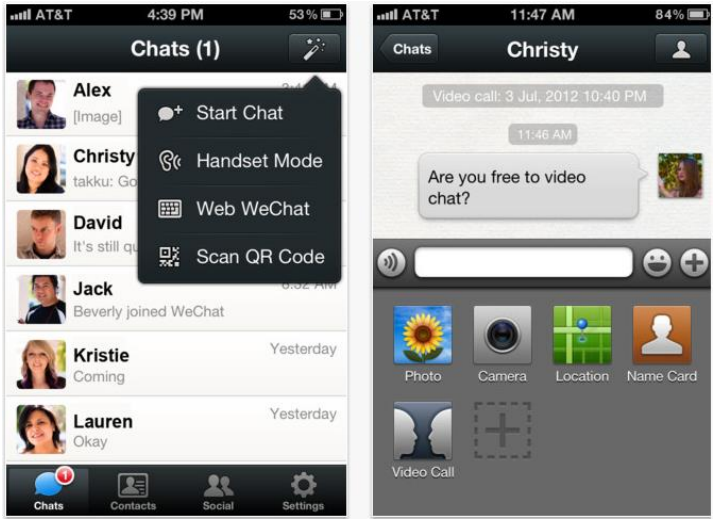

Figure 5 - WeChat interface

Key features:

- IM (1-to-1 and group till 40 people);
- File transfer in Chat and Group Chat;
- Voice messages;
- Location and vCard share;
- Video over IP Call;
- Facebook Integration.

Line

Line [24] is a communication application for free calls and messages that was born after the Tõhoku earthquake in March 2011 [25]. With the telecommunications infrastructure damaged nationwide, NHN Japan employees needed to use Internet-based resources to communicate, and with this in mind, company's engineers developed Line to exchange between them unlimited calls and messages.

Line is growing fast, having now approximately 250 million of users [26] and is available for iOS, Android, BlackBerry, Windows Phone, Windows (PC) and Mac OS platforms.

Figure 6 - Line interface

Key features:

- IM (1-to-1 and group till 100 people);
- File transfer in Chat and Group Chat;
- Location and vCard share;
- VoIP Call;
- Video over IP Call;
- Voice and Video messages;
- Games;
- Facebook integration;
- In-call chat and sharing;

KakaoTalk Messenger

KakaoTalk Messenger is a software application for mobile devices that allows its users to exchange communication using free text messages and calls.

KakaoTalk was launched on March 18th, 2010 [27] and currently counts with about 100 million registered users. This application is available for iOS, Android, BlackBerry, Windows Phone, Bada OS and Windows (PC).

Considering all the analyzed applications, KakaoTalk is far from being the most used among it. Line and WeChat are leaving KakaoTalk behind and are decreasing their distance from WhatsApp [28].

Figure 7 - KakaoTalk Messenger interface

Key features:

- IM (1-to-1 and group);
- File transfer in Chat and Group Chat;
- Location and vCard share;
- VoIP call;
- Video over IP call;
- In-call chat and sharing;
- Voice messages;
- Games.

Viber

Viber [29] was first released in December 2nd, 2010 [30], for iPhone only. Back in those days, Skype was leading almost alone the market of communications over IP, and Viber emerged to stop that monopoly. Currently, Viber has more than 200 million users [31] and is a multiplatform application (with support to iOS, Android, BlackBerry, Symbian, Bada and Windows Phone devices and also has Desktop versions to Windows, Mac and Linux OS).

The investment made by Viber for multiplatform applications brought an increase of users to the company. Right now, Viber has more Android users than iOS, despite having been iOS version the first one to be released.

Figure 8 - Viber interface

Key features:

- IM (1-to-1 and group);
- File transfer in Chat and Group Chat;
- Location share;
- VoIP calls;
- Presence;
- Call transfer between devices;
- Video message;
- Traditional call (using mobile phone's native call application).

Skype

Skype [32] is a *Freemium* software that allows the users to make voice calls and video calls over the Internet. First released in August 2003 [33], Skype is considered the father of the Over the Top Applications and has about 800 million registered users with 280 million of them monthly active [34]. Skype belongs to Microsoft and absorbed MSN Messenger client in 2013.

This application is available in many platforms, such as iOS, Android, BlackBerry, Windows Phone, Windows (PC), Mac OS and Linux.

Skype has different features, depending on the platform where it runs. To make this analysis consistent, Skype was analyzed using the mobile application and not the Desktop application.

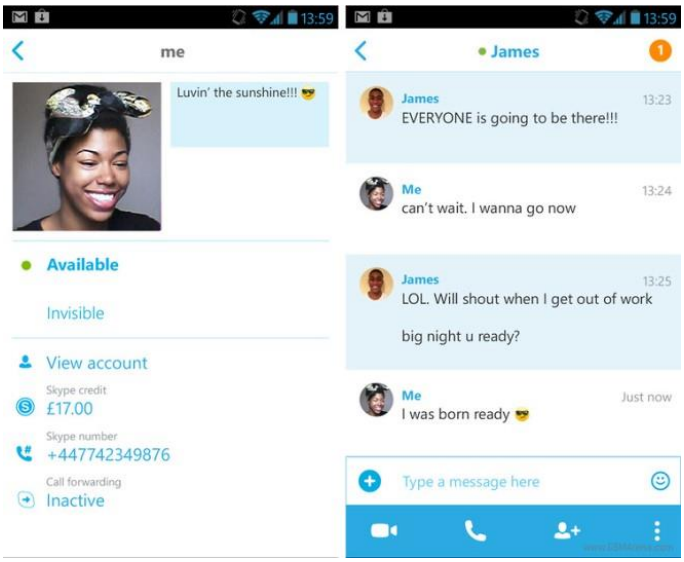

Figure 9 - Skype interface (Android client)

Key features:

- IM (1-to-1 and group);
- File transfer in Chat;
- vCard share;
- VoIP call;
- VoIP conference call;
- Video over IP call;
- In-call chat;
- Video over IP conference (premium feature);
- Traditional calls (using Skype credit);
- Video messages;
- Social network integration.

Google Hangouts

As the name says, Google Hangouts [35] was created by Google and represents the fusion between Google Talk, Google Voice, Google+ chat and Hangouts feature of Google+ (video support), providing to the users an application to communicate with other people by instant messaging and video chat. First released in May 15th, 2013 [36], Hangouts is available in a wide range of platforms: Gmail web client, Google+ web client and in the Android, iOS, Windows and Mac OS devices (these last two as a Google Chrome extension). This application has about 300 million monthly active users, number that is growing fast since the past few months [37].

Google Hangouts is connected with Google+ social network and uses this account to access the contacts and circles, independently of the device where is running.

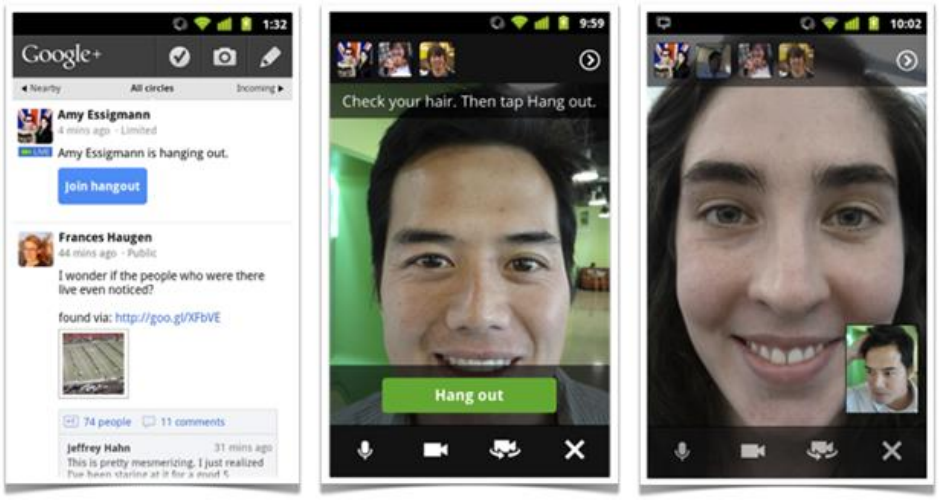

Figure 10- Google Hangouts interface

Key features:

- IM (1-to-1 and group);
- File transfer in Chat and Group Chat;
- IP Voice and Video calls;
- Voice and Video conference call (up to 10 people);
- Call transfer between devices;
- Social network integration;
- Traditional calls (using pre-paid credit);
- Presence status;

Facebook Messenger

Facebook Messenger is the evolution of Facebook web chat for mobile phones. It was first released in August 2011 and one of the purposes to be created was to remove the weight of the Facebook mobile application [38]. The result was the expected, two different applications both with a high performance instead of an overloaded one. About a year ago, a study revealed that Facebook Messenger is available for iPhone, Android and Blackberry and is used monthly by 56.7 million people [39].

Nowadays, Facebook Messenger is more than a simple chat application, as it allows some more operations like calls, file transfer, etc. and is trying to compete against the other OTTs. Its integration with Facebook (the most powerful Social Network existing today) is the main advantage and the differentiating point that allows the fast growth of Facebook Messenger users.

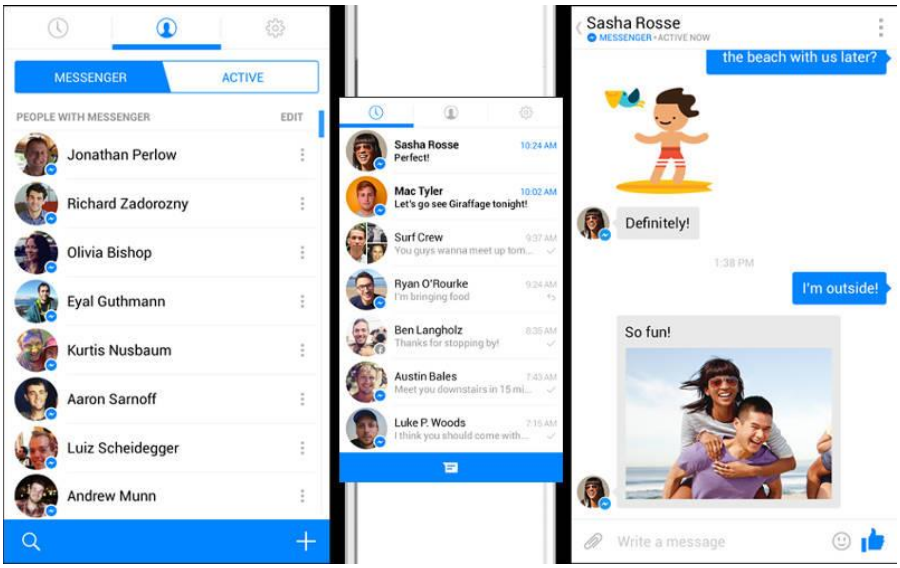

Figure 11 - Facebook Messenger interface

Key features:

- IM (1-to-1 and group);
- File transfer in Chat and Group Chat;
- Location share;
- IP Voice calls;
- Voice messages;
- Social network integration;
- Presence status;

2.1.2. Direct Competitors

As written on the current chapter's introduction, these competitors typify the companies that are developing *joyn* applications to sell to the same clients as WIT-Software: mobile network operators. It is clear that each company tries to improve their product in order to seduce the target clients, but all of them have the same main guideline for the product architecture, and consequently, the same key features: the GSMA RCS specification document. Unfortunately, it was not possible to test these applications because they are not available for download, since they are sold directly to MNOs and they are only accessible to the specific MNOs clients. Consequently, the information that is possible to achieve just by searching for these companies' products is superficial.

The direct competitors are represented by the next four companies:

- **Nable Communications;**
- **Summit Tech Communications;**
- **Neusoft;**
- **Jibe Mobile.**

As previously mentioned, it was not easy to find information on these companies' products, much less differentiating features. So, only the comparative analysis was included in this document. For detailed information about these four companies (history, achievements, etc.) please refer to this chapter's appendix..

2.1.3. Comparative Analysis

Once all the applications have been analyzed, it is important to perform a comparative analysis between them. The diversity of features is high, and most of them will not be in this comparative analysis: only the main features are considered in this section. For a high level of detail, please refer to the complete comparative analysis present in **Appendix A – State of the Art**. Some of the fields corresponding to the Direct Competitors could not be checked due to the lack of information available. The considered features for this small analysis are the following:

- **Voice calls –** ability to perform VoIP calls;
- **Video calls –** ability to perform Video over IP calls.
- **File Transfer –** possibility to exchange contents (images, videos, etc.) between two users
- **Group File Transfer –** possibility to exchange contents (images, vides, etc.) between a group of users
- **In-call Sharing** – ability to share contents during a call
- **Location Share** – possibility to share the current location with other users
- **Chat** – instant messaging between two users
- **Group chat –** instant messaging between a group of users
- **Platforms –** devices supported by the application
- **Network Address Book –** synchronization of contacts with a server to allow the user to access them from any device that uses the service
- **Presence –** possibility to see current state of one user (online, away, etc.).

In the next table, it is possible to see the analyzed competitors and their respective features. WCS is also presented in the comparison.

NOTES:

- Voice and Video Calls in BlackBerry Messenger are only available in BlackBerry devices.
- Group File Transfer in Skype is only available in PC and Mac versions.

Application	Voice Calls	Video Calls	File Transfer	Group File Transfer	In-call Sharing	Location Share	Chat	Group Chat	Network Address Book	Presence	Platforms
WhatsApp Messenger	✗	✗	✓	✓	✗	✗	✓	✓	✓	✓	Android, iPhone, BlackBerry, Windows Phone, Symbian
BlackBerry Messenger	✗	✗	✓	✓	?	✗	✓	✓	✓	✓	Android, iOS, BlackBerry
WeChat	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	Android, iOS, BlackBerry, Windows Phone, Symbian
Line	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	Android, iOS, BlackBerry, Windows Phone, Symbian, PC Windows, Mac
Kakao Talk Messenger	✓	✗	✓	✓	✗	✓	✓	✓	✓	✗	Android, iOS, Windows Phone, Bada, PC Windows
Viber	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	Android, iOS, BlackBerry, Windows Phone, Bada, Symbian, PC Windows, Mac
Skype	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	Android, iOS, BlackBerry, Windows Phone, PC Windows and Mac
Google Hangouts	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	Android, iOS, PC Windows and Mac
Facebook Messenger	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	Android, iPhone, BlackBerry
Nable	✓	✓	✓	✓	✓	✓	✓	✓	?	✓	Android, iOS
Jibe ON (Jibe)	✓	✓	✓	✓	✓	✓	✓	✓	?	✗	Android, iOS
Silta (Neusoft)	✓	✓	✓	✓	✓	✓	✓	✓	?	✓	Android, iOS, Windows Phone, Symbian, PC Windows and Mac
Summit Tech	✓	✓	✓	✓	✓	✓	✓	✓	✓	?	Android, iOS, Windows Phone, PC Windows and Mac
WIT Communications Suite	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	Android, iOS, PC, Web

Table 2 – Comparative Analysis

2.1.4. Conclusions of the Analysis

At the end of the competitors' analysis, it was possible to observe that the number of people nowadays using OTT applications is growing very quickly. This is an expanding market, and the MNOs need to have a product to compete on it - RCS is the chosen way to do that. Most of the indirect competitors already provide some of the RCS main features, so it is important that the RCS providers (companies like WIT Software) develop high quality products with new and differentiating features, or they will be beaten by the competition. The same applies for the direct competitors: new and differentiating features are needed to conquest the customers.

2.2. Technologies

The release of the BlackBerry 10 OS has brought many changes to the way that applications were created for BlackBerry devices. One of the main differences is the language used to create new applications: it was Java and now it is QT (a C++ framework). To WIT Software, this was good news, because the process of implementing a version of WCS for BlackBerry was now simpler – the communication library developed to the other platforms (in C++) could now run almost directly in the BlackBerry 10 devices, without the need of a wrapper like JNI (used in the Android application).

Another great novelty brought by BlackBerry 10 OS relates to the fact that this OS is now running a virtual machine that has Android OS installed. This is an extremely important thing, because now it is easier to bring Android applications to BlackBerry 10 devices (in some particular cases, as it will be explained next).

The new development language allows an easy conversion from Objective C applications (iOS) to BlackBerry, because there are some similarities in the type of objects used by the two platforms.

WCS is already developed for both Android and iOS devices, so one of these two applications could be used as a "guideline" for this project. To do so, it was necessary to analyze both options and choose the one that best fitted our needs.

2.2.1. Porting Applications from Android to BlackBerry 10

Blackberry and Google are working together to facilitate the process of porting Android applications to BlackBerry devices. There were two options to bring Android applications to Blackberry 10:

- Use the repackaging tools for the BlackBerry Runtime for Android applications.
- Port the application code directly and rebuild the application for BlackBerry 10;

2.2.2. Android Repackaging Tools

BlackBerry Runtime for Android Applications is a repackaging tool that uses an existing Android project or Android Application (APK file) and converts it to a BAR file (BlackBerry 10 application) that can run on BlackBerry 10 devices. This is possible because the current operating system that is running on BlackBerry 10 devices runs the Android 4.2.2 (Jelly Bean) as a "virtual machine", enabling the conversion from Android applications to BlackBerry 10 applications.

This conversion enables the application to run on the BlackBerry Runtime Android applications, which supports a number of APIs on BlackBerry 10 but may incur on compatibility or performance differences (for example, the use of native functionalities like the Native Address Book is very unstable, as there is not a perfect mapping between the two platforms). This is not the ideal approach for an Android native application.

When this solution was analyzed for the first time, there was no support for any Android native code[5] at all, for questions of security and system performance (it is important to remember that all content of WCS Communication library is written in C and C++.). So, without the support for Android native code, it is impossible to use this tool without rewriting the entire library in Java to be used by the Android application (that was out of the question in the scope of the internship).

Later, in December 2013, a new version of this tool was released, with support for some Android native code. However, this solution still did not fit the WCS Android application requirements for two main reasons: 1) some of the essential APIs were not supported yet (for example, contacts management still had some problems) and 2) the application performance is still affected (picking the example of the contacts again, the phone needs to map the Android contacts API to the BB10 contacts API and needs to execute procedures to convert data from one platform to another (the Contact class object, for example), which creates overhead).

Since WCS is a powerful application that has high performance requirements and uses a big amount of API's not supported by these tools yet, this option had to be discarded. Port the application directly was the approach to take, using the available iOS application as a guideline. BlackBerry already provides a guide to help mapping Objective C objects and functions to the BlackBerry 10 corresponding ones [40].

2.2.3. Creation of a new application for BlackBerry 10

In addition to the solution discussed above (Android Repackaging Tools), there are three ways to produce BlackBerry 10 applications. The first one uses Adobe Air to bring some application already developed with this technology to BlackBerry 10 (WCS is not implemented with Adobe Air, so this solution was discarded). The second allows the developers to implement the entire

[5] **Android Native Code** – Code that was written in other languages, such as C and C++.

application using HTML 5 with JavaScript. This was a good option, however there was the need to use and support C++ code for the communication library and to access to the platform specific services and API's (like the Contacts API, the SMS/MMS API, etc.) and they were not available for this technology, so this option was discarded too. The last option is called Native, and as the name implies, it consist in the development of the application using the platform's native SDK. That was the right way to do it.

The Native Way (BB10 Native SDK)

This represents the implementation method that was used for the project development – using the platform's native SDK. When this implementation principle was chosen, one of the available project types needed to be picked. These types and their characteristics are described next, and can be seen in the following figure.

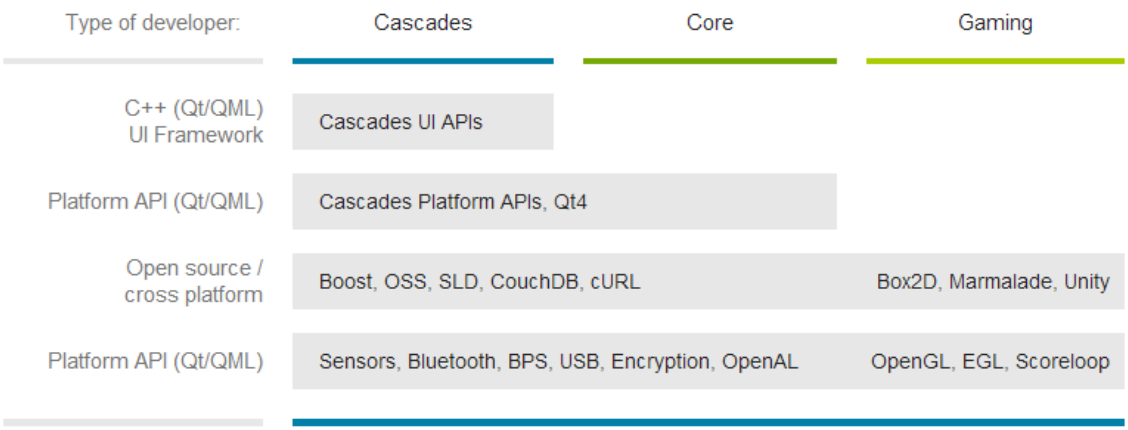

Figure 12 - BlackBerry Native project types

As can be observed in the image above, there are three types of native projects: Gaming, Core and Cascades.

The **Gaming** represents the lower level type of BlackBerry applications. It has access to all the "low level" platform APIs (like Sensors, Bluetooth, etc.) and open source libraries, however it does not have a UI framework included (the entire UI must be implemented using OpenGL and other graphics frameworks) and it does not have access to the Cascades Platform APIs – the "high level" platform APIs (for example, the Contacts API, the Calendar API, etc.).

The **Core** has access to all functionalities available at the **Gamming** type plus the access to the Cascades Platform APIs, however it still does not include the Cascades UI APIs (access to system buttons, lists, text boxes, etc.).

Only the Cascades project has full support for the UI Framework, so that was the type of project chosen.

Another important thing that needed to be defined, was the API level used to develop the project. At the beginning of the implementation process, there were 3 available APIs: 10.0, 10.1 and 10.2. If the application was developed with the 10.2 API, it would not run in old-fashioned devices. However, if it was developed with the 10.0 API, it would run on all devices. The BlackBerry developers were keen to stress the importance of developing the application with the current APIs, because they contain bug fixes, performance improvements, new security

mechanisms, etc. Since all the available BlackBerry 10 devices supported updates up to the 10.2 API, the trainee decided (with the approval of his superiors) to develop the application using this API level. The application is available for any device. If the software version of the target device is outdated, the user must update it, as any data or application compatibility will be lost (all the older applications are supported by the new API, only the opposite is not true).

BlackBerry 10 Cascades project uses Qt and QML as main programming languages. "*Qt is a cross-platform application framework that's used primarily for creating applications that require a UI. Qt uses the standard C++, but extends its functionality using several macros, the most important being the Q_OBJECT macro, which provides an inter-object communication mechanism called signals and slots.*

For building UIs, Qt comes with a specialized markup language called the Qt Modeling Language (QML). QML is a declarative language that's based on JavaScript and is designed for both power and ease of use. Like standard Qt, QML uses concepts such as objects, properties, and signals and slots to let objects communicate with each other." [41]

Cascades applications are built using the Momentics IDE for BlackBerry. The Momentics IDE is an Eclipse-based IDE that is designed for building BlackBerry 10, C and C++ applications. The interface of this application can be explored on the next figure.

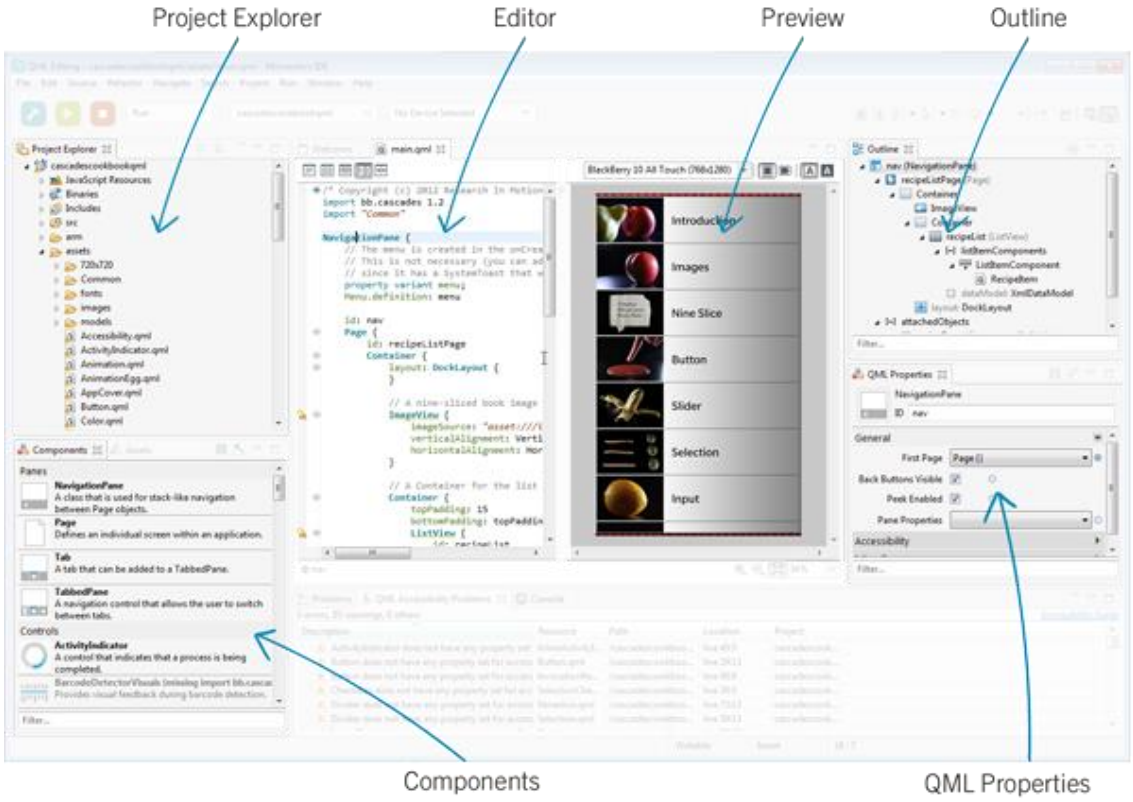

Figure 13 - Momentics IDE user interface and description

2.2.4. Conclusions of the Analysis

From the results of this analysis, it was possible to observe the BlackBerry devices evolution and the development tools currently available to produce applications to this platform. BB10 brought some interesting porting features, like the Android Repackaging Tools. Unfortunately, this feature is still underdeveloped, and it could not be used in this project. Above all, it was interesting to see the quality and robustness evidenced by the BlackBerry 10 devices, smartphones that many people no longer consider.

3. Requirements

This chapter's purpose is to describe the project requirements. They are divided into functional and non-functional requirements. These requirements were later converted into User Stories that represent the project's Product Backlog (this will be detailed in **Chapter 5 – Software Development Methodology**).

The project depends on external libraries, like COMLib. So, the requirements were in constant change during the course of the internship (some API's changed in COMLib, some COMLib's dependencies were added/removed/update, etc.). Some requirements were altered too because of the change of the internship's final output (*joyn* to *Message+*). In this chapter, only the final version of the requirements is presented.

In order to prioritize work, MoSCoW [42] method was used for both categories of the requirements.

The structure used to present these requirements is described next:

- **Req. #**<number> **-** <name> (<priority>): <description>.

3.1. Functional Requirements

A functional requirement represents a functional behavior of a feature to implement. This functional behavior could include intermediate processes that are necessary to execute so the main feature could be completed.

The collected functional requirements for this project are described in the following pages.

COMLib – Dependencies Compilation

COMLib uses a set of external libraries, and some of them are essential to its operation. They first need to be running on BB10 platform so COMLib compilation could be performed. The next requirements represent all the COMLib dependencies, and the corresponding priority/importance to this project.

- **Req. #1 – AVC_H264 (WON'T)**: This dependency represents a video codec used by *joyn* application to make video calls. *Message+* do not need to use it.
- **Req. #2 – Boost (MUST)**: This is an essential COMLib's dependency because it is used by almost every service.
- **Req. #3 – C-ARES (SHOULD)**: COMLib relies on this to resolve DNS queries. However, if for some reason this was incompatible with BB10 platform, it could be replaced.
- **Req. #4 – CURL (MUST)**: A lot of COMLib's services needs to send/receive files through HTTP, so this would be a difficult dependency to replace.

- **Req. #5 – FFMPEG (WON'T)**: Same as AVC_H264.
- **Req. #6 – LIBETPAN (COULD)**: This dependency is used by COMLib's Content Management System (CMS). This service is not essential for *Message+*.
- **Req. #7 – LIBOGG (WON'T)**: This is an Audio codec used by *joyn*. *Message+* does not need it too.
- **Req. #8 – LIBUNWIND (COULD)**: Dependency used by the Android application to manage crash reports. It can be ported in the future, however it was not important for the internship's scope.
- **Req. #9 – LIBVORBIS (WON'T)**: Same as LIBOGG.
- **Req. #10 – LIBXML2 (MUST)**: Library used by COMLib to parse XML files. Very hard to replace.
- **Req. #11 – LOG4CPP (SHOULD)**: Responsible for all COMLib's logging. Could be deactivated, however the existing of logs is very important.
- **Req. #12 – OPENCORE-AMR (WON'T)**: Same as LIBOGG.
- **Req. #13 – OPENSSL (SHOULD)**: Responsible for the encryption of the application's communications. Could be disabled, however that would be translated in a huge security breach.
- **Req. #14 – PJSIP (MUST)**: This is an essential dependency, because the RCS relies on the SIP protocol. Without this, it would be impossible to use COMLib to communicate.
- **Req. #15 – SILK (WON'T)**: Same as LIBOGG.
- **Req. #16 – LITESQL (MUST)**: This was the library responsible for the COMLib's database management. However, it was removed during the second semester, after a refactoring.
- **Req. #17 – SQLCIPHER (MUST)** – This dependency represents the encrypted version of sqlite library. This library is essential to COMLib after the removal of LITESQL.
- **Req. #18 – VO-AMRBENC (WON'T)**: Same as LIBOGG.
- **Req. #18 – X264 (WON'T)**: Same as AVC_H264.

COMLib – Compilation

After the compilation of all dependencies, it was necessary to compile COMLib for BB10. To do that, the following requirement needed to be satisfied:

- **Req. #19 – Add support for BB10 platform in COMLib compilation scripts/code (MUST)**: The available scripts used to compile COMLib were already prepared to Android and iOS platforms. To be used in this project, the support for BB10 platform had to be added.

COMLib – User Input

- **Req. #20 – COMLib user input (MUST) –** If for some reason COMLib needs the user to interact with it, the application must show a user input request and wait for user's answer (for example, send an acknowledgement to the user and wait for its response).

There is a User Input class at COMLib which must be implemented in the application. This class works as a callback (COMLib calls these functions directly).

***Message*+ – Registration**

- **Req. #21 – Accept *Message*+ Terms and Conditions (MUST) –** At the first execution of the application, the first screen must contain the application's terms and conditions. The rest of the app should only be available after the acceptance of these terms.
- **Req. #22 – Request user's phone number (MSISDN) (MUST) –** The application should ask the user for its phone number.

***Message*+ – Settings**

- **Req. #23 – Enable/Disable Chat Service (SHOULD)** – The application should provide a way to enable/disable the chat service (the user can use the application just as a SMS/MMS client).
- **Req. #24 – Chat Service Status (SHOULD)** – The user should be able to know the current session status (Connected, Connecting, Disconnected).
- **Req. #25 – Enable/Disable download of media while using a mobile data connection (SHOULD)** – The user should be able to enable/disable the automatically download of media while using the application connected to a mobile data network.
- **Req. #26 – Enable/Disable download of media while using a Wi-Fi connection (SHOULD)** – The user should be able to enable/disable the automatically download of media while using the application connected to a Wi-Fi network.
- **Req. #27 – Enable/Disable download of media while using the application in roaming (MUST)** – The user must be able to enable/disable the automatically download of media while using the application in roaming.

***Message*+ – Contacts**

- **Req. #28 – Select between "Most Contacted Contacts" and "All Contacts" (COULD)** – At the contacts view, the user could choose between most contacted contacts (list provided by COMLib) and All Contacts (list provided by contact's native application)
- **Req. #29 – Search for a contact (SHOULD)** – When picking up a contact to start a conversation, the user should be able to filter the contact's list by the name and the phone number.
- **Req. #30 – Pick a contact (MUST)** – The user must be able to choose a contact to start/continue a conversation.

***Message*+ – Chat List**

- **Req. #31 – Access to the application settings (SHOULD) –** At this screen (application's main page) the user should be able to access the application's settings.

- **Req. #32 – Start a conversation (MUST)** – The user must be able to start a conversation from this screen. This option will open the application's contact picker.
- **Req. #33 – Navigate through the last chat list entries (MUST)** – The user must be able to scroll through the list of last conversations.
- **Req. #34 – See the latest message preview (chat only) (SHOULD)** – It should be visible to the user a message preview from the last chat message available at a conversation.
- **Req. #35 – Open a conversation (MUST)** – It must be possible to choose one conversation from the list and open it.
- **Req. #36 – See unread messages' count (SHOULD)** – If the user has unread messages at some conversation, the unread messages count should be visible at the respective chat list's entry.

***Message+* – Conversation window**

- **Req. #37 – See conversation history (MUST)** – The user must be able to scroll through the conversation history and see older messages.
- **Req. #38 – Track message status (SHOULD)** – It should be possible to track a sent message status (delivery, pending, etc.).
- **Req. #39 – Share an image file (MUST)** – The user must be able select an image file from the gallery – or take a picture with the phone's camera – and share it with the other contact.
- **Req. #40 – View the contact's capabilities (MUST)** – The user must know if the contact is RCS capable if only SMS/MMS could be exchanged.
- **Req. #41 – Switch Technology (only if the contact has RCS capabilities) (SHOULD)** – If the contact is RCS capable, the user should be able to choose if the communication will occur through RCS or through the traditional mechanisms (SMS/MMS).
- **Req. #42 – Write a new message (MUST)** – It must exist a text area so that the user can write some text to share with the other contact.
- **Req. #43 – Send a message (MUST)** – The user must be able to send a text message to its contact.
- **Req. #44 – Receive a message (MUST)** – The user must be able read a received message at the present conversation.
- **Req. #45 – Accept/Reject an incoming image file transfer (MUST)** – If receiving an image file transfer, if the auto-accept file transfer option is disabled in the settings page, the user must be able to accept/reject this file transfer. However, if the requirements concerning to the auto-accept settings are not implemented, this requirement does not make sense.
- **Req. #46 – Track image file transfer progress (COULD)** – The user could track the image file transference progress through a progress bar.
- **Req. #47 – Cancel an incoming image file transfer transference (COULD)** – For an incoming image file transfer transference, the user could be able to cancel it
- **Req. #48 – See an image preview (SHOULD)** – For an image file transfer, the user should be able to see its thumbnail at the conversation window.

- **Req. #49 – Open image file when the transference is finished (SHOULD) –** After the transfer of an image file is completed, the user should be able to open it by tapping on the image preview.

3.2. Non-Functional Requirements

The non-functional requirements include the conditions that must be met so the features available at this project's proposal could be implemented with the quality and the principles that WIT Software wishes for their projects.

The collected non-functional requirements for this project are described in the following pages. Contrary to previous section, here the requirements will not be grouped according to some subject because, in most of the cases, the subject is already the requirement itself.

- **Req. #50 – Implementation according to both BB10 and WIT Software patterns (SHOULD) –** For a good integration in the Message+ product, the application should, whenever possible, follow the guidelines provided at BlackBerry 10 Developers website [43] and the programming patterns adopted by the company's WCS team.
- **Req. #51 – Application Performance (SHOULD) –** Performance is something to take into account before the implementation of any feature. All the features should be optimized whenever possible.
- **Req. #52 – Application UI/UX (MUST) –** The application's UI/UX must follow the principles of BB10 application and, above all, the *Message+* UI/UX specifications. It is important that, when the user opens the application, he feels the app is running on the same device (for example avoiding other platform specific components). On the other hand, the BB10 application should not be completely different from the other platforms' applications.
- **Req. #52 – Exception Handling (SHOULD) –** No application is perfect – there is always something that could fail. To avoid the application to crash, a good exception handling and error prevention should be done during the implementation of every feature.
- **Req. #53 – Flexibility/Scalability (MUST) –** Every mobile application's architecture should be flexible enough so that features could be added/removed/updated every time needed without giving too much work. For example, for this specific application, it should be possible to change the UI theme, the *strings* contents (and the respective translation) without too much work. To do that, the application's *strings* and image sources are stored in a XML file. At the application's start, these files are loaded and the views are designed with their contents.

These non-functional requirements were taken into account during the entire planning and development time, and they were all fulfilled in most cases.

4. System Architecture

The architecture design represents a crucial phase of every software project. If it is well conducted, the project will be easier to implement and will undoubtedly be better organized.

This chapter aims to describe the architecture of the application to be implemented. It will cover all the features planned and developed during the internship's course. This chapter will be divided in a set of subchapters, explaining the different aspects of the application's architecture.

It is important to highlight that the entire application's architecture were designed by the trainee and it was approved by his supervisor before the implementation of the features.

4.1. Overview

The *Message+* application provides rich contents to the end users through a connection to an IMS network. This network provides services based on network infrastructures and, at the same time, it connects to the circuit-switched network to be able to provide usual communications (CS calls, SMSs, MMSs) keeping in mind the background compatibility. The next image, extracted from WIT's website [44], describes the architecture of WIT's RCS solution in an IMS network.

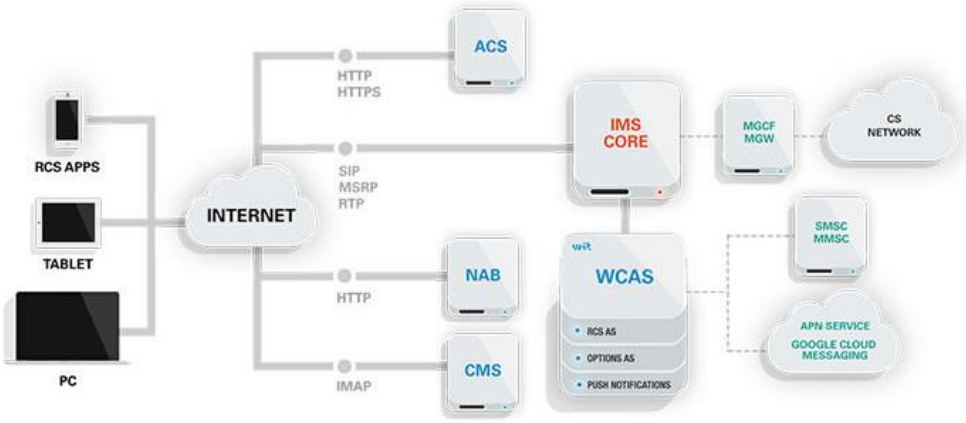

Figure 14 – RCS application deployed in an IMS infrastructure

The communication between the RCS apps and the IMS network must be reliable and handled carefully. This is not easy to achieve, and with the extension of the RCS solution in mind, WIT felt the need to create a way to have a communication stack that could be used by all platforms. That is how COMLib was created.

4.2. COMLib

COMLib (Communications library) is a proprietary library developed by WIT-Software for the development of RCS Clients. The main goal of this communications stack is to handle the entire communication between the RCS applications and the IMS network. It is written in C++ programming language, for portability and performance reasons. Currently, it works both on Android and iOS devices (and was already ported for BlackBerry 10 by the trainee, so it works on this platform too).

Many modules implemented on this stack depends on several third-party libraries. The collection of these libraries is called COMLib SDK and will be described in **Chapter 6 – Implementation and Software Quality.**

The original architecture of COMLib is represented on the following image.

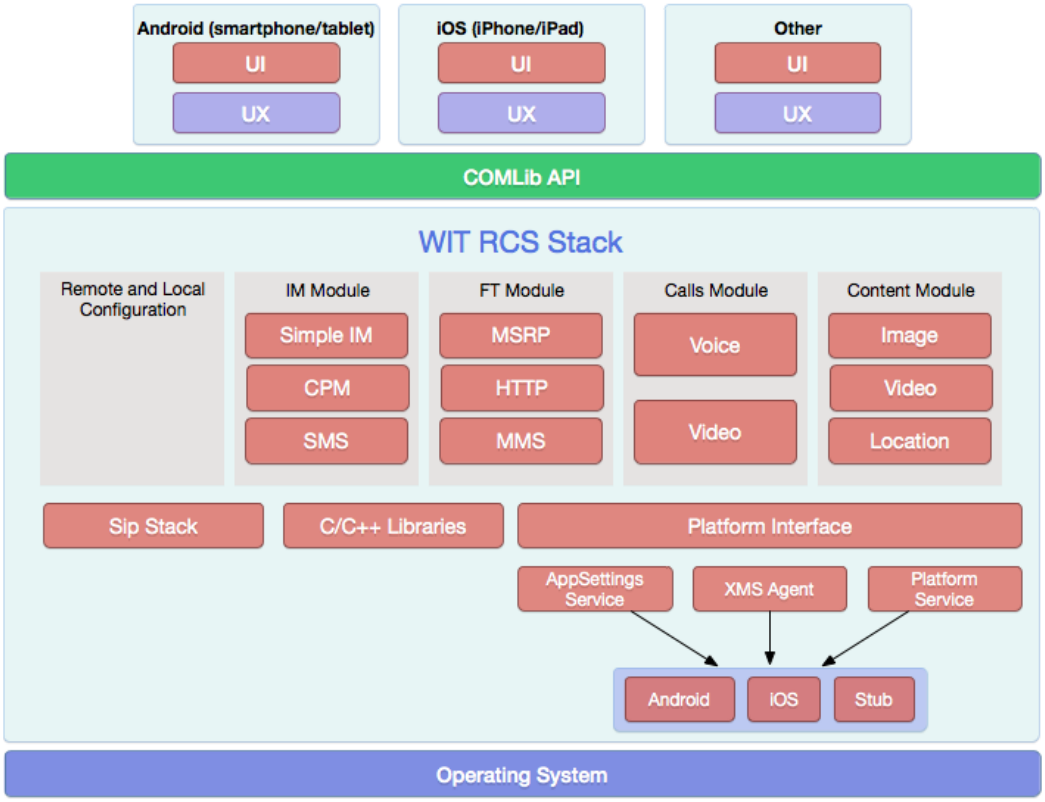

Figure 15 – High-level COMLib's architecture (before the internship)

The entry point of the RCS stack is defined by COMLib API layer, which provides access to all available services. This API allows the application to call COMLib procedures (for example, send a chat message) and subscribe COMLib events. To subscribe these events, the application must send with the subscribe request a callback to be used by COMLib to notify the application when the subscribed event is triggered. Some requests made from the application to the RCS stack may need a callback too. For example, fetch the capabilities of a specific contact can be time consuming. So, to release the app while the request is processed, a callback is sent to COMLib, and when the answer is available, it is executed.

COMLib is divided in 5 principal modules: **Remote and Local Configuration, IM Module, FT Module, Calls Module, Content Module.** None of these modules were changed during the course of the internship.

The internship's tasks related with COMLib were the porting of its dependencies (represented by **C/C++ Libraries** in the image), the porting of the **SIP Stack**, and the implementation of three platform specific services: **AppSettings Service, XMS Agent and Platform Service.** The next figure represents the modifications made to COMLib (in dark blue) and the final state of COMLib's high-level architecture at the end of this internship.

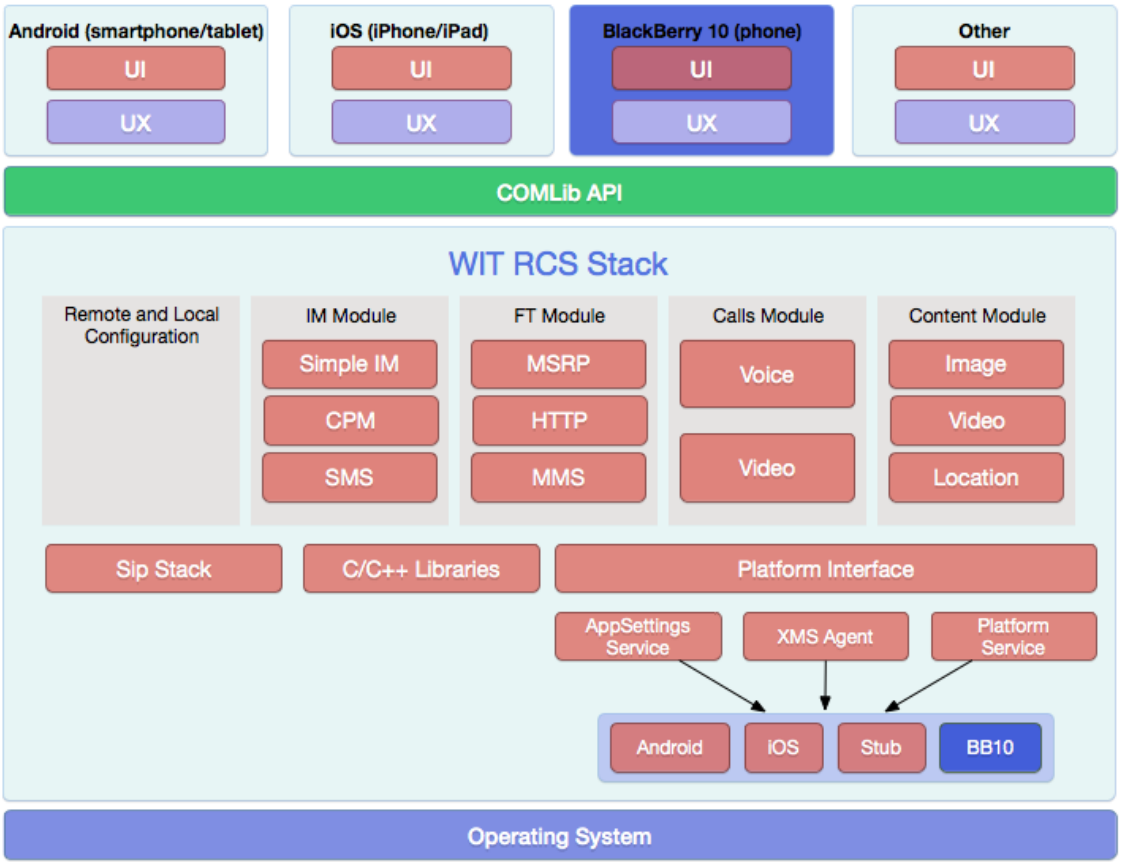

Figure 16 – High-level COMLib's architecture (after the internship)

The **AppSettings Service** contains all the application settings. For example, the path to save the received files, if the auto-download of FT is enabled, etc. The RCS app writes all these settings on the device's storage, and they become accessible from the AppSettings service (which is composed mostly by getters for the stored values). This service provides also the default values for all settings and is used by COMLib to fetch settings' values whenever necessary. For example, if the user is receiving a FT, COMLib needs to know if the download can be started immediately or if the user approval is needed.

Another service that was implemented is the **Platform Service.** This service is responsible to use the platform API to get information from the device. For example, it is possible to get the phone's IMEI, the application state, the hardware vendor, etc. The Platform Service is also responsible to receive and process the binary SMS used to confirm the registration process.

The last service implemented in the RCS stack was the XMS Agent. This service is responsible to send and receive SMS and MMS using the BB10 Messages API.

Each one of these services extend a class with the virtual methods that must be implemented by each platform. The Stub class represents a default implementation of each service, and is the one used when the platform doesn't support a specific service (for example, iOS does not have support for native SMS and MMS, so it uses the Stub implementation of XMS Agent).

4.3. BlackBerry 10 Application Architecture

Each platform has their operation and development principles. The application's source code is written in QT (a C++ framework) and the UI in QML. They communicate using QT signals and slots, as described in the following image.

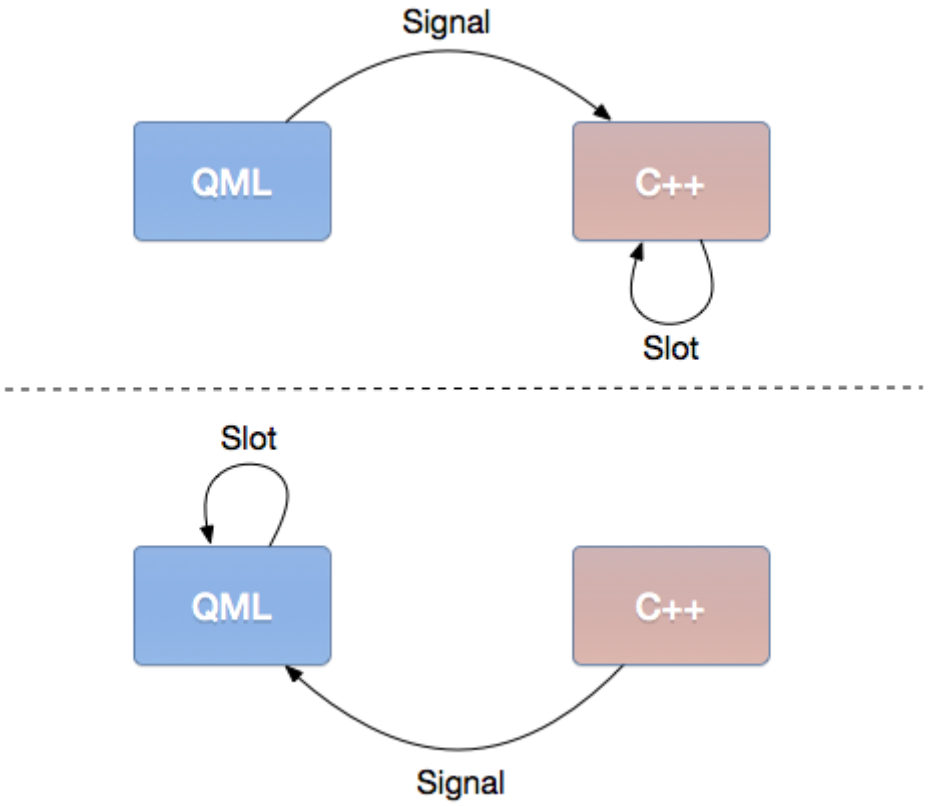

Figure 17 - BlackBerry 10 application typical architecture

The signal/slot mechanism is used to communicate between different threads. For this mechanism to work, it is necessary that the two threads are connected between them (with the connect function provided by QT) and they must use the Q_OBJECT macro. When a signal is

emitted, the threads connected to it will be notified and they will execute the respective slot (defined at the signals destinations)[6].

To maintain consistency between the BB10's application and the Android and iOS applications, the MVC model was used to design the applications architecture. To do so, the trainee, with the approval of his supervisor, decided to use the following application architecture:

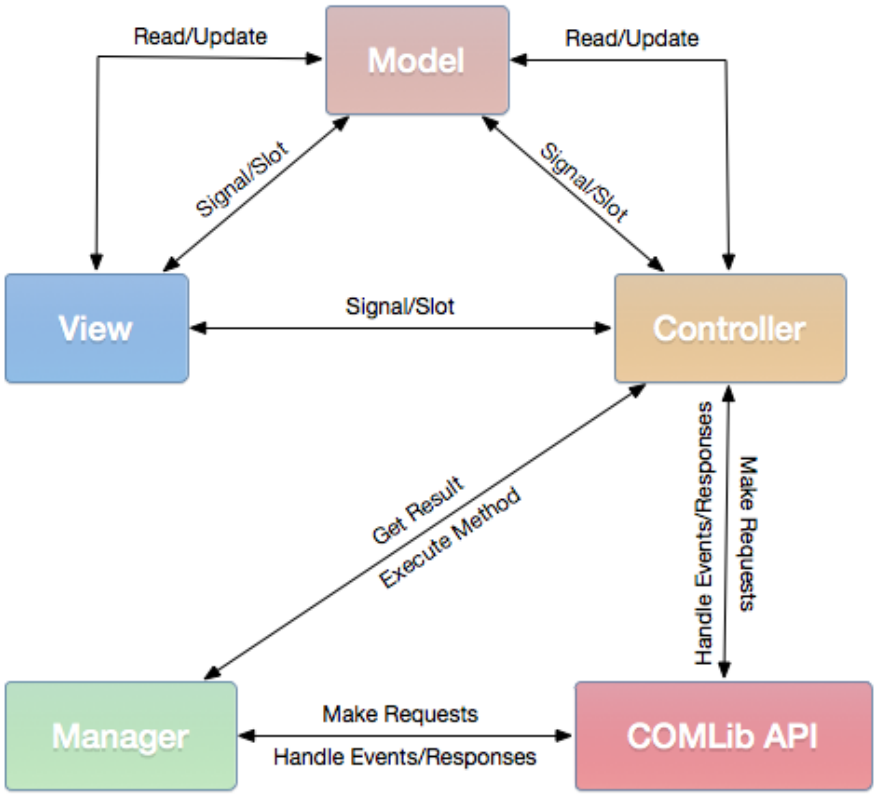

Figure 18 - BlackBerry 10 *Message+* application architecture

A **Model** contains a set of properties that are accessible from the **View** and from the **Controller**. These properties sometimes are bound between the **View** and the **Controller** to specific components, and when they change on the **View**, the **Controller** is notified with a signal. The opposite occurs to. For example, using a bind property for a text box, when the user writes something, the **View** notifies the **Controller** that the value changed. On the other hand, if the **Controller** sets the text of the text box, the **View** receives the signal informing that the corresponding value was changed and that the UI must be refreshed.

Controllers are only instantiated with the respective **View** and **Module**, and their lifecycle depends on the **View's** and **Model's** life cycle (they are all created and destroyed at the same time). **Controllers** are the responsible entities to communicate with the UI components, using a shared **Model** between them to exchange properties. They contain also procedures that can be called directly from the **View** (procedures declared with a QT flag named *Q_INVOKABLE*).

[6] For more information about the signal/slot mechanism, please refer to the available documentation on this subject: http://developer.blackberry.com/native/documentation/cascades/dev/signals_slots/index.html

Beyond that, some procedures where included in the **Controller's** class, because they only make sense when executed in a specific instance of the **Controller.** For example, the chat events provided from COMLib for a specific contact (for example, the "*is typing"* notification) only are necessary when the user has the respective contact's chat thread opened.

Managers work as singletons (only one instance of each manager is available during the entire application's execution) and are instantiated at application's launch. They contain all the static methods that can be executed independently of the **Controller** who invokes them.

Only the Controllers and Managers have access to COMLib's API.

It was important to get familiarized with the BB10 application life cycle to understand the different behaviors that it could present. This allowed the trainee to design and code appropriately to respond to these state changes. The following image describes well the possible states that the application could have and their respective transitions.

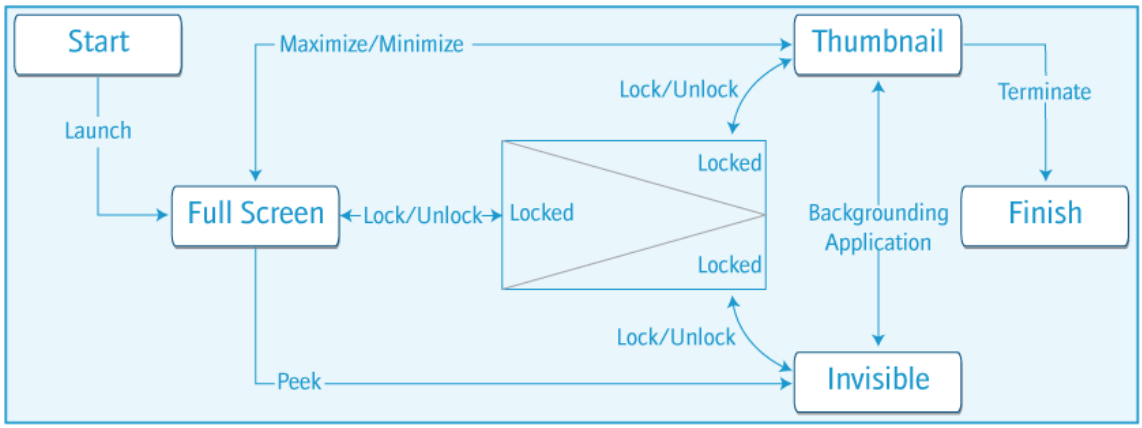

Figure 19 - BB10 application life cycle[7]

The life cycle of the application can be divided into three stages: it starts, runs for a period of time, and then ends. BB10 Native SDK is a multi-threaded, multi-tasking platform that runs multiple applications at the same time. When the application is running in the foreground, it is active. However, it can be interrupted when another application opens and replaces the active one. When the application is no longer in the foreground, it will be sent to run in background. From there, it can either go back to foreground or be closed [45].

4.4. *Message+* for BlackBerry 10

The BB10's application was made from scratch, only using the already existing communications stack. So, an entire process of planning and architecture design needed to be carefully executed, because the entire operation of the application depended on it. The non-

[7] **Image source:** http://developer.blackberry.com/native/files/documentation/core/images/applifecycle.png

functional requirements were taken into consideration while this process was executed, to produce a well-organized and high-performance application.

As mentioned above, the application follows the MVC principle and it's organized in modules, views, controllers and managers. The application has a "main" manager that is instantiated at launch time, and is the responsible to instantiate all the other managers. This manager is used as a controller of the chat list view (this view is always available at the navigation pane stack[8], so it is treated as a "singleton" view) and is responsible for the management of the navigation pane's stack (this is the manager that pushes and pops all the pages on the navigation pane). This administration is done at this particular manager because it is the one connected to COMLib's User Input, (entity responsible for all the COMLib requirements that need user's interaction) which is able to launch new views at any time. So, for synchronization and information consistency reasons, the navigation pane is only administered at this manager. The following image illustrates how the application works.

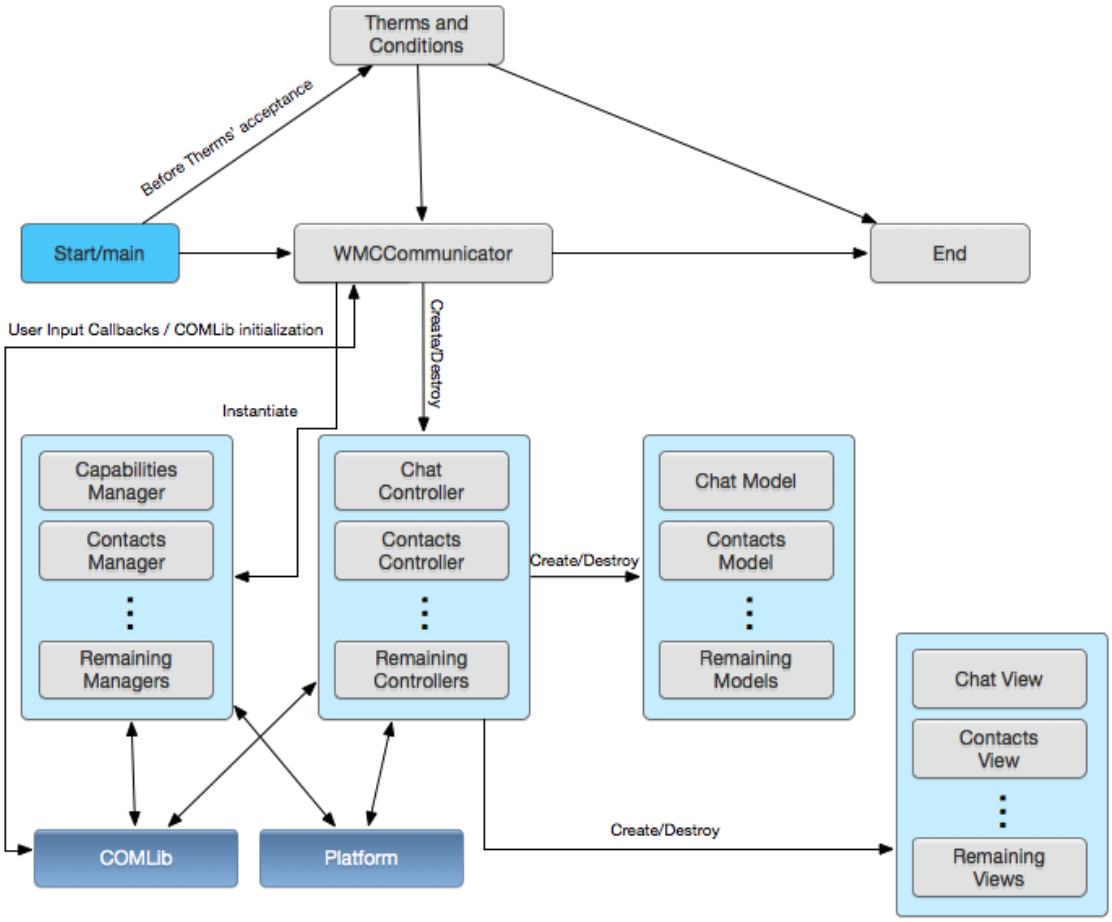

Figure 20- Global application description

When the application is launched, if the Terms and Conditions were already accepted, it goes directly to the **Chat List** view (in the figure, it is represented by its business layer, the

[8] **NavigationPane: "***The **NativationPane** keeps track of a stack of **Page** objects that can be pushed and popped on the stack. Only the topmost page on the stack is displayed to the user.*" [65]

WMCCommunication manager), else the Terms and Conditions page is displayed, and the application remains stopped until the acceptance of these terms. After that, it is redirected for the **Chat List** view. As previously mentioned, the **WMCCommunicator** is the manager responsible to initialize the necessary services. In this case all the project's managers and the COMLib are initialized. After initialized, the COMLib can require the user interaction any time through the User Input callback (sent to the communication stack upon its initialization). For example, the registration process is requested by the COMLib, through the request MSISDN procedure (this will be detailed later). The **WMCCommunicator** also launches the contacts' background synchronization, process that will be detailed later in this chapter. Beyond that, as previously mentioned, this class is the responsible for the NavigationPane stack management. So it is the entity that create/destroy all the Controllers. The models and the views are created inside the controllers, and they are destroyed at the time of "his father's" dead. Lastly, the managers and the controllers are the only entities that are able to communicate with COMLib.

In the next subtopics the main features' architecture will be described. Please note that every **View** has both a **Controller** and a **Module** associated. In some figures only one of the components is visible (to simplify the analysis). However, the reader must assume that the remaining entities are included in the visible one.

4.1.1. Registration

The registration process represents the first feature implemented by the trainee. It is requested by COMLib when it's needed. The following image illustrates the registration procedure.

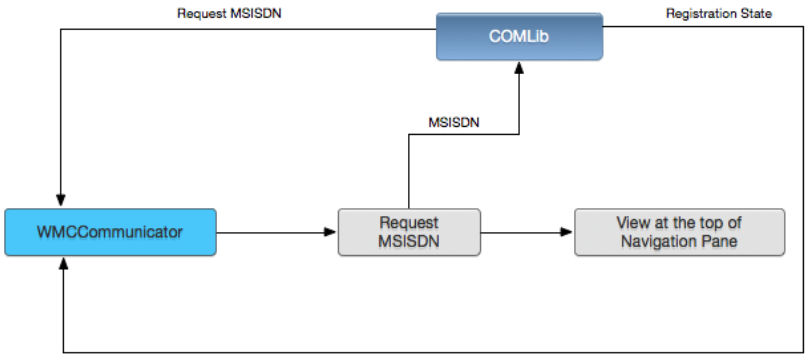

Figure 21 - Registration process

First, COMLib sends a User Input request named "Request MSISDN", where the user must insert its phone number to continue. After that, the phone number is sent to COMLib and the application continues its course (the user goes back to page that was visible when the request arrived). After receive the MSISDN from the user, COMLib handles the remaining registration process. The result of the registration is received in the WMCCommunicator through an event triggered by COMLib

4.4.2. Contacts

Apparently, the access to the contacts would be a quite simple and easy to implement task. And it was! However, the straightforward implementation proved to be a very bad solution, because the process to get the contact details of a contact showed to be quite lengthy. This depended on number of contacts existing in the native address book. For example, for a phone with 500 contacts in the address book, the application took about five minutes to process all contacts. This would make the application almost impossible to use, because every time the user wished to send a message, it would take five minutes to open the contact picker. To fix this problem, the trainee designed a solution to keep some of the contacts information in cache. Just a list of partial contacts[9] is returned each time the user opens the contact picker and the remaining fields are populated from the cache. Querying the phone's address book for 500 partial contacts took about two seconds (which is a really big difference from the last recorded time). A background task was used to fetch the contact details and populate the cache. To store the contact's info, a sqlite3 database was created with two tables: **Contact** and **PhoneNumber**.

The entire procedure is illustrated in the following figure.

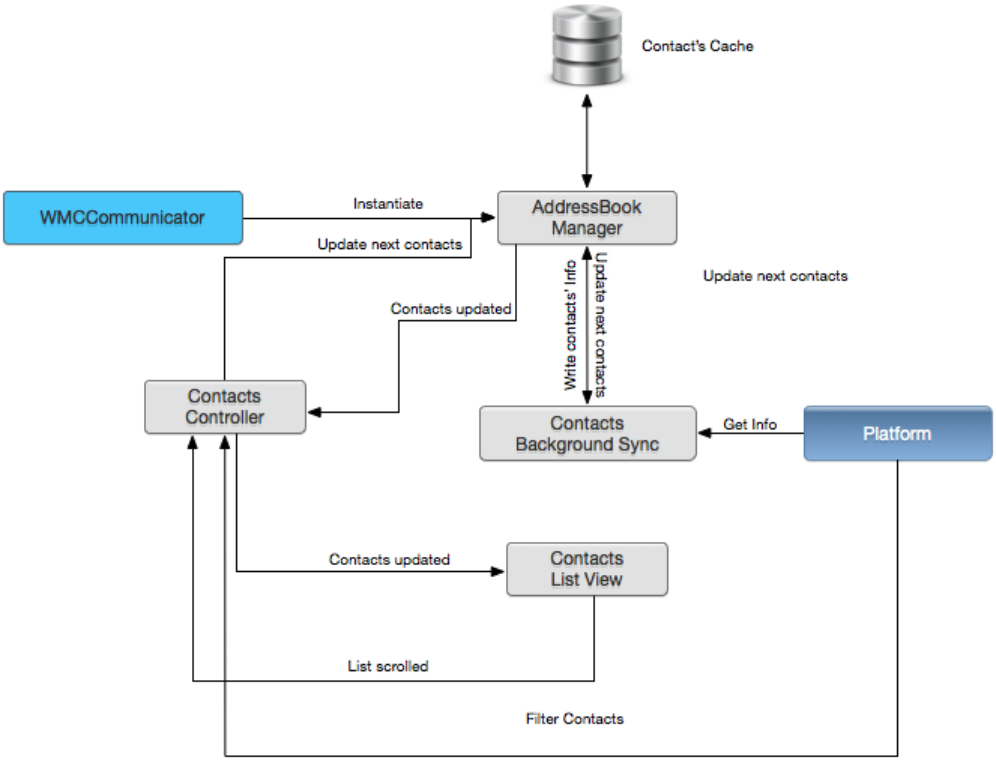

Figure 22 - Contacts solution architecture

When the AddressBookManager is instantiated, a connection to the database is open if the database exists. If not, first the database is created and then the connection is established. After that, the first step consists on loading the entire cache to memory (saved in a Map object, where

[9] **Partial contacts –** when querying the phone's address book for partial contacts, only the contact id and display name is returned

the key is the contact id and the value is an object containing the contact details). After this, the AddressBookManager starts the contacts background sync thread, and synchronizes the contacts from native address book to the application cache (if already exists). If the cache is empty, it will be populated by this thread.

To make this solution user friendly, when the user opens the contact picker and scroll through the list, every time the scroll stops, a signal is sent to the contacts background synch. When this signal arrives to the thread, it pauses its task and updates the contact details of currently visible contacts on the list view. After that it resumes the previous task. If the previous task was already a scrolling event, then it is interrupted. All the communication between different threads is done using signals and slots and the data consistency is assured by synchronization using mutexes.

It is important to mention that the cache is only used to populate the missing fields of a partial contact. The contacts list is loaded directly from the native AB (partial contacts) and the filtering mechanism available at the contact picker view is executed directly on the native address book (because is an already optimized function available at BB10 SDK), so all the data is consistent anytime.

4.4.3. Chat/SMS

It is possible to group the Chat and SMS features because in the application they work in the same way (only the used technology changes).

In the next paragraphs, the designed architecture for this feature will be described.

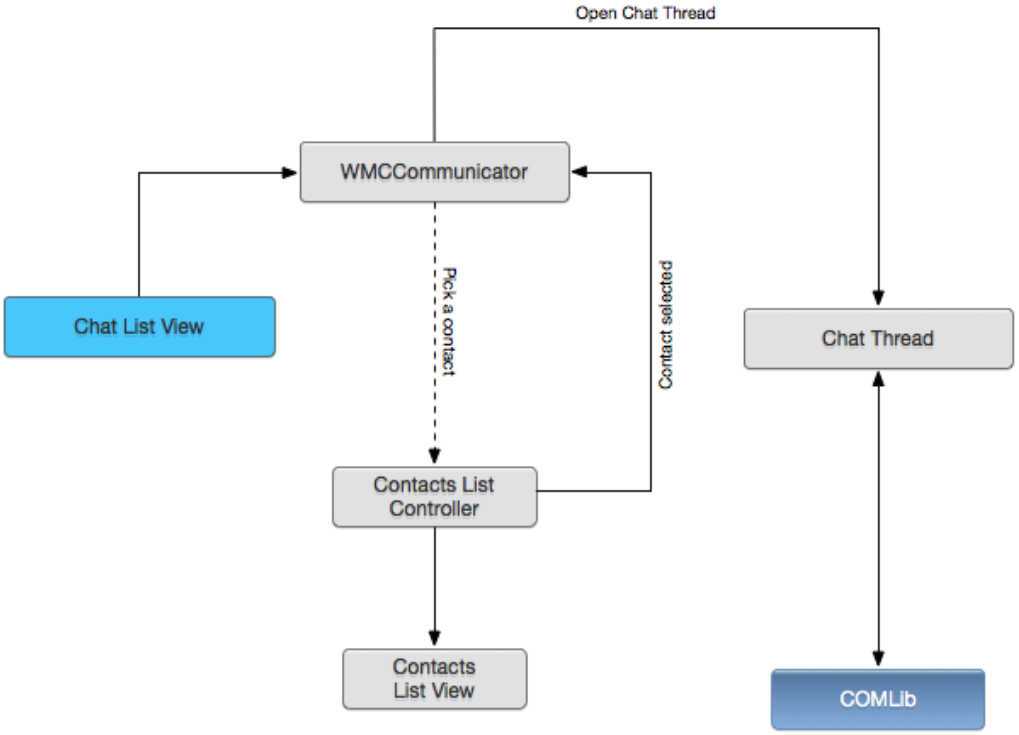

Figure 23 - Chat/SMS feature architecture

There are two ways to enter on the Chat Thread: 1) select an existent conversation from the chat list and 2) start a new conversation. If the second option is the one selected, the application will open the contact picker so the user may select a contact. After, the procedure is the same as if the user selected an existing chat thread.

When a chat thread is opened, the first thing to do is load the history of that conversation (which is done through a COMLib request) and subscribe all COMLib events related with the conversation's contact (for example, "*message added*" event, "*is typing*" event, etc.). These events are responsible to keep the conversation updated. After that, the application asks COMLib for the capabilities of the contact associated with the conversation, to know if it is able to perform RCS communications or if only the traditional mechanisms are available. This request's answer may take a while to arrive. So, to allow the user to keep using the app, this answer comes in a callback. Before the arrival of the capabilities, the user is only allowed to send and receive SMS and MMS. After that, if the contact is not RCS capable, the allowed options remains the same. However, if is RCS capable, the user may choose between send content over RCS or over CS. When arriving messages are displayed on the screen for the first time, the application sends that information to COMLib, so the message can be marked as read on the sender's application.

It is important to understand that all the events related with the contact are subscribed during the time the chat thread is open. So, if the contact's capabilities change during that time, the user is notified.

4.4.4. File Transfer/MMS

The File Transfer/MMS feature uses the same structure of Chat/SMS one – they work together on the same view, only some options change. The following image illustrates the two options added to this feature, and their connection to the feature described above.

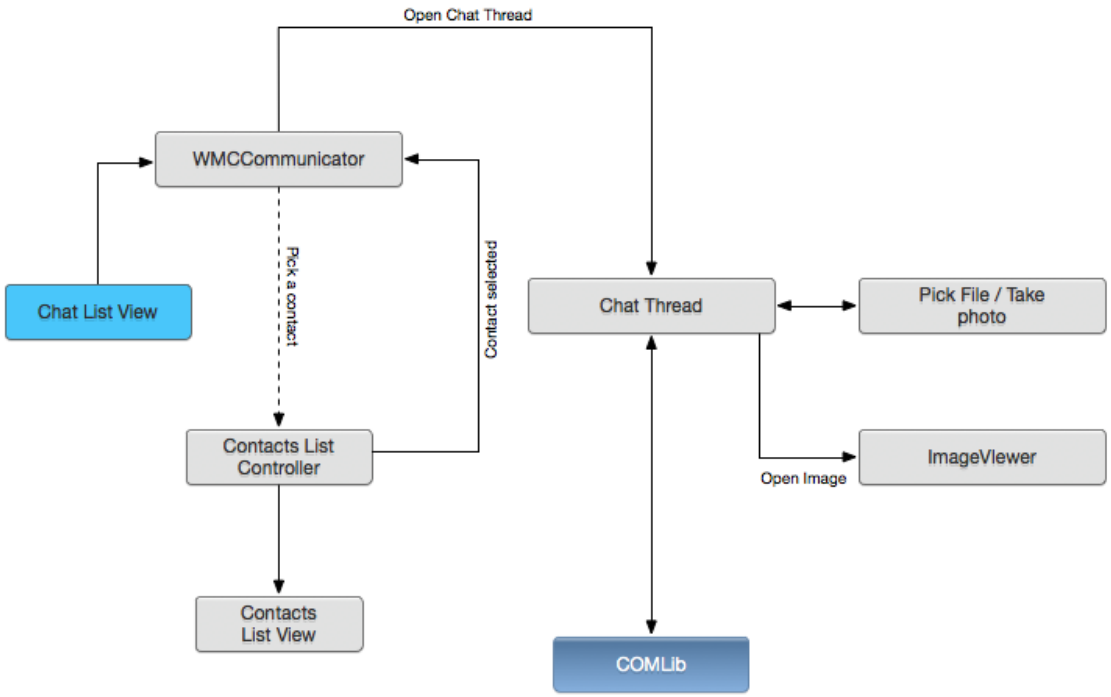

Figure 24 - File Transfer/MMS feature architecture

When the user wants to send an image file transfer, the photo picker is opened and the user can choose between select an existing picture or take a new one. When the image is selected, the file transfer starts. The MMS feature works in the same way. Another available option is open the image (sent or received). This will open the image in a new navigation pane page, adjusted to the entire screen. When a file transfer is incoming, the procedure will depend on the application settings. If the auto-download is enabled, then the download starts automatically. However, if it is disabled, the user must accept the file transfer first to begin the download. During the download (received files) and upload (sent files) process, a progress bar is displayed allowing the user to monitor the download/upload progress.

4.4.5. Application Settings

This section describes how the settings architecture is organized. Only a set of settings selected by the trainee's supervisor were implemented on the UI.

The following figure describes how the settings work.

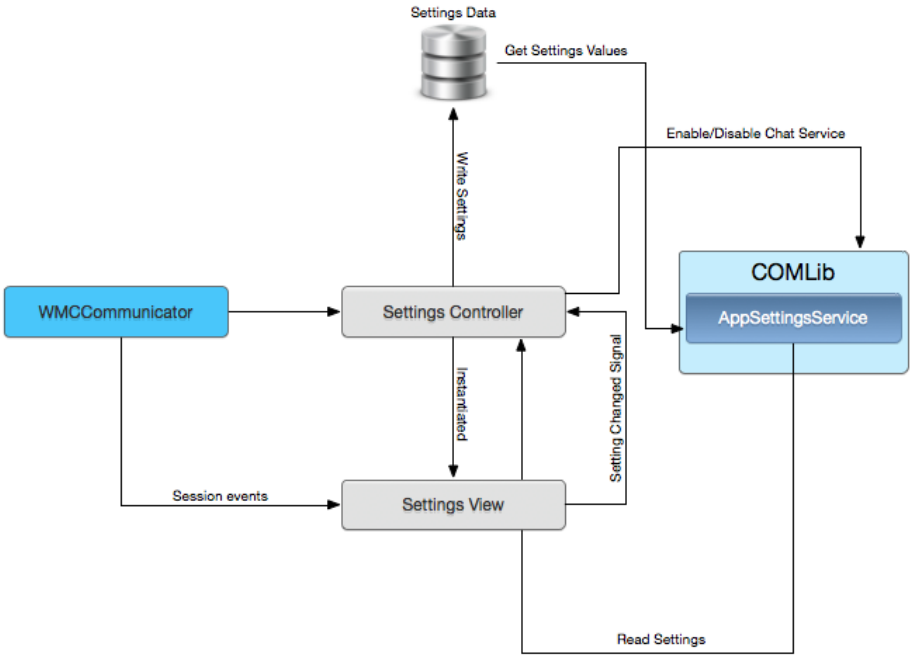

Figure 25 - Settings Architecture

The access to the settings page is available from the Chat List view. The available settings on the UI are:

- Enable/Disable Chat service;
- Download media when using mobile data;
- Download media when connected to Wi-Fi;
- Download media when roaming.

These settings values are read from COMLib's AppSettings service by the UI. When the user changes these values, they are stored in the application settings file. Beyond the settings, in this page is possible to see the current session state. To keep this information updated, the SettingsController is connected to WMCCommunicator manager through a signal/slot mechanism, and when this manager receives a session event from COMLib, it sends a signal to the SettingsController (this just occurs if the Settings page is the current view).

5. Software Development Methodology

This chapter's aim is to describe the software development methodology used during the internship. In it, will be included the planning of the internship, the risk analysis and the discrepancy from the initial plan.

5.1. Scrum Methodology

Over the time, WIT-Software has been adopting iterative development processes based on the *Agile* methodologies, instead of the *Waterfall* methodologies (traditional model where a complete and exhaustive plan is built in the beginning of the project for the entire project's execution time). The adoption of these *Agile* methodologies aims to facilitate the project planning, as it prioritizes people and interactions over processes and tools and working code over comprehensive documentation. The planning is done in phases, not all at once, and it can be changed easily. For all these reasons, Scrum (iterative and incremental *Agile* software development methodology) was the methodology chosen for this project development.

The key components of the Scrum methodology are:

- **Sprint –** It is a period of time (ranging from 2 to 4 weeks, typically) where the team is committed to implement a set of features defined before the sprint's start. In the end of the sprint, all the features should be implemented. If the sprint's result is the expected and is 100% functional, it should be added to final result as a project increment;
- **Product Backlog –** Represents the set of requirements to achieve at the end of the project. These requirements can be organized by priority. The Product Owner of the project defines this component;
- **User Story –** Entry of the product backlog.
- **Sprint Backlog –** Set of entries from the Product Backlog that should be implemented in a particular sprint;
- **Sprint Planning Meetings –** These meetings are used to analyze the result of the last sprint. After this analysis, it is defined the next Sprint Backlog;
- **Daily Meetings –** It is a 5/10 minutes meeting, where the daily results/problems are analyzed and discussed. In the end of the meeting, the Sprint Backlog is revised.

In the Scrum methodology there are three different roles:

- **Scrum Team –** the Scrum Team is represent by the developers of the project. They are responsible for the estimation of each user story's size and they can also make their own design and implementation decisions.
- **Scrum Master –** the Scrum Master is responsible for the Scrum Team. He ensures that the team has all the needed resources for the project implementation. The Scrum Master is also responsible to track the project's evolution and to establish the connection between the Product Owner and the Scrum Team.

- **Product Owner –** The product owner is the main responsible for the project and for all the communication with the client. He also prioritizes the requirements of the product backlog and makes sure that at the end of each sprint the goals were met.

In the next figure, it is possible to observe the Scrum Workflow, to understand better how it works.

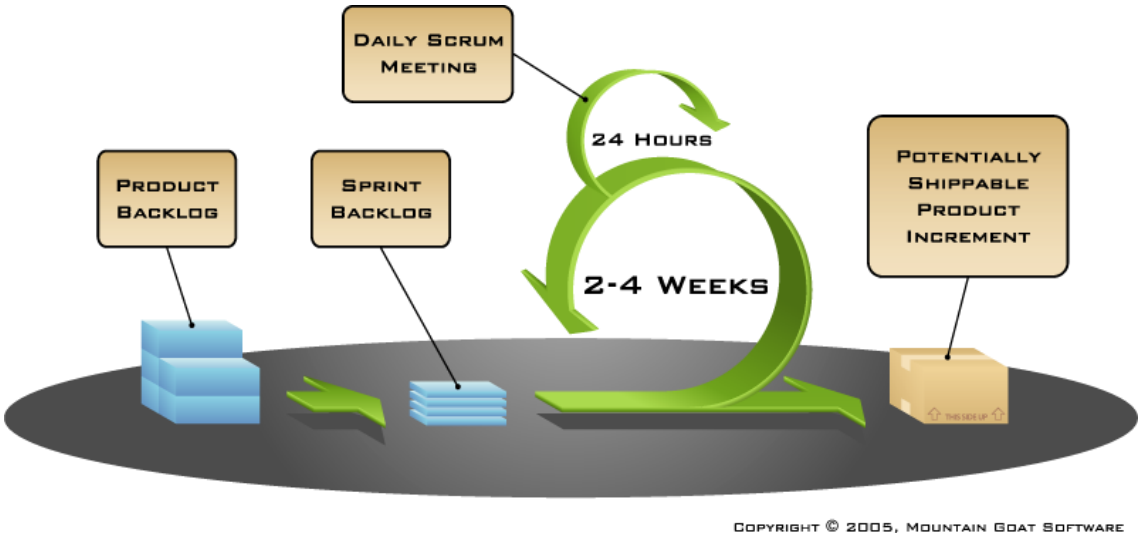

Figure 26 – Scrum workflow

As mentioned above, this was the Software Development Methodology chosen for the project. However, thanks to Scrum's flexibility, it was possible to adapt the methodology to the project needs. For this project, the Scrum's methodology used is characterized this way (the hidden fields were used as defined previously):

- **Sprint –** Each sprint had the duration of two weeks (10 working days). Each sprint started on a Wednesday and ended on a Tuesday.
- **Daily Meetings –** This was not a mandatory requirement of the project. Every time the trainee needed (once a week, once a day, five times per day…) this meeting happened (even if only for five minutes) with his mentor.

For the internship's project, the Scrum roles were defined this way:

- **Scrum Team:** Filipe Figueiredo (trainee)
- **Scrum Master:** Filipe Figueiredo (supervisor)
- **Product Owner:** Rui Gil.

The next subchapter describes the planning and how the work was organized and conducted during the internship course. It is important to highlight that all the planned features for the internship were implemented successfully and before their deadline – the internship was supposed to end at 27th of June, however the trainee finished the development of the project earlier, at 10th of June. From this moment on, given the internship's success, the trainee started to work in a company's real project.

5.2. SVN and Redmine

During the entire project's development, these two tools were used to help the trainee to manage the project and save its development increments.

Redmine is a web base project management application, which contained the entire project tasks and features to be implemented (it was the right hand of the Scrum development methodology used in the project). Each task had an assigned issue id, which was used by the SVN.

Subversion (SVN) is an open source version control system and it was used to keep the project's source code and its respective versions. SVN was connected to Redmine, using each task issue id to identify the respective commits. For the development of the project, a separately branch was created from the project's main repository.

5.3. Planning

In methodologies like Scrum, the use of *Gant* charts for the planning is not practical and easy to use, because the planning is not static and the tasks to a specific Sprint are pulled directly from the Product Backlog (according to the state of the project at the beginning of each sprint). During the internship's course, the sprints were planned with a month in advance (at the end of each sprint, the next two were planned). So, to keep the *Gant* chart updated, a high effort would be necessary. The project planning did not relied on a this type of charts, however, a *Gantt* chart was create to be included in this document, as a representation of the work done and its associated duration. This can be visualized in the next images.

First Semester Planning

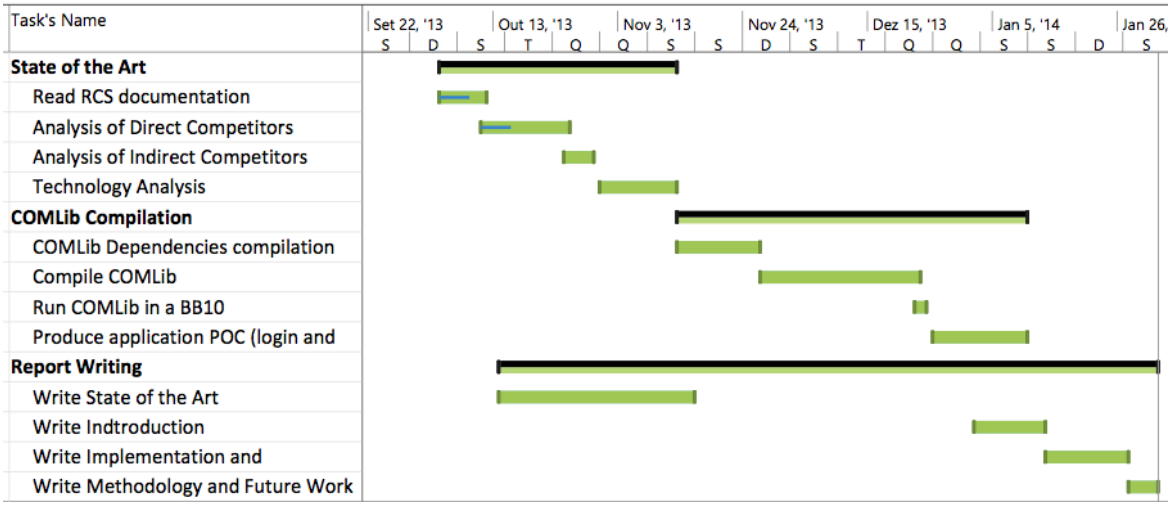

Figure 27 - First semester planning

During the first semester of the internship, the trainee started by reading about the RCS specifications to get familiarized with the project's context. After that, an analysis of the State of the Art was conducted, including both a competitors' and technologies' analysis (subject already discussed at **Chapter 2**). The next task consisted on the compilation of COMLib and its dependencies. This took the rest of implementation time of the first semester, resulting in the compiled COMLib running on a BB10 application's POC.

The report writing was conducted during the semester depending on the information ready to document.

Second Semester Planning

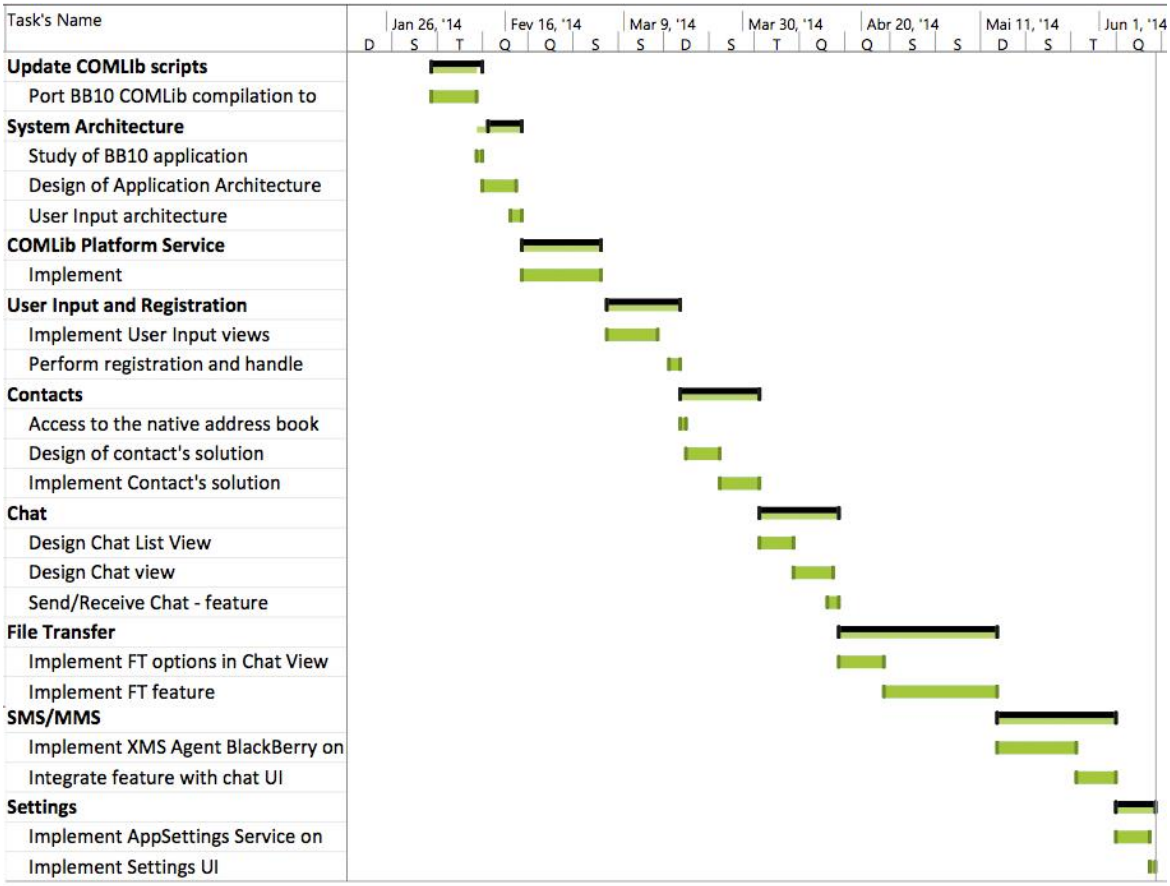

Figure 28 - Second Semester Planning

After the internship's intermediate presentation, the trainee started by updating the scripts used to compile COMLib, which changed for the other platforms during the month of January. At the end of this task, the system architecture for the *Message+* application was designed. The next steps consisted on the implementation of COMLib's Platform Settings Service and User Input, to enable the possibility of implementing the registration feature (which was implemented right after the completion of these tasks). The access to the device's address book followed the registration phase. Initially, this would be straightforward to implement. However, as described at the **System Architecture** chapter, the performance of the unimproved solution was really poor. To fight this problem, it was necessary to design a new solution for the contacts. This feature took more time than what was planned. To make up for this, the Chat and FT features

took less time than what was planned, so the project's implementation time was not affected. These features implied a long UI development time, although, the connection between the application and COMLib was quickly implemented – it consisted in the usage of an API. The implementation of the SMS/MMS feature implied the implementation of the XMS Agent on COMLib. After that, it was only necessary to integrate the SMS and MMS feature with the existing UI. At last, the AppSettings Service was implemented on COMLib. To enable the user to change some of these settings, the settings' view was implemented on the application too.

The trainee did not write the report during the second semester. He was focused on the application's planning and implementation. Therefore, the report was written during the month of August, after the end of the internship.

It is important to justify why the "Functional and Non-Functional tests" category was not included in the Product Backlog. When a project uses an *Agile* methodology, the tests are made through the project, contrary to what happens in the *Waterfall* methodology where the tests are done only at the end of the project. So, it was previously assumed that at the end of each Sprint the respective functional and non-functional tests would be created and performed.

To complement the description of the planning, the next table represents the sprints of the internship and a brief description of each one.

Sprint	Tasks
Sprint #0 29-10-2013 –> 12-11-2013	In this sprint the trainee conducted the analysis of the state of the art and read documentation about RCS specification
Sprint #1 13-11-2013 –> 26-11-2013	The compilation of COMLib dependencies took place at this sprint.
Sprint #2 27-11-2013 –> 10-12-2013	This was the first sprint used for the compilation of COMLib. However, this task was not completed on this sprint.
Sprint #3 11-12-2013 –> 24-12-2013	At this sprint, the trainee finished the compilation of COMLib and started the implementation of POC to test it.
Sprint #4 25-12-2013 –> 07-01-2014	The trainee finished the POC's implementation. After that, he tested the initialization of COMLib.
Sprint #5 08-01-2014 –> 21-01-2014	During this sprint the trainee improved the implemented POC so that it could be demonstrated in the intermediate internship's presentation.
Sprint #6 22-01-2014 –> 04-02-2014	This sprint was used to produce the internship's intermediate documentation.
Sprint #7 05-02-2014 –> 18-02-2014	The COMLib compilation scripts were updated to the last version at the beginning of this Sprint. After that, the trainee designed the architecture for the BB10 application.
Sprint #8 19-02-2014 –> 04-03-2014	During this time, the trainee implemented the PlatformService on COMLib.

Sprint #9 05-03-2014 –> 18-03-2014	At this sprint, the trainee implemented the User Input callbacks used by COMLib. After that, the Registration feature was implemented.
Sprint #10 19-03-2014 –> 01-04-2014	This sprint was used to implement the Contact's solution.
Sprint #11 02-04-2014 –> 15-04-2014	The chat feature was implemented during this sprint.
Sprint #12 16-04-2013 –> 29-04-2013	The implementation of the FT feature started at this sprint. However, this sprint was not enough to finish its implementation.
Sprint #13 30-04-2013 –> 13-05-2013	At this sprint, the trainee finished the implementation of the FT feature.
Sprint #14 14-05-2013 –> 27-05-2013	This sprint started with the implementation of the XMS Agent for the BlackBerry 10 platform. After that, the SMS/MMS feature was integrated in the application.
Sprint #15 28-10-2013 –> 10-06-2013	The last sprint was used to implement the COMLib AppSettings Service for BB10. After that, the settings' view was implemented on the application.

Table 3 - Internship's Planning

The entire Product Backlog (extracted directly from Redmine) can be visualized in **Appendix B – Product Backlog.** For detailed information on each user story please refer to it.

5.4. Risk Analysis

Every software development project is vulnerable to risks. This internship does not represent an exception of that factor. Some risks can be prevented and others can be minimized if they are found at the right time. Therefore, it is necessary to execute a risk analysis to evaluate where the project may fail and try to prevent that. This analysis needs to be made regularly, during the entire project's course. The risks identified for the project are described in the next sections.

Learning curve

Description:

When a software project uses an unknown technology for the developer, there is always a time of learning and adaptation. In the particular case of this project, the trainee was not familiarized with most of the used technologies, such as the C++ programming language (including obviously the respective QT framework) and the QML design language. Beyond that, the application needed to use a dependency called COMLib, which already existed and needed to be adapted to the BB10 platform.

Mitigation:

To mitigate this risk, the trainee read some documentation about BB10 implementation principals and studied the way to bring the existing dependency to the BB10 application.

Project's Dimension

Description:

The initial plan predicted that some of the already implemented modules in the Android application could be used in the BlackBerry 10 project directly. After some research, the trainee discovered it would not be possible. The only thing that could be re-used was the already implemented SIP Stack Library. For this reason, the size of the project increased.

Mitigation:

This risk was assumed by the company at the beginning of the internship, and in fact it became a reality. To mitigate/solve this risk, the company adjusted the internship according to the new conditions (some of the features were removed from the internship's scope).

Device Incompatibility

Description:

Some of the features supported by the iOS and Android applications may not be supported by the BlackBerry 10's platform. For example, the iOS application does not have the SMS and MMS feature, because the API to these services is private.

Mitigation:

To mitigate this risk it was necessary to perform the requirements elicitation and study if they were possible to implement on BB10.

COMLib SDK incompatibility

Description:

As previously mentioned, the COMLib SDK represents a set of external libraries that COMLib needs to work. These dependencies had to be ported so they could run on the BB10 platform. If some irreplaceable dependency was incompatible with the BB10 platform, it would not be possible to use COMLib (at least before some major modifications).

Mitigation:

There was no good mitigation strategy to this risk. Fortunately it did not materialize.

COMLib deprecation

Description:

The RCS stack is always evolving. Every day new features appear, bugs are fixed, some API's change. It is important that the BB10 application uses a recent version of COMLib.

Mitigation:

To mitigate this risk, the trainee regularly updated the COMLib and integrated the changes in the BB10 application

BlackBerry 10 number of clients decrease

Description:

Message+ is already available for Android and iOS. The main goal of the internship was to extend this application do BB10 platform, to be sold later. If the number of BB10 users decrease, it will be difficult to sell the application.

Description:

There was no good mitigation strategy to this risk. During the course of the internship this risk materialized, so the company decided to stop the development of the project at the end of the internship.

5.5. Discrepancy from the Initial Plan

Previously in this chapter, it was mentioned the fact that nothing from the Android application (except the communications library) could be used in the BlackBerry 10's application. That is the reason why the project gained a new dimension. The project implementation started with the compilation of the communications library for the BlackBerry 10's platform and this requirement was not specified in the first internship's proposal. At the end of the first semester, the internship's scope changed too. Firstly, the output of the internship would be a functional demo of the *joyn* application running on BB10. However, the final output is a *Message+* demo running on BB10. When this modification occurred, the application development had not yet begun, so this change did not affect the project's implementation process.

Fortunately, none of the changes or the challenges of the project led to its failure. In the next chapter these main challenges and their solution will be described.

6. Implementation and Software Quality

This chapter has two main objectives: 1) the challenges to overcome during the project's implementation and the way the trainee solved them will be presented. 2) To complete this chapter, a description of the used evaluation mechanisms in this project will be present as well.

During the description of the implementation process of each feature, the results obtained will be presented through some screenshots of the application.

6.1. Implementation: Features, Challenges and Problem Solving

Almost every software project has its challenges. Sometimes they only appear during the implementation stage and they need to be solved as quickly as possible. Thanks to the risk analysis and the specification of the project requirements, some problems were identified right at the beginning of the internship, which allowed the trainee to find and think about solutions to overcome them. However, some other problems were identified only during the implementation process. At this subtopic, both kinds of challenges will be presented.

6.1.1. COMLib compilation

As previously explained, WIT-Software had already developed a communication library for WCS, so the first thing that needed to be done was compile its code with the BlackBerry 10 compiler. Since the communication library has some external dependencies (libraries like *OpenSSL*, *cURL*, etc.), previously to the code compilation of COMLib, it was necessary to compile all its dependencies with the BlackBerry 10 compiler. This process consisted on the compilation of each open source library's code with the BlackBerry 10 compiler to generate new libraries that could run on BlackBerry 10 platform. This proved to be a challenging task, because these libraries are composed by a lot of low-level code and, while some of them compiled at the first attempt without any kind of problems, the others (most of them) had incompatibilities and problems to be solved. To worsen things up, there was almost no documentation about the problems and the incompatibilities found during the process. The trainee discovered that a lot of these libraries probably never had been compiled for the BlackBerry 10 platform before.

To help with this process, a *shell script* was created to compile all the libraries at once. For each dependency, the script extracts its source code to a temporary folder (typically the source code is compressed in a *tar.gz* file), then it applies the necessary patches so the library is able to compile for BB10 (all these patches were create by the trainee, fixing code and configurations directly on the library's source code), then it calls the configure method available in the source code folder (this method is responsible to check if the path to the compiler is correct, to set the

essential macros for the compilation, etc.) and to finish the process it calls *make* and *make install* commands, to effectively compile source code. The result of the compilation (the .a file(s)) and the header files of the dependency are moved to an installation folder, where the COMLib's compilation process will access later.

Some of the libraries were not compiled, mainly because they were not necessary to the COMLib modules used by the application. Some others were compiled but not used (initially they were necessary for *joyn* application, however in *Message+* they are not used).

The next table includes all the dependencies, their compilation state and if they were used or not.

Library Name	Compilation Status	In Use
libOgg	✓	✗
libVorbis	✓	✗
opencore-amr	✓	✗
vo-amrwbenc	✓	✗
Silk	✓	✗
Boost	✓	✓
OpenSSL	✓	✓
cURL	✓	✓
LibXML2	✓	✓
SQLCipher	✓	✓
LiteSQL	✓	✗
Log4cpp	✓	✓
C-ARes	✓	✓
AVC_H264	✗	✗
x264	✓	✗
ffmpeg	✓	✗
libunwind	✗	✗
libetpan	✓	✗
PJSIP	✓	✓

Table 4 - List of Communication Library's dependencies

After the compilation of all the dependencies, it was then possible to compile the COMLib to run on BlackBerry 10. This process was similar to the early one: build a script that compiles the entire communication library's source code and generates the file **libcomlib.so** (the dynamic library ready to be linked to the BlackBerry project). There was already a script built to compile the library to Android and to iOS so, the first step was to edit this file to be able to compile the

library for BlackBerry 10 too. This script is written in the python language, and it generates a gyp[10] file with all the configuration and compilation details necessary to build the project. After the script generation, the compilation process is done making use of a tool called *Ninja*[11]. To build the project it is necessary to run *ninja* with the path for the location of the gyp file as argument. Then it starts to compile the code and, in the end, it generates the library file. Some code needed to be changed so it could run on BB10. This represented a problem, because contrary to the dependencies, a patch would not solve the problem (after is running on BB10, COMLib needs to keep working on the other platforms too). Some of the changes that needed to be done consisted on the addition of a header file, or a namespace prefix (for example, replace *string* for *std::string*). However, some other changes were not so straightforward. For example, BB10 does not have support yet for C++ 11 functions. So, for every C++ 11 function existing on COMLib, the trainee needed to find an alternative able to produce the same result. To introduce this code changes, the trainee could have used the **#ifdef __QNXNTO__ #endif** macro to protect the rest of the platforms (the code inside this block would only run on BB10 devices). However, to keep the COMLib code clean, the trainee decided (with the approval of his supervisor) that the altered code should work on the already existing platforms. This added even more difficulty to this implementation step.

The entire process already described was a big challenge to the trainee that had no experience with these low-level technologies. However, things got even worse, with the appearance of some incompatibilities.

As mentioned before in this document, BlackBerry 10 is based on the QNX Operating System, which by default, uses the Dinkumware C++ Library instead of the GNU Standard Library. The Standard Library in C++ is where the standard objects and functions are implemented (for example, the *std::string* object implementation is allocated in the C++ Standard Library). QNX documentation says that it is possible to change the default library of the project to use the GNU Standard Library. With this information, and taking into account that all the entire COMLib was originally implemented with the GNU STL and that the BlackBerry NDK came with both compilers (one for GNU STL and another for Dinkumware), the trainee decided that all compilations would be made with GNU STL (using the default GCC compiler) instead of the BlackBerry 10 default C++ library (that uses the QCC compiler, a wrapper for GCC where the Dinkumware libraries replace the GNU STL libraries).

When the project was created, all the steps to use GNU STL instead of Dinkumware were followed correctly, however the IDE continued to use the Dinkumware, despite all the settings point to GNU STL. The trainee discovered this problem when he tried to allocate memory for an object that contained a simple string inside of it. It was possible to allocate and possible to use the object. Nevertheless, whenever the destructor of this object was called, the application stopped with a segmentation fault. After a long time trying to fix this issue by changing

---

[10] **GYP (Generate Your Projects) –** is a build automation tool created by Google to build native project files [53].

[11] **Ninja -** is a small build system with a focus on speed. It differs from other build systems in two major respects: it is designed to have its input files generated by a higher-level build system, and it is designed to run builds as fast as possible. [63]

compilation flags, changing compilers, manually including headers for the GNU STL, etc., the trainee decided to go back to the beginning and compile all the COMLib SDK dependencies and the COMLib itself with the Dinkumware library. A lot of incompatibilities were found again, a lot of code needed to be changed but, at the end, it was good value for time, because the problem was completely solved. This problem took almost four weeks to solve, when the COMLib compilation was supposed to take just two weeks (Sprint #2).

At the end of this step, COMLib was ready to run on BB10, and the application development could finally start. It is important to refer that the COMLib support for BB10 were committed directly to the COMLib *trunk* repository, and not to the internship's repository.

6.1.2. COMLib User Input and Registration

The registration represents the first feature implemented on the *Message+* application. The entire registration process is handled by COMLib. However, some UI components have to be implemented to the correct work of this feature.

Before the implementation of the registration feature, the trainee implemented the entire Platform Service on COMLib. As previously mentioned, this service is responsible to access some platform specific information needed by the communication stack. This service was implemented on COMLib using some BlackBerry 10 specific functions. To accomplish this, it was necessary to add the **#ifdef __QNXNTO__** at the beginning of the file and wrap its entire content, or the other platforms would try to compile it, which would not produce a good result... The implementation of this service was quite straightforward because there was a good documentation about the access mechanisms to the necessary data. This service had to be implemented before the registration process because the registration uses a binary SMS to validation purposes. The reception of this type of SMS and its serialization is done at COMLib's Platform Service.

The registration appears to the user as a User Input request. For this particular case, a Request MSISDN. As previously mentioned, the User Input represents an implementation of a COMLib class at the application level. An instance of this class is sent to COMLib at initialization time, so that when necessary, COMLib has the possibility to ask for the user's interaction. The particular case of the request MSISDN can be visualized in the following figure.

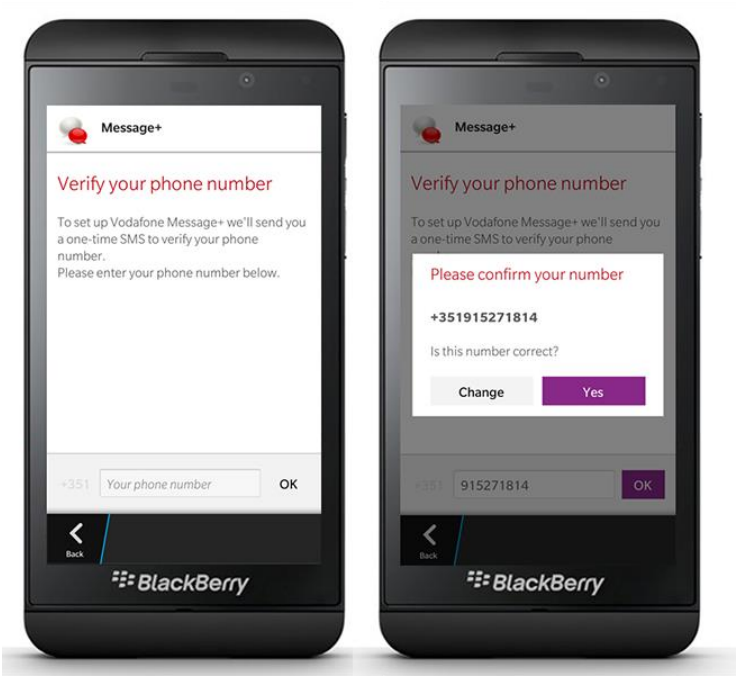

Figure 29 - *Message+* Request MSISDN view

The registration process is automatically handled by COMLib. Whenever is necessary, COMLib sends a request MSISDN to the UI. The user introduces the phone number and COMLib will handle the rest. After that, the application is redirected to the Chat List view.

The implementation of this feature was challenging because it represented the "first impact" with the BB10 technologies. Beyond that, it represented the first impact with the usage of COMLib's API and the familiarization with some technologies used by this library (like the boost shared pointers, used for the COMLib callbacks).

6.1.3. Contacts

The implementation of the contacts feature included a major challenge: improve its performance. The solution to this problem was already documented earlier, at **Chapter 4 – Architecture.**

Another difficulty found by the trainee at this feature's implementation time regards to the UI, particularly to the ListView items. The default items were very limited, and it was not possible to change some of their properties. So, the trainee had to implement a ListView item from scratch, including all the behaviors associated with it (for example, the *onClick* behavior). Even more difficult, still regarding to the ListView, was to find a way to know which were the current visible items (to update them, as described in **Chapter 4**.). In platforms like Android, the ListView object already has methods to get this information. However, in BB10 they do not exist. To solve this problem, the trainee used a function already available which returns the id of the first visible item. After that, the trainee had to calculate the height of each cell, to be able to count the visible ones.

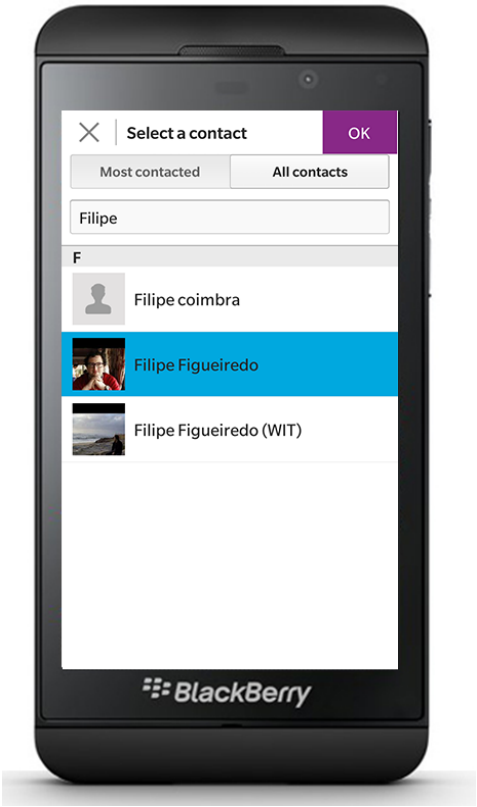

Figure 30 – Contacts page filtered

6.1.4. Chat and File Transfer

These two features represent the main features of the application. Their implementation was more time-consuming than the implementation of any other feature. Once again, the main implementation challenges of these two features were related with the UI layer.

The following figure represents the implemented views. Please note that the Share screen already includes all the options available in the Android application. However, almost all of them are disabled. Only the options “Take photo” and “Picture” are enabled.

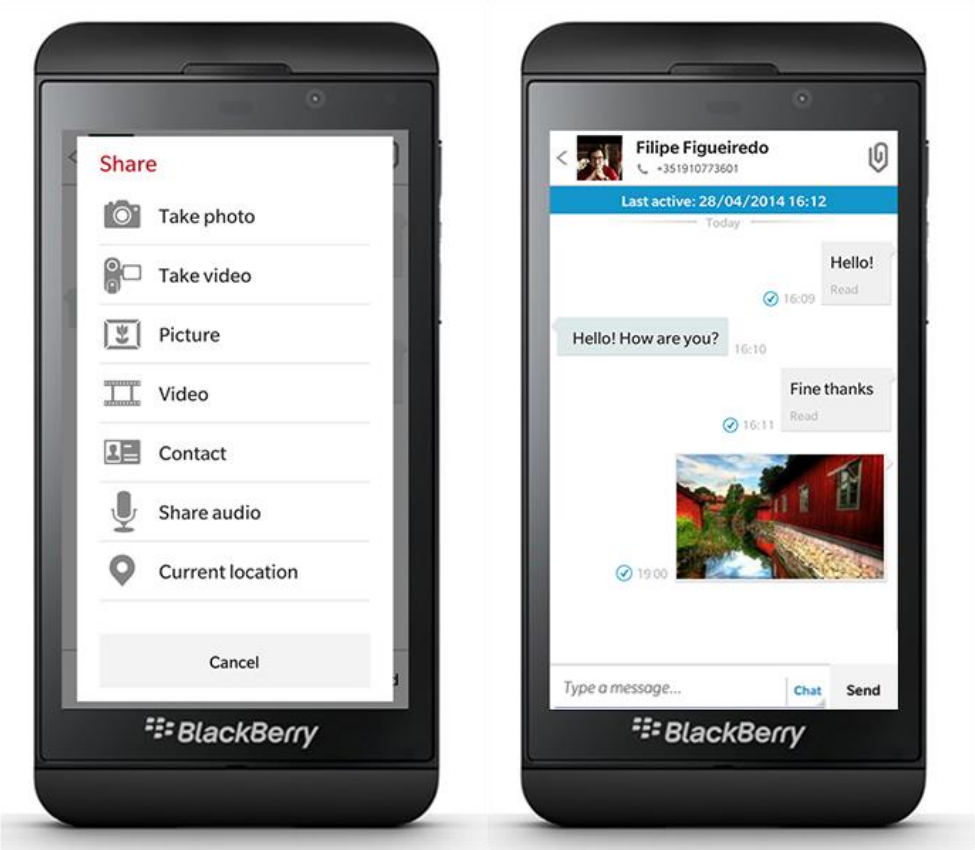

Figure 31 - Chat and File Transfer Feature UI

The UI main challenge of this view consisted on the creation of the chat bubbles. The chat window is composed by a ListView where each bubble represents a ListView item. For this project, four kinds of bubbles were implemented:

- Incoming chat;
- Outgoing chat;
- Incoming FT;
- Outgoing FT.

This means that four different list items needed to be created from scratch, each one representing the contents of each type of bubble. These list items are even more complex than the one implemented in the contacts' feature.

The four list items have one thing in common: the grey bubble (represented at the picture above) where the chat/FT entry is inserted. This bubble's size is not static, it is determined according to the size of the chat/FT entry. The picture used to draw the bubble is not a regular square. It has borders and in one of its sides a little arrow represents the origin of the message. These attributes could not be scaled according to the size of the message content. Only the interior of the bubble could be resized, or the bubble would be deformed. To do that, the trainee needed to use a tool provided by BB10 called 9 slicing [46]. This tool was used to resize the bubble, depending on the cell content size, without resize the borders and the message origin's arrow. After the resize of the bubble, it was still necessary to accommodate everything inside of ListItem cell, depending on the size of it.

Except for the creation of the ListView items, the rest of the UI was easy to implement.

The business logic associated with these features was not difficult to implement. The architecture was already defined, and the entire features were already implemented in COMLib. It was necessary to subscribe the Chat and FT events filtered by the conversation's contact to keep the view updated.

6.1.5. Chat List

The implementation of the Chat List's UI could not be considered a challenge, once the trainee had gained some experience in the implementation of the ChatView. This view is consisted only by a ListView containing a ListView item that was implemented from scratch, containing the necessary properties to hold the information of each Chat List entry. The final outcome of this view can be visualized in the following figure.

Figure 32 - *Message+* chat list

To keep this information updated, it was only necessary to subscribe the COMLib events respecting to chat and file transfer features. When they were triggered, the application would then update the information on this list.

6.1.6. SMS and MMS

This feature represented a challenge in the implementation of the XMS Agent. The Messages API had been introduced in the BB10 SDK shortly before the implementation of this feature. Consequently, a well-organized documentation was not available at the time. Implement the XMS Agent was not a difficult to achieve task, however, when the trainee started to use this feature, he noticed that each time a SMS/MMS was sent through the *Message+* application, a new conversation was created at the device's Messages application. This did not affect the *Message+* application operation; however, it proved not to be user friendly. To fix this situation, the trainee had to study the complete Messages API and implement an algorithm capable of searching for an existing conversation on the phone's Messages application having the same destination of the message that was to be sent. If the conversation existed, the message would be added to it. Else, a new conversation would be created on the device's Messages application. The biggest challenge consisted on the design of this algorithm; however it was not hard to overcome.

These two features use the same UI as the Chat/FT features – only the color of the capabilities message changes (from blue to green). Therefore, the trainee decided that none image illustrating this feature would be displayed here.

6.1.7. Settings

The Settings feature was the last one to be implemented. First, the trainee had to implement the COMLib's AppSettings service. As previously mentioned, this service is responsible to access the application settings and attribute each one of them a default value. The settings values were saved using a QT class named QSettings [47]. This mechanism saves at the application folder a <*key, value*> pair, where the key represents the setting's name, and the value, as the name implies, the setting's value. The settings values are changed from the *Message+* application. COMLib just have read access to them.

The settings page also includes the current session state. To be able to keep this information updated, the COMLib session events are subscribed by the Setting's controller, and when the session state changes, the UI is updated too.

Despite the implementation of the entire AppSettings Service in COMLib, only the settings present in the next figure were implemented on the application.

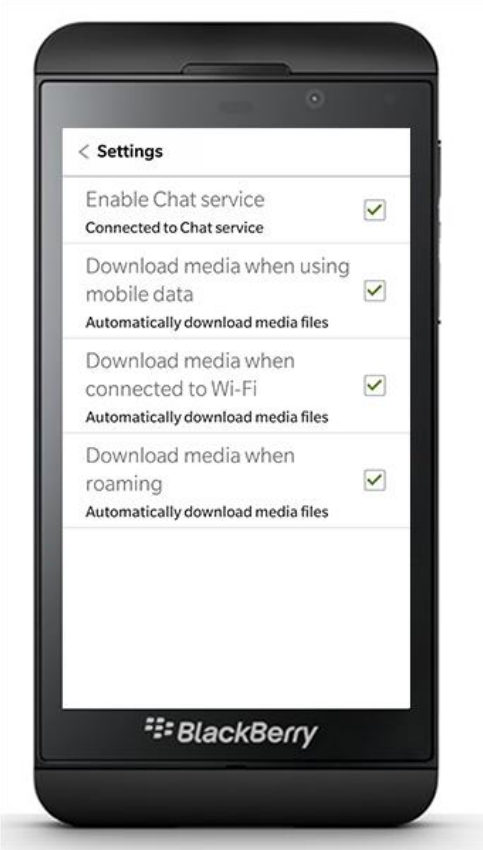

Figure 33 - *Message+* Settings view

This last feature did not presented any implementation problem, so there are not challenges to register.

6.2. Software Quality

This document contains the description of all the tests performed to the final solution. These tests were designed based on the project requirements, and they could be divided in two categories: 1) Functional Tests and 2) Non-Functional Tests. To complement this chapter, the **Appendix C – Software Quality** contains the description of each one of the performed tests.

6.2.1. Functional Tests

This kind of tests represents a very important step of the validation process, giving to the development team the opportunity to fix some application's erratic behaviors, which were not found at implementation time.

These tests are directly related with the Functional Requirements of the application, and they were designed to validate the implementation of each one of them.

The functional requirements related with COMLib compilation were not included in this tests, because they don't represent features of the application, and the usage of these dependencies was not implemented by the trainee (they were already implemented in COMLib).

Regarding to the main features implemented during the internship, a total of 37 functional tests were designed and run. Some of the tests did not pass at the first time, so the trainee had to re-run them until the achievement of a positive result.

The following table demonstrates the output of the first and last runs of the functional tests for all the selected features.

	First Run		Last Run	
Feature	Failed	Passed	Failed	Passed
COMLib User Input	3	3	0	6
Registration	0	2	0	2
Settings	1	4	0	5
Contacts	2	2	0	4
Chat List	1	4	0	5
Conversation Window	6	9	0	15

Table 5 – Functional Tests Results

At the end of the application's development, the trainee observed that all the tests were successfully executed. During the semester some tests failed, and they had to be run again, until they present a positive result. At the end of the internship, the entire set of tests was successfully executed, which means that the implemented features can be considered a success in terms of their operation.

For a detailed description of each functional test, please refer to **Appendix C – Chapter 1**.

6.2.2. Non-Functional Tests

From the non-functional requirements specified, only Application Performance tests and Application UI/UX tests were performed, because the other specified requirements are related with the implementation of the project itself (code indentation, etc.). The results available on this subtopic are concerning to the final tests done to the application.

Application Performance

To measure the application's performance, the trainee used the system tools provided by BB10 OS to get information about the resources allocated to the application in three different application stages:

- Running on background;
- Running on Foreground;
- Getting contacts info (the moment that the application consumes more resources).

The device used to perform these tests was the BlackBerry Z10, which has the next characteristics:

- **CPU:** Dual-core 1.5 GHz Krait
- **RAM:** 2 GB
- **Internal Storage:** 16 GB

The CPU load, Memory RAM and Storage used were measured in this three different stages, producing the next results:

State	CPU (%)	Memory RAM (MB)	Storage (MB)
Background	0.07	18.8	1.44
Foreground	7.34	21.76	1.44
High performance	30.89	40.19	1.44

Table 6 - Performance Analysis

The values on the table represent the average value a set of 10 tests. The detailed results of each running can be found in **Appendix C – Chapter 2.1**.

Usability Tests

To perform this kind of tests, the trainee recruited 31 people to try the main features of the solution implemented.

The test case is described in the following steps:

- For the four main features (Chat, FT, SMS, MMS), a Use Case is selected;
- The phone is given to the user, with the description of the Use Case he must execute;
- All the steps needed by the user to execute the use case are registered;
- At the end, the number of steps utilized by the user will be compared with the minimum steps necessary to fulfil the use case.

All the tests were executed in the same conditions:

- The chat list was cleaned before each test;
- The destination contact was RCS capable.

To provide a representative sample of the population, 31 random users were chosen. It is important to mention that most of them never used a BlackBerry 10 before. The chosen population was divided this way:

- 15 Android users
- 8 iOS users
- 5 BlackBerry users
- 2 Windows Phone users

The next subsections represent the four use cases and the results of the testing phase.

Send Chat Message

To execute this test case, the user had to select a contact, write some text and press the send button.

The minimum steps to reproduce this use case were:

1. Open Contact picker
2. Select the contact
3. Press OK to confirm the selection
4. Write the message
5. Press send

Regarding the results obtained, the next figure shows the number of steps needed by each user.

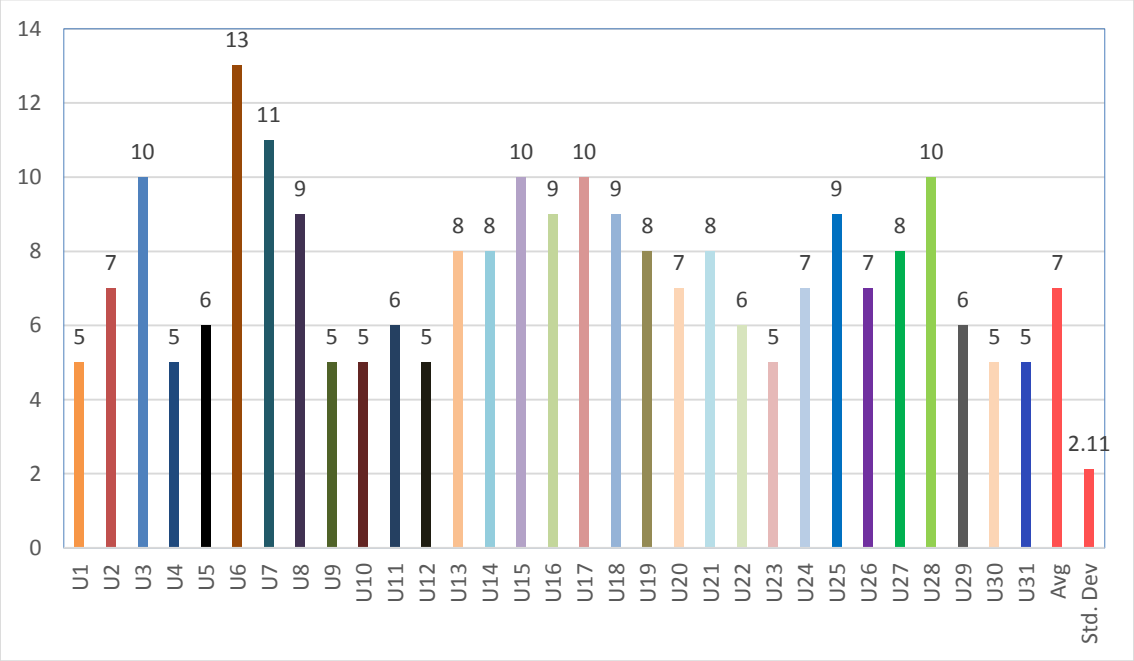

Figure 34 - Results of Send a Chat message (usability test)

The results of this test case were satisfactory, with a great number of users completing the task in the minimum steps required. For this five step use case, the average number of steps was about 7, which represents a good result for the first interaction with the application.

Send FT

To execute this test case, the user had to select a contact and then select an image to send. The image already were in the phone, so it just needed to be selected.

The minimum steps to reproduce this use case were:

1. Open Contact picker
2. Select the contact
3. Press OK to confirm the selection
4. Open the file chooser
5. Select the option “Image”
6. Select the respective image

Regarding the results obtained, the next figure shows the number of steps needed by each user.

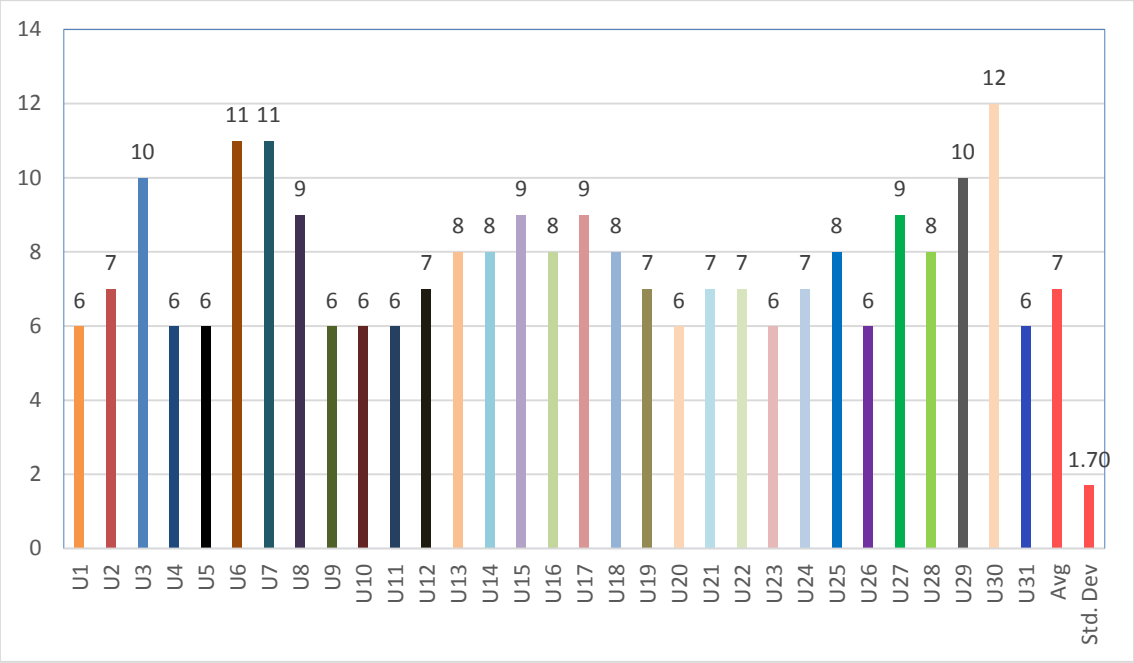

Figure 35 - Results of Send a FT (usability test)

It was possible to see that the users were getting familiarized with the application. For this specific use case, the minimum number of steps needed was six. The average obtained was rounding the seven steps (same as previous use case, however this use case needed one more step than the previous one).

Send SMS

To execute this test case, the user needed to follow the same procedures as to send a chat message. However he needed to switch the technology before sending the message.

The minimum steps to reproduce this use case were:

1. Open Contact picker
2. Select the contact
3. Press OK to confirm the selection
4. Switch technology
5. Write the message
6. Press send

Regarding the results obtained, the next figure shows the number of steps needed by each user.

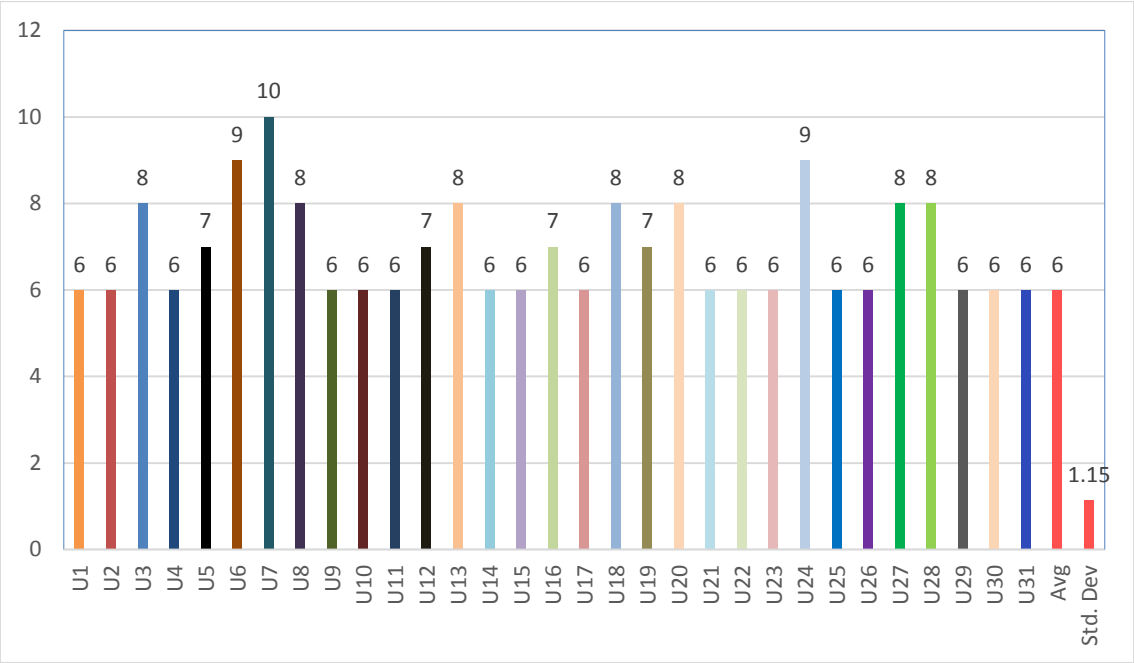

Figure 36 - Results of Send an SMS (usability test)

The results of this test case are satisfactory, with a great number of users completing the task in the minimum required steps. For the 6 step use case, the average number of steps was about 6, which leads almost to the perfection. This result is normal, because it proves that the users are getting familiarized with the app.

Send MMS

To execute this test case, the user had to select a contact, change the technology and then select an image to send. The image already were in the phone, so it just needed to be selected.

The minimum steps to reproduce this use case were:

1. Open Contact picker
2. Select the contact
3. Press OK to confirm the selection
4. Switch the current technology
5. Open the file chooser
6. Select the option “Image”
7. Select the respective image

Regarding the results obtained, the next figure shows the number of steps needed by each user.

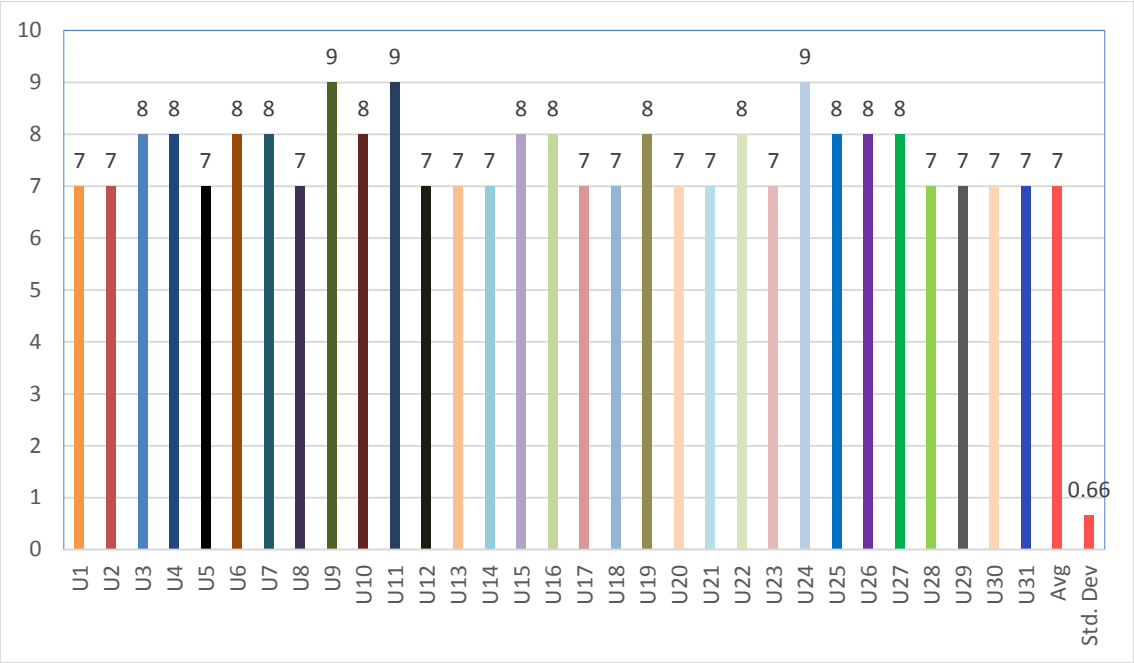

Figure 37 - Results of Send a MMS (usability test)

Once again, the average number of steps needed to perform this action rounded the ideal value (7). At the last test, the standard deviation dropped a lot, which allows the trainee to predict that if the tests continued, this value would tend to 0.

At the end of this analysis, the results were quite satisfactory. They represented a good adaptation of the users with the app (even if they did not had a previous experience with BB10 before). This proves that the UI/UX non-functional requirement was fulfilled successfully, resulting on an easy to use and user-friendly application.

7. Conclusions

At this last chapter, an overview of the work developed during the entire internship is presented, as well as some future work suggestions, if the company wishes to continue with the project implementation. To close this document, the trainee wrote about some final thoughts regarding to the experience lived during this one-year internship.

7.1. Overview

At the beginning of the internship, the initial plan consisted on the porting of the *joyn's* Android application to the BlackBerry 10 platform. This was supposed to be simple to execute, making use of the tools already existing to bring Android applications to BB10's platform. However, a big impediment appeared, and it name was COMLib – the communication stack used by *joyn*. It was not possible to port an Android application that uses native code directly to BB10, so a change to the internship's objectives had to happen: from *joyn* to demo of *Message+'s* main features. This change would imply the entire development of the application, starting by the compilation of COMLib to the BB10 platform – this represented one of the main challenges that the trainee needed to overcome.

After the addition of support for BB10 in COMLib, the implementation of *Message+* for BB10 started. It is important to mention that the entire application was designed and implemented by the trainee.

During the application development, some difficult challenges appeared and the trainee needed to find solutions to overcome them. Fortunately, all the challenges were successfully overcome, and this internship's work can be considered a success.

At the end, the resulting application was fully functional, and all the features planned were implemented successfully. The application was already demonstrated at Vodafone in several countries, and all the clients liked the final outcome.

7.2. Future Work

As previously mentioned, this application is just a demo of the *Message+* main features: Chat, File Transfer, SMS and MMS. If the company intends to continue with this project, the rest of the *Message+* features have to be implemented:

- Exchange IM between more than 2 people;
- Exchange video, audio, vCard or location between 2 or more people;
- Send SMS/MMS broadcast to more than 2 people;

Beyond them, some low-level functionalities existing on the Android application must be implemented too.

At the solution's design time, the trainee prepared everything so the application could be changed from *Message+* to *joyn* easily, just by changing the application configuration files (editing the previously mentioned Strings and Images files) and implementing the remaining missing features. This would need some COMLib dependencies to be compiled as well to support the remaining *joyn's* features (for example, calls and vide calls). However, all the work related with chat, FT, SMS and MMS would already be implemented, which would be a good start.

7.3. Final Remarks

This internship was very important to the trainee, as it represented his first impact with a real project in the labor market. Beyond that, it provided the opportunity to apply all the knowledge acquired during the time the trainee studied at University of Coimbra, as well as to gain new capacities and technical skills. At the beginning of the internship, the trainee did not have any kind of experience with the technologies used during the project development – even the programming language (C++). So he had to learn everything by himself, which turned the internship even more challenging and rewarding. The trainee also understood the importance of all the software engineering process, as it were essential to the project's planning and successful execution.

Above all, the most important was the positive impact that the development of this project had in the trainee's personal and professional life. The fact that it was a success, allowed the trainee to keep working on the company where the internship took place, which marks positively the beginning of a new phase of his life.

References

[1] V. Beal, "webopedia," [Online]. Available: http://www.webopedia.com/TERM/C/circuit_switching.html. [Accessed 21 08 2014].

[2] "GSMA," [Online]. Available: http://www.gsma.com/aboutus/gsm-technology/gsm. [Accessed 21 08 2014].

[3] Techopedia, "Techopedia," [Online]. Available: http://www.techopedia.com/definition/29145/over-the-top-application-ott. [Accessed 05 11 2013].

[4] WIT-Software, S.A, "WIT-Sotftware," [Online]. Available: http://www.wit-software.com/. [Accessed 25 01 2014].

[5] WIT-Software, "WIT-Software, S.A.," [Online]. Available: http://www.wit-software.com/company/company-profile/. [Accessed 25 01 2014].

[6] WIT Software, S.A, "WIT Software, S.A," [Online]. Available: https://www.wit-software.com/company/awards-and-certifications/. [Accessed 23 08 2014].

[7] Apcer, "ISO 9001," [Online]. Available: http://www.apcer.pt/index.php?option=com_content&view=article&id=96:iso-9001&catid=3&Itemid=10. [Accessed 22 01 2014].

[8] Apcer, "ISO 14001," [Online]. Available: http://www.apcer.pt/index.php?option=com_content&view=article&id=117%3Aiso14001&catid=4&Itemid=45. [Accessed 22 01 2014].

[9] Apcer, "NP 4457," [Online]. Available: http://www.apcer.pt/index.php?option=com_content&view=article&id=141%3Anp-4457&catid=10&Itemid=60&lang=pt. [Accessed 22 01 2014].

[10] phone scoop, "Phone Scoop," [Online]. Available: http://www.phonescoop.com/glossary/term.php?gid=131. [Accessed 20 01 2014].

[11] Parks Associates, "Number of smartphone users to quadruple, exceeding 1 billion worldwide by 2014," [Online]. Available: http://www.parksassociates.com/blog/article/pr-march2010-smartphones. [Accessed 24 01 2014].

[12] eMarketer, "Smartphone Users Worldwide Will Total 1.75 Billion in 2014," [Online]. Available: http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536. [Accessed 25 01 2014].

[1 P. Clark-Dickson, "Informa - telecoms & media," [Online]. Available:

3] http://blogs.informatandm.com/12861/news-release-ott-messaging-traffic-will-be-twice-the-volume-of-p2p-sms-traffic-by-end-2013/. [Accessed 24 01 2014].

[1 GSMA, "Rich Communications," [Online]. Available:
4] http://www.gsma.com/futurecommunications/rcs/. [Accessed 03 10 2013].

[1 T. Green, "Mobile Entertainment," [Online]. Available: http://www.mobile-
5] ent.biz/industry/mwc-2012-gsma-unveils-joyn-brand-for-rcs/036849. [Accessed 24 11 2013].

[1 WIT Software, S.A., "WIT-Software," [Online]. Available: https://www.wit-
6] software.com/products/rcs-suite/. [Accessed 20 08 2014].

[1 WhatsApp Inc, "WhatsApp," [Online]. Available: http://www.whatsapp.com/. [Accessed 26
7] 10 2013].

[1 P. Goldstein, "FierceMobileIT," [Online]. Available:
8] http://www.fiercemobileit.com/story/whatsapp-ceo-were-handling-20b-messages-day/2013-04-16. [Accessed 25 01 2014].

[1 "The Verge," [Online]. Available: http://www.theverge.com/2013/10/22/4865328/whatsapp-
9] 350-million-monthly-active-users. [Accessed 26 10 2013].

[2 "BlackBerry," [Online]. Available:
0] http://us.blackberry.com/bbm.html?lid=us:bb:bbm&lpos=us:bb:bbm. [Accessed 10 10 2013].

[2 D. Halliwell, "BBM for iOS and Android to Launch This Summer «Inside BlackBerry - The
1] Official BlackBerry Blog," [Online]. Available: http://blogs.blackberry.com/2013/05/bbm-ios-android/. [Accessed 14 10 2013].

[2 S. Ye, “TechRice,” [Online]. Available: http://techrice.com/2011/09/21/weixin-
2] %E5%BE%AE%E4%BF%A1-tencents-bringing-the-mobile-im-revolution-to-the-mainstream/. [Accessed 19 01 2014].

[2 P. Mozur, “Digits,” [Online]. Available: http://blogs.wsj.com/digits/2013/08/14/number-of-
3] wechat-app-users-triple/. [Accessed 10 10 2013].

[2 “Line,” [Online]. Available: http://line.me/en/. [Accessed 01 11 2013].
4]

[2 “Reuters,” [Online]. Available: http://www.reuters.com/article/2012/08/16/japan-app-line-
5] idUSL2E8JD0PZ20120816. [Accessed 12 11 2013].

[2 N. Lomas, “Tech Crunch,” [Online]. Available: http://techcrunch.com/2013/10/07/line-

6] india/. [Accessed 16 10 2013].

[2 J. H. Kim, “Slideshare,” [Online]. Available:
7] http://www.slideshare.net/junghongkim/insight-about-kakao-games-fin. [Accessed 16 10 2013].

[2 E. Lukman, “TechInAsia,” [Online]. Available: http://www.techinasia.com/messaging-app-
8] war-wechat-line-kakaotalk-whatsapp/. [Accessed 12 10 2013].

[2 V. M. Inc., “Viber,” [Online]. Available: http://www.viber.com/about. [Accessed 14 10
9] 2013].

[3 N. Kawasaki. [Online]. Available:
0] http://helpme.viber.com/Knowledgebase/Article/View/208/35/about-viber-media. [Accessed 15 10 2013].

[3 R. Wauters, “TNW,” [Online]. Available: http://thenextweb.com/insider/2013/05/07/viber-
1] ceo-talmon-marco-interview/#!q9u36. [Accessed 23 11 2013].

[3 S. a. Microsoft, “Skype,” [Online]. Available: http://www.skype.com. [Accessed 10 10
2] 2013].

[3 T. Tänavsuu, “arstechnica,” [Online]. Available:
3] http://arstechnica.com/business/2013/09/skypes-secrets/. [Accessed 07 11 2013].

[3 N. Lomas, “Tech Crunch,” [Online]. Available: http://techcrunch.com/2013/02/26/skype-
4] competitor-viber-hits-175-million-users-up-from-140-million-in-december/. [Accessed 16 10 2013].

[3 Google, “Google +,” [Online]. Available: http://www.google.com/+/learnmore/hangouts/.
5] [Accessed 10 11 2013].

[3 V. Staff, “The Verge,” [Online]. Available:
6] http://www.theverge.com/2013/5/15/4318830/inside-hangouts-googles-big-fix-for-its-messaging-mess. [Accessed 28 10 2013].

[3 A. Barr, “USA Today,” [Online]. Available:
7] http://www.usatoday.com/story/tech/2013/10/29/google-plus/3296017/. [Accessed 23 11 2013].

[3 "192 Million Facebook Mobile Users Are From Android Phone," [Online]. Available:
8] http://www.omeoo.com/industry-news/192-million-facebook-mobile-users-are-from-android-phone/. [Accessed 05 10 2013].

[3 J. Constine, "Facebook Mobile User Counts Revealed: 192M Android, 147M iPhone, 48M iPad, 56M Messenger," [Online]. Available: http://techcrunch.com/2013/01/04/how-many-

9] mobile-users-does-facebook-have/. [Accessed 14 11 2013].

[4 BlackBerry Limited, "Objective C to Cascades," [Online]. Available:
0] http://developer.blackberry.com/native/documentation/cascades/dev/ios_porting/objc_cascades.html. [Accessed 23 01 2014].

[4 BlackBerry Limited, "Introduction to Cascades," [Online]. Available:
1] http://developer.blackberry.com/native/documentation/cascades/getting_started/intro/index.html. [Accessed 12 01 2014].

[4 "ProjectSmart," [Online]. Available: http://www.projectsmart.co.uk/moscow-method.php.
2] [Accessed 23 08 2014].

[4 "Native SDK for BlackBerry 10," [Online]. Available:
3] http://developer.blackberry.com/native/documentation/cascades/. [Accessed 23 08 2014].

[4 WIT Software, S.A, "RCS Suite," [Online]. Available: https://www.wit-
4] software.com/products/rcs-suite/. [Accessed 25 08 2014].

[4 "Native SDK for BlackBerry 10," [Online]. Available:
5] http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native_sdk.devguide/topic/c_appfund_applifecycle.html. [Accessed 24 08 2014].

[4 [Online]. Available:
6] http://developer.blackberry.com/native/reference/cascades/bb__cascades__image.html. [Accessed 29 08 2014].

[4 "BlackBerry Native SDK," [Online]. Available:
7] http://developer.blackberry.com/native/reference/cascades/qsettings.html. [Accessed 29 08 2014].

[4 R. Noldus, U. Olsson, C. Mulligan, I. Fikouras, A. Ryde and M. Stille, "IMS Application
8] Developer's Handbook," Academic Press.

[4 Neusoft Corporation, "About Neusoft," [Online]. Available:
9] http://www.neusoft.com/about/index.html. [Accessed 01 10 2013].

[5 BlackBerry Limited, "BlackBerry 10 samples," [Online]. Available:
0] https://developer.blackberry.com/native/sampleapps/. [Accessed 06 10 2013].

[5 BlackBerry Limited, "BlackBerry 10: First application," [Online]. Available:
1] https://developer.blackberry.com/native/documentation/cascades/getting_started/first_app/index.html. [Accessed 10 10 2013].

[5 GSMA, "GSMA Rich Communications Suite - Release 1 Functional Description," [Online].
Available: http://www.topdoggraphics.com/OtherSites/GSMA_RCS/reference-

2] documents/RCS-deep-dive/RCS%20Release%201/December%202008/functional_desc_v1.pdf. [Accessed 07 01 2014].

[53] Google, "gyp," [Online]. Available: https://code.google.com/p/gyp/wiki/GypUserDocumentation. [Accessed 10 01 2014].

[54] Nable Communications , INC., "IMS Client Solution," [Online]. Available: http://www.nablecomm.com/eng/solution/imsclient.php. [Accessed 13 01 2014].

[55] GSMA, "List of current accreditations for RCS client manufacturers (clients)," [Online]. Available: http://www.gsma.com/futurecommunications/faq/lists-of-current-accreditations-for-rcs-client-manufacturers-clients/. [Accessed 15 01 2014].

[56] Neusoft Corporation, "Neusoft Silta RCS client receives full accreditation from the GSMA," [Online]. Available: http://www.neusoft.com/news/html/20120919/2448154020.html. [Accessed 13 01 2014].

[57] Jibe Mobile, "PR Newswire," [Online]. Available: http://www.prnewswire.com/news-releases/jibe-mobile-selected-for-2013-world-communication-awards-shortlist-229660661.html. [Accessed 10 01 2014].

[58] GSMA, "RCS-e - Advanced Communications: Services and Client Specification," [Online]. Available: http://www.gsma.com/futurecommunications/wp-content/uploads/2012/03/rcs-e_advanced_comms_specification_v1_2_2_approved.pdf. [Accessed 26 01 2014].

[59] Summit Tech Communications, "Rich Communication Suite," [Online]. Available: http://www.richcommunicationsuite.com/. [Accessed 24 01 2014].

[60] GSMA, "Rich Communication Suite: Function Description Release 2," [Online]. Available: http://www.topdoggraphics.com/OtherSites/GSMA_RCS/reference-documents/RCS-deep-dive/RCS%20Release%202/June%202009/R2_090831_RCS_Release_2_Functional_Description_v1_0.pdf. [Accessed 12 01 2014].

[61] GSMA, "Rich Communications Suite: RCS Release 3 Functional Specifications," [Online]. Available: http://www.topdoggraphics.com/OtherSites/GSMA_RCS/reference-documents/RCS-deep-dive/RCS%20Release%203/December%202009/FunctionalDescription_v1.0(1).pdf. [Accessed 12 01 2014].

[62] Summit Tech Communications, "Summit," [Online]. Available: http://www.summit-tech.ca/company.htm. [Accessed 24 01 2014].

[63] Martine, "Ninja," [Online]. Available: http://martine.github.io/ninja/. [Accessed 24 01 2014].

[64] Tencent, “WeChat,” [Online]. Available: http://www.wechat.com/en/features.html#live_chat. [Accessed 10 11 2013].

[65] "Native SDK for BlackBerry 10," [Online]. Available: http://developer.blackberry.com/native/reference/cascades/bb__cascades__NavigationPane.html. [Accessed 25 08 2014].

[66] "Planning Poker," [Online]. Available: http://www.planningpoker.com/. [Accessed 26 08 2014].