POLITECNICO DI TORINO

Collegio di Ingegneria Informatica

Corso di Laurea Magistrale
in Ingegneria Informatica

Tesi di Laurea Magistrale

Android infotainment system

A prototype based on Android Automotive

Relatore
prof. Massimo Violante

Candidato
Francesco Spissu

Settembre 2018

Contents

Summary

1. Introduction 2
2. State of the Art 5
   2.1. In-Vehicle Infotainment 5
      2.1.1. Overview 5
      2.1.2. Architecture 10
   2.2. Operating Systems 14
      2.2.1. QNX 14
      2.2.2. Linux 15
         2.2.2.1. Genivi Alliance 16
         2.2.2.2. Automotive Grade Linux 16
      2.2.3. Android 16
   2.3. Main Components 18
3. Technologies 21
   3.1. Android 21
      3.1.1. Android Platform Architecture 21
      3.1.2. Android Open Source Project 26
      3.1.3. Project Treble 29
      3.1.4. Hardware Interface Definition Language 30
   3.2. Android Automotive 31
      3.2.1. Android Automotive Architecture 31
   3.3. Android Compatibility Definition Document (CDD) 34

3.4. Google Automotive Services (GAS) 35
3.5. Android Auto 36
4. Hardware and toolchain 39
4.1. Hardware 39
4.2. Software 40
4.2.1. Android Studio 40
4.2.2. Software Development Kit 41
4.2.3. Testing 42
4.2.3.1. adb tool 42
4.2.3.2. Logcat 42
4.2.4. Version Control 43
4.2.4.1. Git 43
5. Implementation 45
5.1. Development 45
5.1.1. Overview 45
5.1.1.1. Specifics 45
5.1.1.2. Design 46
5.1.1.3. Logic 49
5.1.2. Radio 50
5.1.2.1. Specifics 50
5.1.2.2. Design 51
5.1.2.3. Logic 52
5.1.3. Multimedia 54
5.1.3.1. Specifics 54
5.1.3.2. Design 54
5.1.3.3. Logic 56
5.1.4. Phone 57

5.1.4.1. Specifics 57
5.1.4.2. Design 58
5.1.4.3. Logic 61
5.1.5. MyCar 62
5.1.5.1. Specifics 62
5.1.5.2. Design 63
5.1.5.3. Logic 63
5.1.6. Preferences 64
5.1.6.1. Specifics 64
5.1.6.2. Design 65
5.1.6.3. Logic 67
5.1.7. SystemUI 68
5.1.8. Different screen sizes management 69
5.1.9. Data Binding 69
6. Future opportunities 71
6.1. Hypervisor 71
6.2. Rotary Pad 72
6.3. Multi-Display 73
6.4. Android P 73
7. Conclusions 75
8. Bibliography 77

1. Introduction

New technologies in recent years are becoming increasingly common in people's everyday lives. In particular, the automotive industry has benefited a lot from these innovations and vehicle models evolve and renew faster and faster with ever more advanced features. In fact, the latest generation cars are no longer just means of transport but have become real computers.

This master thesis project focuses on the development of Android applications for the automotive sector. The aim is to develop a prototype of an infotainment system based on Android Automotive operating system. The goal of the work will be to create a user interface for the cars of the future using a Qualcomm development board with a processor Snapdragon 820Am on which the latest version of the Android Automotive system is available (Oreo 8.1). So, it is needed an interaction with the system at every level in order to customize its operation. To achieve this purpose, it was requested the development of one or more native applications, oriented to the use on board the vehicle, the personalization of applications already available within the Android platform and interaction with Google Automotive Services. Android Automotive is an extension of the standard Android operating system that is developed with the aim of making it work inside an infotainment system. The base level of Android Automotive is the Vehicle HAL, an interface for developing Android Automotive implementations. This interface abstracts how Android communicates with a concrete vehicle and defines the vehicle properties that OEMs can implement. This means that there are many advantages for the user, that rediscovers inside the car the apps that he has inside the smartphone and with which he has great familiarity. But most the benefits are for the manufacturers that don’t have to develop an ad-hoc operating system, that would entail many a big amount of money and time, the would concentrate their resources in the development of new apps or in the customization of those already existing.

So, the work to be done foresaw the creation of some apps, some (like Multimedia and Radio) were already present as released by Google, so for these it was necessary to customize the graphics and add some new features based on the client's requests. While other apps (like MyCar and Preferences) were made directly from scratch. Among these applications there are:

an Overview app which works as the main view of the dashboard, the Radio app, the Multimedia app for playing music, the Phone app, the MyCar app which shows to the driver the driving modes, indicators, pressure of tires and all the settings related to the vehicle, then the Preferences app for choosing the main theme of the system and the others settings like volume and connectivity.

The work done for this project was only a first implementation and new features will be added. Among these there is the use of a hypervisor which makes it easier to obtain and maintain safety certifications by separating safety-critical components from non-safety critical components in separate guest operating systems, in this case Android. For example, this is useful for a safety critical instrument cluster that has security requirements that Android does not guarantee because it is not ASIL certified, in this way no complex hardware is shared. Another feature is the interaction with a rotary pad that helps the driver navigate the various menus and submenus, which is a bit easier than continuously tapping a touchscreen.
Then there is the possibility to have the multiscreen support. Finally, it is desirable the porting on Android P, the next version of Android that Google will release.

The realization of this master thesis project has been developed in collaboration with Magneti Marelli in Venaria Reale. Magneti Marelli is an international company that supplies all the leading car makers with his hi-tech systems and components. The sector where this assignment was achieved is that of Infotainment and Telematics (in particular in the Human Machine Interface team), this group has a lot of years of experience in the development and production of this kind of systems.

Before discussing the work done, the next chapter will present the state of the art regarding the current technologies that are adopted in the implementation of an infotainment system.

2. State of the Art

2.1. In-Vehicle Infotainment

2.1.1. Overview

In-Vehicle Infotainment (or IVI) systems are the heart of the modern cars. In-Vehicle Infotainment is the term used to refer to systems which have the goal of providing entertainment and information contents to drivers and passenger of the vehicle.
There are a lot of different IVI systems, but the main tasks are the same for all of them:

- Managing and playing audio contents
- Navigation for driving
- Making phone calls
- Listen to incoming and sending text messages

The whole system is a combination of components such as control panel, telematics device, and head up display but the main component of system is the infotainment head unit that concentrates the infotainment functions of the vehicle, including tuner reception, media connectivity, audio playback, navigation and human-machine interface.
Since the diffusion of smartphones and their apps is changed the way people absorb information and communicates each other. So, users want to find the same features also while they are driving their vehicles, with a responsive touch-screen, over the air updates and seamless integration of apps (installed on his/her phone) in an IVI system. However, the priority for an OEM (Original Equipment Manufacturer) is to minimize distraction of the driver ensuring the maximum safety.
The requirement to have the same user experience in the vehicle that consumer has in the smartphone has been satisfied. So, manufacturers have to adopt some procedures to develop an infotainment system with these characteristics. [1]
Taking into consideration the system developed by Magneti Marelli which was mounted on the Alfa Romeo Giulia and Stelvio, we now do a more detailed description of all the characteristics that an infotainment system provides. [2]

In particular:
It has to offer the support for multiple cameras, so the driver has a surround view for driving and parking assistant function with rear camera. For this reason, it's necessary to design middleware and application that support switching between multiple cameras and streaming of data from multiple cameras. (Figure 1)

Figure 1. Parking Assistant with rear view camera

To integrate various sources for multimedia streaming like mass storage devices (USB, CD), radio sources (AM, FM, WB, XM or DAB), streaming through Bluetooth or Wi-Fi, internet streaming and connectivity to Apple or Android devices in a single application. In order to playback audio content from devices and smartphones via usb and Bluetooth audio streaming technology and also listen to the radio. (Figure 2)

Figure 2. Media Player

To offer users all critical vehicle data, providing access to vehicle information available on CAN network in the form of graphical interface as well as text-based warnings/alerts. Develop and build intelligence in the system to act according to different states of the vehicle such as moving, at rest and parking mode. This kind of applications for vehicle diagnosis and service management information access give to the driver the possibility to keep under control all the information about the vehicle like the pressure of the tyres, the condition of the brakes, the time and the distance needed for the next maintenance, the level of the oil, the drive mode and so on. (Figure 3)

Figure 3. Vehicle data

To provide an app that shows the performances of the car showing data and statistics related to driving style and the consumptions displaying in order to improve the driving mode of the driver. (Figure 4)

Figure 4. Efficient drive

To implement a full connectivity with all mobile devices with Bluetooth communication in order to have a safe interaction with the phone in hand free mode via Bluetooth connection. In this way the driver can respond to or make a call (even in conference mode), read message, stream music and more. This feature plays a significant role in mitigating driver distraction. (Figure 5)

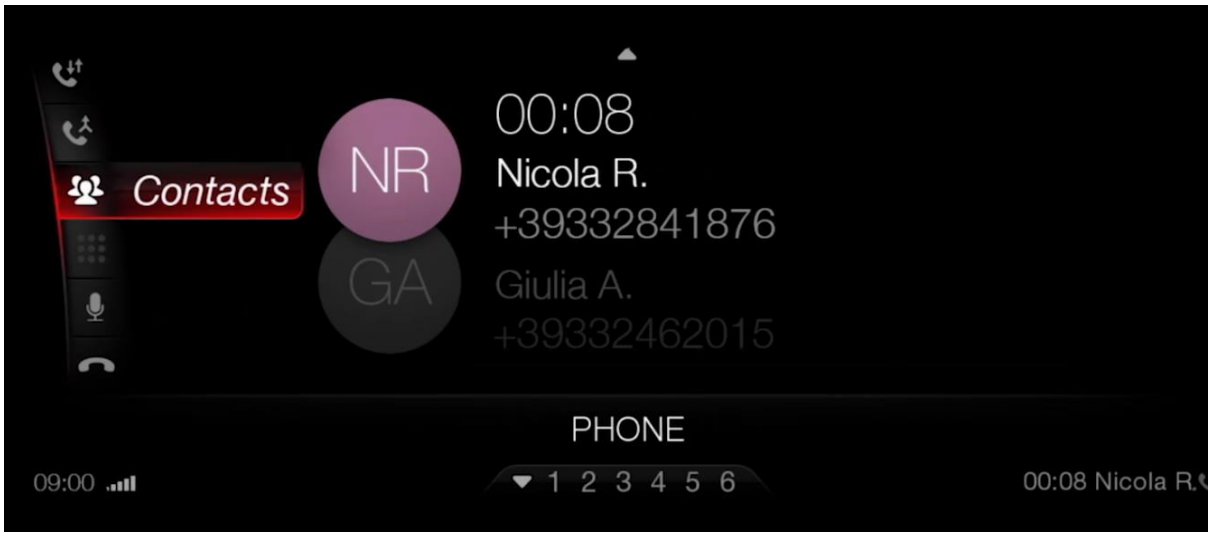

Figure 5. Phone

To give the possibility to install third party apps also based on the geography of the market target to improve user experience. For example, apps for navigation with high resolution 3d maps, with high speed route calculation and real time traffic management. It is also useful to have GPS, this plays an important role in navigation but not fundamental because navigation still working even without GPS signal thanks to dead reckoning technology (which allows to calculate the current position using a previously determined position and advancing that position based upon known or estimated speeds over elapsed time and course). It's also possible to have apps for finding nearest fuel station, finding nearest service station, emergency contact service, weather, internet streaming of multimedia and more. (Figure 6)

Figure 6. Navigation

It's also needed a framework with all the APIs (Application Programming Interfaces) that can be called by the IVI applications, this is the way to have a very high responsive and an intuitive IVI experience. Even with the customization of the look and feel of the screens to display most important information to the user such as weather, date & time, traffic alerts, and other alerts from the vehicle at all times.

Another important characteristic is voice recognition feature that provides an easy way to activate all the components, so the driver can concentrate on driving in order to limit distractions.

The support for connectivity through tethering, Wi-Fi and telematics so the driver and the passengers can be connected to the external world.

Another requirement is to develop and build strategies to help OEMs' and/or Suppliers with certification and secure installation of the applications in IVI system. This enables individual user to deploy applications of their interests in a safe and secured manner.
Despite the multiple choice of applications available the most used applications of infotainment systems are entertainment and navigation. This limitation is due to the connectivity infrastructure that in many countries there is the lack of high-speed internet connection. So, some features like accessing to social media, sending mails, streaming videos and music are possible only in developed markets. [3]

2.1.2. Architecture

The architecture of an infotainment system goes inside a bigger network architecture that is inside the vehicle. It is composed by a CAN bus which interconnects all the ECUs contained in the car with the aim of making them communicate with each other receiving feedbacks from sensors. So, the vehicle has thus become a set of microprocessors that collaborate and exchange information, forming a real network of distributed computers.

- The CAN (Control Area Network) is a multi-master serial bus this means that all the nodes of the network can transmit, and more nodes of the network can request the transmissive channel at the same time. It's a message-based protocol. It allows real-time distributed control with a very high level of security. It was introduced by Bosch in the early 80s for automotive applications in order to allow communication between intelligent electronic devices mounted on a vehicle but now it has spread in many sectors of the industry.

- The ECU (Electronic Control Unit) is any embedded system in a that controls one or more of the electrical systems or subsystems in the vehicle. In modern car there are a lot of ECUs and this number is constantly growing as the cars are becoming more and more sophisticated. For example, inside a car is possible to find a set of ECUs where the most important is the engine control unit and others that are used for transmission, airbags, antilock braking/ABS, cruise control, electric power steering, audio systems and so on.

- Sensors are components capable of producing data autonomously and then placing them on the BUS. The role of sensors is to detect events or changes in its environment and send the information to other electronics. Before sending data, some operations are needed: the amplification of the output signal, the A/D conversion and then emission of the signal on the bus.

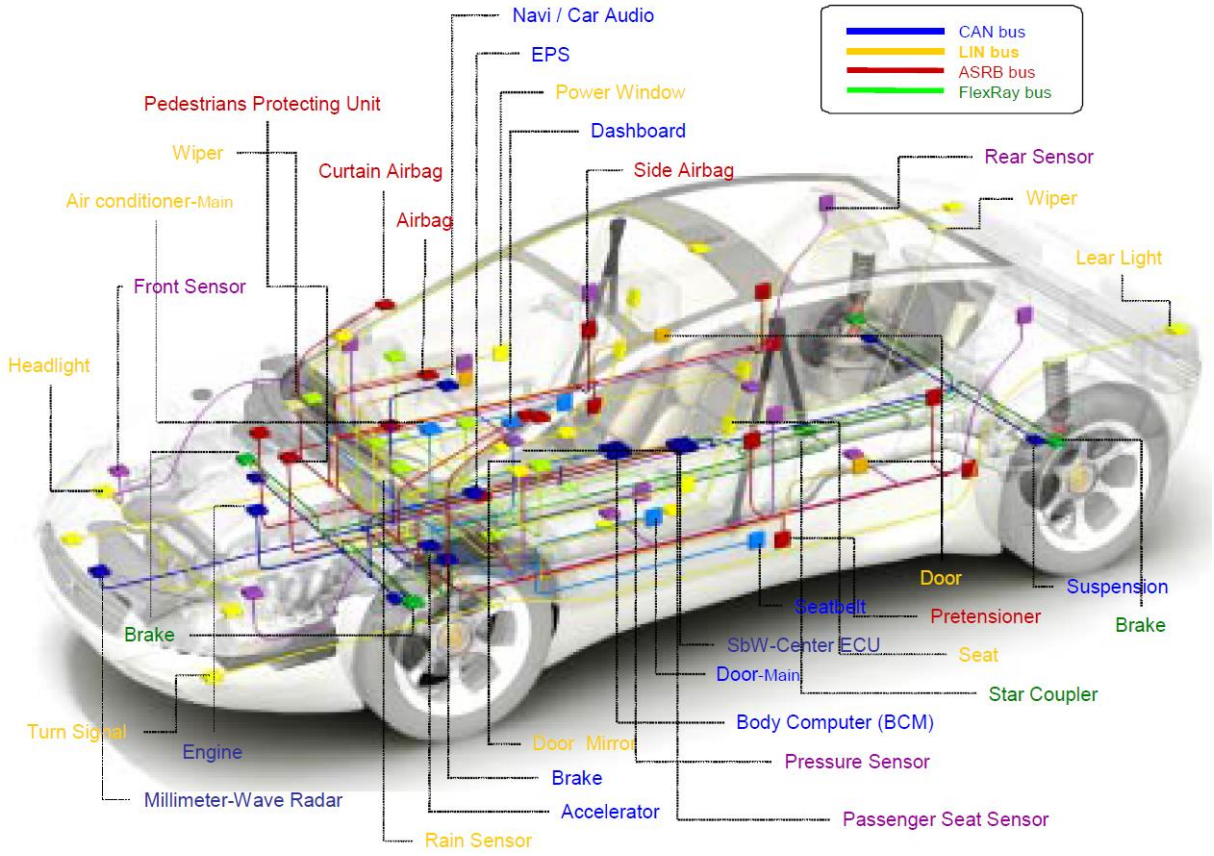

Figure 7.Vehicle network

So, also the infotainment system has his own ECUs. From the architectural point of view an IVI system is composed of a set of linked hardware and software elements. The general architecture of this system is composed of different layers, starting with the hardware layer in the bottom, passing through the operating system layer, the middleware and the application layer to finally arrive in the HMI layer [4]:

- **Hardware Layer:**
  The hardware layer is composed by the CPU with all the necessary hardware and firmware to boot the OS. This layer in addition is equipped with a set of automotive

OEM-specific I/O devices, such as MOST/CAN buses, connected through an industry standard I/O fabric, such as PCI Express.

- **OS Layer:**
  The OS layer is composed by Board Support Package (BSP) and drivers.
  The operating system itself typically manages hardware and software resources and provides applications with common services. The BSP allows the porting of a particular operating system in a particular hardware environment.
  The drivers operate and manages attached hardware devices by providing software interfaces to hardware devices. These drivers are specific to automotive I/O.
  Thus, the operating system can access hardware functions without having to know any detailed information about the hardware currently in use. [5]

- **Middleware Layer:**
  Middleware is defined as the software layer that lies between the operating system and the applications on each site of the system. It includes a rich set of components and interfaces to realize all functional areas of the application layer, such as Bluetooth with support for various profiles and CAN/MOST protocol stacks.
  The Middleware layer provides infotainment-related services to applications. So, this layer helps the implementation of the communication and input/output of data between the application layer and the operating system layer. In this way developers can focus on the particular purpose of their application. [6] [7]

- **Application Layer:**
  The application layer contains a set of applications designed to perform activities or tasks for the benefit of the user and include the ones designed into many mobile Internet devices (MIDs) or handheld devices like Web browsers, calendar, Bluetooth phone, vehicle management functionalities, multimedia entertainment system, and so on. [8]

- **HMI Layer:**
  The Human Machine Interface (HMI) is the main interface of the IVI system with which the user interacts. The main task is to control the display of the IVI system's head unit and has the responsibility to process and react to all user inputs coming into the system, such as speech recognition and touch screen input.

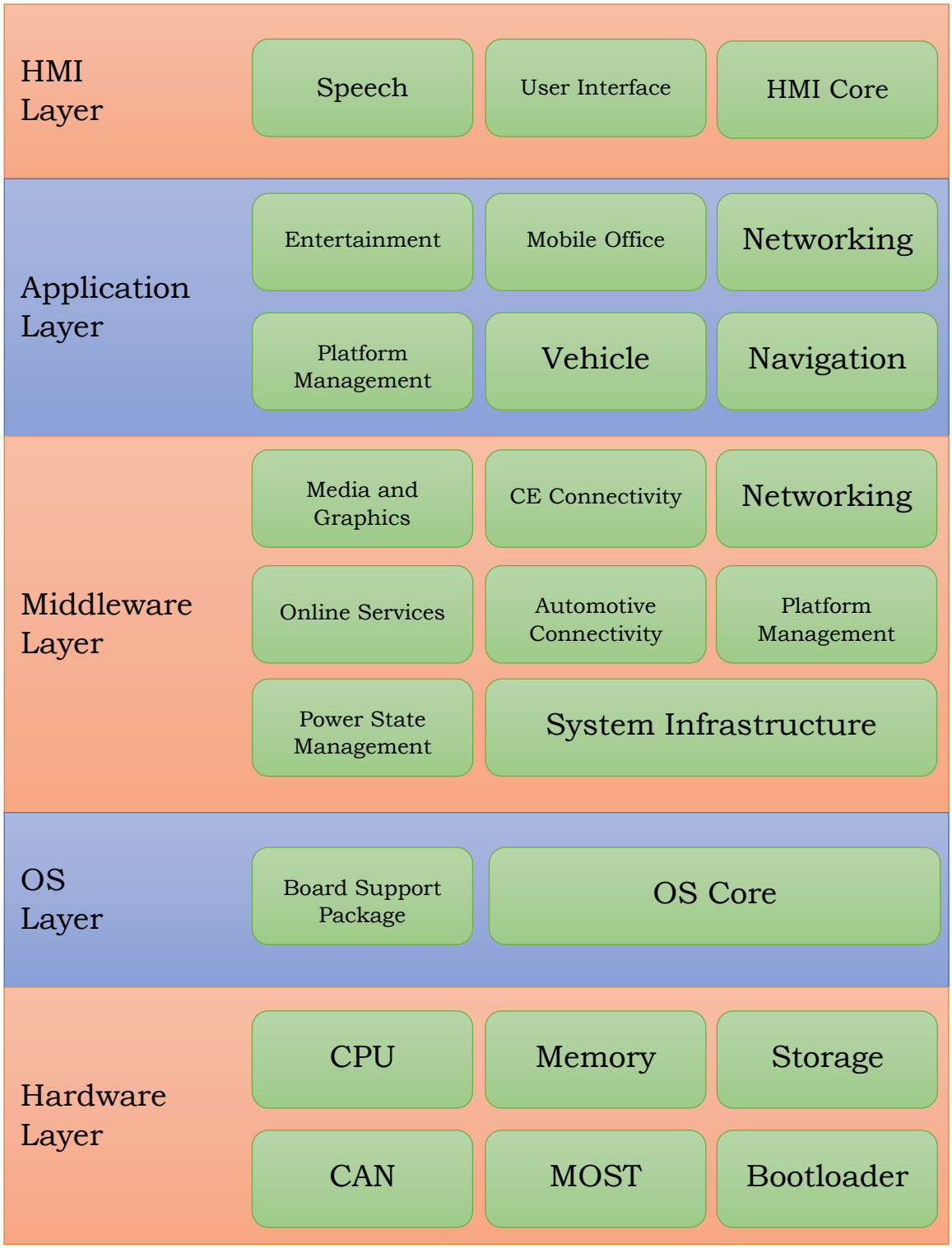

Figure 8. In-Vehicle Infotainment Architecture

2.2. Operating Systems

Currently, in the marker we can identify two main kind of IVI systems: those based on a proprietary software like QNX, and those based on open source software like Linux and Android. QNX is the leader of the market, but with the spread of open source systems this trend in the future is ready to be questioned.

2.2.1. QNX

QNX is a Unix-like real-time operating system (RTOS) developed by BlackBerry that is used mostly in the embedded systems market. It's been implemented into things like in-car control panels, medical devices, and routers.
As regard IVI systems they develop the QNX CAR Platform for Infotainment that is a set of software components with all the functions needed to support automotive industry to create connected infotainment systems with principal automotive technologies.
This platform includes a collection of QNX middleware technology to handle media, web browsing, speech integration, smartphone connectivity, over-the-air (OTA) software updates, Bluetooth and acoustics processing for hands-free calls. It supports also third-party apps for voice recognition, navigation and Natural Language Understanding (NLU).
The platform is made with a modular and extensible architecture, this feature allows to help the personalization for the insertion of new functionalities and the replace of third-party apps in order to satisfy the preferences of the user. [10]
QNX as a different architecture compared to UNIX, Mac OS and Windows, because these have a hybrid kernel while QNX make use of a micro-kernel architecture. Hybrid kernel is a kernel architecture that combines microkernel and monolithic kernel architecture used in computer operating systems. So, there is mixture of the speed and simpler design of monolithic kernel with the modularity and execution safety of microkernel. [11] [12]
Monolithic kernels are implemented entirely as a single process running in a single address space so if something crashes, the entire system crashes. All the operating system tasks such as process management, concurrency and memory management are implemented by means of system calls. Kernel tasks use the processor's most privileged mode.
On the other hand, in microkernel architecture used by QNX, each of kernel tasks has its own address space where it happens, and most of them are run in the processor's least privileged

mode. In this case if something crashes the rest of the system can continue to work, the crashed process can simply be restarted, and operation can continue as normal.
All these characteristics offer greater robustness and protection against programming errors. The operating system itself shares similar APIs and programming methodology with other UNIX and Linux frameworks, but it was built from the ground up with a different underlying architecture.

2.2.2. Linux

Linux is a family of free and open-source software operating systems. These OSs are developed over the Linux kernel. There are loads of distributions each one includes the Linux kernel, supporting utilities and libraries, many of which are provided by the GNU Project, and usually a large amount of application software to fulfil the distribution's intended use.
Linux was originally developed for personal computers based on the Intel x86 architecture but has since been ported to more platforms than any other operating system.
Linux also runs on embedded systems, like devices whose operating system is typically built into the firmware and is highly tailored to the system.
The underlying source code may be used, modified and distributed by anyone under the terms of its respective licenses, such as the GNU General Public License.

Linux is a good solution for building an infotainment system since:

- Linux is independent, and widely accepted in many demanding computing environments.
- The Linux kernel is very extensible, it’s possible to add a hypervisor layer and another operating system (for example Android) can be deployed as a client, making available some of its benefits to the consumer. Others run-time environments can also be added.
- Silicon manufacturers are recognizing Linux as the one OS that can handle both the core elements of infotainment platforms, such as graphics and multicore support, but also new features needed for connected cars: anti-collision technology, voice-activated commands, gesture recognition, and the future requirements of in-vehicle telematics.

For these reasons Linux has been selected as the base operating system for Infotainment from the GENIVI alliance

2.2.2.1. Genivi Alliance

GENIVI is a non-profit alliance whose role is to building middleware to support open source in-vehicle infotainment systems. [13]

The members of this alliance are leading global automakers, electronic companies and vehicle component suppliers.

GENIVI is an open development community producing automotive software components, standard APIs, and a development platform for in-vehicle infotainment and connected vehicle solutions.

The goal is to reduce the challenges automakers and their suppliers face as they deliver the latest consumer requested functionality in their automobiles. GENIVI helps automakers manage the growing complexity and cost of developing IVI software and has successfully introduced open source development models to the automotive industry. GENIVI provides standards and an open source reference that accelerates innovation and delivers value for all links in the connected vehicle supply chain.

In the last years, GENIVI Alliance has been overcome by Automotive Grade Linux as the main driver of Linux-based IVI. [14]

2.2.2.2. Automotive Grade Linux

Automotive Grade Linux (AGL) is a Linux Foundation Workgroup dedicated for the creation of open source software solutions for automotive applications. This collaboration involves automakers, suppliers and technology companies with the target to accelerate the development and adoption of a fully open software stack for the connected car.

The objective is the development of an open platform that has the function of standard for the automotive industry in order to make possible a rapid expansion of new technologies and characteristics. [15]

2.2.3. Android

After the development of Android Auto standard, the system for the synchronization between Android mobile devices and compliant IVI systems, Google has developed an Android-based operating system that fully integrates with the car's on-board computer systems. It is an expansion of its Android Auto project into a full-fledged IVI stack.

Before this some car manufacturers like Hyundai and Honda have ported Android in various IVI systems but without any collaboration with Google. These producers have taken the Android source code (AOSP) and reconfigured it into a car operating system. The result isn't very satisfying because this work requires a lot of development time and when it had released is already out of date.
For example, the Honda Accord of 2017 ships an Android version of 2012, while Hyundai cars (still of 2017) has the Android 2.3 Gingerbread, released in 2010! So, there are very old versions of Android in cars on the market that will never be updated.
The help of Google serves to avoid this waste of time for an ancient operating system. In fact, this support allows car manufacturers involved in the Open Automotive Alliance to have always updated versions of Android.
The last version of Android (Oreo) has been developed with some features, like quicker startup time that improve the use in automotive. [16]
The system provided by Google is called Android Automotive, and carmakers can then build upon with their own branding. It is based on Android Auto, with most of the new features expanding to HVAC and radio controls. Android Automotive has a hardware abstraction layer (HAL) that keeps the interface consistent despite the physical transport layer.
The Vehicle HAL I the base level of Android Automotive, which is controlled by a Vehicle Network Service layer that adds security features. Above this sits a Car Service stack, which is called by a higher-level Car API that interacts with the top application layer. [14]
Even Automotive Grade Linux allows car manufacturers and Tier 1s to customize the User Interface. AGL has the advantage that has started before Android Automotive but this one has the benefit of built-in app ecosystem. Finally, considering strengths of Linux and Android, a very good solution to take into account is the combined use of both together. In this way is possible to mix the advantages of this two, in order to increase the performance of the infotainment system.
Since the work done for this thesis was done on Android in the next chapter we will study in deep this system. First, however, the main components of an infotainment system will be illustrated.

2.3. Main Components

As already said these systems allow the driver to handle all the functionalities provided minimizing the distraction. The Human Machine Interface allows to do this providing a set of different components:

- **Rotary Pad**

  Physical component that allows to handle the items in the display. It helps the driver navigate the various menus and submenus, which is a bit easier than continuously tapping a touchscreen.

Figure 9. Rotary Pad

- **Touch Screen**

  Technology that allows the driver and the passengers to handle the functionalities of the system directly on the screen.

Figure 10. Touch Screen

- **Heads-Up Unit**

  Automotive heads-up display is any transparent display that presents data in the automobile without requiring users to look away from their usual viewpoints.

Figure 11. Heads-Up unit

- **Vocal Recognition Commands**

  Drivers can reach many of the components of the infotainment system thanks to the use of the voice only and the system interprets these commands.
  This feature is very important in order to keep to the driver focused on driving, keeping eyes on the road, hands on the steering wheels and to avoid every possible distraction.

- **Gesture Recognition**

  This is another feature which allows the driver to distract himself as little as possible while he is driving. It consists in the interaction with a machine without a mechanical device, user can communicate with the system through sign languages. Usually this is done using cameras and computer vision algorithms which can interpret the body language.

3. Technologies

3.1. Android

Android is an open source platform designed at first for mobile handsets. It consists in an operating system based on a modified version of the Linux kernel, a middleware and a software platform for the implementation of apps and games. It incorporates the common features found nowadays in any mobile device platform.
This technology is developed by Google through the Open Handset Alliance (OHA) and released in 2007 as beta version (first commercial version in 2008).
Android is open source and for this reason different Android-based phones could have different graphical user interface even they have the same version of the operating system, because developers can modify and customize this OS.
The first implementation was thought for touchscreen mobile devices such as smartphones and tablets, then Google has also developed Android TV for televisions, Android Auto for cars, and Wear OS for wrist watches, each with a specialized user interface. The last release of the operating system (December 2017) is the version 8.1 called “Oreo”. The core Android source code is known as Android Open Source Project (AOSP) and is licensed under the Apache License.
Besides the open source code there is also a suite of proprietary software developed by Google, these are Google Mobile Services (GMS) and consist in a collection of applications and APIs that supports functionality across all Android devices. These apps are licensed by manufacturers of Android devices certified under standards imposed by Google.
Among these there are apps for services like Google Search, Gmail and Google Play, the digital distribution service. [9]

3.1.1. Android Platform Architecture

This open-source software architecture is provided by the Open Handset Alliance (OHA). OHA is a group of 84 technology and mobile companies that includes network operators,

software developers, component and device manufacturers whose objective is to develop open standards for mobile software platforms.
The platform is composed by five main components: Applications, Application Framework, Libraries, Android Runtime and finally the Linux kernel. Now we see each of these in detail, starting from the one at the lowest level. [17] (Figure 12)

The Linux Kernel

The base of the Android platform is the Linux kernel. The Linux kernel is the bottommost layer. It provides the most fundamental system services such as security, memory management, process management and network stack. Using a Linux kernel allows Android to take advantage of key security features and allows device manufacturers to develop hardware drivers for a well-known kernel.

Hardware Abstraction Layer (HAL)

The hardware abstraction layer (HAL) is the layer that is between the physical hardware of a computer and the software that runs on it. So, it has the function of providing standard interfaces that expose device hardware capabilities to the higher-level Java API framework. Since there are different types of hardware components each of these needs a specific interface. The HAL is composed by a set of library modules that implement these interfaces, for example, camera or Bluetooth module. When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

Android Runtime

Android Runtime (ART) is the application runtime environment used by Android OS. ART replaces Dalvik, which is the process virtual machine originally used by Android and performs transformation of the application's bytecode into native instructions that are later executed by the device's runtime environment.
For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART). ART is written to run multiple virtual machines on low-memory devices by executing DEX files, a bytecode format designed especially for Android. If an app runs well on ART, then it should work on Dalvik as well, but the reverse may not be true.

ART include some features like:

- The compilation Ahead-of-time (AOT) and just-in-time (JIT)
- An optimized garbage collection (GC)
- Improved debugger with a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set watchpoints to monitor specific fields

Android also includes a set of core runtime libraries that provide most of the functionality of the Java programming language, including some Java 8 language features, used by Java API framework.

Native C/C++ Libraries

Many core Android system components and services, such as ART and HAL, are built from native code that require native libraries written in C and C++.
The libraries provide core features to Android Architecture. Among all the libraries provided, the most important are:

- libc, the standard C system library tuned for embedded Linux-based devices;
- Media Libraries, which support playback and recording of several audio and video formats;
- Graphics Engines, Fonts, a lightweight relational database engine and 3D libraries based on OpenGL ES.

The Android platform provides Java framework APIs to expose the functionality of some of these native libraries to apps. The Android NDK (Native Development Kit) allows to access to the native platform libraries directly from the native code. NDK is a toolset that lets user implement parts of your app in native code, using languages such as C and C++.

Java API Framework

The Application Framework layer provides the framework Application Programming Interfaces (APIs) used by the applications running on the uppermost layer. Besides the APIs, there is a set of services that enable the access to the Android's core features such as graphical components, information exchange managers, event managers and activity managers, as

examples. Below the Application Framework layer, there is another layer containing two important parts: Libraries and the Android Runtime.
The whole set of features of the Android OS is available through APIs written in the Java language. These APIs are needed in order to create Android apps by simplifying the reuse of code, modular system components and services which include the following:

- A wide View System that can be extended. It allows to create the Using Interface of an app, including lists, grids, text boxes, buttons, and even an embeddable web browser
- A Resource Manager, manage the various types of resources used in the application providing access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that enables all apps to display custom alerts in the status bar
- An Activity Manager that manages the activity lifecycle of applications and provides a common navigation back stack
- Content Providers that manages the data sharing between apps so enable these to access data from other apps, such as the Contacts app, or to share their own data

Developers have full access to the same framework APIs that Android system apps use.

System Apps

Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more. If users want to change these applications can install third-party apps which replace those pre-installed in the device as default apps, except for some as for example the system's Settings app.
The system apps function both as apps for users and to provide key capabilities that developers can access from their own app.

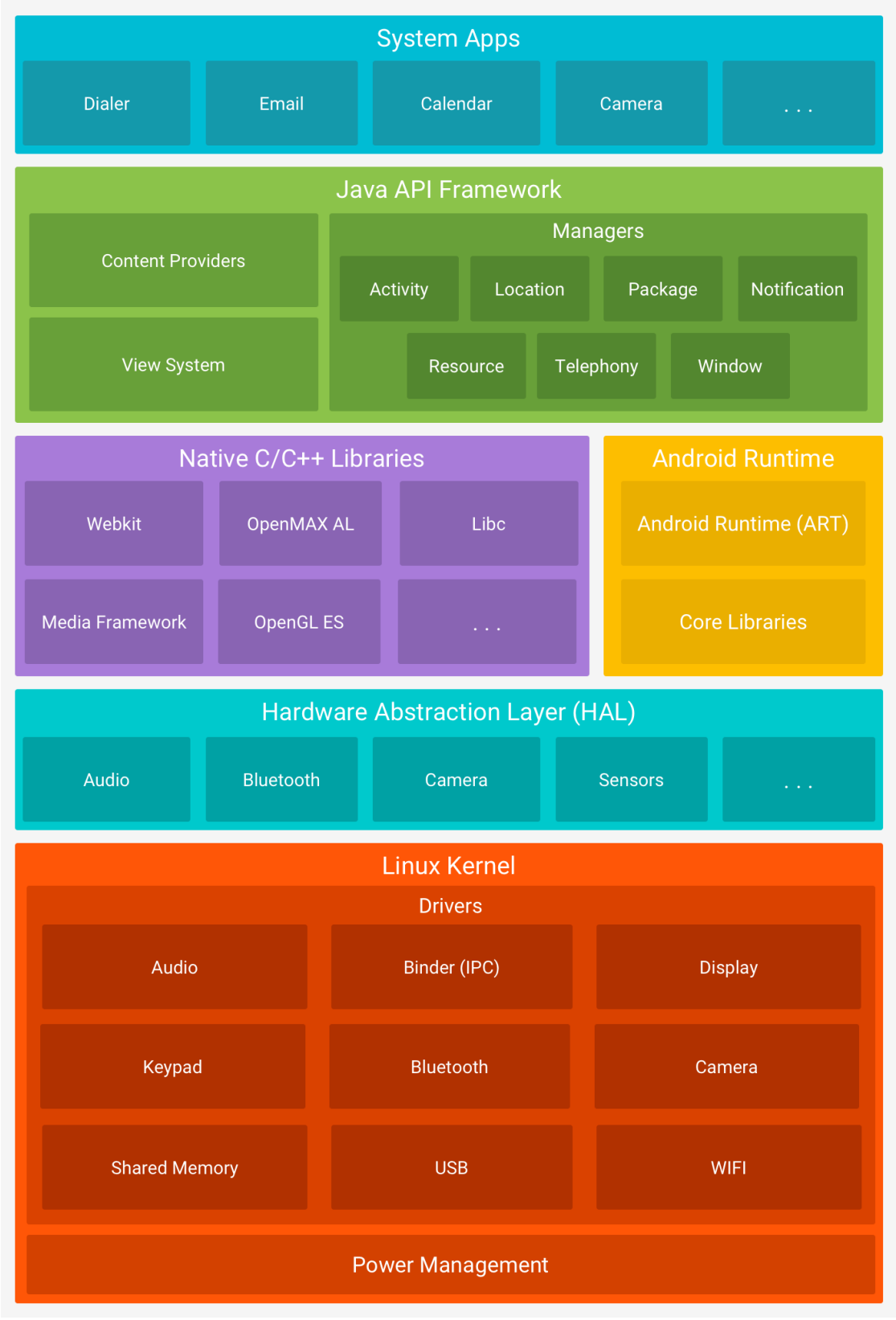

Figure 12. Android software stack

3.1.2. Android Open Source Project

Android is created for a large scale of devices with different design. The primary purpose of Android is to make a platform available for manufacturers and developers with the aim to expand the products with new features creating custom variants of the stack and porting devices to the platform always improving the user experience.
The Android Open Source Project (AOSP) repository offers the source code to achieve these targets. This is free to download, free to change and it's possible to port it in any kind of device.
The result is a full, production-quality consumer product with source code open for customization and porting. So, Android is not only an operating system but this is just a part of what it really is. The fact that it has a very large library of code that developers can modify is a very important characteristic, because it's an advantage for users that can have a lot of applications that are ready to use we they have an Android device.
It's an advantage also for devices manufacturers because they already have an application platform and don't have to build another one on their own.
Furthermore, is an advantage even for Google since increase the number of devices than can use its services. [18]

The architecture is composed by the following elements:

Application framework

The application framework is the software library that sets the structure for developing application software. It provides the framework Application Programming Interfaces (APIs) used by the applications running on the uppermost layer. It is used most often by application developers to write apps. It is important also for hardware developers, because developer APIs are linked to the underlying HAL interfaces and can provide helpful information about implementing drivers. [19]
Besides the APIs, there is a set of services that enable the access to the Android's core features such as graphical components, information exchange managers, event managers and activity managers, as examples. Below the Application Framework layer, there is another layer containing two important parts: Libraries and the Android Runtime.
Many APIs map directly to the underlying HAL (hardware abstraction layer) interfaces.

Binder IPC

The Binder Inter-Process Communication (IPC) is a mechanism that enables high level framework APIs to interact with Android system services. It allows the application framework to overcome the process and call into the Android system services code. At the application framework level, this communication is hidden from the developer and things appear to "just work".

System services

System services are components that act as a link between the functionalities exposed by the application framework APIs and the hardware below. There are two kinds of services, media services (those needed for playing and recording multimedia contents) and system services, like Window Manager, Notification Manager and Search Service.

Hardware abstraction layer (HAL)

The HAL as explained above provides a standard method for creating software link between the Android platform stack and the hardware. In this way there is an interface for hardware producers that which enables Android to not consider the lower-level driver implementations. Using a HAL allow to implement functionality without affecting or modifying the higher-level system.

HAL implementations are packaged into modules and loaded by the Android system at the appropriate time.

Linux kernel

The version of the Linux kernel used is a customized one with some particular features like Low Memory Killer (a memory management system that is more aggressive in preserving memory), wake locks (a PowerManager system service), the Binder IPC driver, and other things needed in mobile embedded platform. These different features don’t change the way in which are developed drivers because are only required for system functionality. So, developers can make drivers in the same way with which typical Linux device drivers are developed.

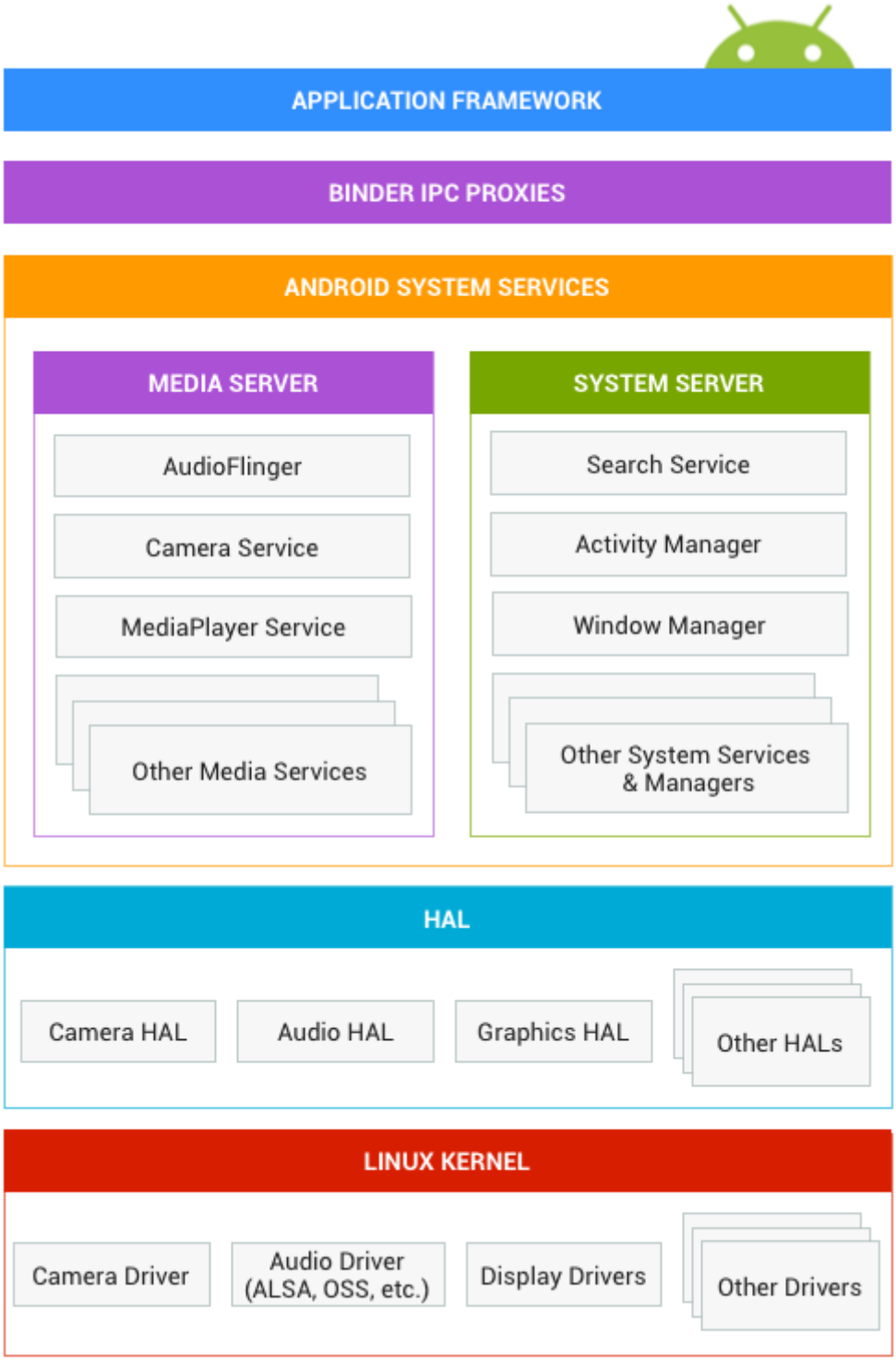

Figure 13. Android Architecture

3.1.3. Project Treble

With Oreo, Google started a project known as Treble which consists in a reorganization of the architecture of the Android OS framework. It is a subdivision of the operative code of Android in order to make it modular. Before Treble in fact the OS framework and low-level software were all part of the same code, because no formal vendor interface exists. The vendor interface is a collection of versioned HAL interfaces, kernel interfaces exposed to user space, and some vendor-specific extensions.

So, when the OS got updated, this low-level software also had to get updated and this required that device makers must update large portions of the Android code. With the separation of these two parts a new stable vendor interface provides access to the hardware-specific parts of Android, so device makers can deliver new Android releases simply by updating the Android OS framework, without additional work required from the silicon manufacturers. So, the Android operating system can be updated without modified the vendor implementation, which can be updated independently at another time.

This will allow hardware manufacturers to work directly and exclusively on the code of their interest without having to intervene on the entire Android code. This therefore does not directly affect the performance or usability of the device but simplifies the steps necessary for the release of software updates, which may be timelier. All the with Android 8.0 or higher can take advantage of this new architecture.

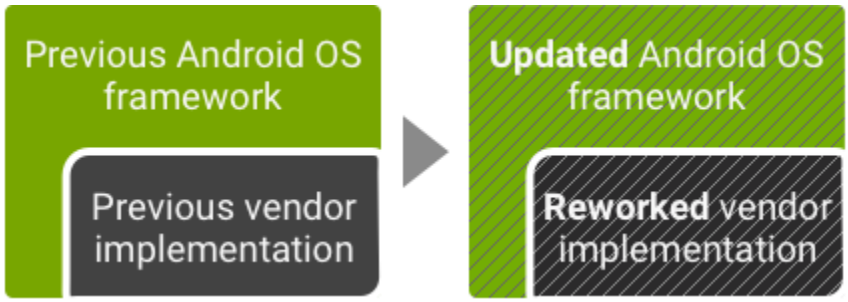

Figure 14. Before Treble

Figure 15. After Treble

3.1.4. Hardware Interface Definition Language

The HAL interface definition language (HIDL, pronounced "hide-l") is an interface description language (IDL) to specify the interface between a HAL and its users. It allows specifying types and method calls, collected into interfaces and packages, HIDL is a system for communicating between codebases that may be compiled independently.

The HIDL separates the vendor implementation (device-specific, lower-level software written by silicon manufacturers) from the Android OS framework via a new vendor interface. Vendors or SOC makers build HALs once and place them in a vendor partition on the device; the framework, in its own partition, can then be replaced with an over-the-air (OTA) update without recompiling the HALs.

In Android versions prior to 8.0 HAL-module interfaces were defined as simple C headers. The syntax of HIDL will look familiar to C++ and Java programmers, though with a different set of keywords. HIDL also uses Java-style annotations.

It enables the Android framework to be replaced without rebuilding the HALs. HALs will be built by vendors or SOC makers and put in a partition on the device, enabling the framework, in its own partition, to be replaced with an OTA without recompiling the HALs.

3.2. Android Automotive

Android Automotive is an Android-based operating system designed for vehicles. Unlike the standard version of the OS, this is much more tightly integrated with the characteristics of a car. As with many Android projects, there are different kinds of auto apps. However, standard android APIs and services don't change in automotive.
Depending on the settings, it can even control various sensors and switches in the vehicle.
It could be used in different ways, as the only operating system but also together with another operating system.

So, there are many options for an infotainment system:

- Running Android directly on a System on Chip as the unique operating system
- Using Android in conjunction with another operating system through a hypervisor
- Utilizing Android via a Linux Container

Many car subsystems are interconnected with the infotainment system with many kinds of bus. Every producer may use different types of bus and protocols, most important are Controller Area Network (CAN) bus, Local Interconnect Network (LIN) bus, Media Oriented Systems Transport (MOST) and as well as automotive-grade Ethernet and TCP/IP networks such as BroadR-Reach. [20] [21]

3.2.1. Android Automotive Architecture

Android Automotive is the core Android operating system that has been optimized and extended into a built-in OS for automotive infotainment systems. Android Oreo is the first public Android version for automotive from Google.

This version of the operating system is an "Android Compatible" version of Android, a device is considered compatible if respects two main prerequisites:

- Satisfy all the requirements of the Compatibility Definition Document
- Pass all test of Compatibility Test Suite

If a device is compliant with these two conditions, it can have branded Android.
So, the architecture is substantially the same of the standard version of the operating system with the addition of all the features needed for automotive. In figure 12 is possible to see the

components that belong to each layer of the classic Android stack while in figure 15 are listed all the new characteristics added for Android Automotive.

In particular, the most important features are:

Vehicle Network Service

A simple native service that allows access to vehicle network (like CAN bus) from all other components, including Car Service and other Android HALs. Access is abstracted into Vehicle HAL and restricted to system components only (non-system components such as third-party apps should use car API instead). It controls Vehicle HAL with built-in security. It provides native and Java APIs. Other HAL can access VNS to access vehicle network and should be launched before any components that rely on it are launched. VNS depends on Service Manager. To start it very early, both should be added to boot image and all libraries should be changed to static.

Garage Mode

Garage mode is an innovative feature that allows the update of the apps when the car is parked in garage. The updates are downloaded when the car isn't running, and the result is that the IVI boot won't be slowed by app updates when you start the car in the morning.

Vehicle HAL

The Android Automotive hardware abstraction layer (HAL) provides a consistent interface to the Android framework regardless of physical transport layer. This Vehicle HAL is the interface for developing Android Automotive implementations. This interface is the between the car and the Vehicle Network Service and defines the vehicle properties that OEMs can implement.

Google declares that system integrators can implement a Vehicle HAL module by connecting function-specific platform HAL interfaces (for example HVAC) with technology-specific network interfaces (e.g. CAN bus). They can also integrate a dedicated MCU (Micro Controller Unit) running a proprietary RTOS for CAN bus access or similar within the Android Automotive framework, which may be connected via a serial link to the CPU running Android Automotive. Instead of a dedicated MCU, it may also be possible to implement the bus access as a virtualized CPU. It is up to each partner to choose the

architecture suitable for the hardware if the implementation fulfils the interface requirements for the vehicle HAL.
It contains property metadata (for example, whether the vehicle property is an int and which change modes are allowed).

The vehicle HAL supports three levels of security for accessing data:

- System only (controlled by VNS)
- Accessible to app with permission (through Car Service)
- Accessible without permission (through Car Service)

Direct access to vehicle properties is allowed only to selected system components with Vehicle Network Service acting as the gatekeeper. Most applications go through additional gatekeeping by Car Service (for example, only system applications can control HVAC as it requires system permission granted only to system apps).
The Vehicle HAL interface is based in accessing (read, write, subscribe) a property, which is an abstraction for a specific function. For example, Vehicle speed, Temperature Set, etc.

Car Service

Implements vehicle specific policy including audio management, power management, etc.

Car API

Collection of car-specific APIs relevant only for automotive. Implemented by Car service. Contains the APIs such as CarHvacManager and CarSensorManager.

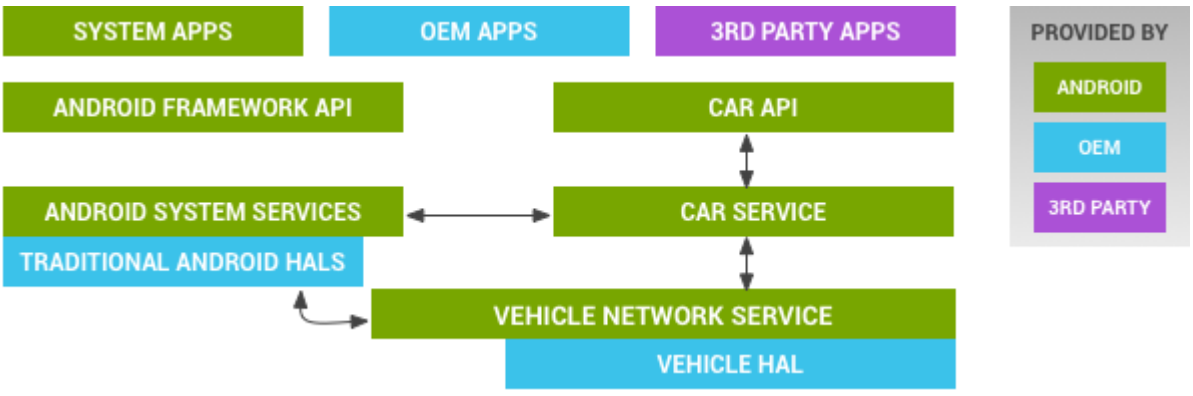

Figure 16. Android Automotive Architecture

Figure 17. Features added for Android Automotive

Since the objective of Android is to have an open platform with which developers can build always new applications. There is set of guidelines in the Android Compatibility program that defines technical details of the Android platform and provides tools for OEMs to ensure developer applications run on a variety of devices.

It is also supplied the Android Software Development Kit which provides built-in tools for developers to clearly state the device features required by their applications. In Google Play are shown the applications only to the devices that can support them.
Considering that device producers can't write all the software needed by users third-app developers are required to realize applications that users wish. With AOSP the development is easy and open as possible.
Building a compatible device allow to have the benefit that comes from the fact that the more applications have to run on those devices. Developers at this point are encouraged to always create new apps. [22]

3.4. Google Automotive Services (GAS)

Google Automotive Services are the automotive correspondent of the Google Mobile Services which are all the apps and services, that are proprietary and licensed by Google, these are provided in order to increase the user experience of the user.
So, Google did the same thing for automotive in order to bring their best apps and services to Android-based automotive infotainment system.
Among these there are:

- **Assistant**
  The automotive version of the Google Assistant a voice-controlled smart assistant that lets the user to perform things like send messages through voice commands. This version has vehicle specific functionalities in addition to the standard ones. The Assistant allows the driver the handle communication, navigation, media player in-car actions just using the voice.

- **Maps and navigation**
  Google Maps is the mapping solution for connected vehicles. It is composed by two main components: Navigation, which includes satellite imagery, street maps, 360° panoramic views of streets (Street View), real-time traffic conditions (Google Traffic) and the Vehicle Mapping Service (VMS), a property introduced in Android 8.1 intended for use only in Android Automotive implementations. It consists in a dynamic map data exchange between an Android Automotive implementation and the

underlying vehicle hardware responsible for managing onboard mapping data. Used for vehicle assistance and autonomous highway driving.

- **Play Store and Core Services**
  The Google Play Store designed for in-car experience where is already possible to find a lot of Automotive applications driver distraction tested (that can be safely operated whilst driving).

3.5. Android Auto

Unlike Android Automotive, Android Auto is not an operating system, it is just a mobile app developed by Google that allows the mirroring of an Android smartphone into the cars head unit. Android Auto is powered by a smartphone using an application.
The phone projects an interface in the car head unit, so it's possible to handle maps, phone calls, play music and all compatible apps.
So, all the stuff like interface, processes and apps run in the smartphone of the driver and the car screen works like a remote display. The phone can be connected via USB cable or wireless.
Android Auto isn't open source, there is a set of proprietary apps inside the phone that are transmitted in the dashboard.
The interface is customized so that it fits the dashboard, and Google is committed to making sure that this follows the best safety practices for computing while driving. So, the user interface of Android Auto doesn't have anything in common with the UI of a smartphone, in fact it is very simplified with larger touch targets, bigger text designed and a focus on voice commands. There is no app drawer, settings, or pull-down notification panel.
There is a status bar on the top of the screen which shows the standard information of a device like: battery level, time, Wi-Fi and so on.
The navigation bar instead is located in the bottom, unlike a smartphone or a tablet there aren't the standard functions like Back, Home, and Recent Apps. This bar lists actual applications in the Android Auto Interfaces, which are Overview, Google Maps, Phone, Media and Car. This allow user to get to every function of Android Auto in a single click, a big improvement over the stock system, which usually requires two-to-three taps to get anywhere (Hit back/home, then the app drawer, then the app).

In particular, the functions are:

- **Overview** is the "home screen" of Android Auto. It's basically a combination of your notifications and a very limited subset of Google Now cards.
- **Google Maps** is exactly what you'd expect: best-in-class, always up-to-date maps and navigation with spoken directions and a searchable POI database.
- **Phone** is contacts, a dialer, and call logs from your phone.
- **Media** is, by default, just Google Play Music, but third-party apps can be installed, allowing this icon to switch to music from other services, podcasts, or whatever other audio source someone feels like building.
- **Car** is the last icon, a little speedometer. Right now, it only lets you exit Android Auto and return to the stock infotainment system. Our experimenting with developer mode leads us to believe this will eventually be a place for OEM-specific apps. Right now, it's just a single exit button.

There are other two apps that not are listed in the bar, the first one is the Voice Search that allows to use vocal command to do something like navigate to an address, make a phone call and so on. It is available through a button on the screen or via a physical button in the steering wheel. The Messages app instead shows notifications as cards on the overview screen, user can tap on them and have them read to him and then can reply by voice.
It’s not possible to customize the user interface and only two kinds of third-party apps are supported: for Media and Messages.
Android Auto is also integrated with the car hardware through steering wheel buttons and gets access to some of the car's sensors. [23]

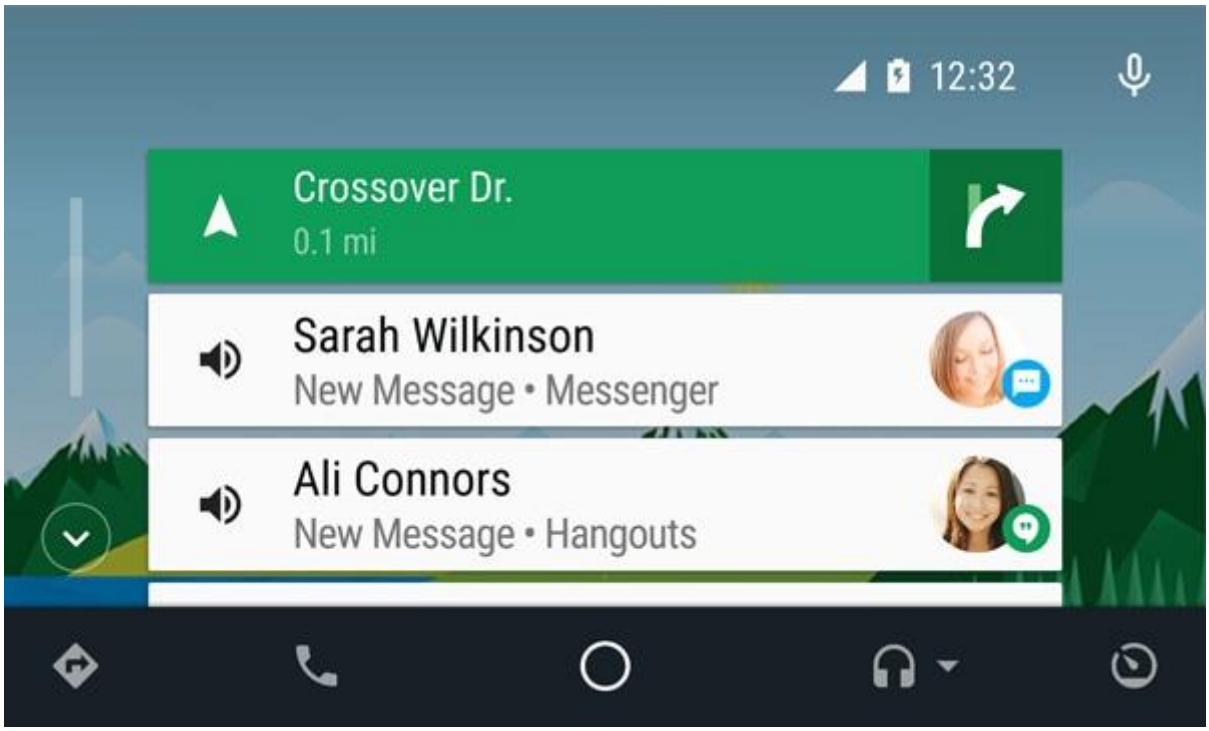

Figure 18. Android Auto

4. Hardware and toolchain

4.1. Hardware

The hardware used for the implementation is composed by Snapdragon Automotive Development Platform (ADP) based on the Qualcomm Snapdragon S820Am processor. It provides OEMs and ecosystem partners with access to Qualcomm's high-performance automotive infotainment, modem and advanced driver assist platform for developing, testing, optimizing and showcasing next-generation in-vehicle infotainment solutions.

The ADP provides an optimized application development environment for rapid deployment of high performance and power efficient connected automotive infotainment offerings.

The S820Am Snapdragon Automotive platform includes a custom-built 64-bit Qualcomm Kryo CPU, custom-built Qualcomm Adreno 530 GPU for virtualization advantages, and Qualcomm Hexagon 680 DSP Vector eXtension to stream high-definition videos seamlessly onto multiple displays. The ADP features rich connectivity through the X12 LTE modem to support Category 12 speeds up to 600 Mbps download, as well as vehicle sensor integration and computer vision to support driver assistance using the Snapdragon Neural Processing Engine. This development platform was supplied together a 10.1-inch capacitive touchscreen from Lilliput with a maximum resolution of 1920x1080 pixels. [24] [25] [26]

Figure 19. Snapdragon platform with Lilliput monitor

4.2. Software

4.2.1. Android Studio

The IDE used for the development is Android Studio that is the official IDE for Android app development, based on IntelliJ IDEA (Java IDE for developing computer software developed by JetBrains), it includes everything necessary to build Android apps. Furthermore, offers many others features to improve the developers work when building apps. [27]
Android Studio divides projects in modules each one with source code files and resource files, they include the Android app modules, library modules and Google App Engine modules.
All the build files are visible at the top level under Gradle Scripts and each app module contains the following folders:

- **manifests**: Contains the `AndroidManifest.xml` file.

- **java**: Contains the Java source code files, including JUnit test code.
- **res**: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

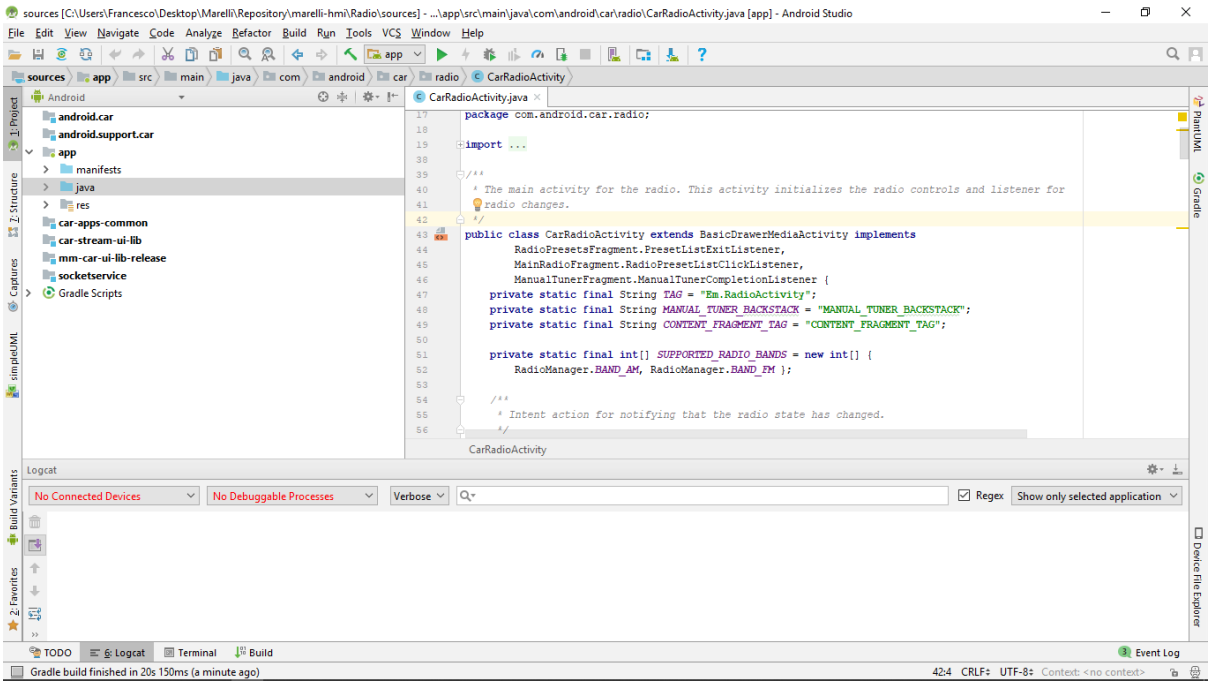

Figure 20. Android Studio

4.2.2. Software Development Kit

The Android SDK (software development kit) is a set of development tools used to develop applications for Android platform. The Android SDK includes the following:

- Required libraries
- Debugger
- An emulator
- Relevant documentation for the Android application program interfaces (APIs)
- Sample source code
- Tutorials for the Android OS

It enables developers to create applications for Android OS. SDK tools are platform-independent and are required to create any Android app, regardless the version you are developing for.

4.2.3. Testing

With the aim of testing the apps have been used some tools useful for installing apps, fixing bugs and accessing to the Unix shell.

4.2.3.1. adb tool

Since is not possible launching the apps directly from Android Studio, for each application it was generated a signed file with extension `.apk`. Then it was installed via command-line interface using the adb tool.

```
adb install -r app.apk
```

Android Debug Bridge (adb) is a command-line tool that allows developers to communicate with a device. `adb` is included in the Android SDK Platform-Tools package. This utility simplifies a set of device actions like install an application or debug it.
The command `adb shell` allows the access to the Unix shell and can be used for running some commands on the device. [28]

4.2.3.2. Logcat

The Logcat is a window of Android Studio that displays system messages. Through the `Log` class the developer can add messages in the classes and saw these in the logcat during the life cycle of the app. It displays messages in real time and keeps a history, so you can view older messages. It's possible to create filters, choose how much information show in messages, set priority levels, display messages produced by app code only, and search the log in this way user can display just information in his/her interest.
When an app throws an exception, logcat shows a message followed by the associated stack trace containing links to the line of code. [29]

4.2.4. Version Control

The project was developed in a team where several people work, so it was necessary to have a version control system in order to coordinate and synchronize the resources among the developers in this way there is consistency in the evolving product.

4.2.4.1. Git

Git is the most commonly used version control system today and is quickly becoming the standard for version control. Git is an open source project originally developed in 2005 by Linus Torvalds, the creator of the Linux.
Git has a distributed architecture, so is an example of a distributed version control system. This means that each user has his own local copy of code is a complete version control repository. These fully-functional local repositories make it is easy to work offline or remotely. A user commits his work locally, and then sync his own copy of the repository with the copy on the server. This paradigm differs from centralized version control where clients must synchronize code with a server before creating new versions of code. In this way every developer's working copy of the code is also a repository that can contain the full history of all changes. [30]

5. Implementation

5.1. Development

As explained in the introduction the objective was the realization of a set of applications in order to have the commons functions of an infotainment system. Some of these applications are already present in AOSP, so it was being only customized with all the features requested by the commissioner, while others were made from scratch as Android native application, always following the specifications of the commissioner. However, we did not use the version of AOSP released directly from Google, but another version customized by Qualcomm that is the same of the original but with some adjustments in order to best fit the board used for development.

5.1.1. Overview

5.1.1.1. Specifics

The Overview app works as a main view of the dashboard. This application shows a view to the other apps installed as Media or Radio (it depends on which one is currently operating), Phone, Maps, MyCar and a box that display the weather and the city.
The Media/Radio box when Media is active displays the cover art of the album in the background, the title of the song, the artist and a play/pause button.
Instead when Radio is active in the background there would be the channel of the station, and in the box below the name of the station and the current program.
The Phone box if no device is connected shows a message which invites the user to pair his/her phone or tablet. When the device is connected in the box appears the name of the user and the profile picture. During a call all the stuff related to this: time, end call button, speakerphone.
The MyCar box displays all the pages of the MyCar app, so let's see all the things related to the driving mode, seat belts, doors, tyre pressure and other information about the status of the vehicle.
The Maps box shows the journey that is being travelled.

The weather box shows the actual weather with temperature and the city in which you are located.

5.1.1.2. Design

This app is structured as a set of `CardView`. A `CardView` is a `FrameLayout` with a shadow and a background with rounded corners. All these cards have different dimensions. The main container consists in a `LinearLayout` (the orange one in the Figure 21) which allows to have a specific orientation of the elements and to set a weight to each child inside it in order to give proportionality to the inside views so, greater is the weight and greater is the space that this child occupies respect to the other children.
So, this layout has `android:orientation="horizontal"` and contains two children, the first one with `android:layout_weight="13"` which will contain other layouts and the second with `android:layout_weight="5"` which will contain the card which shows the view of the Maps application.
This internal layout is also a `LinearLayout` (the green one in the Figure 21) with two children but with `android:orientation="verical"` and `android:layout_weight="7"` for the first one and `android:layout_weight="4"` for the second. These too are in turn `LinearLayout` (the yellow one in the Figure 21) and contains the remaining cards: Media/Radio card on top-left, MyCar card on top-right, Phone card on bottom-left and the Weather card on bottom-right. Every layout is included in his card through the `include` tag.

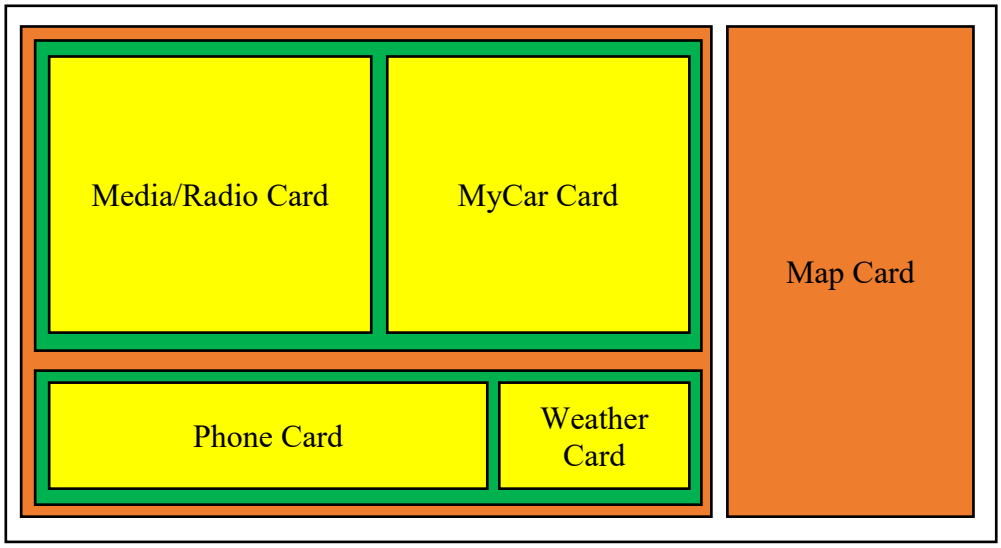

Figure 21. Overview app structure

Each card has its own style:

- `card_multimedia`

  If the media player is activated in the background there is an `ImageView` with the cover-art of the album if present, otherwise there is a sample image. In the bottom there a `LinearLayout` which contains two `TextView` with the info of the current playing as title and artist. There is also a play/pause `Button`.
  While if the Radio is playing in the background appear the channel of the station as a `TextView` and in the bottom remains the `LinearLayout` with two `TextView` but in this case it shows the name of the station and the current program on air.

- `card_phone`

  This card if no phone is connected shows a `TextView` with the message “Please connect a Device” together with a `Button` which allows to open the Bluetooth settings page. When a phone is paired it is splitted in two equal parts obtained with a `LinearLayout` that contains two `RelativeLayout` with `android:layout_weight="1"`. On the left there is an `ImageView` with the profile picture of the user and `TextView` with the profile name. On the right there is a `ListView` with the recent call list.
  When there is an incoming call obviously the Phone application is opened automatically, but if during the call the user opens the Overview app the content of the `card_phone` changes showing information about the current call. The left side remains the same with the profile picture and the name, in the right instead there another `LinearLayout` with `android:orientation="vertical"` where on the top there are three `ImageButton`, one for …, another for the speakerphone and antoher for …
  On the bottom there is a `Chronometer` which display the time of the current call and a `Button` for ending the call.

- `card_weather`

  Card composed by a LinearLayout were on the top there is an ImageView with an icon that shows the weather (a cloud if is cloudy, a sun if sunny and so on) and a TextView for the temperature in degrees. On the bottom other two TextView one for the city where you are and another where the weather is written.

- `card_car_status`

  Inside this card there is a ViewPager that allows the user through the various pages that make up the MyCar application. Each page has his own Layout which shows a summary of the information that can be found within the MyCar app.

- `card_map`

  In this card there is no relevant element from the point of view of the layout, it just contains a portion of the of view that is possible to see opening the Map application.

In the example is possible to see the layout which contains the Media card and MyCar card.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="7">

    <android.support.v7.widget.CardView
        android:id="@+id/media_card"
        style="@style/MainCard"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1">

        <include
            android:id="@+id/card_media_id"
            layout="@layout/card_media"
            app:colorsViewModel="@{colorsViewModel}" />

    </android.support.v7.widget.CardView>

    <android.support.v7.widget.CardView
        android:id="@+id/car_status_card"
        style="@style/MainCard"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1">

        <include
            android:id="@+id/card_car_status_id"
            layout="@layout/card_car_status"
            app:colorsViewModel="@{colorsViewModel}"
            app:statusSelectedPage="@{statusSelectedPage}"/>

    </android.support.v7.widget.CardView>

</LinearLayout>
```

5.1.1.3. Logic

This app is structured with a `MainActivity` which handle all the lifecycle of the application, in particular contains the method `onCreate` that creates the binding with the layouts, set the colors and creates a new instance of the object `MainViewModel`. The class `MainViewModel` has the task of creating the objects corresponding to all the cards belonging to the application, so it creates `MediaCardViewModel`, `PhoneCardViewModel` `RecentCallsViewModel` `CarStatusViewModel` `MapCardViewModel`. Each of these class extends the class `CardViewModel` which implements the common methods that have to take all the cards, then each card implements his own methods in order to handle his logic.

The population of the fields that belongs to each card is a task that is performed by another app called Stream. This app is composed by a `StreamService` that manages all the `StreamCard` being generated by the system and notifies when these are available. A `StreamCard` is used for the communication between various components and it is identified by a unique ID and a type. So, the `SystemService` add the cards and in the Overview, these are bound based on their type to their respective views.

Each card has set an `OnClickListener` which launches an `Intent` that allows to start the corresponding application. In the following the example of the Maps application:

```
container.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        onActionContainer();
    }
});

private void onActionContainer() {
    Intent launchIntent =
context.getPackageManager().getLaunchIntentForPackage("com.mireo.arthu
r.android.magnetimarelli");
    if (launchIntent != null) {
        context.startActivity(launchIntent);
    }
}
```

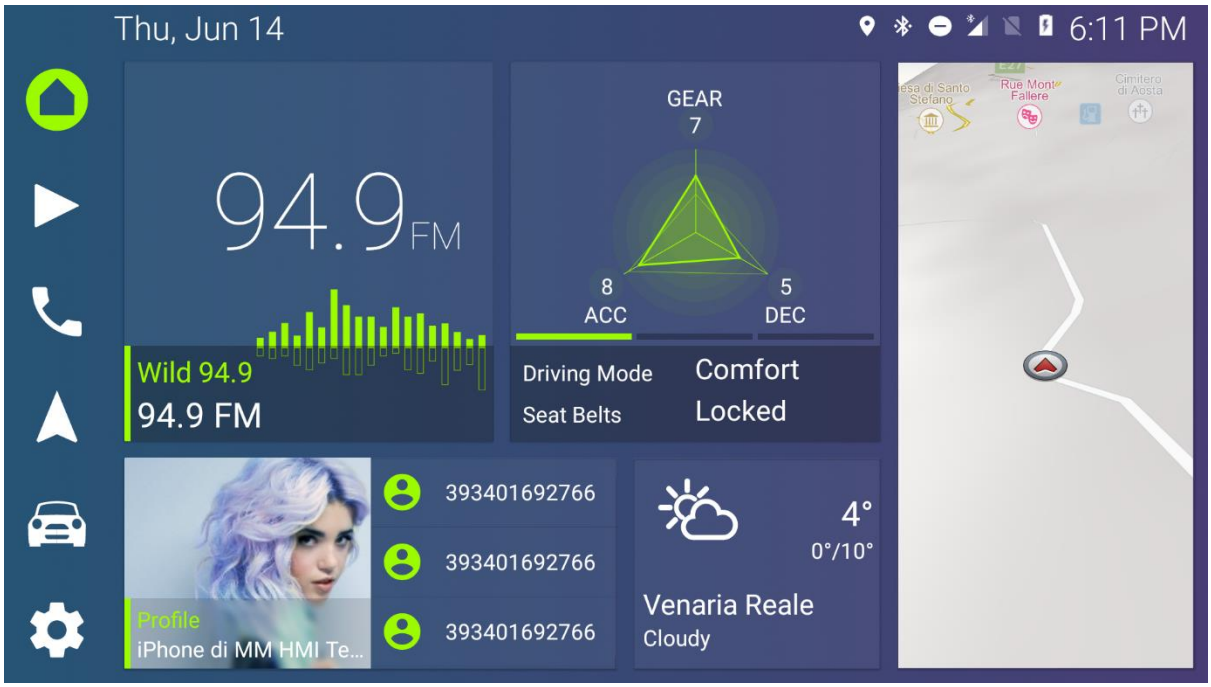

Figure 22. Overview App

5.1.2. Radio

5.1.2.1. Specifics

The Radio app has all the functionality of a common radio.

- Displays the name of the station and the program currently playing.
- Gives the user the possibility to add a radio station to a list of favorite channels through a preference button. This is also possible using a preset bar which provides the addition/removal by a long click on the buttons. If the channel selected is present in the preset bar it become highlighted as we should expect.
- On the right shows the list of all the channels discovered until this time, the user can scroll it and select a new channel.
- Offers buttons for seek back, seek forward and mute/unmute.
- Provides a manual search of the channels, available in a new page opening the drawer.

5.1.2.2. Design

This app is composed of two views the first one is a page that is visible only when the Radio is not available and shows a `TextView` with an error message and an `ImageView` with an error icon. The main view instead is composed by a `LinearLayout` (the orange one in the figure 23) with the property `android:orientation="vertical"` so the layout is divided in other two layouts one over the other. The upper section is another `LinearLayout` (the green one in the figure 23) obviously with `android:orientation="horizontal"` with on the left another `LinearLayout` (the yellow one in the figure 23) for the info about the current station, at the top of this there are two `TextView` with the name of the program, the name of the station and a `Button` for adding that station to the favorites list. At the bottom there are three `Button` aligned whose function is to seek backward, to mute/unmute the volume and to seek forward.

On the right there is a `RecyclerView`, which is a widget needed to display a scrolling list of elements based on large data sets. This `RecyclerView` is populated with the list of radio channels found up to that moment.

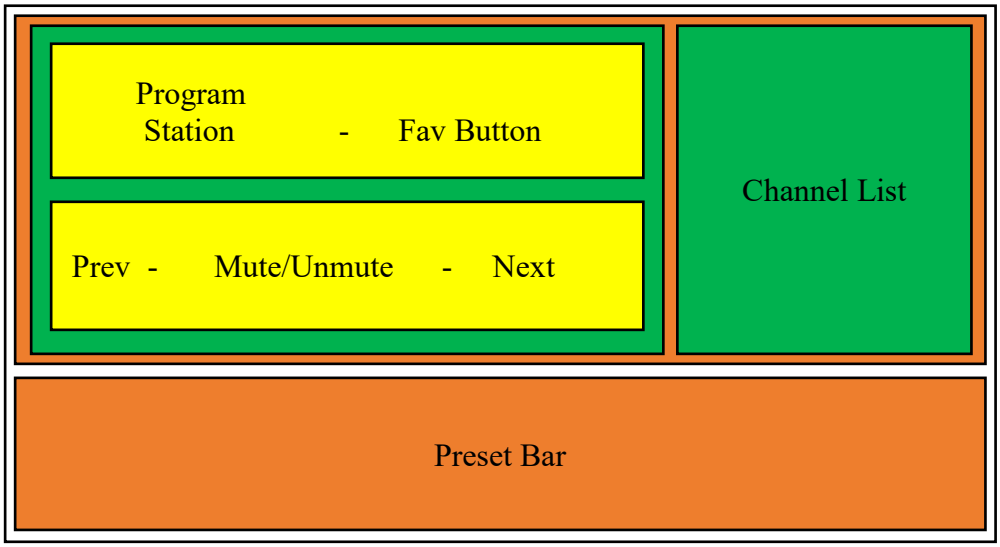

Figure 23. Radio app structure

On the bottom there is the preset bar that is a `CardView` which includes another layout, and this is also a `LinearLayout` that has same behaviour of a `RadioGroup` but this was choosen because is easier to customize. Inside this the buttons are realized once again with

LinearLayout with the attributes android:clickable=”true” and android:longClickable=”true” (that are used to select and save a preset) with two TextView that as before contains the name of the program, the name of the station. Follows an example of a preset button:

```
<TextView
        android:id="@+id/button0txt1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:layout_weight="1"
        android:maxLines="1"
        android:ellipsize="end"
        android:text="+"
        android:textAlignment="center"
        android:autoSizeTextType="uniform"
        android:autoSizeMinTextSize="40sp"
        android:textColor="@color/colorPrimaryDark"
        android:paddingTop="5dp"
        android:textStyle="bold"/>

    <TextView
        android:id="@+id/button0txt2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:layout_weight="1"
        android:textSize="40sp"
        android:ellipsize="end"
        android:textAlignment="center"
        android:textColor="@color/colorPrimaryDark"
        android:visibility="gone" />

</LinearLayout>
```

5.1.2.3. Logic

The class CarRadioActivity is the main activity of the application. It has the role of initializing the radio controls and listener for radio changes. Then this activity creates a new instance of the MainRadioFragment that is the fragment that functions as the main display of the app. It shows the information relative to the current radio station, the controls that allows the user to switch to different radio stations and initializes the RadioController. RadioController is the class which has the task of updating the

user interface of the fragment retrieving information about the current radio station from the `RadioManager` and updating the display based on that information. The `ManualTunerFragment` the is started from the drawer is the fragment that allows the user to manually input a radio station to tune to. Inside it is created the `ManualTunerController` that has the role of initializing the various buttons in the manual tuner screen. The class `RadioStorage` manages persistent storage of various radio options, to achieve this result are used `SharedPreferences` (used to store primitive data in key-value pairs) and a `RadioDatabase`. This last class extends `SQLiteOpenHelper` which is a class that contains a useful set of APIs for managing the database. Using this class is possible to obtain references to the database, the system performs the potentially long-running operations of creating and updating the database only when needed and not during app startup. It is necessary to call only the methods `getWritableDatabase()` or `getReadableDatabase()` (this is done in background threads because they can be long-running) This database is used to save the list of the presets that populate the `RecyclerView` on the right part, this solution was chosen because the list of the channels saved as favorites must be available even when the system is restarted.

`OnClick` and `OnLongClick` are used for selecting and saving/unsaving channels in the preset bar. The `OnClick` is enabled only when a channel is saved in the preset bar, the `OnLongClick` is always enabled. The preset list is synchronized with the favorite button, so when a channel is added/removed in the preset bar the button is filled/unfilled and when is added/removed from the favorite button it appears/disappears in the bar. The scroll view is populated with a recycle view adapter. The logic that manages the behaviour of the mute/unmute, previous and next buttons was already present in the app released by Google, although the mute/unmute buttons were considered as play/pause button but in a FM radio they have little sense of existing. The name of the station, the current program and the channel are metadata.

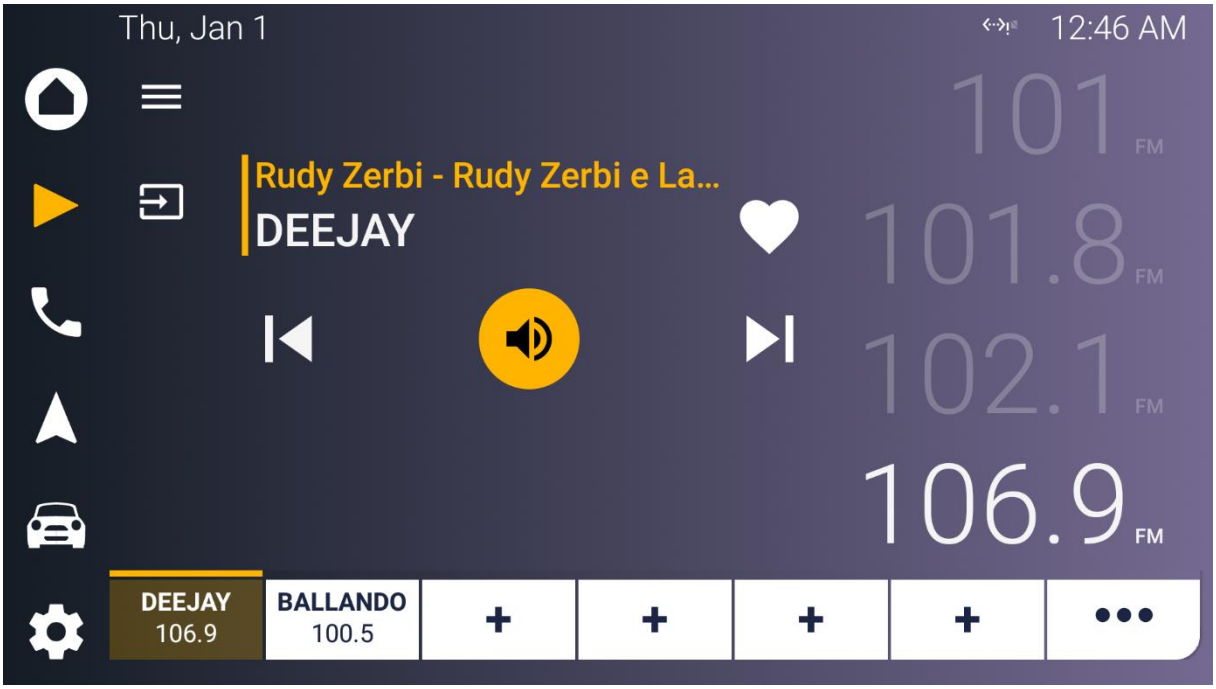

Figure 24. Radio App

5.1.3. Multimedia

5.1.3.1. Specifics

This is the app for playing music. The main view of this app is characterized by rounded corners on the right side and in a blurred view of the album cover art which is the background. Instead in the foreground it displays on one side the album cover art (this time not blurred), on the other side the name of the song that is playing in that moment and the artist, with the possibility to add this song to a list of favorites songs. Then display the media buttons like play/pause, next song, previous song, shuffle, repeat. Other things that appears in the view are the information about the next song that will be played, the timer of the song that is in playing and the relative seek bar. Clicking on the drawer then is possible to access the folders in which the multimedia files are organized by artist, album and track. All multimedia applications inherit the graphics from this Media app so installing 3rd party apps like Spotify or playing music from Bluetooth the same view is maintained.

5.1.3.2. Design

There are two main layouts, the first one is shown when there is still no multimedia content being played, and this is composed by a `RelativeLayout` where inside it is centered

`ProgressBar` with an `ImageView` with an error icon and in the bottom a `TextView` with a message that invites the user to open the menu in the drawer in order to select something to play. The other layout appears when the app is working and is composed by a `CardView` (the orange one in the Figure 25) which contains all the layouts, a `CardView` was used because this allowed to have the corners on the right rounded as in the specifics, obtained setting `app:cardCornerRadius="75dp"`. Inside it there is an `ImageView` that inherits the width and the height of the card, it contains o blurred version of the cover-art of the album of song that is playing at that moment. Above this there is a `LinearLayout` (the green one in the Figure 25) with horizontal orientation where on the left there is an `ImageView` that shows another time the cover-art of the album this time not blurred. This `ImageView` inherits the height of the parent so it has `android:layout_height="match_parent"` with the attribute `android:adjustViewBounds="true"` which allows to have always a squared image with the width equal to the length. On the right there is another `LinearLayout` with vertical orientation which has two `include` for other two layouts: `media_info` and `media_controls`. `media_info` has two `TextView` for the artist and the title of the song with an `ImageButton` to add/remove that song to favorites. `media_controls` instead contains five `ImageButton` for as many buttons needed to play/pause, skip next, skip previous, shuffle and repeat, then there are other two `TextView`, the first one shows the next song the second shows the time of the song that flows. In the bottom of the external card it's also placed a `SeekBar` which displays also this the time of the song that flows.

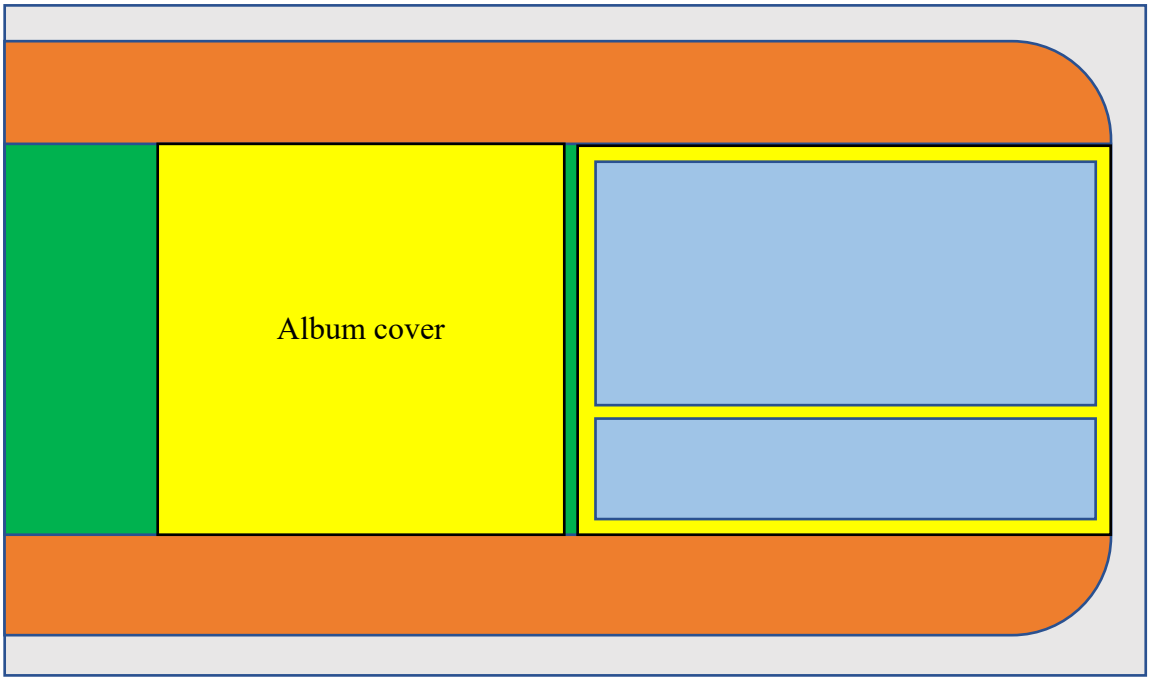

Figure 25. Multimedia App structure

5.1.3.3. Logic

The `MediaActivity` controls the user interface off the app and updates his connection status. When it starts it is created an object `MediaPlaybackFragment()` which is the fragment that displays the media playback UI. This fragment can contain different types of views: no content view, playback controls view, loading view.

When the app starts shows the loading view, after a moment is shown the no content view and at the end when the metadata loads the playback view is shown. Then the Bitmap corresponding to the album is set as background.

This class implements the method `onMetadataChanged()` that has the task of setting the values of the title of the song, the artist, cover-art, duration about the current playing based on the information coming from the class `MediaMetadata`.

Opening the drawer is shown a list of all the folders that contains multimedia files that can be played, this is sorted by song, album, artist or genre.

Inside the app is possible to select the source that the user wants, it's possible to choose among all the multimedia apps: Media, Radio, Bluetooth and third-party apps.

Since there is only one button for multimedia contents, when the user launches another app the last media source is stored. In this way when the multimedia is re-opened, is opened the last source that was selected. There is a timer which marks the time of the song. The time that

flows is represented also from the seek bar whose progress is set through the method `getPosition()` of the class `PlaybackState()`.

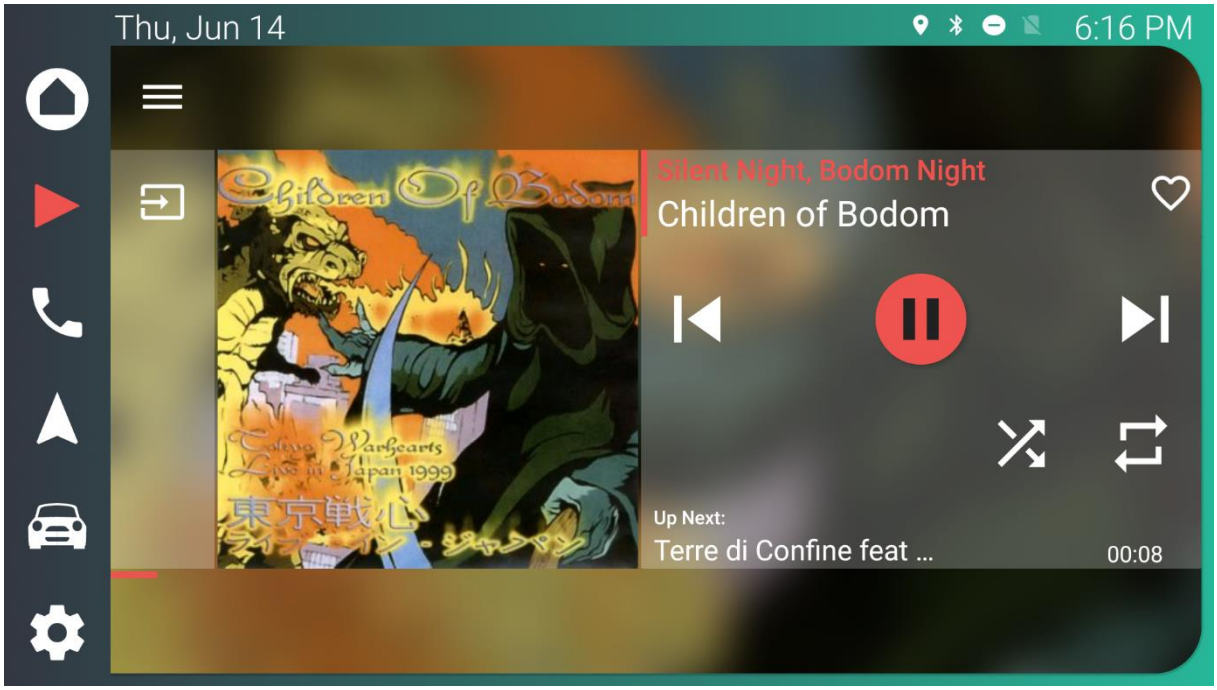

Figure 26. Multimedia App

5.1.4. Phone

5.1.4.1. Specifics

This app is composed by four principal views:

- The first one corresponds in a page that shows an advice to the user saying that no device is connected via Bluetooth, if the user click on it he/she is sent to the settings page where is possible to activate the Bluetooth and pair the smartphone or tablet.
- The main page appears when a smartphone or a tablet is paired through Bluetooth. This app works as a common cell phone. There is a dialpad with which is possible to dial the phone number that appears in a block in the middle between a button to start the call and another for deleting the digits. Another button allows to open the list of contacts.
- Another page appears if there is an incoming call another screen appears with the photo of the caller (if it is not present there is a default image), the number and the name (if present in the contacts list) of the caller. Obviously, there are also two

buttons for accept or reject the call. If the call is outgoing the screen is similar but without the accept button.

- During a call a timer indicating the duration of the call is shown with a mute/unmute button.

Clicking on the drawer is possible to access to the list of contacts, the list of recent and missed calls.

5.1.4.2. Design

This app is structured as a set of different layouts according to the state of the phone. When the phone is not connected there is a `TextView` that shows the message "To make or receive calls, connect your phone to your car via Bluetooth" whit an icon representing the Bluetooth disabled. When there is a connected phone the main view is divided in two layouts, in the one on the right there is the dialpad, this is realized as a set of `LinearLayout` in order to obtain a grid where the phone buttons are placed, three for each line and 4 for each column. These are all realized in the same way that is a Button that as a background a drawable that gives the rounded shape, the size and the right padding to the button. Then, each one has `android:layout_weight="1"` so the space that is taken is for all the same and a margin to distance it from others. Here is an example of button:

```
<Button
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:id="@+id/button1"
    android:background="@drawable/button_shape"
    android:layout_marginLeft="@dimen/margin_button_horiz"
    android:layout_marginRight="@dimen/margin_button_horiz"
    android:layout_marginBottom="@dimen/margin_button_vert"
    android:layout_marginTop="@dimen/margin_button_vert"
    android:autoSizeTextType="uniform"
    android:textColor="@color/window_gradient_start"
    android:text="1" />
```

On the left, instead, there are the call `Button`, the box where the phone number appears which is a `TextView` that is above a line and the `Button` for deleting a digit. These elements are inside a `LinearLayout` so they can stay aligned and with each other and with different weights, so they can better fit the space. Each one of these elements is inside a `RelativeLayout` so that they are easier to place in space.

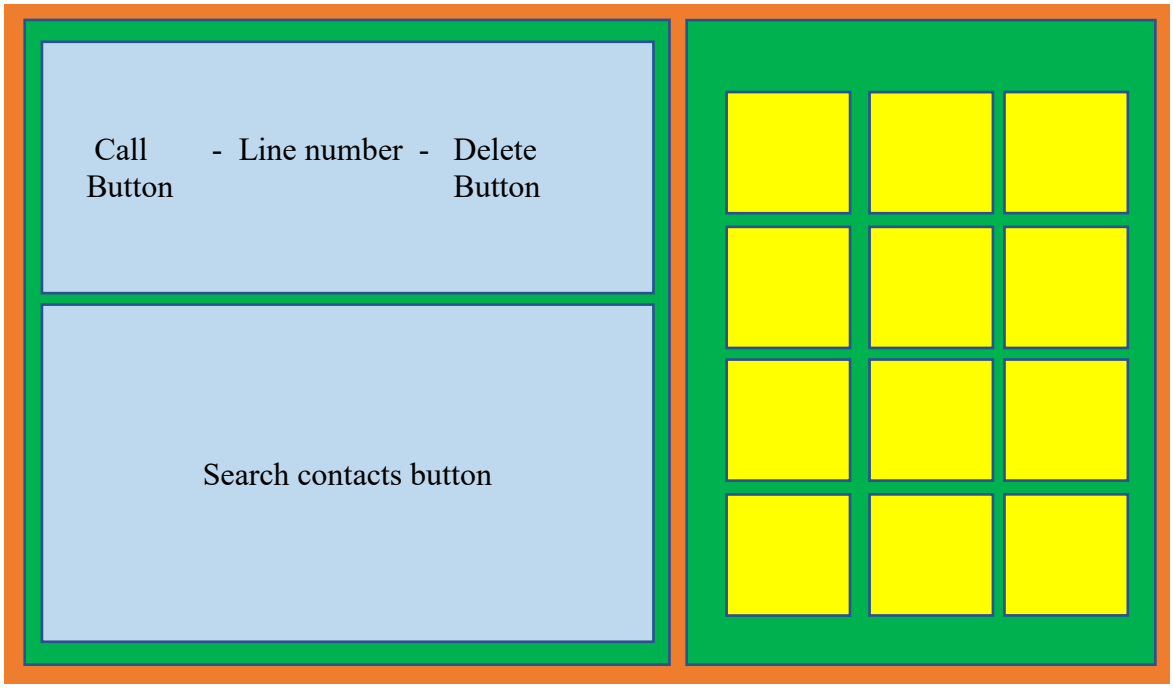

Figure 27. Phone App structure

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:layout_marginStart="70dp">

    <!-- Call Button -->
    <RelativeLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2">
```

```
    <ImageButton
            android:id="@+id/phone_call"
            android:layout_width="@dimen/main_button_dim"
            android:layout_height="@dimen/main_button_dim"
            android:layout_alignParentEnd="true"
            android:layout_centerInParent="true"
            android:layout_marginEnd="20dp"
            android:background="@android:color/transparent"
            android:scaleType="fitCenter"
            android:src="@drawable/ic_phone"
            android:tint="@{colorsViewModel.accentColor}" />

    </RelativeLayout>

    <!-- Line for the number -->
    <RelativeLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="5">

        <!-- Number -->
        <TextView
            android:id="@+id/phone_number"
            android:layout_width="wrap_content"
            android:layout_height="@dimen/dialer_number_view_height"
            android:layout_above="@id/line_number"
            android:gravity="center"
            android:focusable="true"
            android:textStyle="normal"
            android:textSize="@dimen/car_body1_size"
            android:textColor="@android:color/white"
            android:layout_alignParentEnd="true"/>

        <!-- Line -->
        <LinearLayout
            android:id="@+id/line_number"
            android:layout_width="match_parent"
            android:layout_height="5dp"
            android:layout_centerHorizontal="true"
            android:layout_centerVertical="true"
            android:background="@color/car_white_1000"
            android:orientation="vertical">

        </LinearLayout>

    </RelativeLayout>
```

```
<!-- Delete button -->
    <RelativeLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2">

        <ImageButton
            android:id="@+id/phone_delete"
            android:layout_width="@dimen/main_button_dim"
            android:layout_height="@dimen/main_button_dim"
            android:layout_alignParentStart="true"
            android:layout_centerInParent="true"
            android:layout_marginStart="20dp"
            android:background="@android:color/transparent"
            android:scaleType="fitCenter"
            android:src="@drawable/ic_backspace"
            android:tint="@android:color/white" />

    </RelativeLayout>

</LinearLayout>
```

5.1.4.3. Logic

The main activity for the Phone app is `TelecomActivity` which displays different fragments depending on call and connectivity status, these are: `OngoingCallFragment`, `NoHfpFragment`, `DialerFragment` and `StrequentFragment`.

`OngoingCallFragment` is the fragment that displays information about on-going call with options to hang-up.
`NoHfpFragment` is the fragment that informs the user that there is no Bluetooth device attached that can make phone calls. It returns an error message with a button that shows the Bluetooth disabled to display. When the icon that shows the Bluetooth disabled (`mErrorIcon`) is clicked the Bluetooth Setting page is opened through an `Intent`.
`DialerFragment` is the fragment that controls the dialpad.
The button search contact opens a new page with the list if contacts contained in the `StrequentFragment`.
The name of the device is given by the class `UiBluetoothMonitor` that is the class responsible for getting status of Bluetooth connections.

```
mErrorIcon.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intentOpenBluetoothSettings = new Intent();

intentOpenBluetoothSettings.setAction(android.provider.Settings.ACTION_BLU
ETOOTH_SETTINGS);
        startActivity(intentOpenBluetoothSettings);
    }
});
```

Figure 28. Phone App

5.1.5. MyCar

5.1.5.1. Specifics

The application that shows to the driver all the driving modes of the vehicle. It consists of three main pages:

- The first one is dedicated to the display of some parameters like the distance traveled in the day, the time spent on the road and the total mileage. There are also some indicators for the fuel level, oil level and the water temperature.
- In the second one it's possible to see some statistics about the driving: gear, acceleration and deceleration.
- In the last one is reported the pressure of each tire highlighted in a model of the car, then shows if the doors are locked and when it will necessary to carry out maintenance service.

5.1.5.2. Design

The main view is composed by a `viewpager`, a layout manager that allows the user to flip left and right through pages of data. In this case there are three pages.
All the pages of this app are similar, each one of these contains a set of `TextView` which has the role to show the requested statistics, there are also `ImageView` for displays some images of the car with the relative parameters such as the tyre pressure status and the parking sensors.

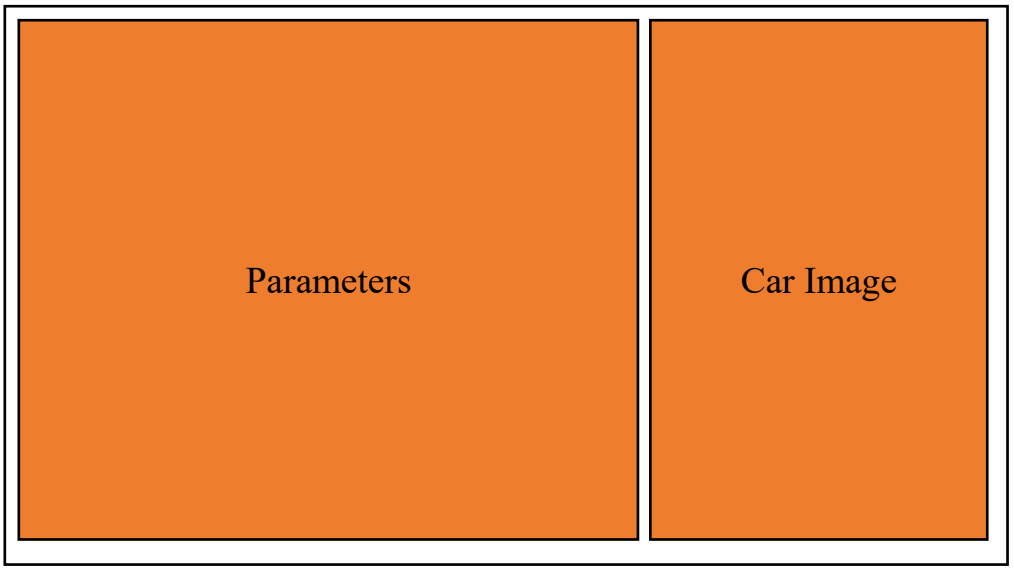

Figure 29. MyCar App structure

5.1.5.3. Logic

The logic of this app has not yet been developed because it is a work that is needed in a second step of the project. So, for the moment only the part related to the graphic aspect has

been implemented with some fake data that will be changed when the real signals will be available.

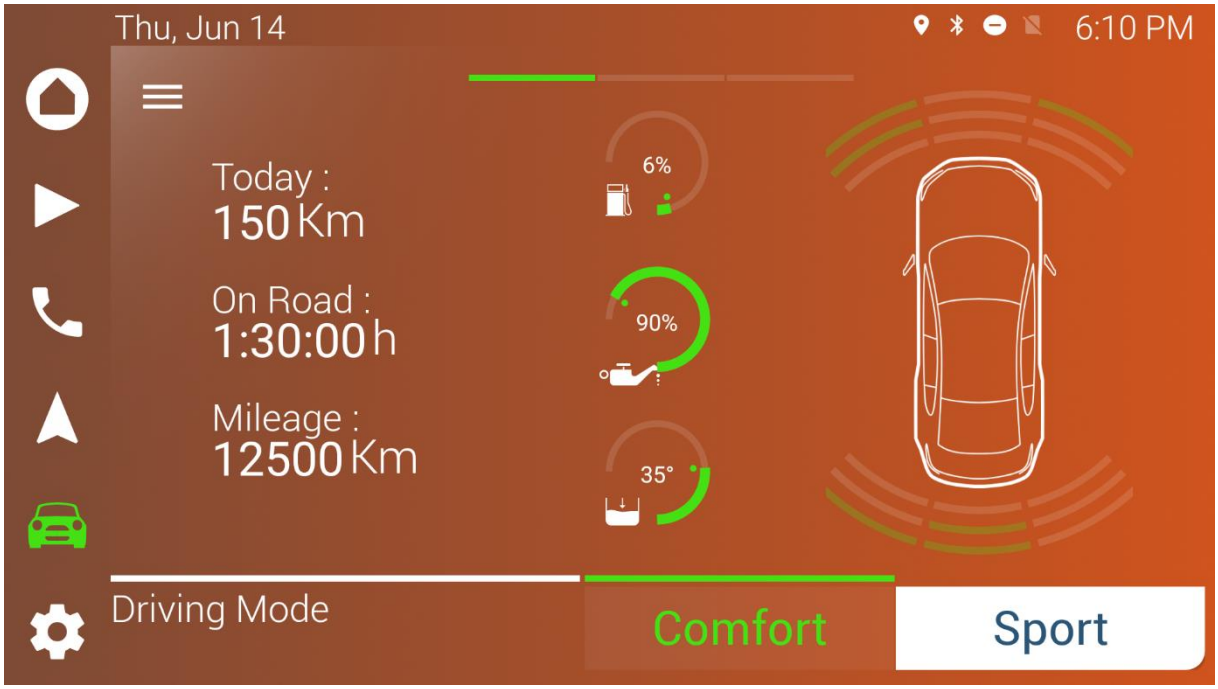

Figure 30. MyCar App

5.1.6. Preferences

5.1.6.1. Specifics

This app must have the function of showing to the user all the possible settings available in the system. It is divided in different pages each one with a for a specific setting.

In particular:

- The first page is intended to allow the user to choose the theme of the whole system. So,he/she can change the color of the background, of the text and of the highlighted elements. This is possible using a color picker realized through a bar and the user can choose manually the combination of colors. It's also possible to choose among five predefined colors pressing the correspondent button.

- The second page is for the volumes. There will be three bars, the first one for volume of media contents, the second is the volume of the phone ring and the last one is that of notifications.
- The last page regards connectivity, in particular Bluetooth and Wi-Fi. It shows for each one if it's active and a list of devices/router with which is possible to connect.

5.1.6.2. Design

As the MyCar application also in this app the main view is a `viewpager` that gives the possibility to slide between fragments. These fragments contain the three pages that this app has to show.

The `setting_themes` page has as principal layout a `LinearLayout` (the orange one in the Figure) that splits the page horizontally. In the top part there is another `LinearLayout` which divides the section in five smaller parts, each of them contains an `ImageButton` (the green circles in the Figure).

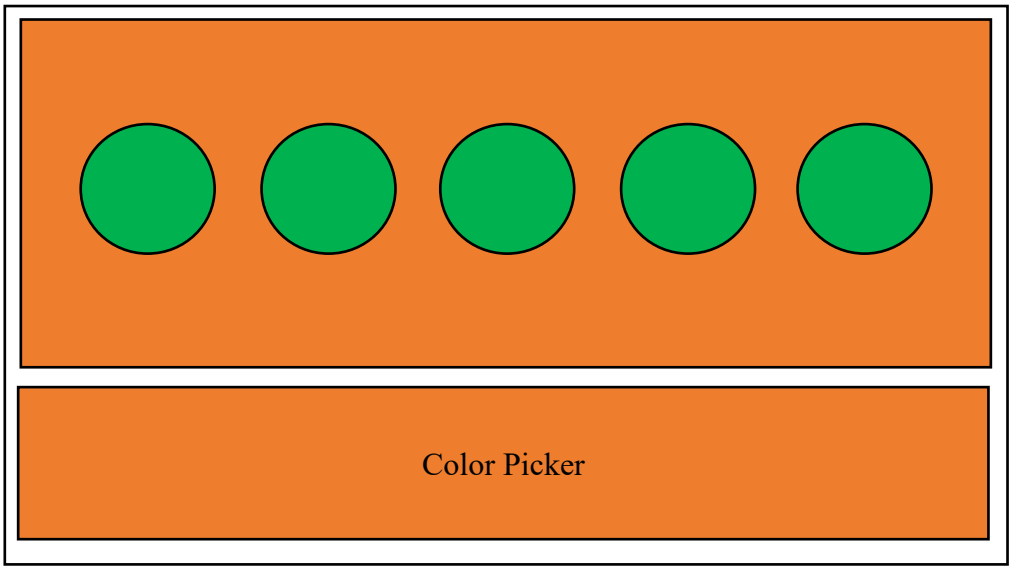

Figure 31. Preferences App structure

Since the buttons are circular in shape each button is realized putting as a background a drawable with a rectangular shape and adjusting the corner radius in order to obtain the desired shape. These buttons are divided inside into three parts in different colors. These colors represents the color of background of all the apps, the accent color for buttons and lines

and the color of the texts. In the example below, it is shown how a button has been made from the graphical point of view.
In the bottom of the layout there is a `SeekBar` for the manually selection of the theme.
The `setting_connectivity` page is composed by a `LinearLayout` that divides vertically the page. On the left side there is the part dedicated to the Bluetooth settings and this contains a set of `TextView` for the texts (as profile name) and Switch for changing the status connected/disconnected. The right part it's analogous to the one on the left but showing the Wi-Fi settings, in addition there is a `ScrollView` that displays all the networks available.
The `volume_connectivity` page contains three `SeekBar` for the for adjusting the volume of multimedia, calls and notifications. Each `SeekBar` has on the left the correspondent `ImageView`.

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >

    <item>
        <shape android:shape="rectangle" >
            <size android:height="@dimen/button_dimension"
                  android:width="@dimen/half_button_dimension" />
            <gradient
                  android:startColor="@color/theme_1_gstart"
                  android:endColor="@color/theme_1_gend"
                  android:angle="0"/>
            <corners android:topLeftRadius="@dimen/half_button_dimension"
                     android:topRightRadius="@dimen/half_button_dimension"
                     android:bottomRightRadius="@dimen/half_button_dimension"
                     android:bottomLeftRadius="@dimen/half_button_dimension">
            </corners>
        </shape>
    </item>

    <item android:bottom="@dimen/half_button_dimension"
          android:start="@dimen/half_button_dimension">
        <shape android:shape="rectangle" >
            <size android:height="@dimen/half_button_dimension"
                  android:width="@dimen/half_button_dimension" />
            <solid android:color="@color/theme_1_accent" />
            <corners android:topRightRadius="@dimen/half_button_dimension">
            </corners>
        </shape>
    </item>

    <item android:top="@dimen/half_button_dimension"
          android:start="@dimen/half_button_dimension">
        <shape android:shape="rectangle" >
            <size android:height="@dimen/half_button_dimension"
                  android:width="@dimen/half_button_dimension" />
            <solid android:color="@color/theme_1_text" />
            <corners
                   android:bottomRightRadius="@dimen/half_button_dimension">
            </corners>
        </shape>
    </item>

</layer-list>
```

5.1.6.3. Logic

The main activity of this app is `SettingsActivity` whose only task is to create a new instance of `SettingsFragment()`. This fragment has the role of displaying the one of the three settings pages. These pages are other three fragments: `VolumeFragment()`, `ConnectivityFragment()` and `ThemesFragment()` that are handled through a `ViewPager` which allows the sliding between them.

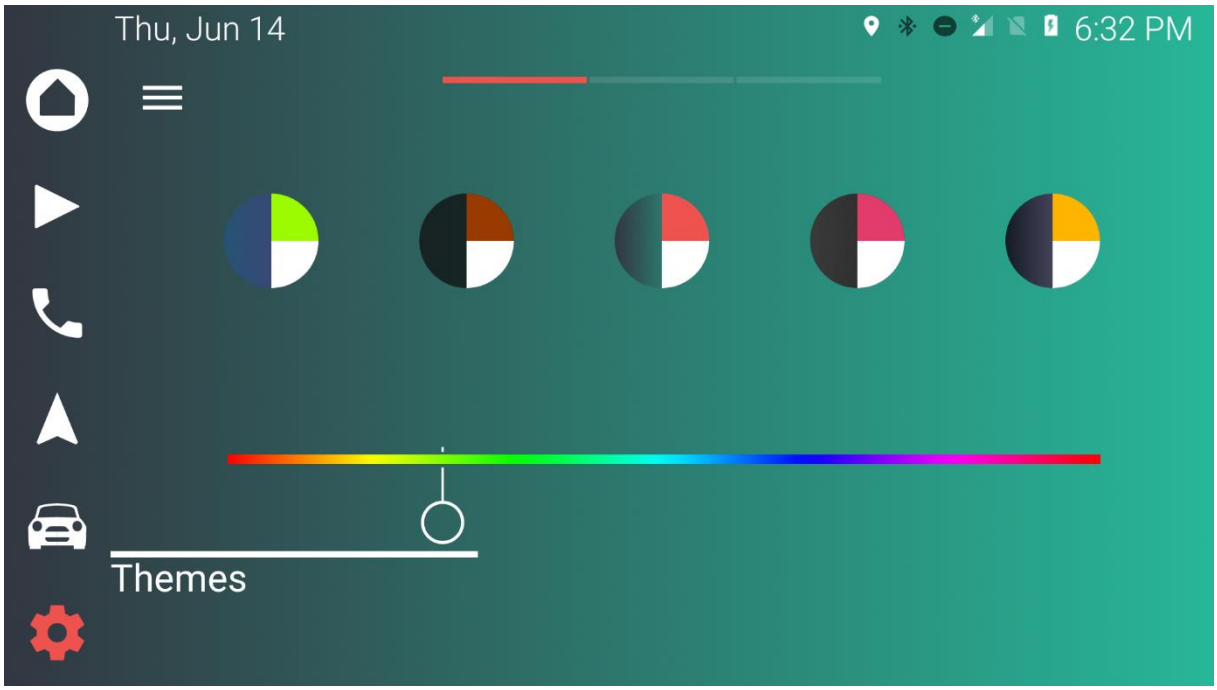

Figure 32. Preferences App

5.1.7. SystemUI

In addition to the applications the systemUI has also been changed. The system UI is the screen area dedicated to the display of notifications, communication of device status and device navigation. It includes both the Status Bar which is the one placed on top of the screen and the Navigation Bar on the bottom that provides a quick away to get the Overview screen (the Home screen of the system) and the other types of activities: Multimedia, Phone, Navigation, MyCar and Preferences. This second bar has been moved on the left side of the screen. This is not a common app like the others so, in order to apply the changes two his layouts it was necessary to edit the `.xml` files directly in the android framework. When an icon present in the System UI is pressed, this send an intent with the index of the pressed button to another app called Lens Picker that, looking this index, choose what kind of app it has to be open and send the correspondent package. If there are more packages of that kind the last one that was opened is launched, if no one was opened a window opens showing all the available applications for that kind. For example, pressing on the Multimedia button appear a list with all the multimedia apps available like Media, Radio and all multimedia third party apps installed.

Originally, also pressing long on a button the Lens Picker presents to the user a list of applications available for that button. This function of opening the window that show the apps available has been eliminated because has been recreated within every single application that has the possibility of choosing among various app. A source menu has been implemented, so the user pressing the source menu button inside the app can open another app. The Lens Picker app has not been deleted because it has the mechanism to keep track of the last app that was opened useful also by changing the perspective. So, when an activity is selected from the system UI the last source that was used will be opened.

The status bar instead remained the same, it was only always made transparent so it can have the same color of the theme chosen by the color picker.

5.1.8. Different screen sizes management

Since the project was not developed for a default system on which to mount it there were no precise dimensions that the screen should have, so for all the apps the different screen sizes were managed. In particular the two dimensions taken into consideration were 1920x1080 pixels (the dimension of the Lilliput monitor supplied together with the board Qualcomm) and 1920x720 pixels which is the most probable size that the final screen will have to have. To achieve this result two ways have been used, the first one consists in the only create different files `dimens.xml` with the different heights of the screens, this is the solution adopted when the layout remains the same, but it is only reduced in dimension. Another way is to create two specifics layouts for each height, this solution was adopted when with the change of the screen change also the position of some elements in the layout.

5.1.9. Data Binding

In the development of these apps at first for finding a view that was identified by an ID attribute in the XML file it was used `findViewById()` that finds the first descendant view with the given ID. The behaviour of this function has been replaced by the Data Binding, that allows you to bind UI components in the layouts to data sources in the app. The difference between these two methods is that the first one uses a programmatically format while the second uses a declarative one. The binding process makes a single pass on all Views in the

layout to assign the views to the fields. When is run `findViewById()`, the view hierarchy is walked each time to find it.

Binding components in the layout file allow to remove many UI framework calls in the activities, making them simpler and easier to maintain. This mechanism for accessing views is not only much easier than `findViewById()`, but can also be faster, in fact this can also improve app's performance and help prevent memory leaks and null pointer exceptions.

6. Future opportunities

The work done for this project was only a first implementation and new features will be added, in order to reach a better integration with the vehicle. Among these there is the adoption of a new board that is directly developed by the company, this will replace the Qualcomm board that was used for this project. The other new components that can be part of the project are listed in the paragraphs following.

6.1. Hypervisor

A hypervisor is a hardware virtualization technique that allows multiple guest operating systems (OS) to run on a single host system at the same time. The guest OS shares the hardware of the host computer, such that each OS appears to have its own processor, memory and other hardware resources.
The use of a hypervisor which makes it easier to obtain and maintain safety certifications by separating safety-critical components from non-safety critical components in separate guest operating systems. For example, this is useful for a safety critical instrument cluster that has security requirements that Android does not guarantee because it is not ASIL certified thus safety-critical information (such as critical warnings, speed) cannot be rendered from Android side. In this way no complex hardware is shared.
A graphical explanation is shown in the Figure 33.

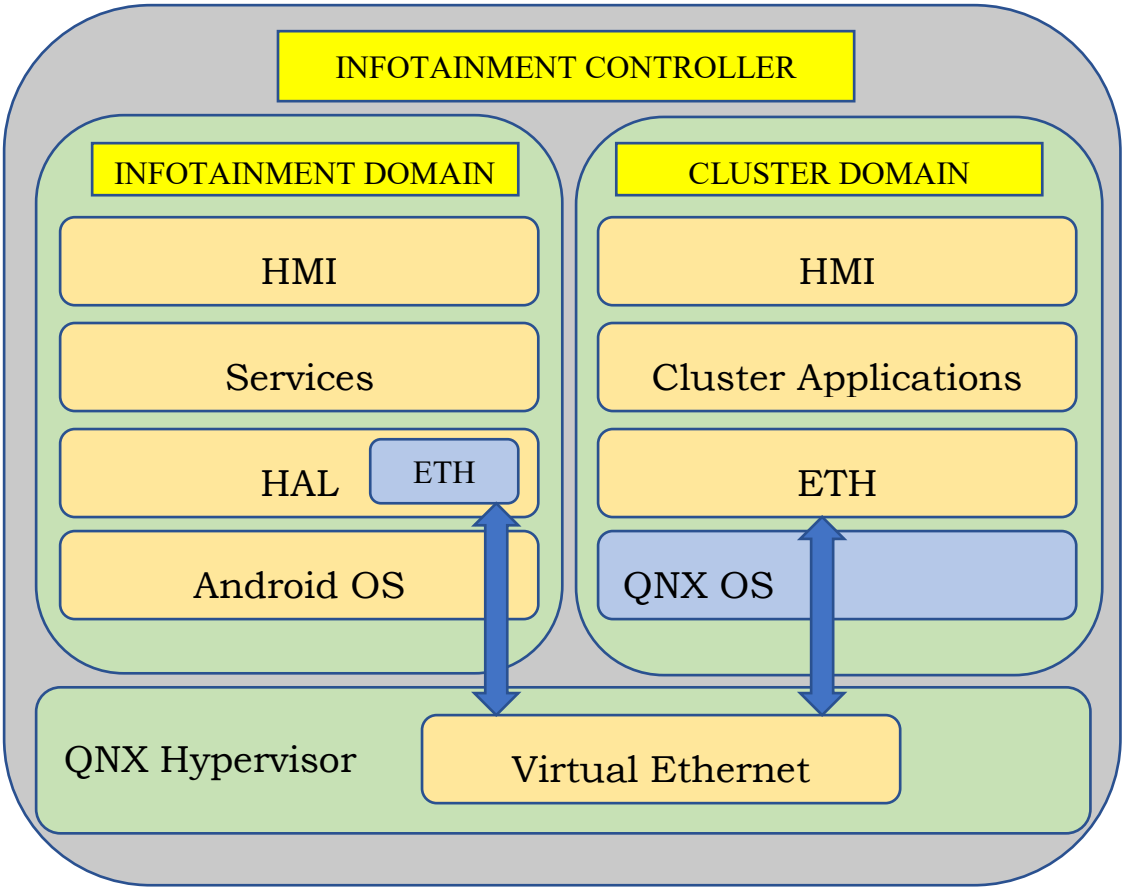

Figure 33. Infotainment and cluster domains with hypervisor

6.2. Rotary Pad

Another feature that could be added to the system is the interaction with a rotary pad, this component is already present in some cars as seen in the first chapter. This feature helps especially the driver because the rotation of the pad is a bit easier than continuously tapping a touchscreen so, he can navigate the various menus and submenus without doing too many movements.

6.3. Multi-Display

Since a car can have more displays, even Android for its system has set up this feature. Not all the displays in a vehicle must have the same characteristics: some can be touchscreen, others not and must respect different levels of driver distraction.
The capabilities for this feature for Android Oreo are the following: support for one primary display with touch input, one directly connected external display without input and a number of virtual displays whose number depends by hardware processing capabilities.
It's possible to have more displays if there is, for example, another real-time operating system to drive the others displays.

6.4. Android P

The project was developed with the latest version available of the Android operating system, Android 8.1 (Oreo), on March 2018 Google release the first developer preview of the ninth update of the operating system: Android P. The official release occurred in the August of the same year.
It is expected that in the next phase of the project the porting of the work done on Oreo also takes place on P. As seen talking about the Project Treble in the third chapter this transition should not be very expensive thanks to the features introduced with treble.

7. Conclusions

As already said in the second chapter, nowdays the market is dominated by IVI systems based on a proprietary software like QNX, but with the spread of open source systems this trend should wane also because with the advent in this field of Google and its resources this is inevitable.

The work done for this master thesis has given the opportunity to evaluate the potential of Android applied to the automotive environment. This system has the disadvantage compared to others to be born recently (for purposes related to the automotive) but with a great experience behind it and a large slice of the market for mobile devices such as smartphones and tablets.

This is a great benefit because the end user is already familiar with the system, its applications and its operation, so it will be able to find inside the vehicle the same functionalities with which it had interacted so far with its portable devices.

The widespread use of these technologies has another big advantage that is to find developers able to develop applications for this system because there are no substantial differences with realizing applications for phones and tablets. In this way these developers can reuse their knowledge even in a new field of application.

But most of all the benefits are for the manufacturers that don’t have to develop an ad-hoc operating system, that would entail many a big amount of money and time, they would concentrate their resources in the development of new apps and in the customization of those already existing. This also thanks to the innovations made by Android with the 8.0 version and the Project Treble with which device makers can choose to deliver a new Android release to consumers by just updating the Android OS framework without any additional work required from the silicon manufacturers. So, there is a big saving of resources.

8. Bibliography

[1] Embitel. [Online]. Available: https://www.embitel.com/blog/embedded-blog/design-strategies-develop-in-vehicle-infotainment-system.

[2] AlfaRomeo. [Online]. Available: https://www.youtube.com/watch?v=ed9mYBZks0I.

[3] Electronicspecifier. [Online]. Available: https://automotive.electronicspecifier.com/driver-assistance-systems/what-s-driving-in-vehicle-infotainment-systems.

[4] Drdobbs. [Online]. Available: http://www.drdobbs.com/embedded-systems/an-architecture-for-in-vehicle-infotainm/222600438.

[5] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Board_support_package.

[6] Objectweb. [Online]. Available: https://web.archive.org/web/20050507151935/http://middleware.objectweb.org/.

[7] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Middleware.

[8] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Application_layer.

[9] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Android_(operating_system).

[10] Qnx. [Online]. Available: http://blackberry.qnx.com/en/products/qnxcar/index.

[11] Arstechnica. [Online]. Available: https://arstechnica.com/gadgets/2013/02/from-the-car-to-your-phone-how-blackberry-ported-over-qnx-for-its-new-os/.

[12] Techopedia. [Online]. Available: https://www.techopedia.com/definition/27004/hybrid-kernel.

[13] Genivi. [Online]. Available: https://www.genivi.org/.

[14] Linuxgizmos. [Online]. Available: http://linuxgizmos.com/linux-based-agl-ivi-stack-to-debut-on-2018-toyota-camry/.

[15] Automotivelinux. [Online]. Available: https://www.automotivelinux.org/.

[16] Arstechnica. [Online]. Available: https://arstechnica.com/gadgets/2017/05/google-brings-android-to-the-car-with-audi-and-volvo/.

[17] Android. [Online]. Available: https://developer.android.com/guide/platform/.

[18] AndroidCentral. [Online]. Available: https://www.androidcentral.com/aosp.

[19] Android. [Online]. Available: https://source.android.com/devices/architecture/.

[20] Harman. [Online]. Available: https://services.harman.com/industries/automotive-connected-car/automotive-engineering-services/android-automotive-platform.

[21] Android. [Online]. Available: https://source.android.com/devices/automotive/.

[22] Android. [Online]. Available: https://source.android.com/compatibility/.

[23] Arstechnica. [Online]. Available: https://arstechnica.com/cars/2015/07/android-auto-review-a-beautiful-but-beta-alternative-to-awful-oem-solutions/.

[24] Linuxgizmos. [Online]. Available: http://linuxgizmos.com/qualcomm-aims-new-snapdragon-820a-soc-at-smart-cars/.

[25] Intrinsyc. [Online]. Available: https://www.intrinsyc.com/vertical-development-platforms/s820am-v2-automotive-development-platform/ .

[26] Qualcomm. [Online]. Available: https://www.qualcomm.com/snapdragon/processors/820-automotive.

[27] Android. [Online]. Available: https://developer.android.com/studio/intro/.

[28] Android. [Online]. Available: https://developer.android.com/studio/command-line/adb.

[29] “Android,” [Online]. Available: https://developer.android.com/studio/debug/am-logcat .

[30] Git. [Online]. Available: https://git-scm.com/.