MSc in Informatics Engineering
Internship
Final Report

MobicarInfo

André Lourenço Santos

andrels@student.dei.uc.pt

DEI Supervisor:
Tiago Baptista

Critical Software Supervisor:
Tiago Ferreira

Date: 12 of July 2012

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

Abstract

The MobicarInfo is an integrated in-car entertainment and information system that will be deployed in the Mobicar vehicle. The MobicarInfo will provide entertainment, navigation, audio and video, location-based services, connectivity to mobile devices and external communications.

This system will communicate with the Internet using GSM and Wi-Fi and with several of the car components through the CAN bus. The MobicarInfo GUI and Services will be integrated with the Visteon IVI platform and the recommended Linux distribution.

The goal for this internship was to develop some modules of the MobicarInfo. Given its complexity, the MobicarInfo needs to be an extensible system, thus allowing new functionalities to be integrated over time. The list of functionalities to be implemented include mobile synchronization, charge control, vehicle-to-grid, remote control and social networks interaction.

Keywords

“Electric Vehicle”, “Infotainment”, ”Mobicar”, “Entertainment”.

Index

Index of Figures

Index of Tables

Acronyms

Throughout this document, the following acronyms are presented:

Acronym	Description
AC	Air Conditioner
AD	Applicable Document
AMP	Asymmetrical multiprocessing
API	Application Programming Interface
BSP	Board Support Packages
CAN	Controller-area Network
CSW	Critical Software, S.A.
CVS	Concurrent Version System
DLNA	Digital Living Network Alliance
EAP	Embedded Automotive Platform
EV	Electric Vehicle
GDI	Windows Embedded Compact standard Graphics Device Interface
GENIVI	Concatenation of Geneva and IVI
GPS	Global Positioning System
GUI	Graphical user interface
IM	Instant Messaging
ISV	Independent Software Vendor
IVI	In-Vehicle Infotainment
HMI	Human Machine Interface
JS	Javascript
JSON	Javascript Object Notation
LBS	Location-based System
MOST	Media Oriented Systems Transport
OEM	Original Equipment Manufacturers
OS	Operative System
PDK	Platform Development Kit
RD	Reference Document
RFID	Radio-frequency identification

Acronym	Description
RT	Real Time
RTOS	Real Time Operating System
SMP	Symmetrical multiprocessing
SRS	Software Requirements Specification
TOT	Time on Task
UI	User Interface
USB	Universal Serial Bus
V2G	Vehicle to Grid
VM	Virtual Machine
YRS	System Requirements Specification

Table 1: Acronyms

1 Introduction

This document reflects the work done during the first and second semester of the academic year 2011/2012 in the Internship of the MSc of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra. The Internship took place in the company Critical Software, S.A.[1], Coimbra Office, specialized in creating and deploying software solutions that guarantee support for key operational functions, and underwent the supervision of Tiago Baptista (DEI) and Tiago Ferreira (Critical Software).

Car's infotainment systems are becoming more and more complex, the need for only the speedometer and an AM/FM radio is long gone. Now the customer expects the car to be connected to all the systems he uses in his daily life, from checking the schedule for the week to planning the route to the closest hotel, or even checking what his friends are saying and doing in any of the social networks.

While making the car a more integrated information system may seem a great idea, there are some dangers that can't be overlooked. First, the driver distraction may increase with the growth in the number of functionalities. Second, connecting the car to the internet may create security issues, allowing anyone with the right knowledge to exploit vulnerabilities. Finally, we can't forget the bugs that may occur at the worst possible time.

Due to this new demand the automotive industry is changing and the days of proprietary single-vendor solutions are coming to an end, since the costs involved and the system complexity is exponentially increasing. Collaboration, open software, interoperable technologies, and connected deployment platforms are the new imperatives.

The MobicarInfo goal is to create a cost-effective infotainment system that incorporates all the core functionalities of the new generation infotainment systems, such as AM/FM radio, music player, video player, GPS, Bluetooth pairing, interaction with social networks and add the required mechanisms to interact with the Mobi.E[2] and the One.Stop.Transport mobility platforms. The Mobi.E services provide a way to locate charging stations and reserve them, while with the One.Stop.Transport the driver can plan a trip using several means of transportation, use bike sharing services and obtain several points of interest.

During the first semester a state of the art study in the subject of infotainment systems was performed, identifying the main solutions currently available and finding similar approaches to the MobicarInfo. Finally, the specification and conception of the functionalities to be developed was accomplished with a requirements analysis for the modules of the infotainment system and with the development of prototypes when needed.

During the second half of the internship, the different modules were coded. This development phase was accompanied with validation activities like testing. Following this phase there was a validation specification, where usability and performance tests were created. The modules were tested and the defects found were fixed. Finally, all the modules were integrated in the Mobicar.

[1] Critical web site - http://www.criticalsoftware.com/

[2] MOBI.E web site - http://www.mobie.pt

Deviations on the initial planning for the second semester caused for the Synchronize module development to be reprioritized, and only the GUI was developed, and the service interactions were simulated.

The document starts with the motivations for this work, addressing the nature of the problems which led to the idealization of this project, its goals, and the approach taken in order to fulfil them. Section 2 presents the overview on the available infotainment systems. This overview includes a bit of what is expected of the new infotainment systems, followed by a study on available solutions and a comparison between the available infotainment systems. Section 3 presents the planning for both semesters, as well as the software development methodology adopted throughout the internship. Section 4 presents the proposed approach for the system. Section 5 presents a risk analysis. Section 6 presents the achievements. Section 7 presents the tests executed. Section 8 presents a retrospective on the work done and future work.

Besides this document, other documents have been created: detailed information about the State of the Art analysis (Annex A), the Software Requirements (Annex B and C), the Usability Test Plan and Results (Annex D and E), the Software Architecture Specification (Annex F and G), the Soap VS Rest Study (Annex H), the Interface Specification (Annex I), the Test Specification (Annex J) and the Test Results (Annex K).

2 State of the Art

This section provides an insight over the available solutions and technologies used for the development of state of the art infotainment systems, and compares them to find the most well-balanced solution.

First we will describe GENIVI [1], an alliance whose mission embodies what is expected of the new generation of infotainment systems, later we will go through the main Operating Systems and Kernel these types of platforms are built on top of. Afterwards, the main solutions available will be described and a comparison between them will be made.

Annex A has a more detailed description about the State of the Art in Infotainment Systems.

2.1 GENIVI

Due to the changes in the automotive industry, an alliance was formed, in 2009, called GENIVI. This marked a new era of cooperation among automakers, suppliers and technology providers.

The GENIVI Alliance is a non-profit organization committed to driving the development and broad adoption of an open source In-Vehicle Infotainment (IVI) reference platform. The purpose of this alliance is to unite industry-leading automotive, consumer electronics, communications and application development companies investing in the IVI market and driving innovation.

The GENIVI mission is to drive the broad adoption of an open source development platform by aligning automotive OEM requirements and delivering specifications, reference implementations and certification programs that form a consistent basis for further open source and ISV development.

GENIVI provides its members with a Linux-based open-source IVI platform and with all the insight needed, as well as a say in further development of its specification, reference architectures, and implementation guidelines. [3]

The GENIVI strategic initiatives are:

- Accelerate the development and implementation of the fully connected vehicle for infotainment applications
- Deliver a platform consisting of standardized middleware, application layer interfaces and frameworks
- Extend open source community innovations to support the automotive domain
- Engage developers to deliver compliant implementations

[3] http://www.genivi.org/why-become-genivi-member

- Sponsor technical, marketing and compliance programs

Currently several platforms are built following the GENIVI way, and they will be covered later.

2.2 Operating Systems and Kernel

This sections goes over the different OS (and Kernel) used in the development of infotainment solutions.

2.2.1 VxWorks

VxWorks [2] is a 32-bit and 64-bit RTOS developed by Wind River, with numerous BSPs and multi-core support, running in SMP or AMP mode. It provides networking solutions for a large variety of embedded systems communications, from device to device via USB, and core to core in an AMP multi-core system with MIPC.

VxWorks RTOS and Wind River Media Library provide an embedded graphics solution. With Wind River Tilcon Graphics Suite, developers have a complete GUI development system for next-generation devices.

Also VxWorks is the first RTOS to be certified under Wurldtech's Achilles certification program[4], an internationally recognized standard for industrial cyber-security.

2.2.2 Linux

Linux [5] is an open source Kernel with hundreds of developers contributing to its development. The core technology is mature and constantly evolving.

Linux has proven to be secure and reliable. Also trusted vendors have added significant value with commercial-grade product offerings.

It has support for Intel IA-64, X86, ARM, ARM V7, MIPS, PowerPC, and SPARC architectures.

Due to being open source, secure and reliable, there are some Linux distributions that are GENIVI compliant, two of which are the Ubuntu IVI Remix and the MeeGo IVI Project.

2.2.3 Ubuntu IVI Remix

The Ubuntu IVI Remix [6] is based on the Ubuntu Core, a sub-set of Ubuntu technologies suited for embedded devices. It supports Intel X86 and ARM-based microprocessors. Using this OS you also get a cloud service, that enables you to access data from anywhere in the world, called Ubuntu One.

2.2.4 Meego IVI Project

[4] Wurldtech's Achilles certification program is designed to assess the network robustness of industrial devices and certify that they meet a formal and comprehensive set of requirements and conformance

The MeeGo IVI Project [7] was being developed with connected mobile devices in mind, therefore it provides fast boot, power efficiency, small footprint, networking and telephony stacks.

In this OS you have Multi-zone Audio (multiple concurrent audio streams), speech synthesis/recognition and an API to access all the services provided, with a discovery mechanism.

It supports X86 and ARM architectures.

2.2.5 Windows Embedded Compact 7

Windows® Embedded Compact 7 [8] is the next generation of Windows Embedded CE[5]. Device manufacturers can use Microsoft's familiar tools to build the next generation of embedded devices with attractive, intuitive user interfaces, real browsing using Internet Explorer® with Flash 10.1, and connections to peripherals, Windows PCs, servers, and networks.

It is a RTOS with a huge list of BSP's. It has support for X86, ARM, ARM V7, MIPS architectures and has SMP support, and you can easily port any existing Windows Embedded CE 6.0 BSPs, hardware and designs to Windows Embedded Compact 7.

In regards to UI it is shipped with Silverlight[6] 3.0-based development framework which supports 3D transformations, Pixel/Shader effects and has multi-touch support. Developers can also use WIN32 and GDI instead of Silverlight.

There is also support for new gestures, the Pan and Flick. Pan gesture is meant for use in list controls, image display and web browsing, while Flick gesture is meant to quickly navigate large lists or to switch between screens on the UI.

2.2.6 Android

Android [13] is a software stack for mobile devices that includes an OS, middleware and key applications. Android relies on Linux Kernel 2.6 core systems, and acts as an abstraction layer between the hardware and the rest of the software stack. There is no RT support yet.

It uses the Dalvik virtual machine. Unlike other Java VMs which are stack-based, this one is register-based, which is optimized for low memory requirements.

It has an integrated browser based on the WebKit[7] engine and in terms of UI, Android provides 2D and 3D custom graphics libraries based on OpenGL ES 1.0 specification[8]. While for data storage it resorts to SQLite.

[5] Windows Embedded CE is a real-time operating system for a wide range of small-footprint consumer and enterprise devices

[6] Silverlight is a development platform for creating Internet applications and media experiences on the web

[7] WebKit is an open source web browser engine

[8] OpenGL ES specification web site - http://www.khronos.org/registry/gles/

There is the usual support for the common audio, video and image formats, and for connectivity there is GSM telephony, Bluetooth, EDGE, 3G and Wi-Fi.

There is a rich development environment including a device emulator, tools for debugging, memory and performance profiling and also a plug-in for the Eclipse IDE.

2.2.7 QNX Neutrino RTOS

QNX Neutrino RTOS [10] is a robust OS that follows a modular architecture. It allows customers to create highly optimized and reliable systems.

The QNX Neutrino RTOS is a microkernel OS, i.e. every driver, protocol stack, filesystem and application runs in the safety of memory-protected user space. Any component can fail and be automatically restarted without compromising other components.

There is also a proven strategy to migrate from single-processing to multi-processing embedded environments.

This OS has support for X86, SH-4, PowerPC, MIPS and ARM processor architectures.

2.3 Infotainment Systems

There are many solutions available for developing infotainment systems. Now we will go through the main platforms, focusing on the features they provide.

2.3.1 Visteon Infotainment Platform

Visteon IVI Platform [10] is a Linux based scalable solution. As such, it rests upon an open architecture and open-source software (to GENIVI standards).

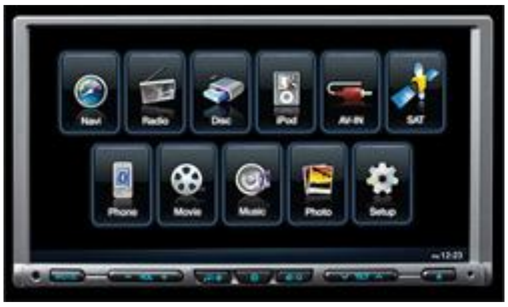


Figure 1: Visteon IVI UI

The platform architecture is divided in three main components:

- Presentation – Manages the GUI
- Services – Provides the data needed by the different GUIs
- VIP – Provides a connection to the hardware

This platform brings to the user, device and internet connectivity. Visteon IVI allows for the remote configuration of the system, and also for software and feature updates. There are also the usual entertainment capabilities of audio and video playback.

Visteon leverages features developed by all open source software GENIVI members, to maximize reuse and robustness, and integrates separate stand-alone modules into one scalable solution. This reduces the vehicle system interface complexity, total cost and engineering cost to maintain once in production.

Visteon is developing a concept car called C-Beyond that uses this platform.

2.3.2 IQon (Saab)

IQon [21] is the work of Saab's innovative spirit. It rests upon Google's Android operating system, taking the open concept to an all new level.

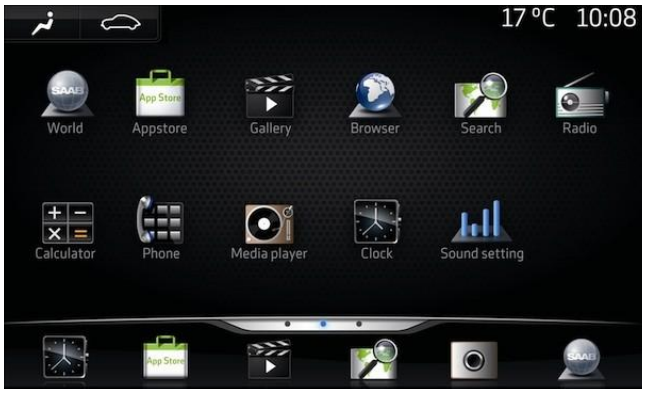


Figure 2: IQon UI

Using Android's OS, IQon inherits its collaborative approach, allowing the end user to download a wide range of applications, online services and multimedia functions provided through a Saab IQon store.

This platform allows for remote communication to and from the car with Saab dealerships, making it possible to upload vehicle data, carry out diagnostics, provide service appointments or even install some in-car options.

Saab also provides an API with more than 500 signals from different sensors in the vehicle (from engine speed to barometric pressure), which allows the Android developer community to use their imagination and ingenuity in order to further tap the potential of this open innovation.

To preserve an high degree of safety and quality, all programs developed must be evaluated and approved by Saab before they are made available to customers (through the Saab IQon store).

2.3.3 QNX Car Application Platform

QNX Car [12] is a pre-integrated software stack with production-proven QNX technologies and dozens of ecosystem partners, and it uses the award-winning QNX Neutrino RTOS. It has many reference implementations, which makes it easier to take to production and reduces the costs. It also provides a re-skinnable HMI.

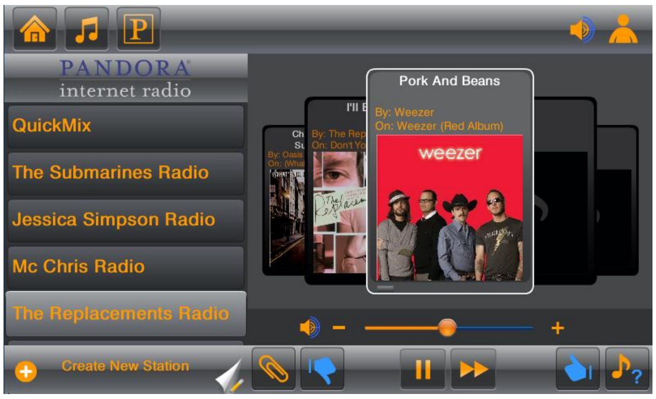


Figure 3: QNX Car Application Platform UI

Like the other platforms here we also have Internet connectivity, which allows for on-demand download (or streaming), Internet radio, on-demand music stores, online gaming, social networking, dynamically updateable navigation, location-based services and GPS augmented by Google Maps.

The HMI framework offers ActionScript extensions for direct access to native C code, an application launcher and also window management. Since its underlying HMI framework is based on Adobe Flash technology, it supports any native Flash application.

Companies who join QNX CAR can participate in joint QNX CAR marketing and sales initiatives, and they can contribute products to the QNX CAR reference implementations. As part of these reference implementations, their products will go out to all QNX CAR automotive tier-one and OEM members.

A proof of concept, the LTE Connected Car[9], was created in under four months, to illustrate how fast and straightforward it is to build an in-vehicle infotainment system using QNX technology.

There are already some car makers that have adopted this platform to develop their infotainment systems. Toyota has developed a platform called Toyota Entune, which will be introduced on most Toyota vehicles over time[10].

2.3.4 Windows Embedded Automotive Platform

Windows Embedded Automotive 7 [9] delivers a large set of integrated and flexible middleware components. Windows Embedded Automotive 7 provides Silverlight for Windows Embedded, an extensible UI framework, which enables seamless integration between designer, developer and reviewer.

[9] LTE Connected Car web site - http://www.ngconnect.org/program/connected-car.htm

[10] Toyota Entune powered vehicles web site - http://www.toyota.com/entune/what-is-entune/vehicle-availability.html

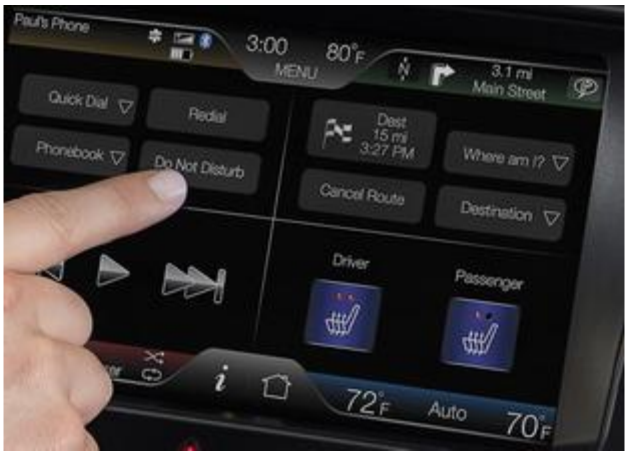


Figure 4: Ford's SYNC GUI

This embedded platform, brings the latest support for multi core architectures, hands free phone control including address book and calendar download with secure simple pairing, and SMS Reply by voice. Reply to text messages using voice controls where the system matches the drivers reply to stored messages. It also has support for media devices as also device updates to car makers so that the platform always works with the latest devices.

The development platform is a hardware implementation of all Windows Embedded Automotive 7 features, which helps to facilitate rapid prototyping. BSPs and drivers are available in the Windows Embedded Automotive 7 PDK and through hardware suppliers. A rich set of middleware and services, including a Bluetooth wireless technology stack, with automotive relevant profiles, and phone, radio, and media modules are also available.

The Automotive System Tools support the stable integration of advanced, high-performance systems. They include improved test modules and easy-to-use product engineering guidelines to help simplify the development process and increase reliability.

These include Windows Internet Explorer and Windows Media technology, required for the development of an automotive multimedia system.

There are many car makers that have adopted this platform to develop their infotainment systems. Ford has developed Ford SYNC[11], and will be available in the 2012 versions of most of Ford's models[12]. SYNC's capabilities vary from model to model. Kia has also developed an Infotainment system powered by Microsoft, called UVO. This system was first released in Optima Hybrid 2011 version, and will be available on most models in 2012[13]. Fiat has also released an infotainment system called Blue&Me[14], and will be available in many car models from Fiat, Lancia and Alfa Romeo. Nissan, Paccar and Alpine Electronics also partnered with Microsoft.

[11] Ford SYNC web site - http://www.ford.com/technology/sync/

[12] Ford SYNC packages and models web site - http://www.ford.com/technology/sync/configurations/

[13] Kia UVO models web site - http://www.kia.com/#/uvo?pageId=availability

[14] Blue&Me packages and models web site - http://www.blueandme.net/blueandme/index.aspx?brand=fiat&lang=en#/support/disponibilita/

2.3.5 Wind River Linux Platform for Infotainment

The Wind River LPI [3] is built upon VxWorks RTOS, a fully tested and validated Linux distribution based on the latest Linux 2.6 kernel technology. It includes the Eclipse-based Wind River Workbench development suite, access to a specialized professional services team, and 24/7 global support. It has support for program state management, fast boot, 3D graphics, multimedia playback, automotive connectivity, consumer electronics connectivity, and an enhanced systems infrastructure.

The Wind River LPI includes an advanced cross-build system that incorporates a structured framework for managing device software components as independent “layers”. This layered cross-build system reduces complexity and delivers added flexibility.

Developers are able to customize the root file system for memory-constrained devices and accelerated kernel boot time for “instant-on” capabilities. This system can process diverse multimedia applications and multiple networking interfaces. It offers support for specialized automotive device processors from leading providers including Freescale and Intel.

Finally this platform comes with Wind River Workbench development solutions, which delivers increased productivity for reduced cost and improved time-to-market. It is Eclipse-based and has support for multiple target connections and process/task/thread debugging. It has on-chip debugging capabilities for custom board bring-up, application debugging and multi-core support.

There is no information available regarding auto makers developing systems using this platform.

2.3.6 MontaVista Automotive Technology Platform

MontaVista brings its many years of experience in Embedded Linux, and provides a platform with graphics, video, sound, USB, Bluetooth, location based services, security and customizable UI frameworks.

This Linux-based framework integrates many components and applications like web browsers, media players, email clients, connection managers, GPS and more.

MontaVista ATP [22] is tuned for high performance, ultra fast boot and power management.

It runs on ARM and Intel® Atom™ processors.

There is no information available regarding auto makers developing systems using this platform.

2.3.7 Mentor Embedded IVI Base Platform

The Mentor Embedded IVI Base platform [13] provides a GENIVI compliant foundation for in-vehicle infotainment software development. It integrates graphics, communication and multimedia middleware with libraries, system infrastructure, and management components, all this on top of Linux.

This platform simplifies the design process, optimizes the benefits of open source, and assists OEMs and Tier One suppliers[15] in building a fully realized IVI product, with the many development tools Mentor has to offer and its expert services.

There is no information available regarding auto makers developing systems using this platform.

2.4 Available Solutions

This section will provide an overview of some Automakers who are developing solutions using one of the previous platforms.

2.4.1 Ford

Ford has developed an Infotainment system using the Windows Embedded Automotive Platform called Ford Sync.

This solution provides the user with:

- Hands-free calling – You are able to control your phone with voice commands, to answer or to make calls.
- Entertainment – Allows plug in of different media types. Provides different voices commands to control the music player. Voice commands to find a radio station. Bluetooth audio streaming.
- Navigation – Turn-by-turn directions delivers voice directions through the audio system speakers. Voice activated navigation.
- Audible text messaging – The system can read incoming text messages from your connected device. Can add preset text message responses.
- Internet connectivity – Has a hub to connect a USB mobile broadband modem or Smartphone to provide the passengers with Internet.
- Video Player – Watch videos from your digital camera or gaming system.
- Voice/Touch Climate control
- Track Driving Efficiencies – Shows how efficient you're driving and can compare it to your average to see if you are improving.

2.4.2 Kia

Kia has also developed an Infotainment system using the Windows Embedded Automotive Platform called Kia UVO.

This solution provides users with:

[15] Tier One supplier is a company who makes products specifically for OEMs

- Advanced Speech Recognition – Intelligent Microsoft speech technology is trained to the system operator's voice, creating a personal profile and allowing for up to two different voice profiles in various languages.

- Natural Interface Advancements – A full-color, easy-to-use in-dash monitor allows occupants to quickly scroll through media and mobile device content through intuitive voice and touch-screen commands.

- Custom Media Experiences with MyMusic – UVO's "Jukebox" function features a 1GB hard drive for media storage, allowing users to rip music from CDs or an MP3 player into personal MyMusic folders and store up to 250 songs sorted by title and/or artist - all through voice commands. The system can shuffle through an MP3 player or AM/FM and SIRIUS(r) radio stations and instantly identify what's playing all through simple voice commands.

- Rear Backup Camera – When the vehicle is put in reverse, a built-in rear backup camera uses UVO's in-dash display to provide clearer images of the environment behind the car assisting the driver to identify certain objects that otherwise may be difficult to see.

- Ability to Continuously Update Features and Services – Based on a flexible Windows Embedded Auto platform, updates and services can be delivered in a number of ways (over-the-air, over-the-Web) for Kia to continue to provide a superior user experience after the system enters the market.

2.4.3 Toyota

Toyota has developed an Infotainment system using the QNX Car Application Platform for the new Toyota Entune.

The solution provides the users with:

- Apps and Data services – Entune seamlessly integrates popular smartphone applications

- Phone Connectivity – Access the Internet through a Smartphone connection. The system provides Hands-free phone and streaming audio through bluetooth

- Multimedia and Navigation – CD player, SiriusXM satellite radio, HD Radio, iPod connectivity, USB storage connectivity, DVD player, Navigation with voice commands.

2.4.4 Fiat

Fiat has also developed a solution based on the Windows Embedded Automotive Platform.

The solution provides the users with:

- Call – Connect to you Bluetooth devices and control it from the vehicle

- Multimedia - Listen to music from a USB storage device, or through Bluetooth

- Navigation - Navigation with voice commands
- Update – The system can be updated
- Eco Drive – Store consumption data and analyze it in the eco:Drive application
- Live Services – TomTom HD traffic, Weather, GPS location

2.5 Comparison

Previous sections described some of the providers of Automotive Infotainment Systems and their capabilities. This section makes a comparison between them based on that information in order to find the most robust platform with a list of features that fits MobicarInfo needs.

The comparison is based upon the information disclosed by each vendor and if it is natively supported and implemented by the vendor, and not by the possibility of expanding the platform to make that service possible.

The final grade of each platform will be given by the following groups of categories:

- GENIVI Compliant – Follows the standards of the GENIVI Alliance - Total of 5 points
- Basic – Total of 25 points
    - Reception (AM/FM) – Ability to receive radio signals - 5 points
    - Audio – Ability to play audio files - 5 points
    - Video – Ability to play video files - 5 points
    - Navigation – GPS navigation system - 5 points
    - Hands-Free – Ability to do hands-free calls - 5 points
- Basic+ - Total of 4 points
    - Streaming (Radio, Video) – Ability to stream radio or video through the Internet - 2 points
    - Voice Recognition – Ability to parse voice input - 2 points
- Connectivity – Total of 18 points
    - Wi-Fi – Wireless capabilities - 5 points
    - GSM / WCDMA/ 3GSM – Contains support to one of these modules - 5 points
    - WiMAX – Wireless broadband access - 1 point
    - USB – Ability to connect peripherals through USB - 2 points
    - Bluetooth – Bluetooth capabilities - 5 points
- Connectivity+ - 4 points
    - Off board option configuration – Ability to remotely configure different options of the car - 2 points
- Plus – 8 points
    - Multi-touch – Ability to sense two or more points of contact with the surface - 2 points
    - Updates – Ability to update different applications of the system - 2 points

- Community Applications – Ability to install applications made by the community not the vendor - 2 points
- Customer Support – Support provided by the vendors - 2 points

Now that we have a way to grade each of the platforms, we will compare them and see how they fare against each other, in Table 2.

	Visteon	IQon	QNX	Windows EAP	Wind River	MontaVista ATP	Mentor Embedded IVI
GENIVI Compliant	**0**	**0**	**0**	**0**	**5**	**5**	**5**
1. Multimedia	15	15	15	15	15	15	15
2. Navigation	5	5	5	5	5	5	5
3. Hands-Free	5	0	5	5	5	5	5
Basic Total	**25**	**20**	**25**	**25**	**25**	**25**	**25**
4. Internet Streaming	2	2	2	2	0	0	0
5. Voice Recognition	2	0	0	2	2	0	2
Basic+ Total	**4**	**2**	**2**	**4**	**2**	**0**	**2**
6. Connectivity	10	5	5	10	11	5	11
7. USB	2	0	2	2	2	2	2
8. Bluetooth	5	0	5	5	5	5	5
Connectivity Total	**17**	**5**	**12**	**17**	**18**	**12**	**18**
9. Off Board option configuration	2	2	0	0	0	0	0
Connectivity+ Total	**2**	**2**	**0**	**0**	**0**	**0**	**0**
10. Multi-touch	0	0	0	2	0	0	0
11. Community Applications	0	2	0	0	0	0	0
12. Updates	2	2	2	2	0	0	0
13. Customer Support	0	0	2	2	2	0	2
Plus Total	**2**	**4**	**4**	**6**	**2**	**0**	**2**
Total	**50**	**33**	**43**	**52**	**52**	**42**	**52**

Table 2: Comparison between platforms

From the final score we can identify the existence of good solutions for the development of infotainment systems.

Some of them are GENIVI compliant, and most of them follow an open source approach in order to speed up the development of cutting edge solutions with much less associated cost, while others define their own way of doing things, resorting only to their know-how.

Each of the different platforms has different key features, but from the sum of all categories, we can see four main balanced contestants, the Visteon IVI Platform, Windows Embedded Automotive Platform, Wind River and Mentor Embedded IVI.

The choice to develop the Mobicar Infotainment will fall upon the Visteon IVI Platform. This forces the use of a Linux-based Operative system, and the services stack provided.

3 Project Planning

This section refers to the project planning for the MobicarInfo internship. First we will see the software development process used in CSW. Afterwards we will go through the tasks planned for the duration of the internship.

3.1 Software Development Process

The approach to software life cycle phases followed by CSW is inspired in ECSS [25] and it is compliant with the abstract life cycle definition described in [24]. Figure 5 presents the phases considered in the software development process.

Phases are an indicator of the focus of a project in a moment in time. For example, in the beginning of a project it is expected that most of the resources are allocated to requirements definition but it does not mean that no other activities can be going on. Some design activities or even construction ones (validating prototypes, etc.) can be carried out before closing the requirements development.

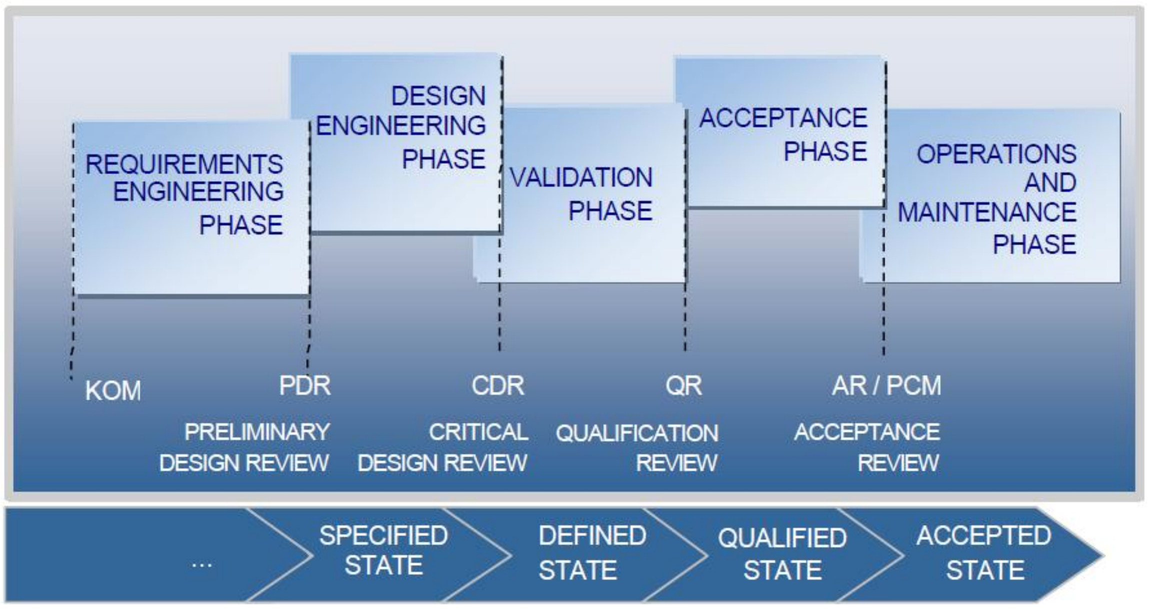


Figure 5: Full software life cycle phases

The features of this life cycle are:

- Requirements Engineering phase consists in the specification of the software to be developed including the software requirements and architectural design (high-level design).

- Design Engineering phase consists of producing the detailed design and source code in parallel allows the design to be generated from source code using reenginerring tools. Producing the unit testing in parallel with the coding allows the errors to be identified and corrected earlier.

- Validation phase purpose is to verify the end-to-end functionality of the system in satisfying all requirements and specifications (mainly system testing).

- Acceptance phase purpose is to demonstrate that the system meets its requirements in the operational environment.

- Operations and Maintenance phase starts after completion of the acceptance review of the software. This phase is very important since the success of software products is dependent from the level of operations that it can provide to the customer organisation.

The MobicarInfo project is a prototyping project. These projects consider a high degree of Research & Development (R&D). Its intention is more focused on creating a proof of concept rather than to build a product (that will be installed in production). It is much more iterative than other type of projects because the project team will be searching for the best choice in terms of project technologies, goals, etc.

These projects may not define a full life cycle depending from the results they have to achieve: GUI prototype, pre-production prototype (includes coding and validation), etc. Figure 6 presents the life cycle phases and milestones for the MobicarInfo project.

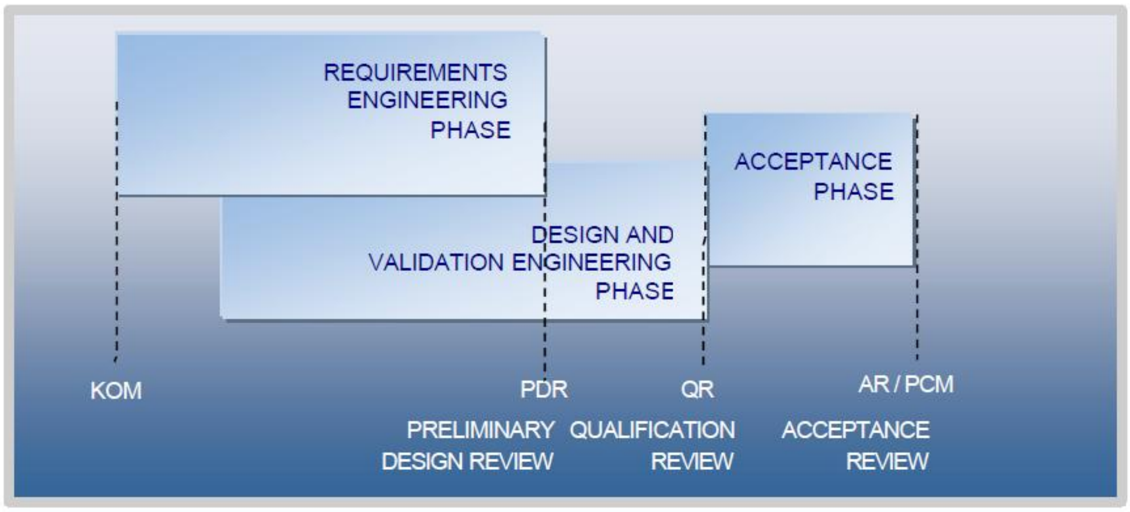

REQUIREMENTS ENGINEERING PHASE
DESIGN AND VALIDATION ENGINEERING PHASE
ACCEPTANCE PHASE
KOM
PDR
QR
AR / PCM
PRELIMINARY DESIGN REVIEW
QUALIFICATION REVIEW
ACCEPTANCE REVIEW

Figure 6: MobicarInfo life cycle

The features of this life cycle are:

- Requirements phase is longer and more iterative than usual because they take longer to consolidate.

- Design and validation starts earlier than usual because requirements need to be designed and validated earlier in order to be verified. These phases are conducted together because the validation of some requirements is performed just after its implementation when other requirements are still in design or implementation.

- The overlap between requirements and design phases is larger to allow the definition, design and validation of requirements in parallel.

- The purpose of the acceptance phase is to demonstrate that the system meets its requirements in the operational environment.

3.1.1 Project Execution Control

There are regular meetings with the Critical supervisor, to monitor project related subjects, such as its execution and possible deviations to the original planning.

3.1.2 Versioning

The project has a centralized repository available for versioning of documents and software. This repository stores all the documentation and code related to the project.

3.2 Planning

This section presents a description and a Gantt chart for each semester, Figure 7 and Figure 8.

3.2.1 First Semester

The first semester had a research component, consisting of the state of the art. This was a basis for the creation of all the documentation for the project, such as software requirements and architecture.

To understand the feasibility of using Ubuntu One[16], Google Calendar[17] and Google Contacts[18], in some of the modules, two simple demos were created.

Afterwards the software requirements were specified for the three modules, Remote Control, Synchronize and Social Networks, that were implemented during the second semester. There was also rapid prototyping for the GUI, in order to better understand the underlying limitations of in-car infotainments.

Besides the specification of software requirements for the three modules, there was a need to create different software requirements for the Remote Control Client and prototypes of its GUI, as well as a Usability Test. There were two versions for this software prototype's, the first one was developed only for Android, the second version was a cross-platform implementation, which was accomplished using Sencha[19] and PhoneGap[20]. The second version of the prototype was developed in order to test the possibility of expanding the support of the application to multiple mobile platforms.

Regarding the architecture, it was initiated during the last month of the first semester and extend to the second semester.

[16] Ubuntu One is a personal cloud system. Web site - http://one.ubuntu.com/

[17] Google Calendar is a free time-management web application

[18] Google Contacts is a contacts management tool

[19] Sencha web site - http://www.sencha.com/

[20] PhoneGap web site - http://phonegap.com/

The original planning suffered some adjustments with the need to fork the project into two, the MobicarInfo and the MobicarInfo Remote Control Client. This meant software specification, architecture definition and rapid prototyping for both, and usability testing for the second.

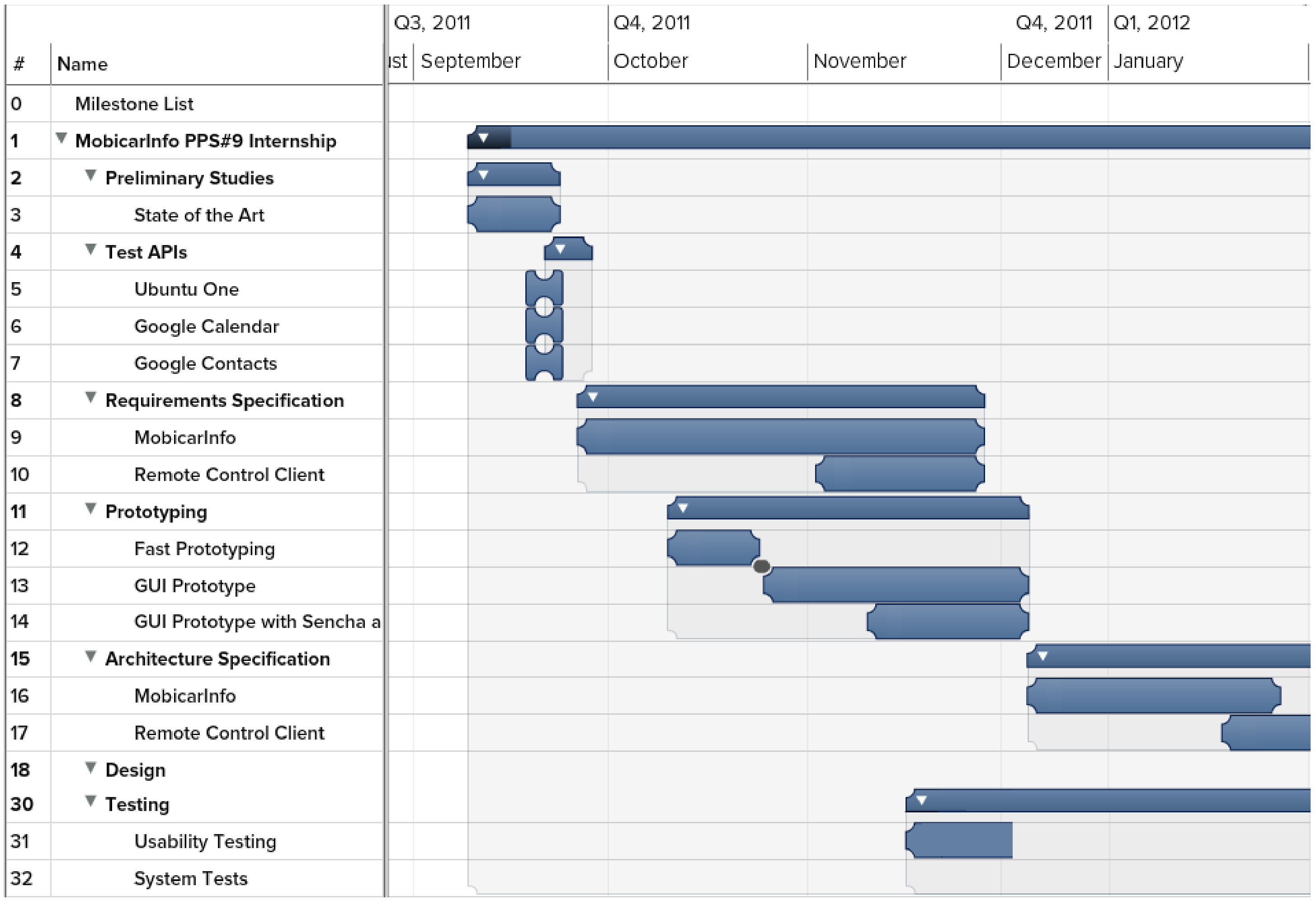
The work done during the First Semester corresponded to the Requirements Engineering Phase.

3.2.2 Second Semester

The goal for the second semester was to finish the specification of the architecture for both the MobicarInfo and MobicarInfo Remote Control Client, and the development of the three modules as well as the remote control client.

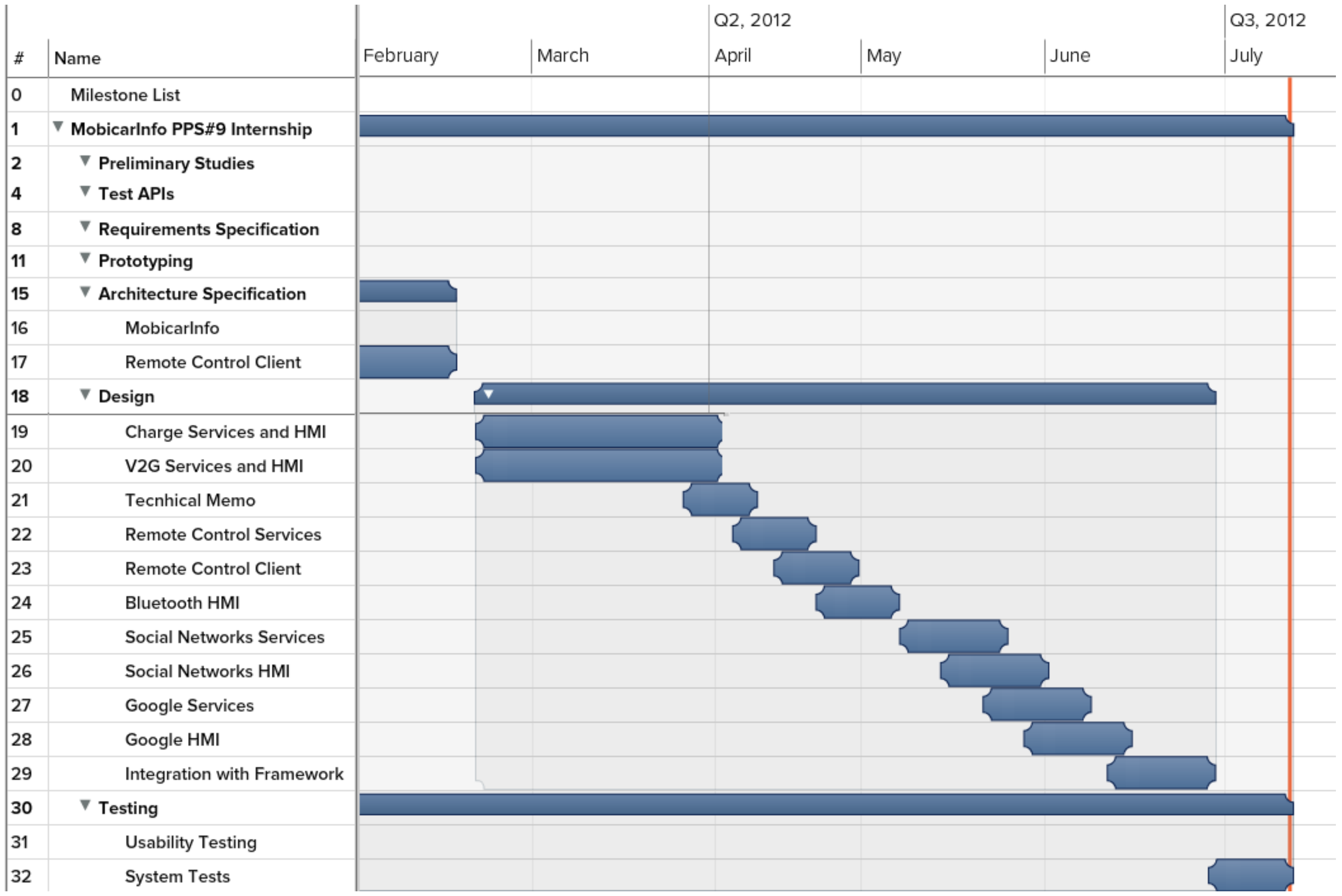
The development phase was accompanied with validation activities and testing. Following this phase there was the validation specification, where usability and performance tests were created. The modules developed were tested and the defects found were fixed. Finally, the coded modules were integrated in the Mobicar.

The work planned for the Second Semester corresponds to the Design and Validation Engineering Phase and Acceptance Phase.



#	Name
0	Milestone List
1	▼ MobicarInfo PPS#9 Internship
2	▼ Preliminary Studies
3	State of the Art
4	▼ Test APIs
5	Ubuntu One
6	Google Calendar
7	Google Contacts
8	▼ Requirements Specification
9	MobicarInfo
10	Remote Control Client
11	▼ Prototyping
12	Fast Prototyping
13	GUI Prototype
14	GUI Prototype with Sencha a
15	▼ Architecture Specification
16	MobicarInfo
17	Remote Control Client
18	▼ Design
30	▼ Testing
31	Usability Testing
32	System Tests

Figure 7: First Semester Plan



#	Name
0	Milestone List
1	▼ **MobicarInfo PPS#9 Internship**
2	▼ **Preliminary Studies**
4	▼ **Test APIs**
8	▼ **Requirements Specification**
11	▼ **Prototyping**
15	▼ **Architecture Specification**
16	MobicarInfo
17	Remote Control Client
18	▼ **Design**
19	Charge Services and HMI
20	V2G Services and HMI
21	Tecnhical Memo
22	Remote Control Services
23	Remote Control Client
24	Bluetooth HMI
25	Social Networks Services
26	Social Networks HMI
27	Google Services
28	Google HMI
29	Integration with Framework
30	▼ **Testing**
31	Usability Testing
32	System Tests

Figure 8: Second Semester Plan

4 Proposed Approach

This section describes the proposed approach for the development of the MobicarInfo.

There will be a listing of all the features of the MobicarInfo, the sub-set of features to be implemented during the internship and the corresponding requirements.

4.1 Features

The MobicarInfo is a complex system that adds more flavour to the current infotainment system. Its features are sub-divided into the following categories:

- Car sharing – allow car sharing between people. This module includes RFID[21] card reading capabilities, checking for black listed cards, unlocking car doors when given a valid card, locking car doors after car sharing use, list of car sharing operations and costs, and pay-per-use of any of the onboard applications

- Entertainment – allow listening to music from the radio, MP3 player or external storage, reproduce video and view photos from external storage. There is also access to the internet, access to social networks, videogames, dashboard widgets and DLNA[22] synchronization

- Fleet Management – provide functionalities to work within a fleet of vehicles, interacting with the fleet management server, and provide remote trip programming

- MobicarInfo – allow controlling and checking different components of the vehicle. These include establishing an internet connection, network configuration, car configuration (excessive speed alerts, AC properties, driving mode, charge, V2G) local and remote, system updates, automatic close doors, Bluetooth synchronization, remote control access, RSS[23], component status, house control (energy consumption monitoring, control some components), route planning, and danger zone detection

- Mobie Interaction – allow checking for and reserving charge stations, as well as checking the energy price and getting a list of the nearby charging stations

- OST Integration – allow getting information about traffic, touristic suggestions, geo-referenced advertisement, information about charging stations, information about public transports, bike sharing, check energy buyers/sellers, and check energy price

- Telemetry – allow storing and sharing telemetry data. These include latitude, longitude, speed, heading, altitude, battery level, distance travelled, energy used and throttle pressure

[21] RFID allows for the automatic identification and tracking of an object, through a tag attached to it

[22] DLNA enables multi-branded products to interoperate and share digital content

[23] RSS is a web feed format used to publish data

- Vehicle Integration – allow controlling the vehicle and getting information about vehicle components, such as to select the driving mode, control the AC, control the windows, display sensor values, control lights, and execute onboard diagnostics.

4.2 Features for the Internship

Due to the duration of the internship a sub-set of features were selected.

The selected features were divided into the following three large modules:

- Social Networks – allow accessing different social networks (Facebook[24], Twitter[25], FourSquare[26]) and publish new content

- Synchronize – allow connecting to and synchronizing with mobile devices as well as to synchronize with Google Contacts and Google Calendar

- Remote Control – allow accessing car functionalities from a mobile phone (V2G, Charge, AC, and Sensor data).

V2G is an external service, provided by the OST. The purpose of the service is for owners to sell or buy energy from one another. The service allows the reservation of a selling or buying operation with another owner and provides the GPS location of that owner, for the transaction to take place.

For each module the background services and the GUI to interact with them will be developed. The Remote Control features require the integration with modules developed by other team members.

4.3 Requirements

Requirements are the basis for the design of the system architecture, since they define the guidelines of the project and what is expected of it.

This section corresponds to the SRS for the three modules described earlier as well as the Remote Control Client application. These requirements were based on the YRS provided by CSW.

A more detailed description can be found in Annex B and C – MobicarInfo Requirements Specification and MobicarInfo Remote Control Requirements Specification.

An overview of the goals for this internship can be seen in Figure 9; these include three modules of the MobicarInfo (Social Networks, Synchronize, and Remote Control) in Red, and the Remote Control Client, in Yellow.

[24] Facebook web site - https://www.facebook.com/

[25] Twitter web site - http://twitter.com/

[26] FourSquare web site - https://foursquare.com/

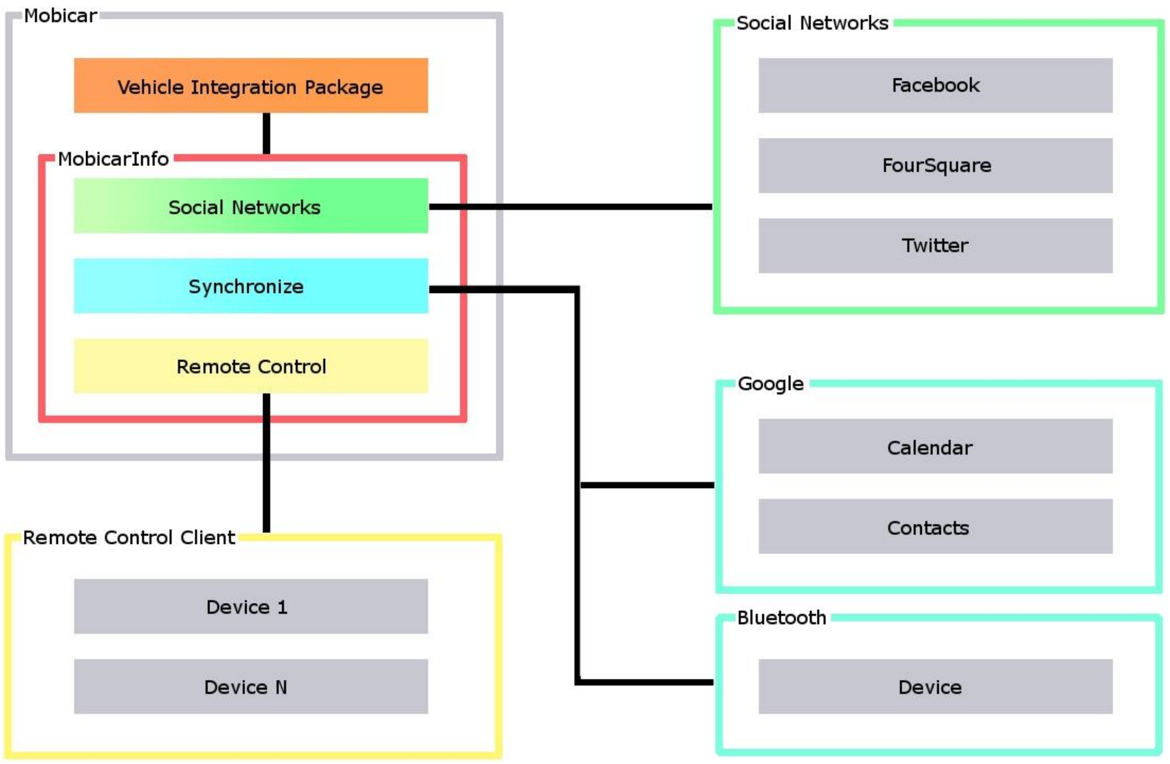


Figure 9: System Environment

4.3.1 MobicarInfo

This section will describe the requirements for each of the modules to be developed during the internship.

General Requirements

- System modularity. The system should be modular, and all modules should have as fewer dependencies as possible. This is to ensure that they can be activated and deactivated as needed.
- Exception handling. The system should handle exceptions that may happen during the execution of the modules.
- Security. Stored information such as user's credentials should be encrypted.

Social Networks

- Configure accounts. The social networks module should allow configuring one account per social network.
- Logout. The social networks module should allow a user to Logout from the service
- Access to different Social Networks. The social networks module should be able to access different social networks and see the information available.

- Publish data. The social networks module should allow publishing new data, such as text messages and photos.

Remote Control

- Authentication. Remote users must authenticate in order to access the functionalities of this module.

- Manage users. The remote control module should only allow managing user accounts from within the vehicle. The module should provide means to create and delete user accounts, as well as changing the user password. The user should also have means to grant and revoke user permissions to the different functionalities provided by this module.

- Change AC State. The remote control module should allow remotely starting or stopping the AC of the vehicle.

- AC Temperature. The remote control module should allow remotely defining the AC temperature of the vehicle, and to get the current AC temperature.

- Change Charge State. The remote control module should allow remotely starting or stopping the current charge.

- Schedule Charge. The remote control module should provide means to remotely schedule a charge for the current day.

- Manage Charge Profiles. The remote control module should provide means to remotely create/update profiles to run at specific times, any day of the week, every week. There should also be provided the ability to delete any existing profile.

- Sell Charge. The remote control module should provide means to remotely sell a percentage of the current charge.

- Monitorization. The remote control module should provide means to remotely check on different components of the vehicle. These include the battery, brakes, powertrain and electrical components.

Synchronize

- Connect Bluetooth module. The synchronize module should connect to the Bluetooth module.

- Synchronize Mobile device. Users should be able to synchronize their mobile devices data with the vehicle, manly the contacts list. This module shall also provide the means to start and stop a call.

- Synchronize with Google. Users should be able to synchronize with their Google Contacts and Google Calendar accounts.

- Logout. Users should be able to logout from the Google service.

4.3.2 Remote Control Client

This section presents the requirements for the Remote Control Client.

General Requirements

- Authentication. Users should authenticate with the vehicle before getting access to the application functionalities.
- Exception handling. The system should handle exceptions that may happen during the execution of the modules.
- Security. Stored information such as user's credentials should be encrypted, and all communications shall be over a secure channel.

Manage AC

- Change AC State. The application should allow starting or stopping the AC of the vehicle.
- AC Temperature. The application should allow defining the current AC temperature of the vehicle, and to get the current AC temperature.

Manage Charge

- Change Charge State. The application should allow starting or stopping the current charge.
- Schedule Charge. The application should provide means to schedule a charge for the current day.
- Manage Charge Profiles. The application should provide means to create/update profiles to run at specific times, any day of the week, every week, and also the ability to delete any existing profile.

Manage V2G

- Sell Charge. The application should allow selling a percentage of the current charge.
- Manage V2G Profiles. The application should provide means to create/update profiles to run when some constraints are met, like the current price for energy and the current percentage of energy available. There should be provided the ability to create profiles both for buying and selling energy. The ability to delete any existing profile should also be provided.

Monitorization

- Component Status. The application should display a list of the most recent status data, separated by type (battery, brakes, powertrain, electrical components).

4.4 High Level Architecture

This section provides a more detailed overview for both the MobicarInfo modules and the Remote Control Client.

The complete specification can be found in Annex F and G – MobicarInfo Software Architecture Specification and MobicarInfo Remote Control Software Architecture Specification.

4.4.1 MobicarInfo

The MobicarInfo is comprised of several modules, and presents itself in a three-layered architecture, with the presentation, processing and data layers separated. The abstraction of all the modules allows future system enchancement regardless of the technologies used on the implementation.

Figure 10 presents the layers and corresponding modules, which will be described next.

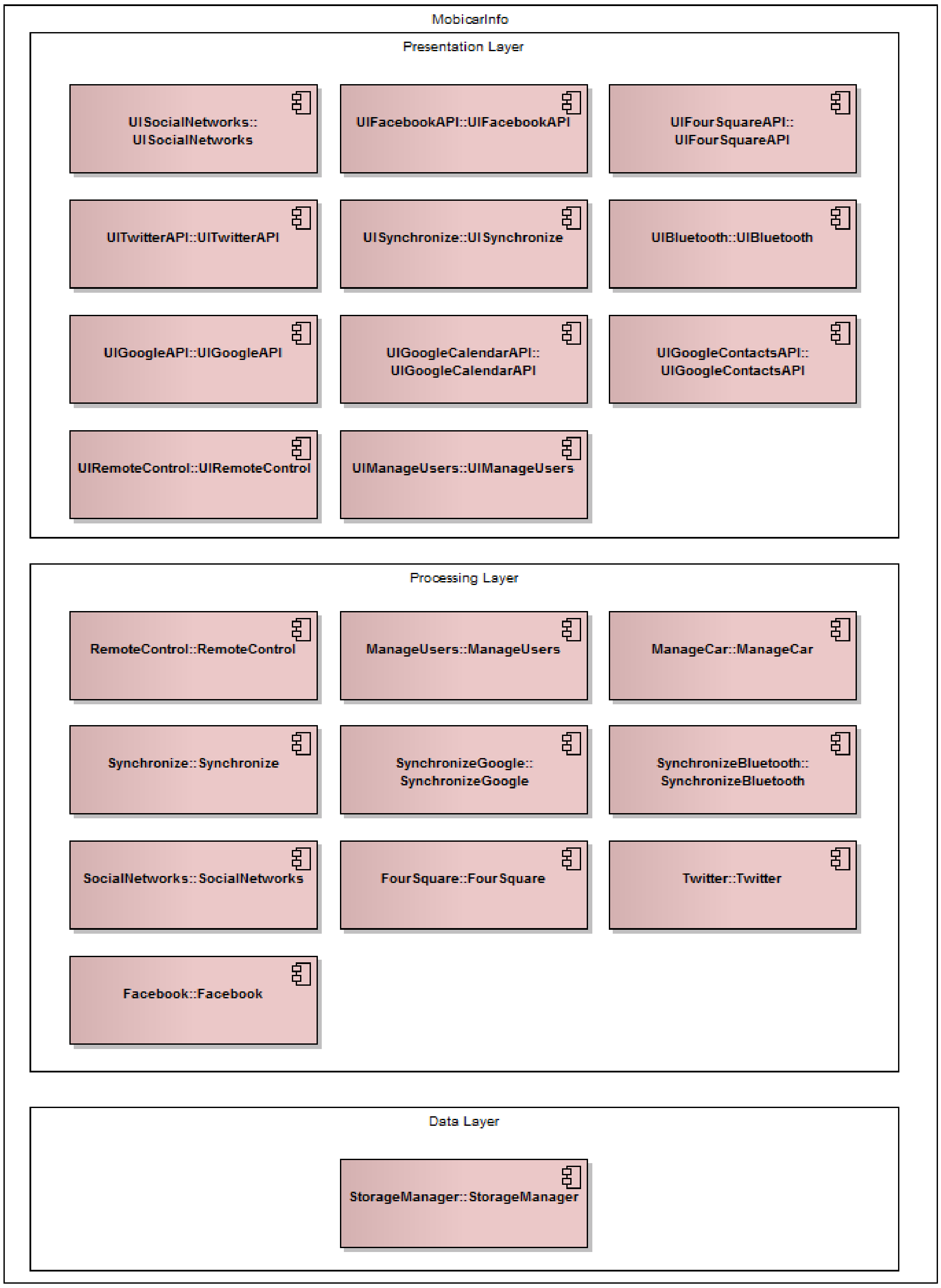

MobicarInfo
Presentation Layer
UISocialNetworks::UISocialNetworks
UIFacebookAPI::UIFacebookAPI
UIFourSquareAPI::UIFourSquareAPI
UITwitterAPI::UITwitterAPI
UISynchronize::UISynchronize
UIBluetooth::UIBluetooth
UIGoogleAPI::UIGoogleAPI
UIGoogleCalendarAPI::UIGoogleCalendarAPI
UIGoogleContactsAPI::UIGoogleContactsAPI
UIRemoteControl::UIRemoteControl
UIManageUsers::UIManageUsers
Processing Layer
RemoteControl::RemoteControl
ManageUsers::ManageUsers
ManageCar::ManageCar
Synchronize::Synchronize
SynchronizeGoogle::SynchronizeGoogle
SynchronizeBluetooth::SynchronizeBluetooth
SocialNetworks::SocialNetworks
FourSquare::FourSquare
Twitter::Twitter
Facebook::Facebook
Data Layer
StorageManager::StorageManager

Figure 10: MobicarInfo modules

4.4.2 Data Layer

This section shall present the data layer modules.

Data modules are responsible for the acquisition or storage of data from/to files.

4.4.2.1 StorageManager

The data storage is handled by this component. All the requests for reading or writing a file by the other components of the application are made through the StorageManager.

4.4.3 Processing Layer

This section shall present the processing layer modules.

Processing modules are responsible for processing data and executing their respective functions.

4.4.3.1 SocialNetworks

This component enables the interaction with social networks. The component is responsible for loading the Facebook, FourSquare and Twitter modules.

4.4.3.2 Facebook

This component enables the interaction with Facebook. This interaction includes authenticating the user, fetching data from Facebook and posting new data. The idea of this component is to act as a proxy between the Facebook Rest API and the GUI.

4.4.3.3 FourSquare

This component enables the interaction with FourSquare. This interaction includes authenticating the user, fetching the user profile, the checkins made by the user and search for places, from FourSquare, as well as submitting checkins. The idea of this component is to act as a proxy between the FourSquare Rest API and the GUI.

4.4.3.4 Twitter

This component enables the interaction with Twitter. This interaction includes authenticating the user, fetching the user profile and the Timeline, from Twitter, as well as submitting new Tweets. The idea of this component is to act as a proxy between the Twitter Rest API and the GUI.

4.4.3.5 Synchronize

This component is responsible for loading the Bluetooth and Google modules.

4.4.3.6 SynchronizeGoogle

This component enables the interaction with Google. This interaction includes authenticating the user, fetching the list of contacts, calendars and events for a given calendar, as well as creating, updating and deleting contacts, calendars and events. The idea of this component is to act as a proxy between the Google Rest API and the GUI.

4.4.3.7 SynchronizeBluetooth

This component enables the interaction with mobile devices, through Bluetooth. This interaction includes discovering devices, pairing with devices, getting the list of contacts and managing calls made to the device.

4.4.3.8 Remote Control

This component is responsible for loading the ManageCar and ManageUsers modules, and configuring the network and managing the user sessions.

4.4.3.9 ManageUsers

This component is responsible for managing the users that can access the vehicle from the Remote Control Client. The module provides ways for creating, updating and deleting users as well as granting and revoking permissions from a registered user.

4.4.3.10 ManageCar

This component enables the management of the vehicle Charge and AC, and provides access to the V2G functionalities. It also provides a way to monitor some of the car components. The idea of this module is to act as proxy between the CAN and Core modules, developed by other members of the team, and the GUI, but at the same time add some extra functionality like creating Charge profiles, or Scheduling a charge from within the MobicarInfo interface. The profiles and schedule are meant to execute automatically whenever all the conditions are verified.

4.4.4 Presentation Layer

This section shall present the presentation layer modules.

Presentation modules are responsible for presenting a user interface that enables interaction with the system.

4.4.4.1 UISocialNetworks

This component provides a GUI to select the service to enter. The services provided are Facebook, FourSquare and Twitter.

4.4.4.2 UIFacebookAPI

This component provides a GUI to interact with Facebook. The interface provides an Authentication, Profile, Albums, Create Album, Photos, Update Photo, Wall, Create Wall Entry, Entry Comments, Create Entry Comment and Logout screens.

4.4.4.3 UIFourSquareAPI

This component provides a GUI to interact with FourSquare. The interface provides an Authentication, Profile, Checkins, Search Places and Logout screens.

4.4.4.4 UITwitterAPI

This component provides a GUI to interact with Twitter. The interface provides an Authentication, Profile, Timeline, Create Tweet and Logout screens.

4.4.4.5 UISynchronize

This component provides a GUI to select the service to enter. The services provided are Bluetooth and Google.

4.4.4.6 UIBluetooth

This component provides a GUI to interact with Bluetooth devices. The interface provides a Discover Devices, Pair Device and Paired Devices screens.

4.4.4.7 UIGoogleAPI

This component provides a GUI to select the service to enter or to Logout from the service. The services provided are Calendar and Contacts.

4.4.4.8 UIGoogleCalendarAPI

This component provides a GUI to interact with the Calendar service. The interface provides a Calendars List, New Calendar, Edit Calendar, Delete Calendar, Events List, New Event, Edit Event and Delete Event screens.

4.4.4.9 UIGoogleContactsAPI

This component provides a GUI to interact with the Contacts service. The interface provides a Contacts List, New Contacts, Edit Contact and Delete Contact screens.

4.4.4.10 UIRemoteControl

This component provides a GUI to access the Manage Users component.

4.4.4.11 UIManageUsers

This component provides a GUI to interact manage users. The interface provides a List Users, Add User, Delete User, Edit User and Edit User Password screens.

4.4.5 Remote Control Client

The Remote Control Client is comprised of several modules, and presents itself in a three-layered architecture, with the presentation, processing and data layers separated. The abstraction of all the modules allows future system enchancement regardless of the technologies used on the implementation.

Figure 11 presents the layers and corresponding modules, which will be described next.

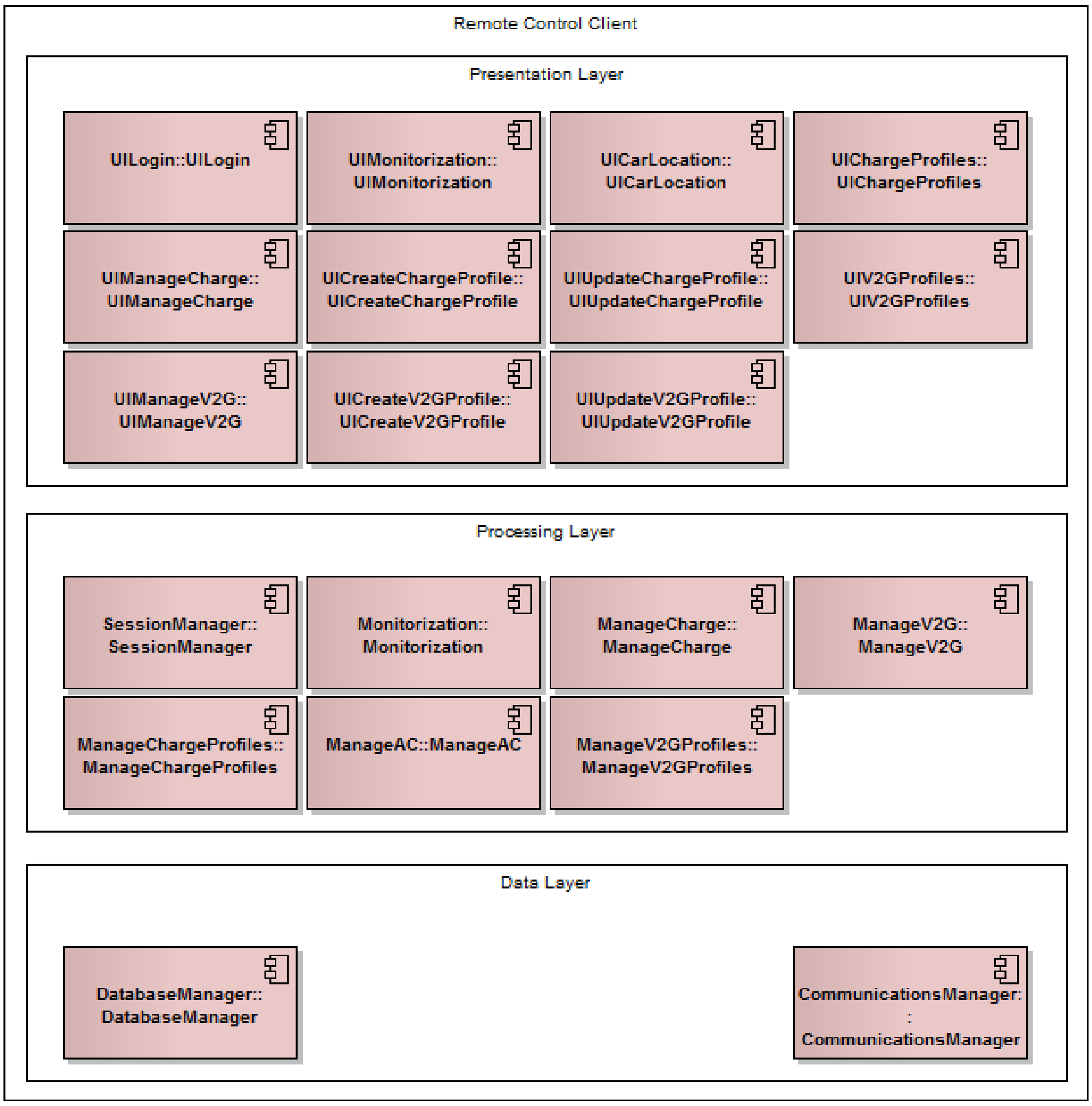


Figure 11: Remote Control Client modules

4.4.6 Data Layer

This section shall present the data layer modules.

Data modules are responsible for the acquisition or storage of data from/to files, and all the communications.

4.4.6.1 DatabaseManager

This component handles all the communications made with the database. All the queries required by the other components of the application are executed through this component.

4.4.6.2 CommunicationsManager

This component handles the communication with the vehicle. All the requests made to the vehicle are executed through this component. This component is responsible for handling the communications errors.

4.4.7 Processing Layer

This section shall present the processing layer modules.

Processing modules are responsible for processing data and executing their respective functions.

4.4.7.1 SessionManager

This component provides the means to authenticate the user, store user sessions and logout a user from the system.

4.4.7.2 Monitorization

This component provides the means to see the status of multiple components of the vehicle.

4.4.7.3 ManageCharge

This component provides the means to Start/Stop a charge and to set the speed of the charge. This component is responsible for starting the ManageChargeProfiles component.

4.4.7.4 ManageChargeProfiles

This component provides the means to create, update and delete charge profiles.

4.4.7.5 ManageV2G

This component provides the means to manage different aspects of the vehicle's V2G functionalities. The ManageV2G component is responsible for starting the ManageV2GProfiles component.

4.4.7.6 ManageV2GProfiles

This component provides the means to create update and delete V2G profiles.

4.4.7.7 ManageAC

This component provides the means to manage the AC of the vehicle.

4.4.8 Presentation Layer

This section shall present the presentation layer modules.

Presentation modules are responsible for presenting a user interface that enables interaction with the system.

4.4.8.1 UILogin

This component provides a GUI to authenticate with the Vehicle. The interface provides a Username and Password fields.

4.4.8.2 UIMonitorization

This component provides a GUI to visualize the monitoring data of the Vehicle.

4.4.8.3 UICarLocation

This component provides a GUI to visualize the Vehicle GPS location.

4.4.8.4 UIChargeProfiles

This component provides a GUI to visualize the Charge profiles that exist in the system.

4.4.8.5 UIManageCharge

This component provides a GUI to manage the vehicles Charge, to select the charge Speed and to Start/Stop a charge.

4.4.8.6 UICreateChargeProfile

This component provides a GUI to create a new charge profile.

4.4.8.7 UIUpdateChargeProfile

This component provides a GUI to update an existing charge profile.

4.4.8.8 UIV2GProfiles

This component provides a GUI to visualize the V2G profiles that exist in the system.

4.4.8.9 UIManageV2G

This component provides a GUI to manage the vehicle V2G operations. Sell or buy energy to or from another car owner.

4.4.8.10 UICreateV2GProfile

This component provides a GUI to create a new V2G profile.

4.4.8.11 UIUpdateV2GProfile

This component provides a GUI to update an existing V2G profile.

4.5 MobicarInfo Interface

The MobicarInfo provides external access to some of the vehicle services, through the RemoteControl component. This section will provide an overview of the Interface specification.

A brief study about the REST and SOAP web services can be found in Annex H – Soap VS REST.

The complete specification can be found in Annex I – MobicarInfo Interfaces Specification.

4.5.1 WSRemoteControl

The RemoteControl component provides a web service following a REST architecture, where we have resources referring to the functionalities we want to expose. A resource or collection of resources is referred as a URI.

In order to manipulate these resources, the standard HTTP methods can be performed:

- GET – Get a resource information or collection of resources,
- PUT – Create a new resource,

- POST – Update an existing resource,
- DELETE – Delete an existing resource.

A URI in REST architecture follows the following structure:

1. http://www.domain.pt/resources/item

Where the "resources" represents a collection of resources, and "item" represents the ID of a single resource.

Table 3 describes all the resources available in the WSRemoteControl and the possible operations for each resource.

Operation	Resource	Description
PUT	sessions	Create a new session
DELETE	sessions	Delete an existing session
GET	users	Get a list of users
GET	users/userId	Get a user profile
POST	users/userId	Update a user information
POST	charge/start	Start a charge
GET	charge/start	Check if a charge can be started
POST	charge/stop	Stop a charge
GET	charge/stop	Check if a charge can be stopped
POST	charge/schedule	Schedule a charge for the current day
GET	charge/schedule	Check if a charge is scheduled
DELETE	charge/scheduled	Cancel a scheduled charge
GET	charge/profiles	Get a list of charge profiles
PUT	charge/profiles	Create a new charge profile
GET	charge/profiles/profileId	Get a charge profile
POST	charge/profiles/profileId	Update a charge profile
DELETE	charge/profiles/profileId	Delete a charge profile
GET	v2g/prices	Get the current charge prices
GET	v2g/sell	Check if it's possible to start selling
POST	v2g/sell	Start selling an amount of energy
GET	v2g/profiles	Get a list of V2G profiles
PUT	v2g/profiles	Create a new V2G profile
GET	v2g/profiles/profileId	Get a V2G profile

Operation	Resource	Description
POST	v2g/profiles/profileId	Update a V2G profile
DELETE	v2g/profiles/profileId	Delete a V2G profile
GET	ac/start	Check if the AC can be started
POST	ac/start	Start the car AC
GET	ac/stop	Check if the AC can be stopped
POST	ac/stop	Stop the car AC
GET	ac/temperature	Get the AC current temperature
POST	ac/temperature	Update the AC current temperature
GET	Monitoring	Get a list of the available components
GET	monitoring/componentId	Get the component information
GET	monitoring/location	Get the location of the vehicle
GET	monitoring/charge	Get the vehicle's charge

Table 3: WSRemoteControl Resources

Each request has a list of required parameters which can be seen in Annex H, as well as the data elements definition.

4.5.2 Example Messages

An example of an authentication message:

```
POST /sessions HTTP/1.1

Host: www.domain.pt

Content-Type: application/json

Content-Length: length


{"username":"admin","password":"
240be518fabd2724ddb6f04eeb1da5967448d7e831c08c8fa822809f74c720a9"}
```

An example of the server response to a successful authentication:

```
HTTP/1.1 200 OK

Content-Type: application/json

Content-Length: length
```

```
{"statusCode":200,"sessionToken":"123abc444123abcabccdacdaolaolaxp
to"}
```

An example of a get charge profiles message:

```
POST /charge/profiles HTTP/1.1

Host: www.domain.pt

Content-Type: application/json

Content-Length: length

{"sessionToken":"
123abc444123abcabccdacdaolaolaxpto","limit":2,"offset":0}
```

An example of the server response to a get charge profiles request:

```
HTTP/1.1 200 OK

Content-Type: application/json

Content-Length: length



{"statusCode":200,"chargeProfileList":[

      {"profileId":"123","profileName":"test1","profileDays":[0,1
      ],"startTime":"14:00","endTime":"15:00:00"},
      {"profileId":"321","profileName":"test2","profileDays":[2,3
      ,4],"startTime":"9:00","endTime":"10:00:00"}

]}
```

4.5.3 Message Description

The communication between the Clients and the MobicarInfo is through the exchange of messages. The client executes a HTTP operation (GET, POST, PUT, DELETE) to any of the available resources, with the parameters required, in a JSON object, in its body, and the web service returns a JSON object, with the requested information or with the status of the operation.

The messages have a specific structure, described in Annex I – MobicarInfo Interfaces Specification. This structure has to be known to both sides, Client and Server.

5 Risk Analysis

This chapter describes the risks involved in the project and the steps taken to mitigate them.

The major risks detected, ordered by degree of impact, were the following:

1. Delays in the implementation by the other parties involved in the project
2. Delays in the layout design
3. Changes in the APIs used in the system

Risk 1 occurred. This led to delays on the implementation of the Bluetooth Screen, and was implemented a simulated version of the service in order to test the functionalities of the screens depending on this service.

Risk 2 was mitigated by doing the layouts that weren't provided by the designers.

Risk 3 occurred. This happened in a later stage of the internship which made it impossible to re-write the affected services.

6 Achievements

As described in the Project Planning section, the goals for the internship were to develop three modules for the MobicarInfo, the SocialNetworks, Synchronize and Remote Control, and a Mobile Application to remotelly interact with the vehicle. Due to some circumstances of the project some features weren't fully developed.

The prototypes for the Remote Control Client developed during the first semester provided some valuable information for the development of the final version of the application.

This chapter describes the work done during the internship, and presents the problems found, how they were solved, the tests made during development and, finally, the future work.

6.1 Prototypes

As was previously stated, there were two prototypes made for the Remote Control GUI. One was an application only for Android, another was a cross-platform implementation using Sencha Touch 2[27] and PhoneGap[28]. Both have the same features and similar interface.

Sencha is a cross-platform library aimed at touch-enabled devices. Sencha allows to build native-feeling web apps for iPhone, Android and Blackberry.

PhoneGap is a framework that creates a bridge between the device's APIs and web apps, with support for multiple mobile platforms. Currently PhoneGap provides access to the device's accelerometer, camera, compass, contacts, files, geolocation, media, network, notifications (alert, sound, vibration) and storage.

Both prototypes were a simple implementation of the GUI for the Remote Control Client, because of this they don't communicate with any server, and all data is stored locally on the devices DB or in memory.

The main purpose of the prototypes were to execute Usability Tests, in order to understand the limitations of developing for Mobile devices. In the next section a brief description of the test and the insights taken from it.

6.1.1 Usability Test

This section describes the usability test conducted using the Remote Control Client prototype.

[27] http://www.sencha.com/

[28] http://www.phonegap.com

The goals of usability testing include establishing and validating user performance measures, and identifying potential design concerns to be addressed in order to improve the efficiency, productivity, and end-user satisfaction.

The next sections will describe the methodology, list the tasks preformed, and present the results obtained. A more detailed description on the Usability Test Plan can be found in Annex D and on the Usability Test Results in Annex E.

6.1.1.1 Methodology

For the realization of the tests 4 participants were recruited, from which three have a background in Informatics.

The tests occurred in three days. On the first day only one participant was tested, on the second day two were tested, and on the final day the last participant was tested.

Participant	Date of the Test	Computer Expertise	Android Expertise
1	12/12/2011	2	0
2	13/12/2011	2	3
3	13/12/2011	2	1
4	14/12/2011	4	0

Table 4: Test Participants

Each individual session lasted approximately 20 minutes. During the session, the test facilitator explained what would be tested and handed a list of the tasks to be performed. Participants read the task scenarios and tried to execute them. During the execution of tasks the data logger recorded the time taken to complete each task and logged any inconsistency found.

After each task, the facilitator asked the participants to rate the system on a 5-point System Usability Scale [26] with measures ranging from Strongly Disagree to Strongly Agree.

6.1.1.2 Tasks

Test participants attempted completion of the following tasks:

Task 1 - Authenticate User

Task 2 - Monitorization Interaction

Task 3 - Access Manage Charge

Task 3.1 - Manage Charge Interaction

Task 3.2 - Schedule Charge

Task 3.3 - Access Charge Profiles

Task 3.3.1 - Update a Charge Profile

Task 3.3.2 - Delete a Charge Profile

Task 3.3.3 - Create a Charge Profile

Task 4 - Access Manage V2G

Task 4.1 - Sell Energy

Task 4.2 - Update a V2G Profile

Task 4.3 - Delete a V2G Profile

Task 4.4 - Create a V2G Profile

Task 5 - Access Manage AC

Task 5.1 - Manage AC Interaction

Task 6 - Logout

See Annex E for a detailed list of tasks/scenarios

6.1.1.3 Results

This section presents a summary of the tests results for the Functionalities Testing and Overall Design testing.

Almost all of the tasks were completed by all participants, except for Task 2 and Task 3.2, which had a 50% completion rate.

The time (seconds) taken to complete each task was recorded. Task 3.3.1 required participants to update an existing profile and took the longest time to complete. However most of the tasks had a low completion time ranging from 5 seconds to 30 seconds.

The Test Observer captured the number of errors participants made while trying to complete the task scenarios. Task 2 and Task 3.2 add two critical errors each which made it impossible to complete without help. The first one was to interact with the Monitorization tab, which involved expanding/retracting the group elements of the list, while the last was to Schedule a Charge, which involved opening the menu, enter the Schedule activity and define some values.

All the other Tasks, except for Task 3.1 and Task 4.1 which add one non-critical error each, were completed without errors.

See Annex E for a more detailed list of the test results.

6.1.1.3.1 Summary of Data

The table below displays a summary of the test data. Low completion rates, critical errors and high values for time on task are highlighted.

	Task Completion	Errors	Time on Task (s)
Task 1	4	0	11,25
Task 2	2	**2**	32,5

	Task Completion	Errors	Time on Task (s)
Task 3	4	0	5
Task 3.1	4	1	23,75
Task 3.2	2	**2**	40
Task 3.3	4	0	13,75
Task 3.3.1	4	0	47,5
Task 3.3.2	4	0	5
Task 3.3.3	4	0	31,25
Task 4	4	0	6,25
Task 4.1	4	1	30
Task 4.2	4	0	30
Task 4.3	4	0	5
Task 4.4	4	0	38,75
Task 5	4	0	12,5
Task 5.1	4	1	20
Task 6	4	0	5

Table 5: Test Results Summary

6.1.1.3.2 Overall Design

After the test was finished, participants rated the application for 10 overall measures (see Annex E).

These measures include:

- Frequency of use
- Complexity
- Ease of use
- Need for support
- Functions well integration
- Inconsistency
- How quickly most people would learn how to use the system

- System very cumbersome to use
- Level of confidence using the system
- Need to learn a lot of things before using the system

All the participants strongly agreed that they would use the system in a regular basis and there was no need to learn anything new to be able to use it. They agreed that the system was easy to use and easy to learn.

The following table shows the complete results.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Score (%)
Participant 1	5	2	4	1	5	2	5	2	4	1	**87.5**
Participant 2	5	2	5	1	4	1	5	1	5	1	**95**
Participant 3	5	2	4	1	4	1	4	1	4	1	**87.5**
Participant 4	4	1	4	1	5	1	5	1	4	1	**92.5**

Table 6: Overall Design results

6.1.1.4 Findings and Recommendations

The problems found, the recommendations to solve them and their severity are presented in this section. The recommendations are to be treated as a possible solution to that specific problem, but the developer is free to find other solutions. The same cannot be said about the problem severity. This attribute is to be mandatorily used to establish the priorities of problems solving. The severity level follows the next scale:

- 1: If the problem isn't fixed, the users might not be able to use some functionality.
- 2: Users will be frustrated if this problem is not solved; they may give up.
- 3: If the problem isn't fixed users might end doing wrong actions or actions that they wouldn't need.
- 4: Users are annoyed, but the problem doesn't keep them from completing the use of certain functionality.

6.1.1.4.1 Monitorization Interaction (Task 2)

Change	Justification	Severity
Show a wizard on the first time running the application explaining how the expandable list works	Two participants didn't interact with the Expandable list	2

Table 7: Monitorization Interaction Recommendation

6.1.1.4.2 Manage Charge Interaction (Task 3.1)

Change	Justification	Severity
Change the "Set Speed" label to "Set Charge Speed"	One participant found that the Set Speed label was not clear on the meaning of its purpose, preventing the user to	3

Change	Justification	Severity
	correctly configuring the Charge	

Table 8: Manage Charge Interaction Recommendation

6.1.1.4.3 Schedule Charge (Task 3.2)

Change	Justification	Severity
Show a wizard explaining how the sub-menu works on the first time running the application	Two participants didn't know how to access the sub-menu	1

Table 9: Schedule Charge Recommendation

6.1.1.4.4 Sell Energy (Task 4.1)

Change	Justification	Severity
Change the positioning/width of the "Amount to Sell" seekbar	One participant found that the Seekbar was not completely visible, preventing the user to correctly define the sell options on the first try	4

Table 10: Sell Energy Recommendation

6.1.1.4.5 Manage AC Interaction (Task 5.1)

Change	Justification	Severity
Add a description to the AC Control section explaining its purpose	One participant found that the AC Control button purpose was not clear	4

Table 11: Manage AC Interaction Recommendation

6.1.1.5 Usability Test Conclusion

Most of the participants found the MobicarInfo Remote Control Prototype easy to use, comprehensive, clean and uncluttered. There were some aspects that limited its usability for those who were not used with Android applications, and some non-critical errors that could be an annoyance for the users if not solved.

Implementing the recommendations and continuing to work with users ensures a continued user-centric system.

6.2 MobicarInfo

The modules implemented for the MobicarInfo follow a three-layered architecture:

- Services – Provide the wrappers for the REST APIs from Facebook, FourSquare, Twitter and Google, and the Remote Control web service. The services are loaded at the start-up of the MobicarInfo.
- Daemons – Proxy between the GUI and the Services. The daemons are loaded after the Services.
- GUIs – GUI presented to the user. The GUI is the last to be loaded.

The communication between each abstraction layer was implemented using DBus[29]. DBus is a message bus system that allows an application to register methods and signals, which can be accessed and caught from any application, thus allowing applications to talk to one another.

The following sections will go into detail about each service, daemon and GUI developed.

6.2.1 Services

The services provide the wrappers needed by the Social Networks and Synchronize modules and the web service needed by the Remote Control module.

These services are implemented using Python[30].

6.2.1.1 Social Networks

The Social Networks service is divided into three modules, Facebook, FourSquare and Twitter.

6.2.1.1.1 Facebook

This service was developed using the Pythonforfacebook[31] SDK to handle the Facebook Graph API and Authentication.

The Facebook Service provides methods for requesting:

- User profile
- News feed
- Feed entry comments
- Albums
- Photos in an album

It also provides methods for publishing:

- New albums
- New photos into an album
- New entry in the user wall
- New comments into feed entries

These methods are registered into DBus and accessible by any application.

[29] http://www.freedesktop.org/wiki/Software/dbus

[30] http://www.python.org/

[31] http://www.pythonforfacebook.com/

The Authentication with Facebook is done through OAuth 2 [32], and the authentication end-point is through a web-site, which forces the authentication procedure to be divided between the service and the GUI. The flow is as follows:

- If there is no account already authenticated with the application (stored in file), an Auth URL is sent to the GUI
- When an Auth URL is returned a WebView is displayed for the user to authenticate and authorize the Mobicar Application
- On a successful authorization a Token is returned to the Facebook Service
- The Service finishes the OAuth 2 flow by requesting the access token and storing it

After the OAuth procedure is done the service can do the normal operations in name of the user.

Due to the Facebook API restrictions, in order to get the Picture of any object (Feed Entry, Album, Photo, User), an extra request is needed, which delays the retrieval of information. To accelerate the process of retrieving the Picture URL for each of the objects, without blocking, a Thread Pool was created. Whenever a picture for an object isn't available locally (in memory), a task is added to a Queue so the Pool of threads can fulfill the task. When all tasks are finished a signal is emitted in order for the daemon to refresh the data and notify the GUI.

The service is responsible for handling authentication, request and connection errors. When an error is caught a signal is emitted, this way any application listening can treat it accordingly.

6.2.1.1.2 FourSquare

This service was developed using the FourSquare[33] library to handle the FourSquare endpoints and authentication.

The FourSquare Service provides methods for requesting:

- User profile
- Checkins
- Places

It also provides a method for making a Checkin into a place.

These methods are registered into DBus and accessible by any application.

The Authentication with FourSquare is done through OAuth 2, and the authentication end-point is through a web-site, which forces the authentication procedure to be divided between the service and the GUI. The flow is as follows:

[32] http://oauth.net/2/

[33] https://github.com/mLewisLogic/foursquare

- If there is no account already authenticated with the application (stored in file), an Auth URL is sent to the GUI
- When an Auth URL is returned a WebView is displayed for the user to authenticate and authorize the Mobicar Application
- On a successful authorization a Token is returned to the FourSquare Service
- The Service finishes the OAuth 2 flow by requesting the access token and storing it

After the OAuth procedure is done the service can do the normal operations in name of the user.

The service is responsible for handling authentication, request and connection errors. When an error is caught a signal is emitted, this way any application listening can treat it accordingly.

6.2.1.1.3 Twitter

This service was developed using the Tweepy [34] library to handle the Twitter endpoints and authentication.

The Twitter Service provides methods for requesting:

- User profile
- Timeline

It also provides methods for creating new Tweets.

These methods are registered into DBus and accessible by any application.

The Authentication with Twitter is done through OAuth 2, and the authentication end-point is through a web-site, which forces the authentication procedure to be divided between the service and the GUI. The flow is as follows:

- If there is no account already authenticated with the application (stored in file), an Auth URL is sent to the GUI
- When an Auth URL is returned a WebView is displayed for the user to authenticate and authorize the Mobicar Application
- On a successful authorization a Token is returned to the Twitter Service
- The Service finishes the OAuth 2 flow by requesting the access token and storing it

After the OAuth procedure is done the service can do the normal operations in name of the user.

The service is responsible for handling authentication, request and connection errors. When an error is caught a signal is emitted, this way any application listening can treat it accordingly.

[34] http://tweepy.github.com/

6.2.1.2 Google

The Google service was divided into two modules, Contacts and Calendar.

These services were developed using the GData API[35] library.

6.2.1.2.1 Contacts

The Contacts Service provides methods for:

- Authentication
- Logout
- Getting the Contacts list
- Creating Contacts
- Updating Contacts
- Deleting Contacts

These methods are registered into DBus and accessible by any application.

Due to the GData API restrictions, in order to get the Picture of a contact, an extra request is needed, which delayed the retrieval of information. To accelerate the process of retrieving the Picture for each of the contacts, a Thread Pool was created. Whenever a picture for a contact isn't available locally (in memory), a task is added to a Queue so the Pool of threads can fulfill the task. When all tasks are finished a signal is emitted in order for the daemon to refresh the data and notify the GUI.

The service is responsible for handling authentication, request and connection errors. When an error is caught a signal is emitted, this way any application listening can treat it accordingly.

6.2.1.2.2 Calendar

The Calendar Service provides methods for:

- Authentication
- Logout
- Getting the Calendars list
- Getting the Events of a calendar
- Creating Calendars
- Updating Calendars
- Deleting Calendars
- Creating Events

[35] https://developers.google.com/gdata/docs/client-libraries

- Updating Events
- Deleting Events

These methods are registered into DBus and accessible by any application.

When searching for events of a calendar only the events of a specific month are fetched. This is to keep the time needed to retrieve the information as low as possible.

The service is responsible for handling authentication, request and connection errors. When an error is caught a signal is emitted, this way any application listening can treat it accordingly.

6.2.1.3 Remote Control

The Remote Control service was developed using the CherryPy[36] framework. This framework is web oriented, and is designed for rapid development of web applications.

This service implements the interface described in Section 4.5.

In order to provide all the required functionalities this service communicates with the following daemons:

- Charge – Start/Stop charges, Charge Speed list, Manage profiles
- RemoteControl – AC Management, Monitoring data
- Core – V2G Buy/Sell operations, V2G List of Buyers/Sellers.

6.2.2 Daemons

The daemons were developed using C++ and the Qt 4.7 [37] library.

There is a daemon for each screen. Daemons are in charge of requesting information from the service, as well as keeping this information up to date, with the use of Threads.

All requests made to the various services are asynchronous. This allows the daemon to keep running and processing requests without blocking.

6.2.2.1 Charge

The Charge daemon acts as a proxy between the Charge Screen and the Remote Control Client and the CAN Daemon. The need for this daemon comes from the extra functionality added with the creation and management of charge profiles.

The daemon provides methods for:

- Starting a Charge
- Stopping a Charge

[36] http://www.cherrypy.org/

[37] http://doc.qt.nokia.com/

- Scheduling a Charge
- Creating a Profile
- Updating a Profile
- Deleting a Profile

To persist the profiles a SQLite database is used.

In the background a thread is, at a set time interval, checking for any profile in the database that should be executed. Whenever the time conditions are verified the start charge command is sent to the CAN daemon, and the thread enters a waiting mode, where the current profile stays running until the end time defined. When the end time arrives, an end charge command is sent to the CAN daemon, and the thread resumes the normal functionality.

6.2.2.2 RemoteControl

The RemoteControl daemon acts as a proxy between the RemoteControl Screen and the Remote Control Client and the CAN Daemon. The need for this daemon comes from the extra functionality added with the creation and management of users.

The daemon provides methods for:

- Starting the AC
- Stopping the AC
- Set the AC Temperature
- Creating a User
- Updating a User
- Deleting a User
- Grant Permissions
- Revoke Permissions
- Get Telemetry data

To persist the user profiles a SQLite database is used.

6.2.2.3 SocialNetworks

The SocialNetworks daemon acts as a proxy between the Facebook, FourSquare and Twitter Screens and the Facebook, FourSquare and Twitter services.

The daemon provides methods for accessing all the functionalities provided by each service.

When a request is made to this daemon, an Asynchronous call is made to the corresponding service method. The reply is handled by a Low Priority Thread which checks, at a set time interval, if the reply is finished. When the reply finishes, the resulting data is sent to the "father" process for processing.

Whenever new data is fetched from a service, the "father" process emits the corresponding signal to notify the GUI. The errors emitted by any of the services are also forwarded to the GUI.

If there are no requests to process, the Thread refreshes the current data at a constant interval.

6.2.2.4 Synchronize

The Synchronize daemon acts as a proxy between the Google Screen and the Contacts and Calendar services.

The daemon provides methods for accessing all the functionalities provided by each service.

When a request is made to this daemon, an Asynchronous call is made to the corresponding service method. The reply is handled by a Low Priority Thread which checks, at a set time interval, the reply until it finishes. When the reply finishes, the resulting data is sent to the "father" process for processing.

Whenever new data is fetched from a service, the "father" process emits the corresponding signal to notify the GUI. The errors emitted by any of the services are also forwarded to the GUI.

If there are no requests to process, the Thread refreshes the current data at a constant interval.

This daemon was also meant to act as a proxy between the Bluetooth Screen and a daemon or service with the ability to manage Bluetooth devices. Due to some delays this functionality wasn't implemented. For demonstration purposes the methods were implemented but with static data.

6.2.3 GUIs

The GUIs were developed using C++, Qt 4.7 library and QML[38].

QML is a declarative language designed to describe the user interface of a program. We are able to share objects between C++ and QML, as well as calling methods, accessing properties and catching signals.

6.2.3.1 Bluetooth Screen

This screen provides a GUI for discovering Bluetooth devices and pairing with them, as well as accessing the contacts list in a device and calling any of the contacts.

All requests are made through DBus to the Synchronize daemon. In order for not blocking the GUI, the requests are made asynchronously. The reply is handled by a Low Priority Thread which checks, at a set time interval, if the reply is finished. When the reply finishes, the resulting data is sent to the "father" process for processing.

[38] http://doc.qt.nokia.com/4.7-snapshot/qdeclarativeintroduction.html

6.2.3.2 Charge Screen

This screen provides a GUI for scheduling a charge.

All requests are made through DBus to the Charge daemon.

6.2.3.3 Social Networks

The Social Networks menu provides access to the three social networks available.

All requests made from the different GUIs are made through DBus to the Social Networks daemon. In order for not blocking the GUI, the requests are made asynchronously. The reply is handled by a Low Priority Thread which checks, at a set time interval, if the reply is finished. When the reply finishes, the resulting data is sent to the "father" process for processing.

6.2.3.3.1 Facebook Screen

This screen provides a GUI for accessing Facebook. It allows a user to browse their albums of photos, see their News Feed, and to publish new data.

6.2.3.3.2 FourSquare Screen

This screen provides a GUI for accessing FourSquare. It allows a user to browse their Checkins, to search for places and to Checkin to those places.

6.2.3.3.3 Twitter Screen

This screen provides a GUI for accessing Twitter. It allows a user to browse their Timeline and to publish new Tweets.

6.2.3.4 Google Screen

The Google menu provides access to the two services available.

All requests made from the two GUIs are made through DBus to the Synchronize daemon. In order for not blocking the GUI, the requests are made asynchronously. The reply is handled by a Low Priority Thread which checks, at a set time interval, if the reply is finished. When the reply finishes, the resulting data is sent to the "father" process for processing.

6.2.3.4.1 Calendar Screen

This screen provides a GUI for accessing the authenticated user Calendars. The user can edit any of the calendars or list the events contained in them. When listing the events of a calendar, the user is able to navigate by month or week. The user can select a day of the month in order to see the list of events of that day, and create, edit or delete any of the events.

6.2.3.4.2 Contacts Screen

This screen provides a GUI for accessing the authenticated user Contacts. The user can edit any of the contacts or create new entries. The user can also call a contact if a device was previously paired through the Bluetooth Screen.

6.2.3.5 MakeCall Screen

This screen provides a simple GUI for calling contacts.

All requests are made through DBus to the Synchronize daemon.

6.2.3.6 RemoteControl Screen

This screen provides a GUI for managing users which can access the vehicle through the Remote Control Client. It provides an interface to create new users and edit existing ones. There is also an interface to select the functionalities, from a predefined list, that will be available to a user in the Remote Control Client.

6.2.3.7 V2G Screen

This screen provides a GUI for buying or selling energy.

All requests are made through DBus to the Core daemon.

6.3 Remote Control Client

The Remote Control Client was developed based on the GUI prototype implemented during the first semester.

This client is a cross-platform web application, developed using Sencha Touch 2 and PhoneGap.

The application follows a two-layered architecture:

- Processing Layer – Responsible for processing data and executing their respective functions
- Presentation Layer – Responsible for presenting a user interface that enables the interaction with the system.

The following sections will go into detail about each layer.

6.3.1 Processing Layer

The processing layer was developed as a normal JavaScript object, which implements all the methods required to fetch and process information for the views.

This object provides methods for:

- Authentication
- Getting the user permissions
- Getting the monitoring data
- Getting the Charge status
- Getting the Charge profiles
- Getting the Charge schedule
- Getting the AC status
- Getting the V2G lists

- Starting/Stopping a Charge
- Creating Charge profiles
- Update Charge profiles
- Setting a Charge schedule
- Starting/Stopping the AC
- Selling/Buying energy
- Logging out

There was a need to keep the information in the application up to date. To accomplish this, a function was added with a constant refresh, in order to keep running as long as the application lives, polling data from the server. Since some information changes more frequently than other, this polling takes that into consideration.

To perform the calls to the REST web service running in the vehicle we used JQuery[39].

6.3.2 Presentation Layer

The presentation layer was developed using Sencha Touch 2.

There are multiple views for the different functionalities available. The views use the information retrieved by the processing layer and inserted into Store objects, or inserted directly into the view elements.

The usability test performed during the first semester provided some insights about how to structure the Views, which were greatly changed since the prototype.

In the prototype the information was displayed using expandable lists, allowing the users to expand/retract elements as needed. This wasn't straight forward, and was removed from the final version of the application.

In the Manage Charge View the Speed and Charge Controls were separated, forcing the user to perform two actions in order to start a charge at the desired speed. For the final version, there is a single section with the Speed of the charge and a button to Start/Stop it. When a Start is requested, the speed that is selected is sent with that same request.

The prototype used a sub-menu to access extra functionalities like scheduling a Charge or managing the Charge Profiles, and during the usability this caused some problems for users. For the final version this hidden menu was removed, and a fixed bottom bar was added. The previous version also posed some problems for the application being cross-platform, since some devices do not have a Menu button.

[39] http://jquery.com/

7 Tests

This section presents the tests results for the MobicarInfo and Remote Control Client applications.

The complete test case specification and results can be found in Annex J and K - Test Cases Specification, Test Results.

The tests tried to cover most of the functionalities. There were some aspects that weren't fully tested, manly the exception handling of some screens.

During the first test run, 30 tests were executed, and from those 30 tests 13 gave errors. From these 13 errors, 3 were critical errors that completely prevented the use of some of the functionalities, also preventing the execution of further test cases.

During the second test run, 60 tests were performed, from those 12 gave errors. Most of those errors were design issues with minimal impact.

There are some aspects of the application that need to be fixed before its deployment into a real environment.

The issues detected in the first Test Run completely prevented the use of some screens. There were also some issues that caused a bad user experience.

There were major improvements between the first Test Run and the second. Fixing the issues found in the second Test Run will make the application much more stable and user friendly, and ready for a real environment.

8 Conclusions

The work done throughout the internship helped the development of the MobicarInfo, the Infotainment system for the next generation of electrical vehicles. The components developed provide owners the social component that is so important in the current daily life, with access to social networks and google services. A mobile application was also developed to give users the ability to interact with the vehicle from anywhere, and to check on different component status. To support this application, the required services were also implemented in the infotainment side, mainly a REST web service and a Remote Control screen for managing the users. There was also some integration with modules developed by other members of the team.

The MobicarInfo goal is to create a cost-effective infotainment system that incorporates all the core functionalities of the new generation infotainment systems, such as AM/FM radio, music player, video player, GPS, Bluetooth pairing, interaction with social networks and add the required mechanisms to interact with the Mobi.E and the One.Stop.Transport mobility platforms.

On the first semester the focus was on documentation and prototyping. A State of the Art Study was made in order to find the best development platform for the MobicarInfo. There was a major set back during this phase due to the scarce of detailed information about the solutions, making it harder to compare them. There was a usability test done using one of the prototypes, this provided with some feedback that was used during the development phase.

On the second semester the different modules were developed, first of all, the services that provide the Social Network and Synchronize daemons all their information. The Social Networks services introduced a minor set back, with the use of OAuth 2 with a web based authentication point. This forced the authentication process to be divided between the GUI and the Service.

Secondly the implementation of the daemons. The communication between the daemons and the services was first with synchronous requests, but this proved to be a problem, since the services may take longer than expected contacting the web APIs, resulting in timeouts. In order to minimize this issues the requests were changed to asynchronous and threads were created to handle the replies.

During testing an issue was detected while getting the Google Contacts images and the Facebook images. There was a big delay while fetching information, making requests to timeout. To overcome this a thread pool was added to those services. Caching of images was also introduced in order to reduce the number of requests.

The prototyping and usability testing done during the first semester proved to be valuable during the development of the Remote Control Client, allowing for a better and more intuitive interface to be implemented, taking into account all the feedback received from the test subjects.

There were some deviations from the planning, during the first semester and the second semester. The first semester was due to the introduction of prototyping and the specification of the Software Requirements and Architecture for the Remote Control Client. The second semester was due to design delays and the delay on the implementation of the Bluetooth service.

8.1 Future Work

The deviations on the second semester caused the development of the Bluetooth module to be postponed. The Synchronize daemon provides static data for the Bluetooth, Google and MakeCall screens. A service should be implemented and connected to the Synchronize daemon, in order to provide real information. This service should have the following capabilities:

- Discover Bluetooth devices
- Pair with devices
- Get a device contacts
- Make a call through a device
- Receive a signal when a call is received by a paired device

The Google Contacts and Calendar were Google Data APIs, but recently Google as started to adapt their APIs to REST web services which makes the Google Data APIs deprecated. This creates an urgent need to change this services to connect to the new APIs.

The layout for the MobicarInfo and Remote Control Client was not finished by the end of the Internship, due to some delays in the Design department. There are some minor changes required in the presentation of data in the Monitoring view of the Remote Control Client, in order for the application to be more eye-candy. The screens of the MobicarInfo also need some adjustments to make the layout more intuitive and user friendly.

There is room for some improvement in terms of error handling in the Remote Control Client, in order to prevent any data loss whenever there is a problem with the connection to the server.

Also, the application should be deployed to iPhone, since it was developed with being cross-platform in mind the work required to accomplish this is minimal.

9 References

[1] San Ramon, GENIVI Alliance Unveils Member Compliance Program, August 2, 2011.

[2] Wind River VxWorks RTOS. Available in the URL http://www.windriver.com/products/vxworks/ accessed September, 2011.

[3] Wind River, Wind River Platform for Infotainment 3.0, 2010.

[4] Wind River, Wind River VxWorks Platforms 6.9, 2011.

[5] What is Linux: An Overview of the Linux Operating System. Available in the URL https://www.linux.com/learn/resource-center/376-linux-is-everywhere-an-overview-of-the-linux-operating-system accessed September, 2011.

[6] Ubuntu IVI Remix receives GENIVI Alliance Compliance Approval. Available in the URL http://www.canonical.com/content/ubuntu-ivi-remix-receives-genivi-alliance-compliance-approval accessed September, 2011.

[7] MeeGo In-Vehicle. Available in the URL https://meego.com/devices/in-vehicle accessed September, 2011.

[8] Windows Embedded Compact 7 (Formerly CE). Available in the URL http://www.microsoft.com/windowsembedded/en-us/evaluate/windows-embedded-compact-7.aspx accessed September, 2011.

[9] Microsoft, A Technical Companion to Windows Embedded Automotive 7, July, 2010.

[10] Visteon, Infotainment and Internet Platform: Technology Data Sheet.

[11] QNX Neutrino RTOS. Available in the URL http://www.qnx.com/products/neutrino-rtos/neutrino-rtos.html accessed September, 2011.

[12] QNX, QNX CAR Application Platform: Solution Brief, 2010.

[13] Mentor Graphics, Tools and Services for IVI Developers, Mentor Embedded In-Vehicle Infotainment (IVI): Datasheet, 2011

[14] Android Developers. Available in the URL http://developer.android.com/guide/basics/what-is-android.html accessed September, 2011.

[15] Ubuntu One Wiki. Available in the URL https://wiki.ubuntu.com/UbuntuOne/TechnicalDetails accessed September, 2011.

[16] Bluez Documentation. Available in the URL http://harmattan-dev.nokia.com/docs/platform-api-reference/ accessed October, 2011

[17] JBluez. Available in the URL http://jbluez.sourceforge.net/ accessed October, 2011.

[18] Twitter Developers. Available in the URL https://dev.twitter.com/docs accessed November, 2011.

[19] FourSquare Developers. Available in the URL https://developer.foursquare.com/ accessed November, 2011.

[20] Facebook Developers. Available in the URL https://developers.facebook.com/docs/reference/api/ accessed November, 2011.

[21] Saab IQon. Available in the URL http://www.google.pt/url?sa=t&rct=j&q=saab%20iqon&source=web&cd=1&ved=0CCEQFjAA&url=http%3A%2F%2Fnewsroom.saab.com%2Fnews%2Fnews%2Fworldfirstfromsaabsaabiqonopeninnovationincarinfotainment.5.741c75ab12da7f448807ffe719.html&ei=PZgdT_79G8jX8QOE3-ygCw&usg=AFQjCNE54v2JPpzQGVSrrhauPQbYkq1pZw&cad=rja accessed September, 2011

[22] MontaVista Automotive Technology Platform. Available in the URL http://mvista.com/sol_detail_ivi.php accessed September, 2011.

[23] John Lehmann,Understanding the Importance of GENIVI and Open Source IVI Development, 2011.

[24] David Schultz, Judith Bachman, Linda Landis (CSC), Mike Stark, Sally Godfrey (GSFC), Maurizio Morisio (Univ. of Maryland), Space Engineering (Software), ECSS-E-40b Draft 1, February 15, 2002

[25] Rui Cordeiro, Carlos Coutinho, Project Life Cycles, April 8, 2011

[26] J. Brooke, SUS: A quick and dirty usability scale, in Usability evaluation in industry, P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, Eds. London: Taylor and Francis, 1996.