Masters in Informatics Engineering
Internship
Final Report

Development of a browser-based Audio/Video Application using HTML5 and WebRTC

Luís Manuel Tavares de Matos
lmmatos@student.dei.uc.pt

WIT Software Supervisor:
Eng. Frederico Lopes

DEI Supervisor:
Dr. Mario Rela

Date: 3rd July 2013

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

Department of Informatics Engineering

Faculty of Sciences and Technology

University of Coimbra

Pólo II, Pinhal de Marrocos, 3030-290 Coimbra

☎+351239790000 | 🖷+351239701266 | 🖃 info@dei.uc.pt

| 🌍 http://www.uc.pt/fctuc

WIT Software, S.A.

EN1/IC2, Km 185,6, Banhos Secos, Santa Clara, 3040-032

Coimbra

☎+351239801030 | 🖷+351239801039 | 🖃info@wit-software.com | 🌍 http://www.wit-software.com/

Candidate:

Name: Luís Manuel Tavares de Matos

Student Number: 2007183871

Contact: lmmatos@student.dei.uc.pt

DEI Supervisor:

Name: Mário Rela

Contact: mzrela@dei.uc.pt

WIT Software Supervisor:

Name: Frederico Lopes

Contact: frederico.lopes@wit-software.com

Abstract

Rich Communication Services (RCS) is an industry effort focused on the use of IP Multimedia Subsystem (IMS) for providing enhanced mobile phone communication services. RCS's goal is to enable subscribers to establish calls between eachother despite their mobile operator and device type. Since this is only possible through unified standards RCS is a reliable base for communications standardization.

The Web is an environment where users communicate with each other frequently. Web-based solutions can reach a wide variety of devices. New technologies such as WebRTC and Firefox OS come to push the web to the next level.

WebRTC is the next step in browser-based real-time communications. It consists in adding native support for real-time audio and video communications in a web-browser. Firefox OS is an Operative System for mobile devices built upon HTML5 in development by Mozilla.

WIT Software is a RCS Unified Communications solution provider, WIT's Communications Suite (WCS) is complete set of communication products, for web, mobile and desktop scenarios.

In this internship, the main goal is to develop two RCS client applications to be added to the WCS family of products. A new Web-Based client built atop WebRTC and a new mobile client for Firefox OS.

Keywords

"WebRTC", "Communications", "Real-time", "Browser", "Rich Web Applications", "Rich Communications Services", "RCS", "joyn", "FirefoxOS", "HTML5", "W3C", "Boot2Gecko"

Index

List of Figures

Acronyms

AB	Address Book
API	Application Programming Interfaces
B2G	Boot to Gecko
CRM	Customer Relationship Management
CSS	Cascading Style Sheets
FF OS	Firefox OS /mOS
FT	File Transfer
GSMA	Global System for Mobile Communications Association
HAL	Hardware Abstraction Layer
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
HVGA	Half Video Graphics Array
IMS	IP Media Subsystems
ISDN	Integrated Services Digital Network
joyn	Costumer facing brand for RCS-e services
JSON	JavaScript Object Notation
LTE	Long Term Evolution
MSISDN	Mobile Subscriber ISDN Number
MSRP	Message Session Relay Protocol
MWC	Mobile World Congress
OEM	Original Equipment Manufacturer
OTT	Over-the-top
P2P	Peer to Peer
PBX	Public Branch Exchange
PSTN	Public Switched Telephone Network
RCS	Rich Communications Suite (or Services)
RCS-e	Rich Communications Suite (enhanced)
REST	Representational State Transfer
RTC	Real-Time Communications
RTCP	RTP Control Protocol
RTMP	Real Time Messaging Protocol
RTP	Real-Time Transport Protocol
SGM	Socket Gateway Module
SIM	Subscriber Identity Module
SIP	Session Initiation Protocol
SMS	Short Message Service
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UC	Unified Communications
UI / UX	User Interface
UNI	User to Network Interface
VOIP	Voice Over IP
WCAS	WIT Communications Application Server
WCS	WIT Communications Suite
WebRTC	Web Real-Time Communication
WWC	WIT Web Communicator
XCAP	XML Configuration Access Protocol
XML	Extended Markup Language

1. Introduction

This document reflects the work done during the first and second semesters of the academic year 2012/2013 of the Internship of the Masters of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra. The work took place on WIT Software, S.A., a company founded in 2001 as a spin-off from the University of Coimbra, specialized in the development of applications and services for telecommunication and media companies, and underwent the supervision of Dr. Mário Rela (DEI) and Eng. Frederico Lopes (WIT Software).

The initial goal of this internship was to conduct a detailed study about HTML5 and WebRTC technologies in order to develop prototypes of a voice and video communication application that runs in a web-browser. However during the start of the second semester of the Internship it was decided to develop a native client for Firefox OS. This new client will be a new branch of the WebRTC prototype that started to be developed during the first semester.

HTML5 is a cooperative effort between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG). HTML5 will be the new standard for HTML. HTML5 brings native browser support for audio and video elements as well as complex data manipulations that used to require various technologies such as Adobe's Flash or Shockwave. HTML5 is still work in progress however the major browsers already support many of HTML5 elements and API's.

WebRTC is the bleeding edge of browser enabled rich communications and a part of HTML5. Its objective is to bring the power of native desktop communication applications such as softphones[1] to the web-browser by invoking a series of simple JavaScript API's.

WebRTC was first proposed to W3C by Google after the acquisition of Global IP Solutions (GIPS) and with it its real time audio and video products and technology. Google provided and licensed GIPS's former voice and video media engines by wrapping then up with a set of JavaScript API's under the WebRTC open-source project. Finally they proposed the first draft for "WebRTC 1.0: Real-time Communication between Browsers" to the W3C in August 23th 2011.

The possibilities of having a rich communication application that's not bound to a specific platform can benefit several business areas, from Customer Relationship Management (CRM), online commerce and of course inter-device communication. All this is possible by using nothing more than a WebRTC capable web-browser and a web connection.

Currently WebRTC is behind standardized by two standard bodies, the IETF and W3C, it is also backed by major communications companies including Cisco and Ericsson.

Motivation

WebRTC has a great potential to disrupt OTT service providers and radically change the development of unified communications platforms. WebRTC enables the users to establish calls between each other quickly and without the sign-up process often required in OTT communications applications such as Skype. WebRTC presents potential opportunity for experienced rich communication solutions providers has they can quickly integrate WebRTC

[1] A Softphone can be described as software that simulates a cell phone. The purpose is to provide features normally available on mobile devices on a computer

clients into their own communications suite. However, this ease of deployment also removes the entry level barriers typical in the VOIP market by presenting startups with the means to quickly create web-based communication applications without the need for any additional technologies such as browser plug-ins.

Goals

WIT Software is a unified communications solution provider, WIT's Communications Suite (WCS) comprises a complete set of communication products, for web, mobile and desktop scenarios.

The goal for this internship is to explore the development a new product to integrate with the WCS family of applications. By the end of its development cycle the new product should consist of a browser-based, WebRTC powered communications platform with support for voice and video calls, one-to-one chat, file transfer and address book operations.

There are two major components that will be developed in the course of this project. The browser-side component, which consists in the JavaScript API that communicates with the user interface (UI), and the server-side component that consists in the module that will be integrated in WIT's Communications Application Server (WCAS) a legacy SIP Application Server. Detailed information about the project can be found in Annex C – Solution Architecture.

The client-side Software Development Kit (SDK) is intended to be independent from the UI it is attached to. It will provide a set of services that the UI can asynchronously invoke to establish communication with the server and manage all communications, and an event-based API to receive incoming calls and any other information from the server.

The server-side library will consist of the required mechanisms to transform WebRTC client requests into WCAS internal request format and subsequently transform the server responses in the client specific protocol.

The communication between these two entities is done using HTML5 WebSockets. The messages are structured using JavaScript Object Notation (JSON) according to a new proprietary communication protocol specific for WebRTC client.

During the second phase of this internship there is also planned the development of a RCS client for the Firefox OS mobile operative system (joyn FF OS). Detailed information regarding this project can be found in Annex D - FFOS.

Document Structure

The document starts with an introduction presenting the motivations behind this work. The planning for the first and second semesters of the internship is presented afterwards.

Then, an overview on the technologies involved on this project is given, which alternatives were considered and what studies were conducted. An analysis of some WebRTC solutions already available on the market is also presented in this chapter.

Apart from the document the following documents were created: details on state of the art and background knowledge (Annex A – State of the Art Document), detailed information on the project requirements (Annex B – Requirements Analysis) also detailed information on the system architecture (Annex C – Solution Architecture)

Detailed information regarding the joyn for Firefox OS client is found in (Annex D – FFOS).

2. Project Planning

This section makes reference to the overall planning of the internship project. It concerns the software development process adopted as well as an overview of the planned tasks. The project will be referred from now on as WIT Web Communicator WebRTC (WWC WebRTC). The RCS joyn client for Firefox OS developed during the second semester will be referred as joyn for Firefox OS (joyn FF OS).

2.1. Software Development Methodology

The WWC WebRTC project is of Strategic nature, its objective is to explore new technological possibilities in order to gain technological advantage in a new area of the business market. In this type of environment the needs and requirements of a project shift constantly making traditional development methodologies poorly suited for this task. This reasons conjugated with the fact that the internship also has a limited duration led to an iterative and incremental agile software development methodology being chosen.

The development methodology adopted was *Scrum* [1]. Sprint lasted two weeks. Daily status meetings were conducted with the Team Leader to discuss task status and plan the next step.

Project Execution Control

Besides the normal Scrum meetings, there are also scheduled weekly meetings that take place each Tuesday with the project owner to demonstrate the current developments. The sole purpose of these meeting is to show the current prototype functionality against the planned features.

Versioning

The project has a centralized repository available for documents and software versioning. This repository stores all the documentation and source code related to the project. WIT Software is an ISO-9001 [2] certified company which means that there are several rules and procedures that must be followed. These include naming conventions and specific software development guidelines for all project related artifacts.

2.2. Project Team

Bellow we have a table with the members that constituted the WWC WebRTC development team for the duration of the internship.

Product Owner	Frederico Lopes	
Team Leader	André Silva (*Intern Scrum Master*)	
Team	***Member***	***Project Area***
	Luís Matos	*WCAS Socket Lib and Client JS SDK*
	Bruno Leite	*WWC WebRTC User Interface*
	Nuno Campos	*WCAS ICE support*

Table 1 – WWC WebRTC team members

Bellow we have a table with the members of the joyn for Firefox OS development team for the second semester of the internship.

Product Owner	Frederico Lopes	
Team Leader	André Silva (*Intern Scrum Master*)	
Team	***Member***	***Project Area***
	Luís Matos	*WCS joyn for Firefox OS client*
	André Silva	*WCS joyn for Firefox OS client*

Table 2 – joyn FF OS team members

In the course of this internship André Silva is the person in charge of assigning tasks and reviewing the intern progress for each sprint, therefore he also acts as Scrum Master.

2.3. Planning

Gantt Charts are typically used to illustrate a project's schedule. They are also typically used in classical approaches to software development like the *Waterfall* [3] model. Despite what was said previously regarding the nature of the WWC WebRTC project three Gantt charts will be presented. The first chart illustrates the work developed during the first semester, the second represents the initial plan for the second semester and the last shows the actual work developed during the second semester. The Gantt charts were decided to be included both for a cleaner and quicker learning of the project status and as it was said previously the duration of this internship is limited to two semesters.

First Semester

This internship's first semester planning mainly focuses on investigation for the WWC WebRTC project. This consisted in the study of the WebRTC [4] and HTML5 video technology. A detailed analysis of possible solutions for the WWC WebRTC was conducted as well as an analysis of competitor solutions. This investigation effort was of a great help for the specification of the requirements and the architecture.

Parallel to this research, experiments with a previously developed prototype were also conducted with the objective of evaluating the feasibility of the WWC WebRTC project and learn how to use the project's technologies.

The original planning for the first semester suffered some modifications after a product roadmap reunion in October 26th. The scope of features originally planned have changed in order to have a working prototype for presentation in the 2013 edition of Mobile World Congress (MWC) [5] that was held in Barcelona in past February.

Upon reviewing the scope of features planned for this first semester it was decided the feature regarding information about contact availability would not be completed in the time window available and as such it was decided to swap it for RCS-E [6] contacts capabilities discovery. Capabilities discovery mechanisms allow the user to be notified about voice, video, and content sharing capabilities of the contacts in his/her address book. The availability information feature was shifted for the final release.

On a traditional waterfall model, these tasks would have represented the Requirements, Design and part of the Implementation stages.

Second Semester

Throughout the second semester significant functionality was added to the WWC WebRTC project. File transfer and 1-to-1 chat with typing and delivery notifications was implemented.

After a roadmap reunion that took place on the 3rd of March it was decided that the development of the other planned features (Group Chat and Video and Image Share) were to be postponed in order to start the developments of a WWC WebRTC prototype for Firefox OS which later became the joyn for Firefox OS project. The development of this new client took place during the second semester of this Internship.

Project Roadmap Events

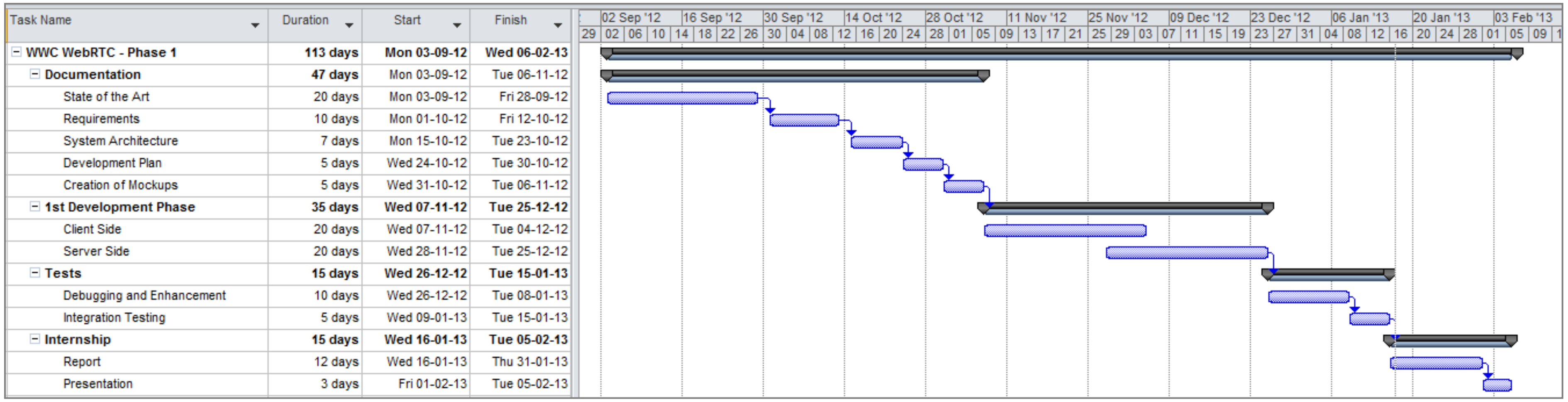
The following table illustrates the major roadmap events that took place during the internship.

Date	Overview
26 October	-Initial Project Milestones Delimited -It was decided that Bruno Leite would lead the development of the new client's UI (WWC WebRTC)
16 November	-Discussion of the JavaScript SDK Architecture -Presentation on the initial sketches for the new WWC WebRTC Client -It was decided that Nuno Campos would enable support for ICE connectivity in WCAS
23 November	-Unravel of the new WWC WebRTC Client Mockups -Discussion of the Features to present in Mobile World Congress (MWC)
3 January	-Further Refinement of the JavaScript SDK
25 -28 February	-Reveal of the WWC WebRTC client during the MWC 2013
3 March	-Intern moved from WWC WebRTC Client to the new joyn for Firefox OS client -joyn for Firefox OS application architecture discussion
13 July	-Internal Demonstration of the joyn FF OS client prototype

Table 3 – Project Roadmap Events

2.4. Overview of the First Semester

The following Gantt chart illustrates the tasks developed so far and the timeline involved in their execution.

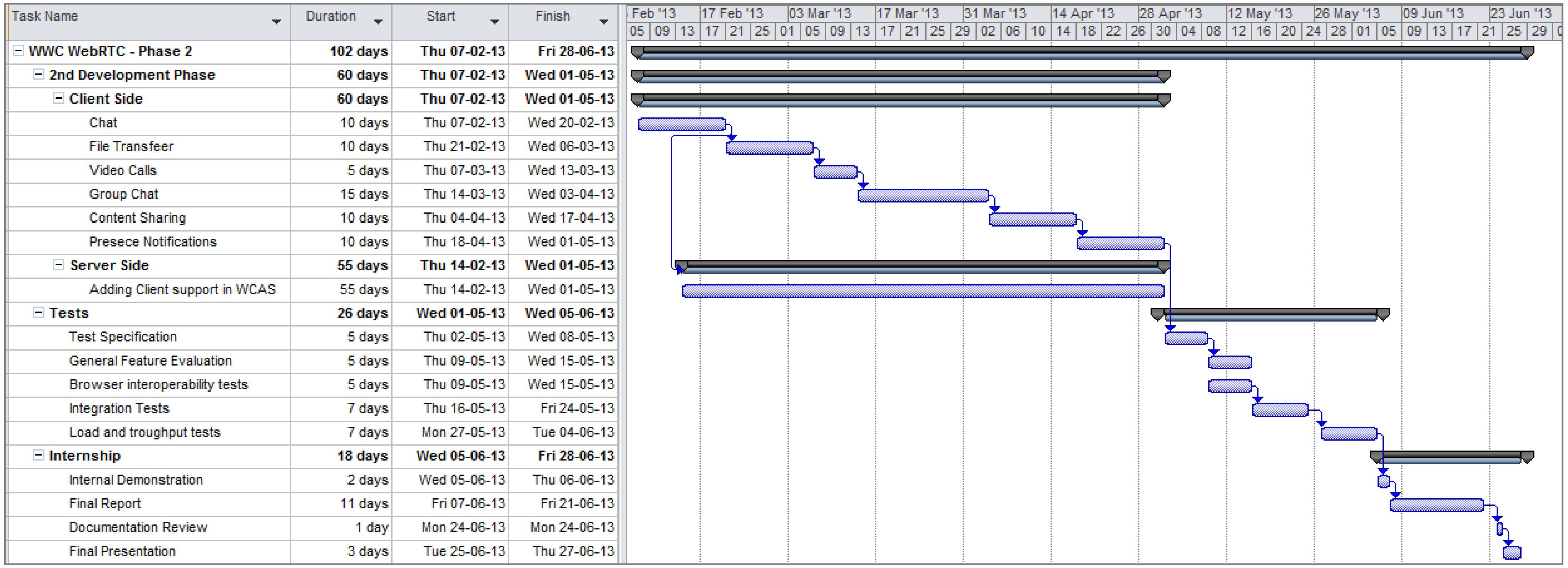


Task Name	Duration	Start	Finish
WWC WebRTC - Phase 1	**113 days**	**Mon 03-09-12**	**Wed 06-02-13**
Documentation	**47 days**	Mon 03-09-12	Tue 06-11-12
State of the Art	20 days	Mon 03-09-12	Fri 28-09-12
Requirements	10 days	Mon 01-10-12	Fri 12-10-12
System Architecture	7 days	Mon 15-10-12	Tue 23-10-12
Development Plan	5 days	Wed 24-10-12	Tue 30-10-12
Creation of Mockups	5 days	Wed 31-10-12	Tue 06-11-12
1st Development Phase	**35 days**	**Wed 07-11-12**	**Tue 25-12-12**
Client Side	20 days	Wed 07-11-12	Tue 04-12-12
Server Side	20 days	Wed 28-11-12	Tue 25-12-12
Tests	**15 days**	**Wed 26-12-12**	**Tue 15-01-13**
Debugging and Enhancement	10 days	Wed 26-12-12	Tue 08-01-13
Integration Testing	5 days	Wed 09-01-13	Tue 15-01-13
Internship	**15 days**	**Wed 16-01-13**	**Tue 05-02-13**
Report	12 days	Wed 16-01-13	Thu 31-01-13
Presentation	3 days	Fri 01-02-13	Tue 05-02-13

Fig. 1 – Gantt chart for the First Semester

2.5. Planed Work Plan for the Second Semester

The following Gantt chart illustrates the predicted timeline for the tasks expected to be completed during the second semester.

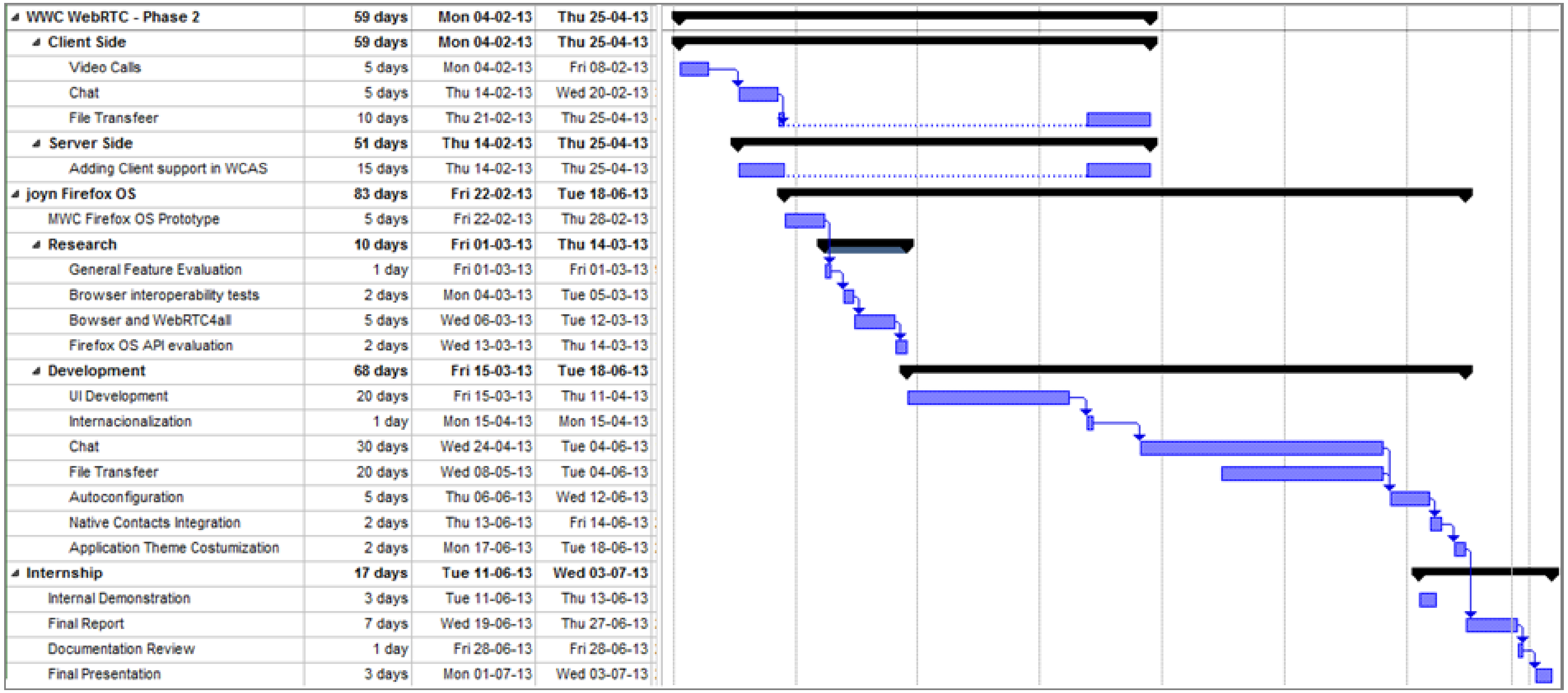


Task Name	Duration	Start	Finish
WWC WebRTC - Phase 2	**102 days**	**Thu 07-02-13**	**Fri 28-06-13**
2nd Development Phase	**60 days**	**Thu 07-02-13**	**Wed 01-05-13**
Client Side	**60 days**	**Thu 07-02-13**	**Wed 01-05-13**
Chat	10 days	Thu 07-02-13	Wed 20-02-13
File Transfeer	10 days	Thu 21-02-13	Wed 06-03-13
Video Calls	5 days	Thu 07-03-13	Wed 13-03-13
Group Chat	15 days	Thu 14-03-13	Wed 03-04-13
Content Sharing	10 days	Thu 04-04-13	Wed 17-04-13
Presece Notifications	10 days	Thu 18-04-13	Wed 01-05-13
Server Side	**55 days**	**Thu 14-02-13**	**Wed 01-05-13**
Adding Client support in WCAS	55 days	Thu 14-02-13	Wed 01-05-13
Tests	**26 days**	**Wed 01-05-13**	**Wed 05-06-13**
Test Specification	5 days	Thu 02-05-13	Wed 08-05-13
General Feature Evaluation	5 days	Thu 09-05-13	Wed 15-05-13
Browser interoperability tests	5 days	Thu 09-05-13	Wed 15-05-13
Integration Tests	7 days	Thu 16-05-13	Fri 24-05-13
Load and troughput tests	7 days	Mon 27-05-13	Tue 04-06-13
Internship	**18 days**	**Wed 05-06-13**	**Fri 28-06-13**
Internal Demonstration	2 days	Wed 05-06-13	Thu 06-06-13
Final Report	11 days	Fri 07-06-13	Fri 21-06-13
Documentation Review	1 day	Mon 24-06-13	Mon 24-06-13
Final Presentation	3 days	Tue 25-06-13	Thu 27-06-13

Fig. 2 - Gantt chart for the Second Semester (planned)

2.6. Overview of the Second Semester

The following Gantt chart illustrates the tasks developed during the second semester and the timeline involved in their execution.



WWC WebRTC - Phase 2	**59 days**	**Mon 04-02-13**	**Thu 25-04-13**
Client Side	**59 days**	**Mon 04-02-13**	**Thu 25-04-13**
Video Calls	5 days	Mon 04-02-13	Fri 08-02-13
Chat	5 days	Thu 14-02-13	Wed 20-02-13
File Transfeer	10 days	Thu 21-02-13	Thu 25-04-13
Server Side	**51 days**	**Thu 14-02-13**	**Thu 25-04-13**
Adding Client support in WCAS	15 days	Thu 14-02-13	Thu 25-04-13
joyn Firefox OS	**83 days**	**Fri 22-02-13**	**Tue 18-06-13**
MWC Firefox OS Prototype	5 days	Fri 22-02-13	Thu 28-02-13
Research	**10 days**	**Fri 01-03-13**	**Thu 14-03-13**
General Feature Evaluation	1 day	Fri 01-03-13	Fri 01-03-13
Browser interoperability tests	2 days	Mon 04-03-13	Tue 05-03-13
Bowser and WebRTC4all	5 days	Wed 06-03-13	Tue 12-03-13
Firefox OS API evaluation	2 days	Wed 13-03-13	Thu 14-03-13
Development	**68 days**	**Fri 15-03-13**	**Tue 18-06-13**
UI Development	20 days	Fri 15-03-13	Thu 11-04-13
Internacionalization	1 day	Mon 15-04-13	Mon 15-04-13
Chat	30 days	Wed 24-04-13	Tue 04-06-13
File Transfeer	20 days	Wed 08-05-13	Tue 04-06-13
Autoconfiguration	5 days	Thu 06-06-13	Wed 12-06-13
Native Contacts Integration	2 days	Thu 13-06-13	Fri 14-06-13
Application Theme Costumization	2 days	Mon 17-06-13	Tue 18-06-13
Internship	**17 days**	**Tue 11-06-13**	**Wed 03-07-13**
Internal Demonstration	3 days	Tue 11-06-13	Thu 13-06-13
Final Report	7 days	Wed 19-06-13	Thu 27-06-13
Documentation Review	1 day	Fri 28-06-13	Fri 28-06-13
Final Presentation	3 days	Mon 01-07-13	Wed 03-07-13

Fig. 3 - Gantt chart for the Second Semester (actual)

3. State of the Art

This section discusses the state of art analysis that took place for the two RCS clients developed during the course of this internship, the Wit Web Communicator WebRTC and the joyn FF OS Client.

3.1. Rich Communication Services

Rich Communication Services (RCS) is an industry effort focused on the use of IP Multimedia Subsystem (IMS) for providing enhanced mobile phone communication services.

RCS is a joint effort of network infrastructure vendors, handset manufacturers and Telecom Operators. RCS is seen as an industry initiative to fight the threat of OTT apps (such as Whatsapp, Viber, Skype, etc.) and to leverage the capabilities of IMS networks towards the evolution to Long Term Evolution (LTE).

3.1.1. What is RCS-e

RCS-e is a commercial specification for RCS (version 1.2.1). It was designed by a leading group of Mobile Operators, including Vodafone, Telefonica, Orange, Deutsche Telekom and Telecom Italia, under the governance of GSMA. The official announcement of RCS-e was unveiled at the 2011 Mobile World Congress.

RCS-e delivers an experience beyond voice and SMS, enabling users to use instant messaging, live video sharing and file share across any mobile phone on any network operator.

The service can be natively integrated in the mobile phone or via a downloadable app. The main features are:

- **Service discovery**: joyn identifies automatically other joyn contacts on the contact list;
- **Chat**: enable instant messaging with one or more contacts;
- **File Share**: transfer files (e.g. pictures, videos) during a chat or a voice call;
- **Live Video Share**: share live video during a voice call.

Joyn is the consumer-facing brand for the RCS-e services. The joyn brand is used by all operators deploying accredited RCS services.

Fig. 4 – Joyn Logo

3.2. WWC WebRTC

In this section the WebRTC technology will be discussed in detail. Topics such as where it came from, what it enables and its advantages and disadvantages will be covered. An overview of the current state and open issues will also be in discussion.

It will also be given an overview of the current technologies that enabled results similar to what will be expected from the WWC WebRTC client and an analysis of competitor solutions which already use the WebRTC technology.

This section ends with an overview of the technologies that will be used on the course of this project.

3.2.1. Overview

WebRTC is an API definition being carried out within W3C and IETF to standardize the support for real-time capabilities in web browsers, allowing web applications to send data and media streams between devices over IP networks in a peer-to-peer fashion [7]. WebRTC empowers the developers with the ability to create Rich Internet Applications (RIA's) without relying on third party plug-ins.

3.2.2. WebRTC Architecture

Bellow we have the diagram of the WebRTC architecture.

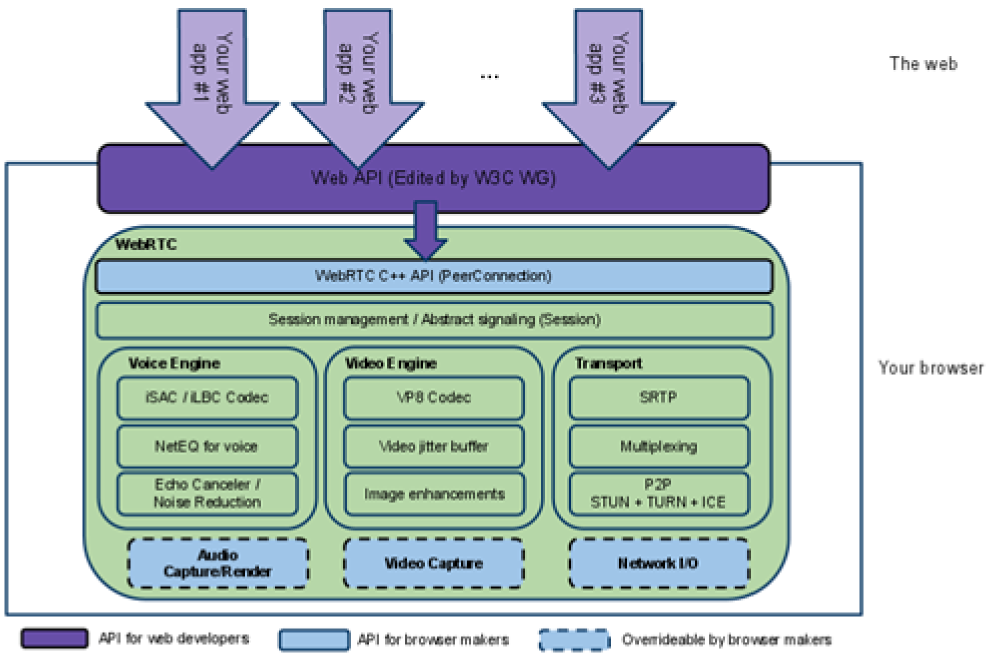


Fig. 5 - WebRTC Architecture [8]

This architecture is attractive for the Web Developer Community for various reasons. The Transport Layer (PeerConnection) is compact and easy to use. With a few lines of code we can establish connectivity between two web browsers. The Codec processing is automatically managed, removing the need to acquire these from third parties. The WebRTC project uses the VP8 [9] codec for video and Opus [10], G.711 [11], iLBC [12] and iSAC [13] for audio. All these codec's are royalty free.

Besides, having this technology natively accessible in the web browser reduces the chance of having integration problems at the expense of not having much control over the real-time data transmission flow. Other benefits include a reduced load on Web Servers previously used to relay the communication, automatic adaptation to network conditions changes and browser embedded echo cancelation and noise suppression mechanisms.

Other features include the possibility to send different media types are sent over the same channel using RTP multiplexing, reducing the time it takes to establish a session and improving the overall efficiency of the data channel. The audio and video streams are encrypted using Secure Real-time Transport Protocol (SRTP) [14] with DTLS [15] algorithms.

Bellow we have a diagram showing an example of WebRTC browser model.

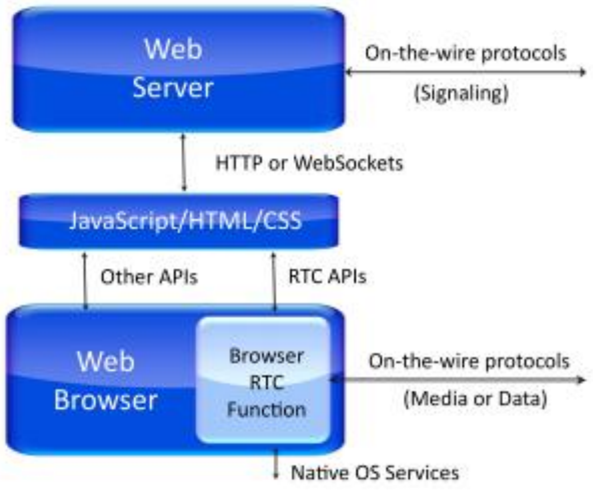


Fig. 6 – WebRTC Browser Model [16]

We will discuss the main components (API's) behind the WebRTC technology. These components reside in the browser (Browser RTC Function component in Fig.4), its implementation may vary depending on the browser manufacturer.

GetUserMedia

A WebRTC capable browser accesses the user's audio and video devices using the GetUserMedia API. This API is an implementation of the W3C Media Capture and Streams working draft [17]. This API can access multimedia data (video, audio or both) from local devices (video cameras, microphones, webcams) and represent it using data streams which can then be played using HTML5 video or audio elements or sent over the web using the PeerConnection API.

PeerConnection

The PeerConnection API is a top level API for WebRTC media exchange. This API is intended to implement the W3C WebRTC API [18] proposal for the PeerConnection JavaScript API in C++. The PeerConnection API lies at the heart of real time communication. It allows P2P (browser-to-browser) streaming of audio and video information. In order to setup a PeerConnection the browser must exchange information regarding its codec support. This is done by exchanging extended SDP (Session Description Protocol) [19] messages. The session establishment is done by using ICE (Information and Content Exchange protocol) [20].

3.2.3. Conducted Studies

During the initial stage of the internship a study to evaluate the feasibility of the WWC WebRTC project was conducted. More information regarding this study can be found in Annex A – State of the Art Document.

The purpose of the study is to evaluate what solutions existed for managing SIP (Session Initiation Protocol) in the web-browser since browsers cannot manage UDP/TCP sockets directly. This study attempts to give an overview of what solutions exist that do not require the utilization of third party plug-ins.

Possibilities Considered

As the browser cannot send SIP directly, we need to pass the communication through a capable component.

The following options were deemed viable:

- Create a **new JavaScript API** to communicate with WCAS via Websockets;
- Use the **current ActionScript/JavaScript API** that communicates with WCAS using Real Time Messaging Protocol (RTMP).
- Create a new client that uses the **RCS REST API** [21] model;
- Use a library to **create SIP requests in the browser** and send them to a SIP proxy via Websockets.

The approach that consisted into reusing the existing ActionScript/JavaScript API was deliberately left out of the study since it is only attractive as a fallback option and the main objective of this project is promote the use of HTML5 and WebRTC technology.

Study Conclusions

In this sub-section the conclusions regarding each approach considered initially will be discussed.

The RCS API approach was rendered not suitable for a short term development effort because it conflicts greatly with the logic flow of the existing API. For example the commands to pass to the Wit Communications Application Server (WCAS) cannot be directly invoked in a similar way to the existing API. This translation changes would lead to additional development effort.

The libraries that exist for generating the SIP packets in the web browser (JavaScript based SIP Adapter) are not a good option yet due to complexity and stabilization issues. At the moment we are still far from an interesting SIP stack. Besides the SIP stack built in WCAS is time-proven and toughly tested.

Using the existing Flash/JS API is only relevant when trying to interact with clients that do not support WebSockets. Therefore it was selected as a fallback option.

This leaves us with the implementation of WebSocket support in WCAS. The flexibility and reutilization of various components makes it the most attractive option for short term development.

3.2.4. Current WebRTC Browser Support

This sub-section concerns to the current support of the WebRTC API in the main browsers.

Browser	Notes
Opera	Implemented part of WebRTC on the laboratory version on January 2012. The current stable version already has support for the GetUserMedia API.
Chrome	Integrated WebRTC into its dev channel release in January 2012. It's also included in the stable release since version 20 (June 2012)
Firefox	Included WebRTC into the Alpha version in early 2012 and it is expected to be included in the stable release of Firefox 18 in January 2013 [22]
Internet Explorer	Microsoft stated that they have started working on implementation of the API. News of the current state of implementation can be found in HTML5 Labs [23]
Mobile Devices	Ericsson announced the world's first WebRTC-enabled browser for mobile devices, called "Bowser" currently available for iOS and Android (October 2012) [24]

Table 4 - Main browser's WebRTC support

3.2.5. Analysis of Competitors

WebRTC applications benefit from the advantage of being able to follow a "develop once deploy anywhere" philosophy. In the future the same web application can be used to connect clients in a variety of devices, from desktop computers to tablets and Smartphone's. Only the media devices, a web connection and a WebRTC enabled browser will be required

As browser makers continue to shift toward WebRTC more and more companies tend to invest in this area in order to get a foothold in this new business market and take advantage of its possibilities.

In this section some web-based voice and video clients that use WebRTC will be presented and compared.

***Frisb* – Frisb™**

FrisB [25] is a web-based voice service that creates a voice channel to ring and invite a regular telephone user to talk. FrisB is a free service that uses WebRTC to send the user's audio media to a lightweight Public Switch Telephone Network (PSTN) gateway.

***Phono* – Voxeo**

Phono [26] is a jQuery plugin and JavaScript library that turns a web browser into a softphone, capable of making phone calls and sending instant messages.

***Sipml5* – Doubango**

Doubango [27] claims that Sipml5 [28] is the world's first open source HTML5 SIP client entirely written in JavaScript. Sipml5 can be used to connect to any SIP or IMS network using no extensions or plugins to make audio/video calls and instant messages. Doubango also has a WebRTC to SIP gateway called webrtc2sip.

***Twelephone* – GetVocal**

Twelephone [29] adds real-time communications (voice, video, presence and messaging) to Twitter using HTML5 and WebRTC. Upon installing an extension in Chrome the user interface of Twitter is changed so that it now provides information about the presence of a person (that must also be using Twelephone) and allows the user to initiate a call or send a private message.

***Open tok* – Tokbox**

OpenTok [30] is an API developed by tokbox to enable developers to build face-to-face video to any web browser, iOS, or Android device. OpenTok Servers manage all communication and streaming between devices.

Clientless Web Softphone – Thrupoint

The Thrupoint Clientless Web Softphone [31] is one of the solutions of the Thrupoint Unified Communications (UC) Suite. Thrupoint's WebRTC solutions provide the signaling interworking, media transcoding and applications needed to connect to Thrupoint's UC network. Thrupoint also offers APIs for WebRTC to SIP interworking.

***Connect* – Utribo**

Connect by Utribo [32] is Software as Service (SaS) platform that enables subscribers to receive calls made in a web browser (using WebRTC) to their computer, phone, or Private Branch Exchange (PBX).

***Zingaya* – Zingaya, inc**

Zingaya [33] is a software as service (SaS) platform especially suited for online sales or customer service. Zingaya provides an application that can be embedded into e-retailer's website. Using this application the costumer can talk directly through the browser to a business representative who can be working in the sales department, at his own computer or on a mobile phone.

***TenHands* – TenHands**

TenHands [34] is a web used to deliver real-time communications in a browser-based application. At the moment Tenhands requires a browser plug-in despite the fact that the media is sent using WebRTC.

***Bistri* – bistri ©**

Bistri [35] is a web based service that provides its members with a customizable link that works like an online phone number. The application user interface is built using HTML5 and CSS and the media is processed using WebRTC. Bistri provides integration with numerous services like Gtalk, Facebook, Windows Live and Yahoo.

Considerations

Of the solutions analyzed the application that most resembles what is expected of the WWC WebRTC client is the Clientless Web Softphone by Thrupoint. WIT Software and Thrupoint are both unified communications providers so out of this group this application is the closest to a competitor that we get. Thrupoint's Clientless Web Softphone hasn't been released yet and as such it could not be analyzed in detailed.

Therefore as a measure of quality the current version of WWC has been selected as the prime competitor although it does not use HTML5 or WebRTC Technology it still delivers the features expected to be implemented during this internship.

Competitor Comparison

Bellow we have a comparative table of the capabilities and underlying technology of some of the market available WebRTC solutions.

Client	Company	Protocol	Communication with the Server	PSTN Interworking	SIP Interworking	License	Audio	Video
FrisB	FrisB	Proprietary	HTTP	**Yes**	**No**	Not Released	**Yes**	**No**
Phono (WebRTC)	Voxeo Labs	XMPP	HTTP	**Yes**	**Yes**	Not Released	**Yes**	**Yes**
Sipml5	Doubango Telco	SIP	WebSockets	**Yes**	**Yes**	GPL V3	**Yes**	**Yes**
Twelephone	GetVocal inc	Proprietary	WebSockets	**No**	**No**	Not Released	**Yes**	**Yes**
Open.Tok	Tokbox	Proprietary	HTTP	**Yes**	**Yes**	Proprietary	**Yes**	**Yes**
Thrupoint	Thrupoint	Proprietary	HTTP	**Yes**	**Yes**	Proprietary	**Yes**	**Yes**
Utribo	Utribo	Proprietary	HTTP	**Yes**	**Yes**	Proprietary	**Yes**	**Yes**
Zingaya	Zingaya	Proprietary	RTMP	**Yes**	**Yes**	Proprietary	**Yes**	**No**
TenHands	TenHands	Proprietary	HTTP	**No**	**No**	Proprietary	**Yes**	**Yes**
Bistri	Bistri Engineering	XMPP	HTTP	**No**	**No**	Proprietary	**Yes**	**Yes**
WWC WebRTC	WIT Software	Proprietary	WebSockets	**Yes**	**Yes**	Proprietary	**Yes**	**Yes**
WWC RCS-e (Flash)	WIT Software	Proprietary	RTMP	**Yes**	**Yes**	Proprietary	**Yes**	**Yes**

Table 5 – Comparison of Competitor Solutions

3.2.6. Project Technologies

This section describes the technologies that will be used in the course of this project, each of the following sections identifies how will they be applied, and the alternatives studied.

Q Promises

Q Promises [36] are one of three ways available in JavaScript for asynchronous operations handling. The other two are events and callbacks. Promises can be combined with callbacks and the promise object can be passed around as an argument to functions and fulfilled when the appropriate time comes.

A promise is an object that represents the return value or the thrown exception that a given function may eventually provide. Promises invert the "inversion of control" approach of callbacks, cleanly separating the input arguments from flow control arguments.

Promises were also chosen to be used because they add extra security to the web application since all the code to be executed upon fulfilling the promise can't mess with the global scope of the SDK.

Angular JS

Angular JS [37] is a HTML framework that enables the creation of dynamic views in web-applications. Simply put it acts as an extension to the HTML language. It automatically synchronizes data from the UI (view) with JavaScript objects (model) through 2-way data binding. Since we cannot use Flash for rendering the user interface, Angular JS was chosen since enables the creation of dynamic web applications with various views with little complexity.

WebSockets

The HTML5 WebSocket JS API enables bi-directional communication between a browser and a server over one TCP Socket. HTTP is only half-duplex, communication using Websockets is quicker and more scalable because of a reduced number and size of the requests. The WebSocket API is still being standardized by the W3C however the major browsers have already support for this technology.

The following WebSocket libraries were analyzed:

- **jWebSocket** [38]
- **Atmosphere** [39]
- **MigratoryData** [40]
- **Bristleback** [41]

The following table summarizes the features of the WebSocket frameworks analyzed.

jWebSocket	Atmosphere	MigratoryData	Bristleback
• Cross-Browser Compatibility • Clear and Simple API • Support for Text and Binary Data • Fallback Support using Flash • Full Socket Session Management • Good Documentation	• Cross Browsers Compatibility • Active Development • Fallback Support using Comet and JASONP • Good Documentation	• Support for Text and Binary Data • Highly Scalable (horizontally and vertically) • API's for multiple Languages • Extensive Documentation	• High Level Abstraction • Integration with Spring • Different development branch of jWebSocket. • Good Documentation
GNU LGPL	**Apache v.2**	**Commercial**	**GNU LGPL**

Table 6 – Comparison of WebSocket Frameworks

The purpose of this study was to learn the current state and maturity of WebSocket Frameworks. The choice of the jWebSocket library was conditioned due to integration factors with WCAS. As a requirement in Session Initiation Protocol (SIP) transactions multiple responses are given for a unique request. For example on a SIP Invite the sequence is the following:

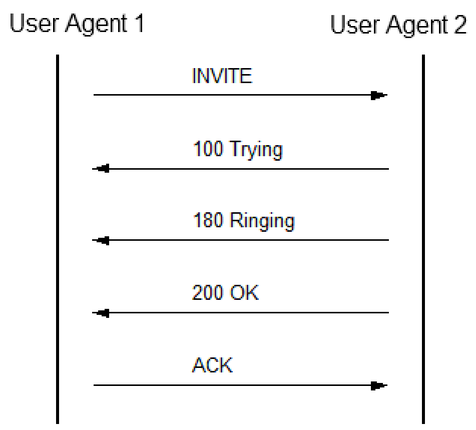


Fig. 7 – SIP Message Exchange for a SIP INVITE

In order to support this kind of operations the WebSocket library had to be modified. The jWebSocket library was chosen because it provided good documentation that allowed this modification.

Before the beginning of this internship the jWebSocket library was already used with success in WCAS, therefore no reason to change was identified at this point.

Chrome WebRTC Implementation

Development will take place in the Chrome Web Browser because it is the leading browser in the WebRTC API implementation since January 2012. Chrome Canary implements the latest changes in the WebRTC framework and the features provided (Chrome Web Tools) make developing and debugging an easier task.

Grunt JS

Grunt JS [42] is a task-based command line build tool for JavaScript projects. Grunt automates several tasks in a JavaScript project typical workflow. For instance it automatically concatenates the entire project's files into a single JavaScript (JS) file, validates the code with code quality tools, minifies the JS code and automatically launches a static preview server. Grunt JS also does all these tasks automatically whenever it detects that any of the files in the project directory have been changed.

JSON

JavaScript Object Notation (JSON) is a data-interchange format for representing data structures and associative arrays. JSON is an alternative to the XML format, and is gaining its importance since it is more lightweight than the previous, which is important when it comes to passing data over network connections.

JSON is the data type that is exchanged with WCAS's WebSocket Module. Since the SDK is built in JavaScript objects can be directly passed to the server without being serialized and we will also use less bandwidth in comparison to other data interchange formats like Extensible Markup Language (XML).

3.3. Joyn for Firefox Client

This section discusses the state of the art that directly concerns the joyn for Firefox OS Client.

3.3.1. The Firefox OS Operative System

Previously known as Boot to Gecko, Firefox OS is a Linux based Operative System for mobile devices in development by Mozilla. Its core concept is to allow HTML5 applications to communicate directly with the device's hardware using a series of JavaScript API's (Web API's).

3.3.1.1. Firefox OS Architecture

Bellow we have a diagram that depicts the high level architectural components of Firefox OS [43].

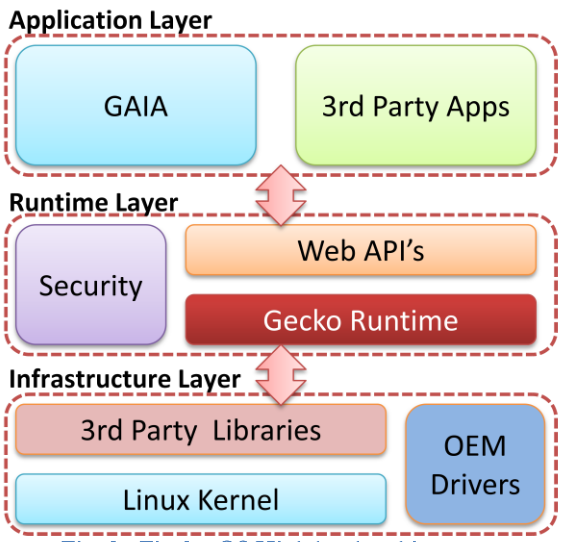


Fig. 8 - Firefox OS High level architecture

We can identify three major layers from this diagram:

Application Layer

The Application layer includes Gaia which is the User Interface of Firefox OS. Gaia implements all the standard system components such as the lock screen, home screen and default system applications. Third party applications are also stored here.

Runtime Layer

The Runtime layer includes the Firefox OS runtime (Gecko). Gecko provides support for HTML, JavaScript and CSS processing and includes important components such as the network stack, user interface layout stack and JavaScript engine. This layer also includes the Web API's that are accessed by Gaia and 3rd party applications and the Security layer that moderates access to Gecko and the Infrastructure layer.

Infrastructure Layer

The Infrastructure layer also called Gonk is the lower layer of the Firefox OS operative System. It consists of a Linux kernel and userspace and the Hardware Abstraction Layer (HAL) that includes the Original Equipment Manufacturer (OEM) firmware and drivers as well as 3rd party device drivers.

3.3.1.2. Firefox OS Applications

The entire user interface of Firefox OS is a Web Application (Gaia) written in HTML, JavaScript and CSS. Gaia is responsible for rendering content on the screen and interacting with the FF OS runtime (Gecko). Gaia is also capable of displaying and launching other Web apps. Any application that runs on Firefox OS is also a Web page, although with enhanced access to the mobile device's hardware and services through the use of Web API's.

3.3.1.2.1. Web API's

Web API's are methods exposed by the runtime layer that enable the web applications to access the device hardware. The level of access to these API's varies according to the type of the application. In order to access sensitive device data the applications must be "trusted" and in some cases require the user permission.

3.3.1.2.2. Application Security

Application security in FF OS is enforced trough Content Security Policy (CSP). This CSP policy dictates what type of content the web application can run and which device API's it has access to.

There are two types of applications, hosted and packaged. An application is called hosted if its resources are obtained remotely from a remote web server. The application is said packaged if all its resources stored in the device encapsulated in a single zip file.

There are three security levels that can be attributed to Firefox OS applications.

- **Web (Hosted)**: Typical Web Content;
- **Privileged:** Trusted Applications, obtained through Firefox OS App Stores;
- **Certified:** Highly Trusted Applications, certified by a Carrier or the OEM;

3.3.2. Competitor Analysis

In this section we will present and compare commercial Rich Communication applications available for Firefox OS.

joyn for Firefox OS (Solaiemes)

Fig. 9 - by Solaiemes application logo

Joyn for Firefox OS [44] was announced by Solaiemes[2] on January 2013 and is currently the only publically known RCS joyn application available for Firefox OS.

The application is a JavaScript/Ajax RCS thin client. Major computation operations are done remotely. After logging in the subscriber is identified through a URL identifier. RCS services are accessed through a SOAP [45]/REST [46] API exposed by the server.

3.3.2.1. Overview

Bellow we have a comparative table of the capabilities and underlying technology of the joyn clients of Solaiemes and WIT Software.

	Joyn Solaiemes	Joyn WIT Software
Protocol	Proprietary	Proprietary
Communication with the Server	HTTP	WebSockets
License	Proprietary	Proprietary
RCS Version	RCS 2	RCS-e
Layout Customization	CSS	Angular JS and CSS
Client Type	Thin	Thick
Capabilities Discovery	**NO**	**YES**
Availability Information	**YES (RCS 2)**	**SIP OPTIONS**
Chat 1-1	**YES**	**YES**
File Transfer	**YES**	**YES**
Integration with Device Contacts	**NO**	**YES**
Support for Network Address book (NAB)	**YES**	**YES**

Table 7 – RCS Client applications for Firefox OS Overview

[2] http://www.solaiemes.com/

4. Proposed Approach

4.1. Wit Web Communicator WebRTC

This section concerns the proposed architecture for the development of the WWC WebRTC project.

In this section the features and the associated requirements of this project will be described. Lastly an overview of the system architecture will be discussed.

4.1.1. Features

The WWC WebRTC client is designed to be integrated with the WCS family of products. The goal is to create a fully capable web-based communication application without the use of third-party plug-ins.

The initial features required for the WWC WebRTC client are the following:

- Voice and Video Calls;
- Possibility of media sharing (images and video) during a voice call;
- One to One Chat and Group Chat with File Transfer;
- Information about contacts availability and media capabilities;
- Contact management;
- Possibility of sending SMS / MMS if the contact can't receive IM messages;

4.1.2. Requirements

In this section the requirements for the WWC WebRTC project will be discussed. An overview on the user stories and use cases for a typical user will also be given.

The detailed requirements specification can be found in Annex B – Requirements Analysis.

4.1.3. User Stories

In this section the User Stories for a normal WWC WebRTC user are depicted.

As a User I want to…
"…Make a call"
Using a Microphone and Webcam the user must be able to do audio and video calls to his friends.
"…Send an SMS/MMS"
The User must be able to send text/image messages to friends using mobile devices.
"…Chat with a friend
The User must be able to chat with a friend through the browser.
"…Add Contacts to my contact list"
The User must be able to insert new contacts in his/her contact list.
"…Manage Contacts"
The User must be able to manage his/her contact list (delete, block, edit, etc...)

4.1.4. Use Cases

Below is represented a Use Case diagram representing the actions that a user can take in the WWC WebRTC application.

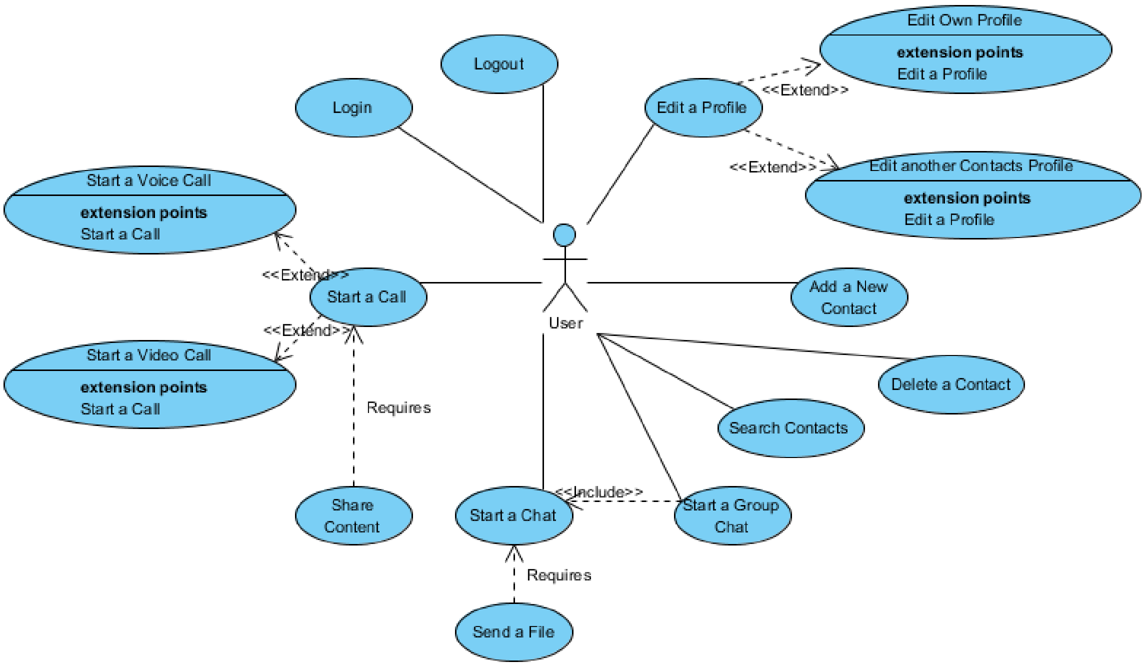


Fig. 10 – User Use Cases Diagram

Client

This subsection reefer to the requirements identified for the WWC WebRTC JavaScript SDK.

- The WWC WebRTC SDK must be able to establish a WebSocket connection with WCAS.
- The SDK must provide an interface layer with services so that the higher abstraction layer (UI) can invoke methods from the SDK and communicate and receive messages from the server.
- All the internal components of the SDK **must** not be directly accessible from outside. Instead all interactions **must** be done trough the services layer API. All internal components of the SDK **should** be able to talk to each other transparently.
- The UI **must** remain responsive despite any delays induced by the SDK. Most operations **should** be asynchronous.
- All components **should** be implemented as modules on the SDK in order to allow different configurations upon initialization.
- All SDK components **should** be as independent as possible and work without dependencies.
- The SDK **must** provide an API for the registering of events so that the UI can be notified when a certain action has taken place.

The function of the Events API is explained in the following diagrams.

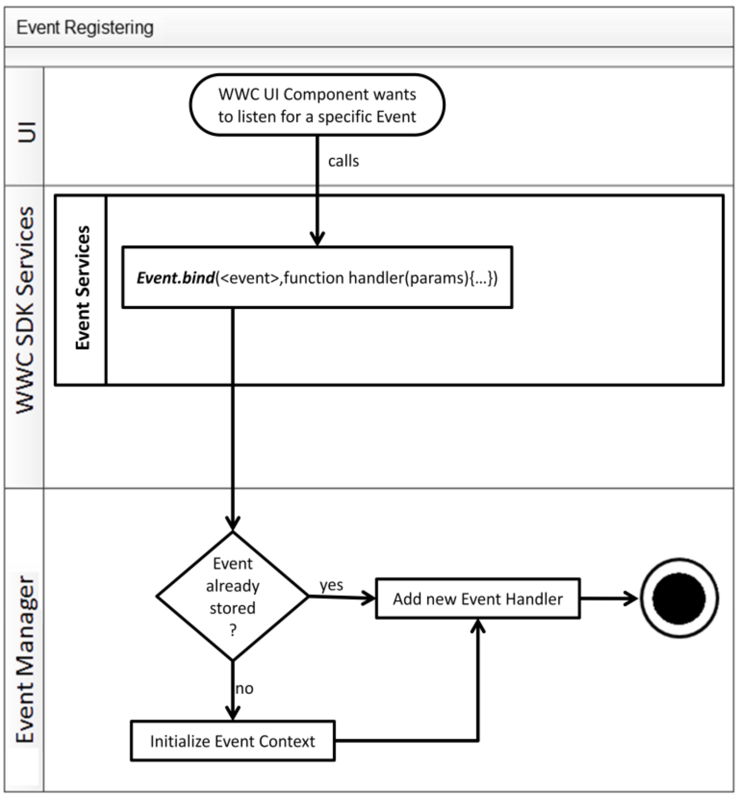


Fig. 11 – Event Registering Flow Diagram

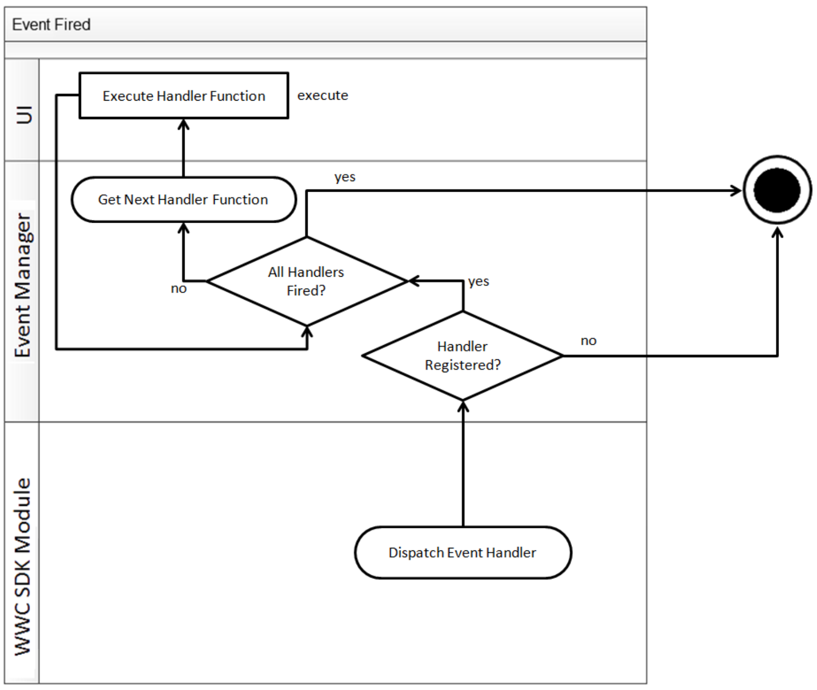


Fig. 12 – Event Firing Flow Diagram

Server

The server comprises of WCAS and the Socket Lib Module. This is the most complex module of the system as it must follow WCAS design guidelines.

- The Socket Lib module **must** be integrated with WCAS according to its modular design methodology.
- WCAS **must** be able to receive and recognize commands send by the WWC SDK.
- WCAS **must** be able to recognize when a user terminates the session either explicitly by sending a SESSION_TERMINATE Request or when WCAS detects that the WebSocket connection has been closed.
- Each transaction between WCAS and WWC SDK **must** be numbered with a unique transaction number.
- The Socket Lib module **must** implement the necessary logic to interact with WebRTC clients as well as the necessary Command Translators that enable communication with WCAS core.

4.1.5. Architecture

This subsection concerns the defined architecture for WWC WebRTC project. This concerns both the WebRTC JavaScript SDK and the Server Socket Lib library.

The detailed architecture specification can be found in Annex C – Solution Architecture.

Overview

The following figure (Fig. 5) shows an overview of the Wit Web Communicator (WWC) WebRTC solution Architecture. As we can see, the WWC WebRTC solution has two major components/sectors: the client and the server.

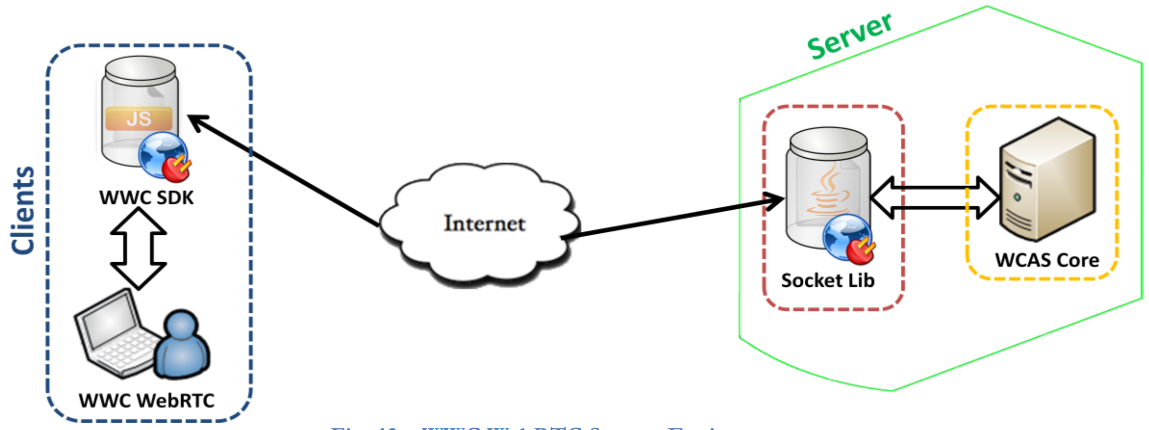


Fig. 13 – WWC WebRTC System Environment

The **Server** (in green) is composed by:

- The **Socket Lib** library imported in WCAS.
- WIT's **WCAS** (Wit Communications Applications Server) on top of which the Socket Lib Module is deployed.

The architecture specification regarding the WWC SDK (Client) and the Socket Lib (Server) will now be discussed.

Client

Bellow we have a high level view of the WWC WebRTC SDK.

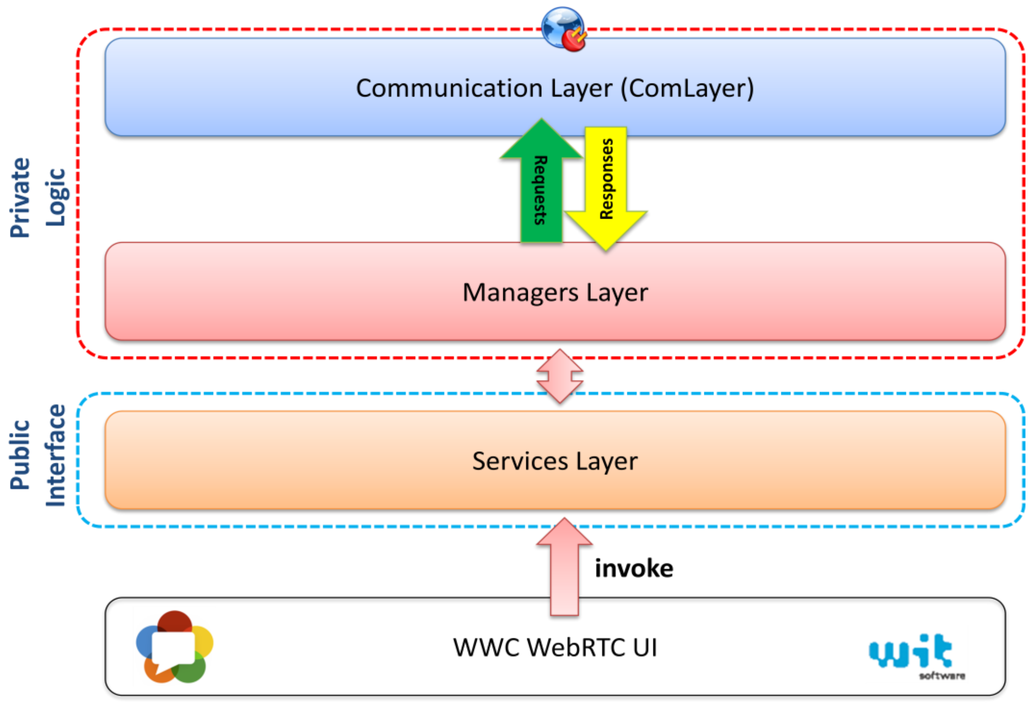


Fig. 14 –High Level Design of the Client SDK

There are three major layers in the WWC WebRTC JavaScript SDK. Detailed Information regarding the components that constitute these layers can be found in Annex C Architecture Specification.

a) Communication Layer (ComLayer)

This is the component responsible for the communication with the server. It also manages and notifies the Manager Layer of incoming requests and responses.

b) Managers Layer

This is the layer where the internal logic of the SDK is stored and is not directly accessible for the UI. This layer consists of various classes (Managers) that have different competencies. There are managers for Session Management, Call Management, Chat Management, Content Share Management, Capabilities Discovery and Events Management.

c) Services Layer

The Public Services Interface contains functions that the UI can invoke in order to trigger a given action. These functions have privileged access to the manager's public methods and are organized according to their respective manager.

Server

This section refers to the architecture design of the Socket Lib module integrated in WCAS.

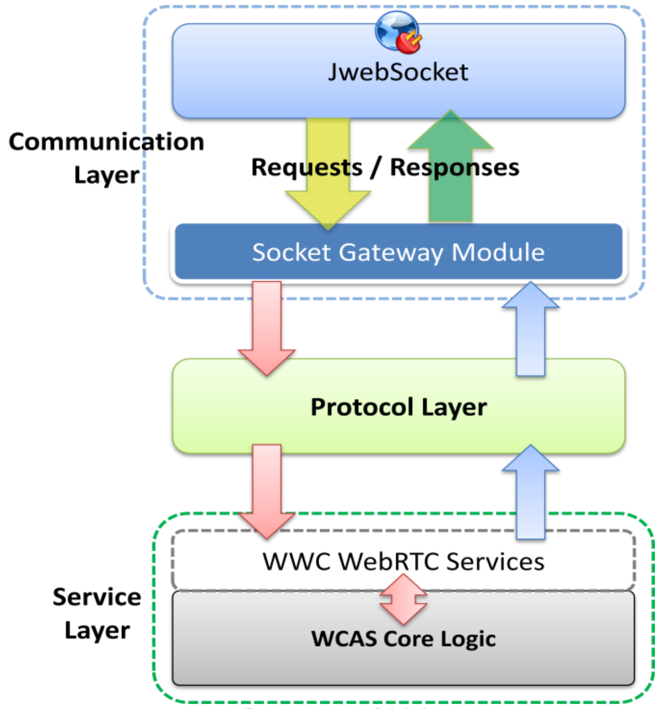


Fig. 15 – High Level Design of WCAS Socket Lib module

There are three major components layers that make up the WCAS Socket Lib Module. Detailed Information regarding the components that constitute these layers can be found in Annex C Architecture Specification.

a) Communication Layer

This is the middleware layer responsible of delegating requests incoming from WWC WebRTC clients to the Protocol Layer for further processing. It is also responsible for sending the responses and commands received from the Protocol layer back to the client using the WebRTC client specific protocol using the jWebSocket library.

b) Protocol Layer

This layer is where all the messages between the client and the server are constructed and deconstructed. This layer is made up by command Translators that are responsible for translating client requests into WCAS internal command syntax and WCAS responses into each client's explicit protocol.

c) Service Layer

The services layer is stored in the Core of WCAS. The core exposes a set of API's which must then be overridden by client specific services. This results in a clear separation between the business logic of the WCAS core and domain-specific problems such as client interoperability.

4.2. Joyn for Firefox OS Client

This section concerns the overall architecture of the joyn for Firefox OS application.

4.2.1. Architecture Overview

Bellow we have a simple diagram depicting the high level architecture of the joyn for Firefox OS client application.

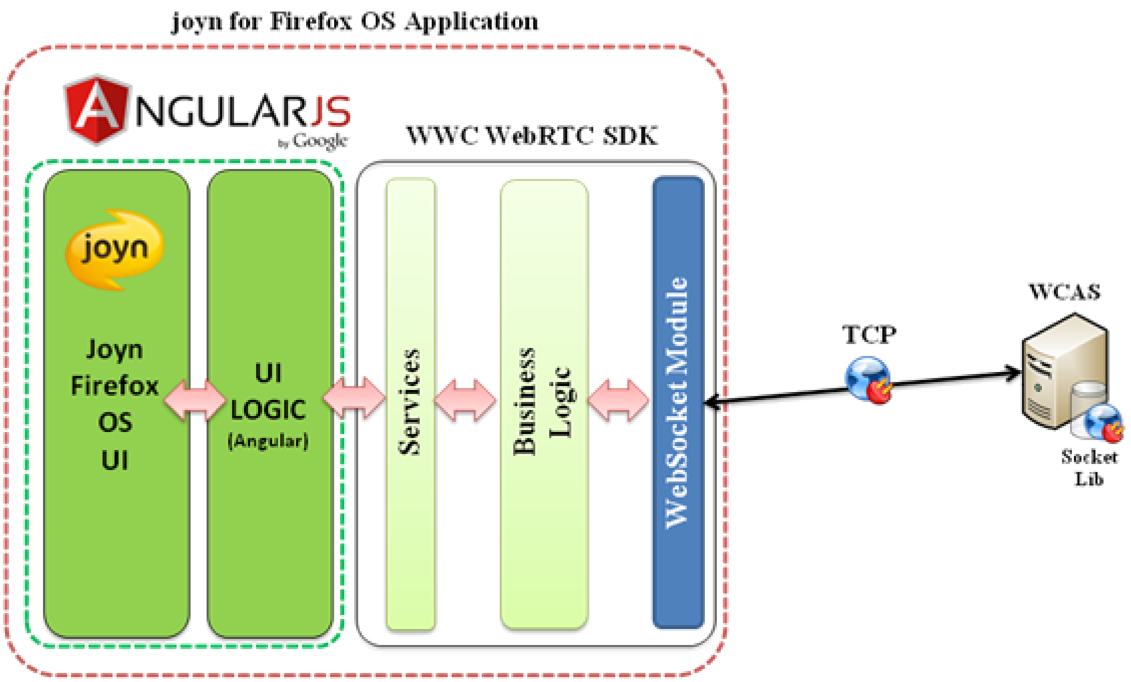


Fig. 16 - Architecture overview of the joyn FF OS application

In order to develop the joyn FF OS application with the imposed constraints (limitations regarding both team members and overall project time) several components were re-used from the WWC WebRTC project. Namely the WWC WebRTC SDK remains the same for both projects since both applications share a web – based environment and the WWC WebRTC SDK was designed to be independent of the UI it is attached to.

Angular JS is a web MVC framework used in both applications. The flexibility and component isolation provided by it ensured that almost all the services used for data manipulation and WWC SDK management are similar to identical in both the joyn FF OS and WWC WebRTC projects.

4.2.2. Design Principles

FF OS applications are written in HTML5. The visual components are arranged in HTML files and styled trough CSS. All client logic is computed using JavaScript. JavaScript is also used to access the device data such as the contacts and data storage.

The application will be built atop the Angular JS framework, same as the WWC WebRTC version. This way we maximize reutilization of components such as Controllers and Services having only to develop the views from scratch according to the design document specified for the joyn FF OS client. Detailed information regarding the joyn FF OS client design can be found in Annex D FFOS.

4.2.3. Architecture

Bellow we have a diagram that describes the architecture of the presentation layer for the joyn FF OS client. Detailed information regarding the joyn FF OS client architecture can be found in Annex D FFOS.

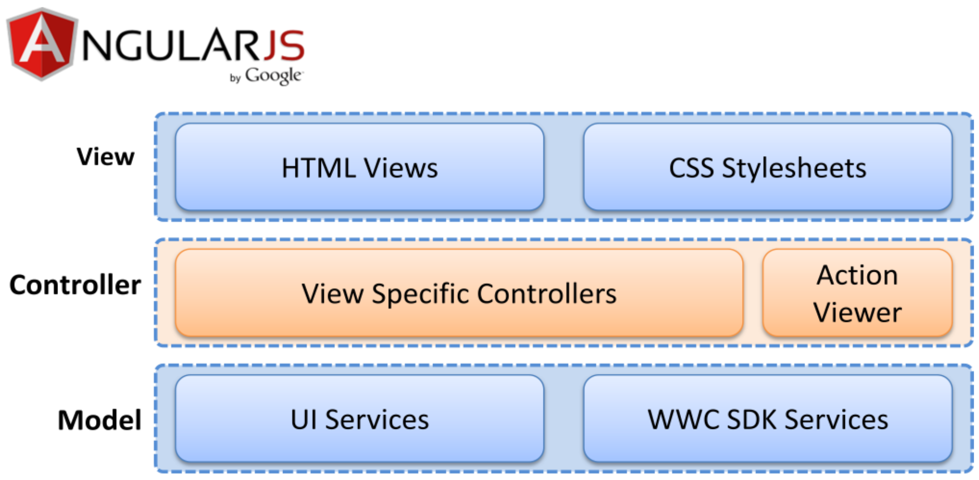


Fig. 17 - Presentation layer high level Architecture

View Component

The View Component includes the several partial HTML files that compose the joyn FF OS application such as the Contact List, Chat window, Chat List and Contact Details Screens and its respective Cascading Style Sheets (CSS) used for appropriate styling.

Controller Component

The Controller Component refers to the JavaScript code responsive for the application logic, each view as an associated controller. The Action viewer controller is the global application controller and is in charge of navigation across the various application views.

Model Component

The model component is the UI service layer. Services are always in execution and concern data (Model) used across different controllers. There are two types of services. Services used for interaction with the WWC SDK and are responsible for managing events for various chat and file transfer sessions and UI services used for storing information related to visual components such as the number of chats with unread messages.

5. Results Obtained

5.1. WIT Web Communicator WebRTC Client

The work on the prototype had two principal flows, the development of the JavaScript SDK (client) and WCAS Socket Lib Library (server).

WWC WebRTC JavaScript SDK (Client)

Currently offers support for Voice and Video Calls between WebRTC clients and Voice Calls for PSTN devices. SMS messages and 1-1 chat with typing and delivery notifications are also available. File Transfers were also implemented.

Socket Lib WCAS Library (Server)

Currently offers support for address book operations, session creation and termination, capabilities discovery information, WebRTC voice calls, chat 1-1, SMS and file transfer.

Bellow we have some screenshots that describe the current look of the WWC WebRTC client used for demonstrational purposes. This is the look that was presented at the MWC last February.

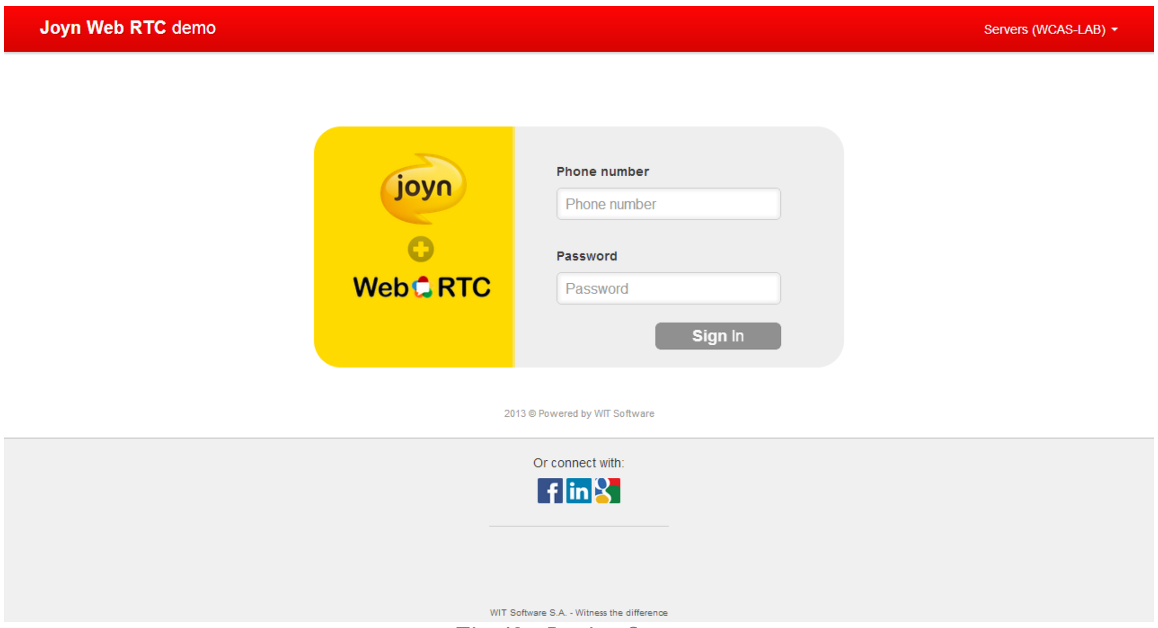
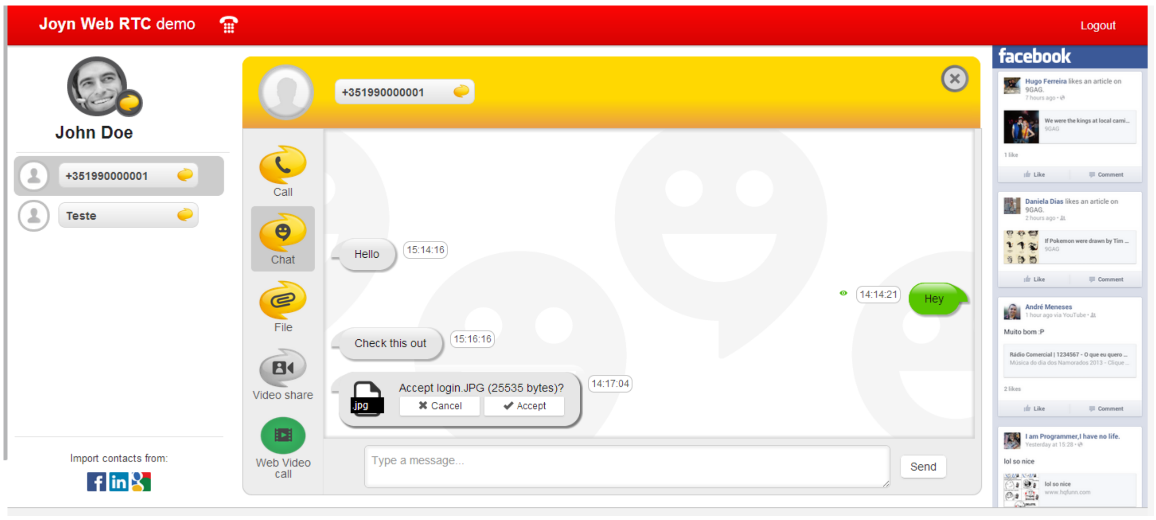


Fig. 18 – Loging Screen



Hello 15:14:16

14:14:21 Hey

Check this out 15:16:16

Accept login.JPG (25535 bytes)? 14:17:04
✖ Cancel ✔ Accept

Fig. 19 – 1-1 Chat Conversation

5.1.1. Tests

In order to develop the new joyn client for Firefox OS the load and throughput tests planned for this semester for the WWC WebRTC client had to be postponed to after the duration if the internship. The development followed a testing – and – developing methodology and ad hoc tests were conducted as necessary.

5.1.1.1. Browser Interoperability

Bellow we have a summary of the required HTML5 features required by the Wit Web Communicator HTML5+WebRTC web application.

- WebSockets
- Media Capture and Streams (GetUserMedia)
- HTML Audio and Video Elements
- WebRTC (PeerConnection)
- WebGL Binary Data
- HTML5 File API
- Web Workers

More detailed information about how each of these components is needed can be found in section 7 of Annex C Architecture Specification.

Bellow we have a table that summarizes the results of the Browser interoperability tests for the WWC WebRTC client.

API	**Feature**	**Browser Support**				
		Chrome	Firefox	Safari	Opera	IE
Communication	**WebSockets**	6+	8+	5+	12.10+	10+
Local Multimedia	**GetUserMedia**	21+	17+	n/a	12+	n/a
	<audio> element	6+	3.6+	5+	10.60+	9+
	<video> element	6+	3.6+	5+	10.60+	9+
WebRTC	**PeerConnection**	21+	Nightly[3]	n/a	n/a	n/a
WebGL	**ArrayBuffer**	10+	4+	5.1+	11.64+	10+
	uInt8Arrays	10+	4+	5.1+	11.64+	10+
File API	**File Reader API**	6+	3.6+	6+	11.10+	10+
	File Slice	6+	3.6+	6+	11.10+	10+
	Blobs	6+	3.6+	6+	11.10+	10+
Other	**Web Workers**	6+	3.6+	5+	10.60+	9+
Potential Feature Set						
Google Chrome	Voice and Video Calls 1-to-1 Chat Group Chat File Transfer					
Mozilla Firefox	Same as Google Chrome					
Safari	All but Voice and Video over IP calls					
Opera						
Microsoft IE						

Table 8 – WWC WebRTC browser feature requirements

[3] Online available in Firefox Aurora and Nightly builds (first announcement in February 4th 2013)

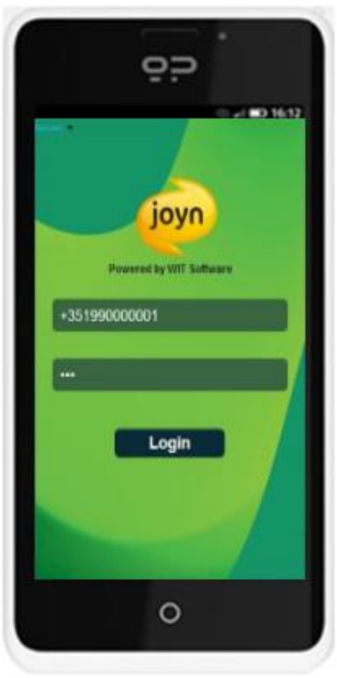
5.2. Joyn for Firefox OS client

The joyn FF OS client consists of a modified branch of WWC WebRTC. This version possesses a fluid interface that is aimed for mobile devices. It also possesses additional functionally that enable the application to access functionally exposed by the Firefox OS device.

Currently the joyn FF OS client offers support for 1-1 Chat with typing and delivery notifications. It also supports address book operations integrated with the device contact list and File Transfer. Detailed feature evaluation can be found in section 7 of Annex D – FFOS.

The WWC JavaScript SDK and Socket Lib Server library are shared between the two projects proving the flexibility and robustness of the implemented solution.

Bellow we have some screenshots that describe the look of the finished functional prototype.



+351990000001

Fig. 20 – Login Screen

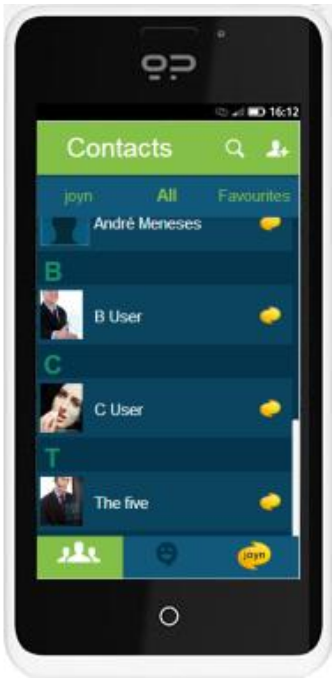
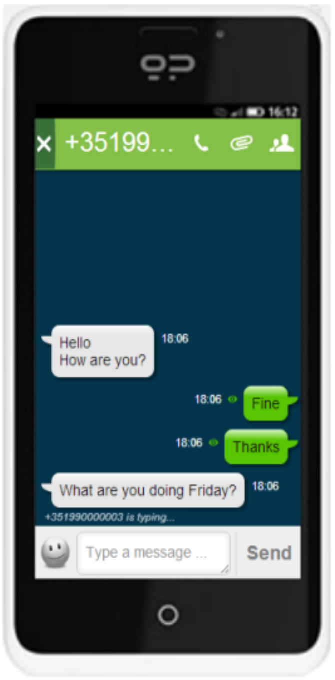


Fig. 21 – Address Book



Hello
How are you? 18:06

18:06 Fine

18:06 Thanks

What are you doing Friday? 18:06

+351990000003 is typing...

Fig. 22 – 1-to-1 Chat Window

5.3. Internship Outcomes

This table briefly summarizes what features where implemented in the two clients that were developed during the course of this Internship.

WIT Web Communicator WebRTC		Joyn Firefox OS Client	
Feature	***Status***	***Feature***	***Status***
Contact management	✓	Native Device Contacts Integration	✓
Chat 1-1	✓	Chat 1-1	✓
SMS Messages	✓	File Transfer	✓
MMS Messages	✗	Application Theme Customization	✓
Voice Calls to PSTN	✓	Client Auto configuration XML	*partial*
Video Calls to other WebRTC Clients	✓		
Video Calls to other Clients	✗		
File Transfer	✓		
Content Share during a Call	✗		
Group Chat	✗		

Table 9 – Overview of the implemented features

Support for MMS, Content Share and Group Chat in the WWC WebRTC was postponed in order to advance the developments of the joyn FF OS client. Group Chat Communication is also being developed in another project that also uses the WWC WebRTC SDK developed during this internship.

Video Calls to other devices were also not possible to implement during this Internship because that would require several modifications in WCAS due to codec incompatibilities between clients (Chrome only encodes video using VP8 [9], most RCS clients, especially mobile clients prefer AVC/H.264 [47] for video).

Support for IP voice and video calls in the joyn for Firefox OS client was not considered during this internship due to the fact that Firefox OS doesn't yet possesses support for WebRTC applications. Calls to the PSTN network are made by invoking the native device dialer application. Support for SMS messages is done by invoking the Firefox OS SMS composer.

6. Future Work

Future developments will now be shifted towards the WWC WebRTC once the tasks of stabilization and demonstration preparation of the joyn for Firefox OS client are finished.

7. Conclusions

This document reflects the work done during the first and second semesters of the internship concerning the development of the WWC WebRTC solution. This solution comprises the client side JavaScript API and the Socket Lib library that is integrated in WCAS.

The original planning suffered some modifications during the first semester due to the need to implement some features earlier for the MWC. During the second semester further changes had to be made in order to further the advances of the new prototype client joyn for Firefox OS

At this point both the WWC WebRTC and the joyn FF OS client are available for demonstration and both already give a good idea of what the finished products will be.

WebRTC is still a bleeding-edge technology and, as such, it is still constantly undergoing changes. Many more changes are expected in this year as major market players will start to pronounce regarding the WebRTC's roadmap. This volatility is one of the risks of this project however as the major browser vendors have started to invest in WebRTC the API should reach a stable state in the coming months. The Architecture proposed for this project was also designed in order to achieve both maximum flexibility and mitigate any major changes to the WebRTC API that could compromise the application.

The benefits of WebRTC can range from interesting to transformational. Business areas such as CRM can benefit greatly from the ease of use provided by WebRTC. For example we have an undecided costumer shopping online. At the distance of a click he can be in touch with a company representative and have his/her doubts cleared increasing customer satisfaction and possibly the company's revenue.

The impact in unified communications solutions will be even more significant, although at a longer term. For existing vendors, the ability to easily support soft clients on a range of devices will increase the overall usability of their UC solution.

Although there are still some challenges regarding WebRTC adoption, mainly security and interoperability concerns, ultimately the value of open browser based real-time communications will drive the industry to overcome them.

The prototype client of WWC WebRTC that was prepared by the end of February was successfully presented at the 2013 edition of Mobile World Congress and was divulged in several technological news websites.

During the second part of the internship developments shifted towards the development of a RCS client for Firefox OS, the flexibility of the architecture of the WWC WebRTC client made possible to finish the joyn FF OS client prototype by the end of the internship with the expected functionality. Once the client is finished and prepared for demonstration it will be showcased to possible clients by the sales team of WIT Software.

The fact that both clients developed during the internship share most of its internal components proves the robustness and flexibility of the approach proposed initially for the application architecture. As a matter of fact the WWC JavaScript SDK that communicates with the WCAS server is already being used in a Corporate Communication demo application and will be further enhanced in the future.

References

[1] Scrum.org, “Improving the Profession of Software Development,” [Online]. Available: http://www.scrum.org/. [Acedido em 16 1 2013].

[2] ISO, “Quality management systems -- Requirements,” 2008. [Online]. Available: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber =46486. [Acedido em 16 1 2013].

[3] W. Model, “ADVANTAGES, EXAMPLES, PHASES AND MORE ABOUT SOFTWARE DEVELOPMENT,” [Online]. Available: http://www.waterfall-model.com/. [Acedido em 16 1 2013].

[4] Google inc., “WebRTC,” [Online]. Available: http://www.webrtc.org/. [Acedido em 16 1 2013].

[5] GSMA, “Mobile World Congress 2013,” [Online]. Available: http://www.mobileworldcongress.com/. [Acedido em 16 1 2013].

[6] GSM Association, “RCS-e - Advanced Communications: Services and Client Version 1.2.2,” 04 July 2012.

[7] S. H. Göran AP eriksson, “WebRTC: enhancing the web,” *Ericsson Review,* p. 1, 2012.

[8] Google inc., “WebRTC Architecture,” [Online]. Available: http://www.webrtc.org/reference/architecture. [Acedido em 30 10 2012].

[9] WebMProject, “VP8 Bitstream Specification License,” [Online]. Available: http://www.webmproject.org/license/bitstream/. [Acedido em 6 11 2012].

[10] Xiph.Org, “Opus Interactive Audio Codec,” 2012 12 21. [Online]. Available: http://www.opus-codec.org/.

[11] ITU-T, “Pulse Code Modulation of Frequencies - ITU-T Recomendation G.711,” ITU-T, 1990.

[12] A. D. H. A. e. a. S. Andersen, “ Internet Low Bit Rate Codec (iLBC) - RFC 3951,” 12 2004. [Online]. Available: Internet Low Bit Rate Codec (iLBC).

[13] Google Inc., “WebRTC Components,” 2011. [Online]. Available: http://www.webrtc.org/reference/webrtc-components#TOC-iSAC. [Acedido em 1 7 2013].

[14] E. C. K. N. M. Naslund, “The Secure Real-time Transport Protocol (SRTP) - RFC 3711,” IETF, 2004.

[15] N. M. E. Rescorla, “Datagram Transport Layer Security - RFC 4347,” 4 2006. [Online]. Available: http://tools.ietf.org/html/rfc4347.

[16] A. B. Johnson, WebRTC APIs and RTCWEB Protocols of the HTML5 Real-Time

Web, Digital Codex LLC, 2012.

[17] W3C, “Media Capture and Streams,” W3C, 28 6 2012. [Online]. Available: http://www.w3.org/TR/mediacapture-streams/.

[18] W3C, “WebRTC 1.0: Real-time Communication Between Browsers,” 21 8 2012. [Online]. Available: http://www.w3.org/TR/webrtc/.

[19] C. Perkins, “SDP: Session Description Protocol - RFC 4566,” IETF, 2006.

[20] C. O. B. H. R. L. L. P. G. L. Neil Webber, “The Information and Content Exchange (ICE) Protocol,” 26 10 1998. [Online]. Available: http://www.w3.org/TR/NOTE-ice.

[21] GSMA, “RCS API Detailed Requirements,” 9 11 2012. [Online]. Available: http://www.gsma.com/rcs/wp-content/uploads/2012/10/RCS_API_requirements_v2_2.pdf. [Acedido em 28 1 2013].

[22] R. Nyman, “Full WebRTC support is soon coming to a web browser near you!,” Hacks.Mozilla.Org, 11 9 2012. [Online]. Available: https://hacks.mozilla.org/2012/09/full-webrtc-support-is-soon-coming-to-a-web-browser-near-you/.

[23] Microsoft, “HTML5 Labs,” [Online]. Available: http://html5labs.interoperabilitybridges.com/prototypes/media-capture-api/media-capture-api/info. [Acedido em 2 11 2012].

[24] Ericsson, “Experimental WebRTC mobile browser released by Ericsson,” 19 10 2012. [Online]. Available: http://www.ericsson.com/news/121019-experimental-webrtc-mobile-browser-released-by-ericsson_244159017_c.

[25] frisB, “frisb.com,” 2012. [Online]. Available: http://www.frisb.com/.

[26] Voxeo Labs, “PhonoSDK WebRTC,” 2012. [Online]. Available: http://phono.com/webrtc.

[27] Doubango, “Doubango Telecom,” [Online]. Available: http://www.doubango.org/. [Acedido em 7 11 2012].

[28] Doubango Telecom, “World's first HTML5 SIP client,” 2012. [Online]. Available: http://www.sipml5.org/.

[29] Twelephone, 2012. [Online]. Available: http://twelephone.com/.

[30] OpenTok, [Online]. Available: http://www.tokbox.com/opentok/api/documentation/v2. [Acedido em 8 11 2012].

[31] thrupoint, 2012. [Online]. Available: http://www.thrupoint.com/solutions/webrtc-sip.html.

[32] Utribo, “Utribo Connect,” [Online]. Available: http://www.utribo.com/. [Acedido em 8 11 2012].

[33] Zingaya, 2012. [Online]. Available: http://zingaya.com/.

[34] TenHands, "Free, Easy Video Collaboration for Business," 2012. [Online]. Available: https://www.tenhands.net/Home.htm.

[35] Bistri, "Bistri," [Online]. Available: https://bistri.com/. [Acedido em 8 11 2012].

[36] Github, "Kirskowal Q," [Online]. Available: https://github.com/kriskowal/q. [Acedido em 18 1 2013].

[37] Google, "HTML enhanced for web apps!," [Online]. Available: http://angularjs.org/. [Acedido em 18 1 2013].

[38] jWebSocket, "Boosting Web Communication," [Online]. Available: http://jwebsocket.org/. [Acedido em 28 12 2012].

[39] Atmosphere, "The only Portable WebSocket/Comet Framework supporting Scala, Groovy and Java," [Online]. Available: https://github.com/Atmosphere/atmosphere. [Acedido em 28 12 2012].

[40] Migratory Data, "MigratoryData WebSocket Server," [Online]. Available: http://migratorydata.com/migratorydata-websocket-server.html. [Acedido em 28 12 2012].

[41] Bristleback Server, "A fast, high level WebSocket Server powered by Spring Framework," [Online]. Available: http://bristleback.pl/. [Acedido em 28 12 2012].

[42] B. Alman, "Grunt," [Online]. Available: https://github.com/gruntjs/grunt/tree/0.3-stable. [Acedido em 18 1 2013].

[43] Mozilla, "Firefox OS Architecture," 22 5 2013. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Platform/Architecture.

[44] Solaiemes, "joyn for Firefox OS available as HTML5 app using Solaiemes RCS network API," 14 1 2013. [Online]. Available: http://blog.solaiemes.com/2013/01/joyn-for-firefox-os-available-as-html5.html.

[45] "SOAP specification," W3C, 27 4 2007. [Online]. Available: http://www.w3.org/TR/soap/.

[46] "Representational State Transfer," Wikipédia, [Online]. Available: http://pt.wikipedia.org/wiki/REST. [Acedido em 24 6 2013].

[47] MPEGLA, "AVC/H.264 License Agreement," MPEG LA, LLC , 2009. [Online]. Available: http://www.mpegla.com/main/programs/AVC/Pages/AgreementExpress.aspx.

[48] "RFC 4975," 9 2007. [Online]. Available: http://tools.ietf.org/html/rfc4975.

[49] GSM Association, "Rich Communication Suite 5.1 Advanced Communications Services and Client Specification," GSMA, 13 August 2012.

[50] "Gaia is a HTML5-based Phone UI for the Boot 2 Gecko," [Online]. Available: https://github.com/mozilla-b2g/gaia. [Acedido em 23 06 2013].

[51] GSMA, Rich Communication Suite Release 2 Service, 14 February 2011.

[52] "Firefox OS Web API," [Online]. Available: https://wiki.mozilla.org/WebAPI. [Acedido em 24 6 2013].

[53] "B2G - Milestone 5 Plan," [Online]. Available: https://docs.google.com/spreadsheet/ccc?key=0AiBigu584YY7dGlNSlY0QzhJb3M5anRBa1gxalV0Y3c#gid=0. [Acedido em 24 6 2013].

[54] Mozilla, "Firefox OS Simulator 3.0.1," [Online]. Available: https://addons.mozilla.org/pt-pt/firefox/addon/firefox-os-simulator/. [Acedido em 24 6 2013].

[55] IETF, "Javascript Session Establishment Protocol," 22 10 2012. [Online]. Available: http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-02. [Acedido em 18 1 2013].

[56] IETF, "RTCWeb Offer/Answer Protocol (ROAP)," 30 10 2011. [Online]. Available: http://tools.ietf.org/html/draft-jennings-rtcweb-signaling-01. [Acedido em 18 1 2013].

[57] W3C, "Media Capture and Streams," 28 6 2012. [Online]. Available: http://www.w3.org/TR/mediacapture-streams/. [Acedido em 18 1 2012].

[58] W3C, "WebRTC 1.0: Real-time Communication Between Browsers," 21 8 2012. [Online]. Available: http://www.w3.org/TR/webrtc/. [Acedido em 21 1 2013].