Bachelor’s thesis

Information and Communications Technology

2019

Miikka Lukumies

QT FRAMEWORK IN MODERN EMBEDDED USER INTERFACE DEVELOPMENT

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2019 | 47 pages, 2 pages in appendices

Miikka Lukumies

QT FRAMEWORK IN MODERN EMBEDDED USER INTERFACE DEVELOPMENT

Graphical user interfaces are becoming a more prominent feature in the development of new machines and devices. As the cost of performance is decreasing, an increasing amount of new systems are manufactured with the capability of providing an approachable window into the underlying data. The wider variety of environments and an expanding selection of connectivity options generate more demanding requirements for the modern user interfaces, especially in the embedded environment, where the resources may be limited.

The Qt Framework is a cross-platform software development framework for embedded and desktop environments. It enables creating user interfaces and applications for a wide range of devices and platforms using the C++ programming language for application logic, complemented by its custom markup language along with JavaScript for the creation of dynamic user interfaces.

The objective of this thesis was to assess the Qt Framework's suitability for the development of modern user interfaces in embedded Linux and Android mobile environments. To provide a comprehensive, all-round analysis of the capabilities of the framework, a fully functioning automotive in-vehicle infotainment system and a mobile companion application were created, with an extensive set of features that could appear in a modern embedded user interface.

A list of requirements gathered at the beginning of the projects alongside a testing plan was used as a performance metric for assessing the framework. The various useful features the framework provided enabled the creation of highly modular user interfaces for both platforms, which greatly increased code re-use and maintainability. The framework's capability of functioning as the sole development kit for the entire project proved the maturity of the toolkit the framework provided.

KEYWORDS:

C++, Qt Framework, embedded user interface, agile software development

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2019 | 47 sivua, 2 liitesivua

Miikka Lukumies

QT FRAMEWORK MODERNIEN SULAUTETTUJEN KÄYTTÖLIITTYMIEN KEHITTÄMISESSÄ

Graafisista käyttöliittymistä on tullut huomattavan yleinen ominaisuus uusien koneiden ja laitteiden kehityksessä. Suorituskykykustannuksien laskiessa yhä useampi uusi laite päätyy markkinoille mukanaan jonkinlainen näyttöpääte, joka auttaa tarjoamaan käyttäjälle helpommin lähestyttävän ikkunan alla olevaan informaatiotulvaan. Erilaisten asennusympäristömahdollisuuksien ja yhteystapavaihtoehtojen lisääntyminen tuo mukanaan yhä vaativampia edellytyksiä moderneille käyttöliittymille, varsinkin sulautetuissa järjestelmissä, joissa resurssit saattavat olla muutoinkin rajoitetut.

Qt Framework on laitteistoriippumaton sovelluskehitysalusta sekä sulautettuihin, että työpöytäympäristöihin. Se mahdollistaa käyttöliittymien sekä sovellusten kehittämisen laajalle valikoimalle laitteita ja alustoja, hyödyntäen C++ ohjelmointikieltä ohjelmistologiikan toteuttamiseen, sekä kehitysalustan omaa kuvauskieltä täydennettynä JavaScript-ohjelmakoodilla dynaamisten käyttöliittymien toteuttamiseen.

Tämän opinnäytetyön päämääränä oli arvioida Qt Frameworkin soveltuvuutta modernien käyttöliittymien kehittämisessä käyttäen kohdealustana sulautettua Linux-käyttöjärjestelmää, sekä Android-pohjaista mobiilialustaa. Kattavan analyysin suorittamiseksi toteutettiin sovelluskehitysprojekti, jonka tavoitteena oli luoda täysin käyttökelpoinen ajoneuvon info- ja viihdejärjestelmä, sekä täydentävä mobiilisovellus. Järjestelmää kehitettäessä mukaan sisällytettiin mahdollisimman suuri määrä erilaisia toimintoja, joita nykypäivän käyttöliittymissä voisi esiintyä.

Ennen järjestelmän kehityksen aloittamista projektille pyrittiin keräämään kattava lista vaatimuksia sekä testaussuunnitelma, joita voitiin käyttää sovelluskehitysalustan ja -ympäristön soveltuvuuden ja suorituskyvyn mittaamiseen. Opinnäytetyön tuloksena huomattiin Qt Frameworkin olevan hyvin kypsä ja suorituskykyinen työkalupaketti laitteistoriippumattomien modulaaristen nykyaikaisten käyttöliittymien kehittämisessä. Lukuisat kehitysalustan tarjoamat hyödylliset ominaisuuden mahdollistivat sen käyttämisen ainoana työkaluna jopa laajamittaisien järjestelmien luomisessa.

ASIASANAT:

C++, Qt Framework, sulautettu käyttöliittymä, ketterä sovelluskehitys

CONTENTS

APPENDICES

FIGURES

TABLES

LIST OF ABBREVIATIONS

Abbreviation	Explanation of abbreviation
ALSA	Advanced Linux Sound Architecture
API	Application Programming Interface
CSS	Cascading Style Sheets
GHz	Gigahertz
IVI	In-Vehicle Infotainment
JSON	JavaScript Object Notation
LPGL	GNU Lesser General Public License
QML	Qt Modeling Language
SDK	Software Development Kit
UI	User Interface

1 INTRODUCTION

Lately the most prominent new feature in the development of human operable machines and devices has been the graphical user interfaces, which are being embedded into an increasing number of newly developed systems. The purpose of these improved interfaces is to provide a much more approachable window into the underlying mountains of data from different origins, not to mention possible entertainment capabilities.

The wide variety of models and variants of machines and devices targeted creates a set of performance and flexibility requirements for the toolchains used in creating such interfaces, the main focus being on targeting as many different platforms as possible while keeping the cost and effort at a minimum. The customizable sizes, features, and flavors in which these systems are manufactured create endless possible configurations, requiring powerful build systems and flexible development toolkits to be able to adjust to the different configurations and the individual needs of the end-users.

This thesis takes a closer look into a set of commonly used tools to assess their suitability in creating modern embedded user interfaces, using the automotive world as an example environment. A list of requirements for the project is devised by creating user stories and used alongside a functional testing plan as a performance metric for the assessment.

The selected technologies, The Qt Framework for software development, running on both, a custom embedded Linux distribution, and the Android mobile platform, are used to create a demonstrative automotive in-vehicle infotainment system with an extensive set of features to conduct a comprehensive analysis of the performance of the framework.

2 SOFTWARE DEVELOPMENT WITH QT

Qt is a framework for creating native cross-platform software for a multitude of supported platforms. The main feature of the framework is its preprocessor, capable of turning the projects created with Qt into native C++ software code, which can be compiled for the target devices using any supported build system. The Qt Framework includes also its own build system, qmake, which can be used to create the application binaries. [1]

2.1 Development methods

The Qt Framework provides two methods for creating applications; the Qt Quick, and Qt Widgets modules. Both modules can be used to create desktop applications, however, they have some differences in supported features and the way they can be developed with. [2]

Qt Widgets

The Qt widgets module provides the developer with components needed to build classical-looking desktop style applications, mainly for PC environments, where the navigation and interaction happen using a keyboard and a mouse [2]. The Widgets module is especially useful when creating data-driven applications using the model/view user interface architecture, where the user interface implements only a view of the underlying data model, often supplied from elsewhere [3] [4]. The Widgets module does not support touch screens, and animation support is limited [2].

Qt Quick

The Qt Quick module is the Qt Framework standard for creating modern and smooth graphical user interfaces with support for touch input and complex animations [2]. Qt Quick applications are mainly developed using the declarative QML language, reminiscent of JSON and CSS, which is rendered to the user using the QML engine [5]. The easily readable QML code can be extended with imperative inline JavaScript expressions [6] and C++ plugins [7].

2.2 Cross-platform Development

The qmake build system included in the Qt Framework enables application development for multiple different target platforms with few, if any, changes to the project [8]. The project files that are created as an instruction set for the build system provide support for straightforward simple projects for a single platform, as well as complex collections of software code supporting multiple platforms, that require intricate control over the build process.

2.3 Useful features

The following sections list a collection of useful features of the Qt Framework, which make it particularly useful in modern user interface-oriented application development, especially for different embedded environments, and led to choosing the framework as a base for the project described in this thesis.

2.3.1 The Qt Resource System

The Qt Resource System is a method for storing application-related data in binary format inside the application executable [9]. The Qt resource compiler, *rcc*, creates a C++ source file containing static arrays of raw, compressed data, from which the application can load its resources, such as icons and images, during runtime. This possibility is particularly useful on platforms, which do not support application resource management natively. A good example of this kind of platform is the Embedded Linux environment.

As of the time of writing this thesis, the application resources are by default embedded into the executable using the resource system. The process is highly configurable, and parameters such as compression rate and method can be customized for each resource. [9]

2.3.2 Property bindings

The Qt Markup Language has built-in support for dynamic object behavior, which can be utilized by using property bindings. Property bindings are a way to create relationships between the different components of the user interface. This can be leveraged to instruct the GUI to react dynamically to interaction, such as a touch event.

The property bindings can consist of anything simple algebraic relations to complex JavaScript expressions, of which the QML engine keeps track and updates the corresponding elements when a change is detected.

2.3.3 Signal handling

To react properly and immediately to user input, the graphical user interface must be able to create events. Additionally, in the embedded environment, hardware may contain components that might require immediate attention at any given point of time, especially if there is a need to communicate with systems constrained with real-time requirements. The Qt Framework provides a powerful signaling system for both the user interface and the supporting backend. It is possible for the two layers to send signals to each other [10]. The Qt Company also provides proprietary modules, such as the Qt Safe Renderer, for applications having to meet functional safety requirements [11], however, including these modules fell outside the scope of this thesis.

The user interface run by the Qt QML engine handles different events created by the user or the interface components with the framework’s Signal and Handler Event System [10]. The signals, bound to a certain property of a component, when triggered, invoke an assigned signal handler containing JavaScript code to be executed in the event of the corresponding change in the state of the application. The signal handlers may change the properties of other components, such as width and height, or trigger other signals based on a variety of configurable conditions.

The user interface signals can also trigger events in the backend. Along with signals created by hardware or for example timing functions, these signals are handled in the C++ backend libraries by the Signal & Slot system, powered by the Qt meta-object system [12] [13]. The Qt Framework’s meta-object compiler creates accompanying C++ code to the different objects derived from the framework’s base object class to implement a signaling system between the various objects. It is possible to create many-to-many relationships between the signals (senders) and slots (receivers) to create one or two-way signaling pathways to react to changes in the application state. Figure 1 visualizes the possible connections between the signals and slots of separate objects.

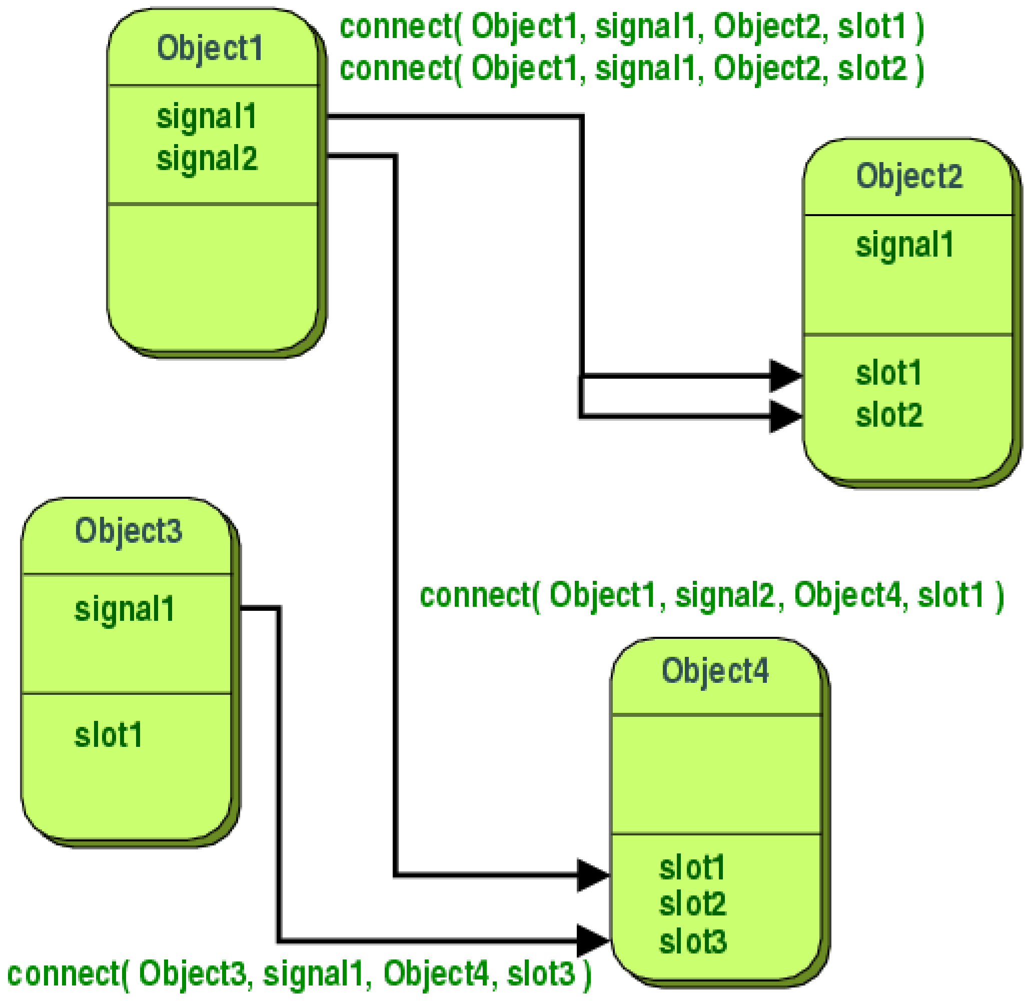

Object1
signal1
signal2
connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect(Object1, signal2, Object4, slot1)
Object4
slot1
slot2
slot3
connect(Object3, signal1, Object4, slot3)

Figure 1. Abstract connections (https://doc.qt.io/qt-5/signalsandslots.html).

2.3.4 Yocto support

The Qt Framework provides well-documented support for integration in the Yocto Project-based embedded Linux distribution creation [14]. The Qt Company maintains and updates a Git repository for its Yocto Project recipe layer, *meta-qt5,* for an easy and up-to-date starting point for using the framework in embedded device creation [15]. The following chapter concentrates on introducing The Yocto Project.

3 THE YOCTO PROJECT

The Yocto Project is an open-source, collaboratively designed tool for creating custom embedded Linux distributions. Its architecture and hardware independency are the key elements that have fueled its adoption as a standard for creating custom Linux-based systems for any platform. It has become widely adopted across the industry, especially the automotive industry has grown interested in the project as a base for device creation. [16] [17] [18]

3.1 About the Yocto Project

The Yocto Project was first released in 2010 by the Linux Foundation. Upon the project’s first major release 1.0, the previously separated OpenEmbedded Linux build framework joined the collaboration and shared their OpenEmbedded-Core set of metadata, which is nowadays responsible for the core functionality of the project [19]. In addition to the OpenEmbedded-Core, the second main component of the project is the BitBake task execution engine, which handles most of the tasks required for building embedded operating system images.

3.2 Yocto development and architecture

A typical Yocto Linux distribution build project consists of a customizable collection of layers, sets of recipes containing instructions for the BitBake build system on how to compile and assemble an operating system. The layers are usually divided into logical parts of the final system, such as hardware layer, graphics layer, and an application layer. This division, visualized in Figure 2, makes it possible to separate the machine and application specific features and configuration into smaller parts, which can be - quickly and easily swapped to produce a different system, to target a wider variety of platforms and applications, while maintaining the state of the core components.

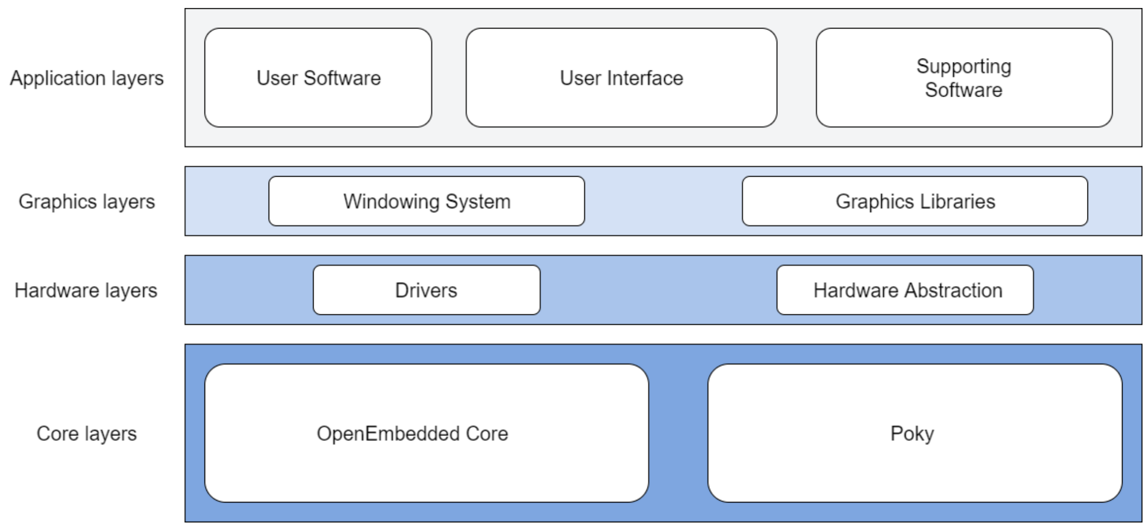


Figure 2. Yocto Layered model example.

Most of the layers needed to create a functioning system can usually be found from the OpenEmbedded Layer index, thus the designers' responsibilities are often limited to determining the correct set of recipes for the targeted system, and to tweaking the configuration parameters of those recipes. Silicon manufacturers that support the Yocto project usually supply their own BSPs (Board Support Package) for all supported architectures, and the Poky reference distribution provides a comprehensive starting point for creating the rest of the system. This means that it is often only the application layer and supporting libraries that need configuration, and unless custom software is included in the final product, creating own recipes is often unnecessary. [19]

The development process may be divided into a series of steps, many of which enable customization. Figure 3 gives an overview of the whole process of creating an operating system, and an accompanying software development kit. The main build system is barely a task execution engine, capable of fetching resources and compiling them into distribution-specific package formats and eventually into the operating system image and SDK.

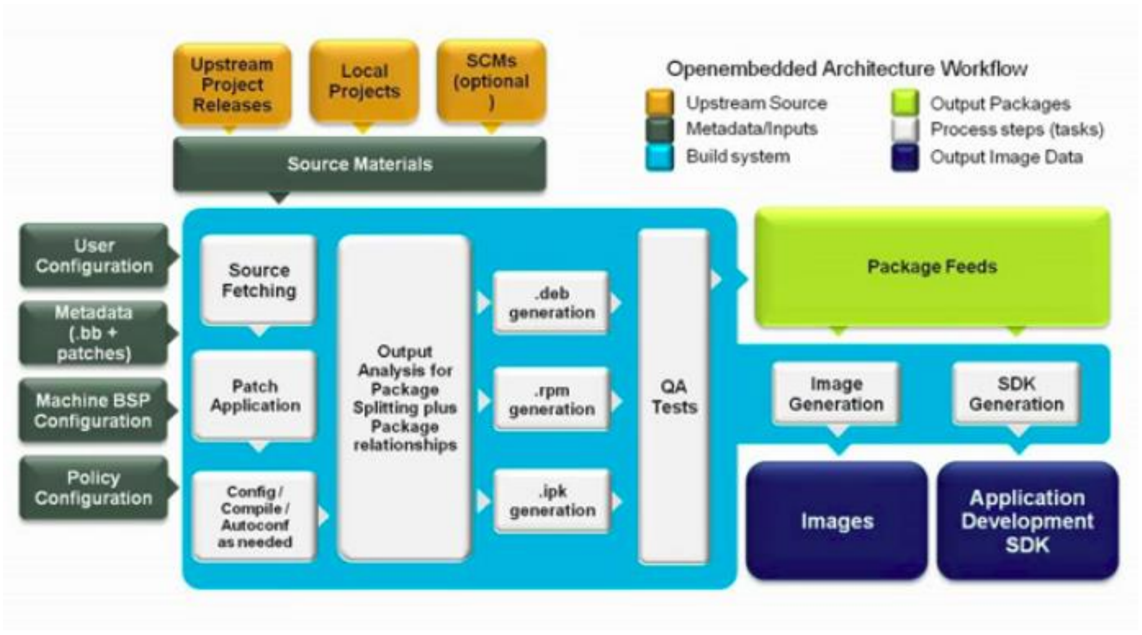


Figure 3. OpenEmbedded Architecture Workflow (https://yoctoproject.org).

The recipes, delivered in layers or separately, enable the users to create policies, patches and configuration details, which determine the properties of the created operating system. These recipes can also be used to define separate software resources, that should be included in the final image. These resources can be configured to be fetched from many different sources, such as Git, or local repositories. [19]

4 BLUETOOTH TECHNOLOGIES

Bluetooth is a short-range communication protocol, used nowadays in many different applications, where low cost and energy consumption are of the essence. The protocol was originally developed by Ericsson in 1994, and in 1998 a Special Interest Group (Bluetooth SIG) was formed to continue governing the development of the protocol, promote its usage and protect the trademark itself [20]. In 2018, the Bluetooth SIG had nearly 35,000 members, and 3.7 billion new devices were shipped with Bluetooth support [21].

4.1 Introduction to Bluetooth

The Bluetooth technology utilizes the license-free frequency band of 2,4GHz, used also in many other applications such as wireless hotspots to access the Internet. This frequency band provides decent transmission rates and ranges up to 100m. [22]

The Bluetooth standard specifies a wide variety of protocols and profiles for communication between different devices and variable types of information [23] [24]. The communication protocols are responsible for specifying the means for communication on the wireless media, and the profiles are specifications used to identify the capabilities of the various kinds of devices.

The Bluetooth protocols are usually divided into two categories; the controller and the host stack [23]. The controller stack is responsible for the low-level link and timing-oriented communication between the microchips responsible for the radio transmission. The host stack protocols, on the other hand, comprise of more higher-level, controller-agnostic protocols usually implemented in software. Figure 4 visualizes the possible host and controller combinations. The Bluetooth protocol includes both connectionless and connection-oriented protocols, to fit a maximum amount of use cases.

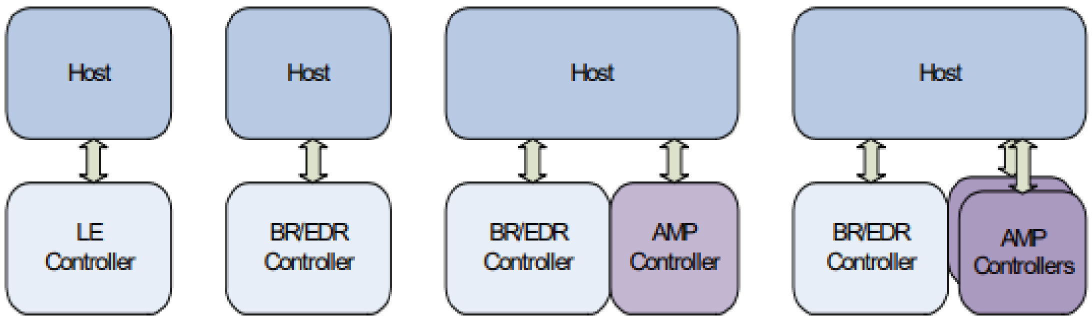


Figure 4. Bluetooth Host and Controller combinations [25].

The almost forty different profiles specified in the Bluetooth standard make it possible for various kinds of devices, such as printers, cameras, headphones, and other wearables to correctly communicate with each other [24]. By registering a subset of these profiles, a Bluetooth device can accurately describe and broadcast its properties to be discovered by and connected with desired endpoints. The following subchapters introduce a subset of these profiles and corresponding software, relevant to the features implemented as a result of this thesis.

4.2 Media playback

The Advanced Audio Distribution Profile (A2DP) specifies a format over which audio can be streamed using Bluetooth between devices [24]. The communication profile is implemented as a strictly unidirectional master-slave system, where the Source device establishes a synchronous, connection-oriented link to feed audio stream to the Sink, a destination device [26]. This profile is often implemented together with Headset and/or Handsfree profiles.

4.2.1 A/V remote control profile

To control the playback of media, usually music or video, the devices can implement a profile called Audio/Video Remote Control Profile (AVRCP). This protocol enables controlling the data streams by stopping or starting the stream, as well as changing tracks. Additionally, custom metadata, such as artist and song name, and media source status can be sent over to the target device [24].

4.2.2 Handsfree profile

While the A2DP and AVRCP profiles provide the means to stream music and control its playback, the Hands-Free Profile (HFP) enables handling voice calls. The hands-free profile defines a protocol to create a bidirectional, one-channel audio link to transfer voice call audio between the devices, and also some control, such as initiating or answering to calls [24]. The hands-free profile defines two roles in the communication; an Audio Gateway, and a Hands-Free Unit, where the gateway is normally a mobile phone as source of the voice data, and the hands-free unit is usually the remote device, such as car audio kit, with optional controls [26].

4.2.3 Pulseaudio software

To access, mix, control and reroute audio streams to and from multiple sources effectively, a sound server can be installed on the system to provide an abstraction layer, and to automate management. PulseAudio is an LGPL 2.1+ licensed open source sound server, which has wide support over multiple operating systems and is available for many Linux distributions. Its features include software mixing and routing audio streams, which can be utilized to route Bluetooth audio streams to hardware speakers. [27]

4.3 Object Exchange profile

In addition to providing an audio link for voice calls, another important profile to implement for voice call functionality is the exchange of contact information. The OBject EXchange (OBEX) profile is a protocol defined initially for infrared data transmission, later adopted into Bluetooth, to exchange any forms of binary data in the form of objects [28].

The Phonebook Access Profile (PBAP) is built on the OBEX profile and can be used to allow the car kit to see the contact information of an incoming caller and even for downloading the phonebook of the connected mobile phone [24].

4.4 Serial communication profiles

In some cases, it is necessary to define a custom profile to exchange arbitrary data between the devices, to create own protocols and to enable finer control over the communication. The Serial Port Profile (SPP), based on the RFCOMM protocol, is a profile which enables the creation of virtual serial ports, and thus application-defined two-way communication [24] [26].

4.5 BlueZ Bluetooth protocol stack

To effectively manage and control the underlying Bluetooth hardware as well as to manage different connection lifecycles, additional software can be installed on the device to provide an additional layer of abstraction. The Official Linux Bluetooth protocol stack, BlueZ, is an awarded collection of many layers of software, from kernel drivers to testing and configuration software, which supports many Linux distributions and processor architectures. The protocol stack provides real hardware abstraction, as well as support for multiple Bluetooth devices and an interface to all the different Bluetooth stack layers. [29]

5 DEVELOPMENT ENVIRONMENT

Application development for a remote host such as the embedded Yocto Project Linux distribution is quite different from developing desktop applications. The most significant difference is that the developed software cannot be run on the host machine, instead, a target-machine-specific build system must be installed on the host computer to develop for different hardware. In the next chapters, the steps required to develop for the target hardware and to build and configure the operating system itself are introduced.

5.1 Cross-Compilation Build System

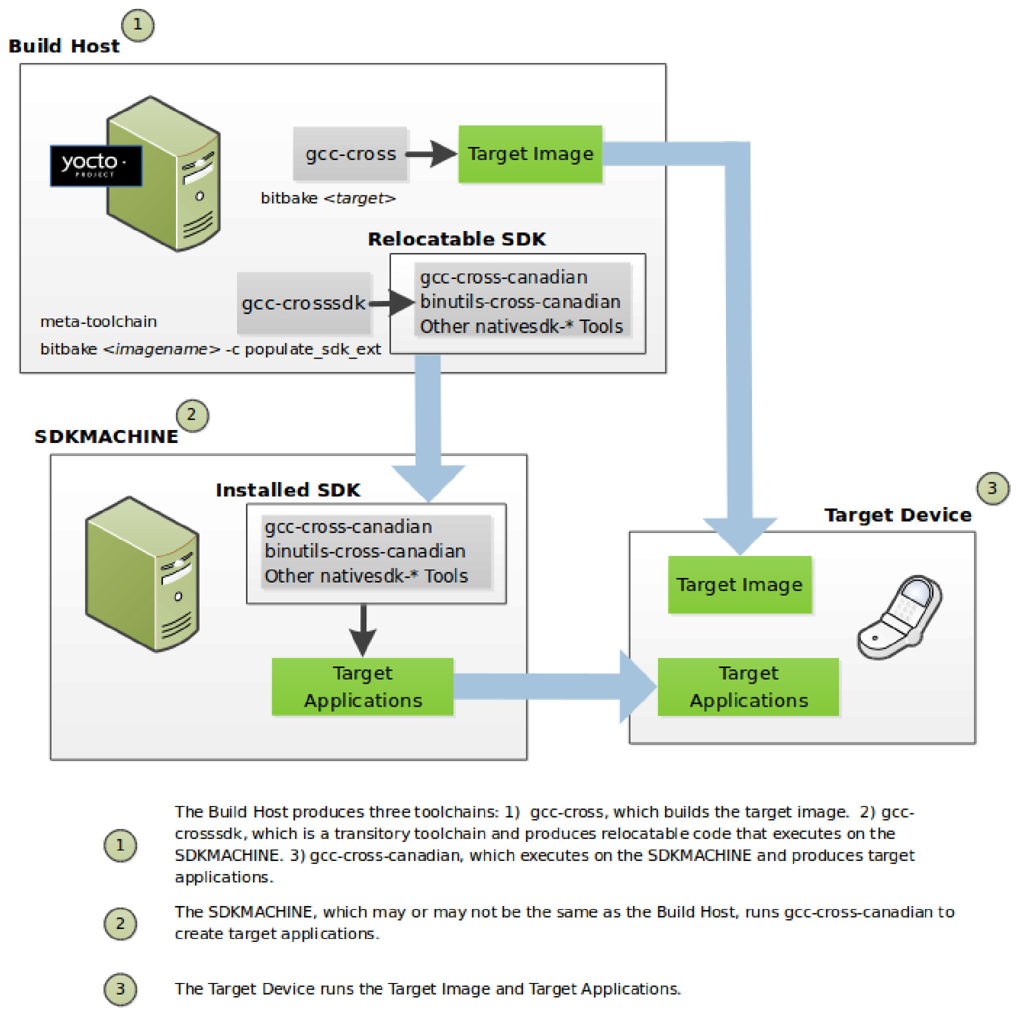
Unlike in the desktop computer environment, an embedded operating system rarely comes equipped with a compiler, needed to produce software to run on specific hardware. An embedded system often lacks a graphical user interface and developing on such system would be rather unpleasant.

What is cross-compilation?

To produce software for a specific hardware and system architecture, a system-specific version of a piece of software called a compiler must be used. A compiler is a program that takes human-readable program code as an input, and together with a linker produces software for the target device. This process sometimes includes libraries built for the target device, which, if used in the program, must be obtained and deployed to the target device as well.

The compiler, the linker, and associated software and libraries are very important parts of a functioning computer system; therefore, it might not be a good idea to make modifications to them. Instead of altering the working configuration, it is common to use virtualization to create a separate environment for cross-compilation.

The Yocto Project build system also relies heavily on cross-compilation. The build system configures a cross-compiler and associated libraries for each target machine to produce the target operating system image. The same configuration can also be packaged in an SDK to create software for the target platform from any host machine. This process is visualized in Figure 5. [28]



1 The Build Host produces three toolchains: 1) gcc-cross, which builds the target image. 2) gcc-crosssdk, which is a transitory toolchain and produces relocatable code that executes on the SDKMACHINE. 3) gcc-cross-canadian, which executes on the SDKMACHINE and produces target applications.

2 The SDKMACHINE, which may or may not be the same as the Build Host, runs gcc-cross-canadian to create target applications.

3 The Target Device runs the Target Image and Target Applications.

Figure 5. Cross-development toolchain generation [30].

Introducing the PELUX platform

PELUX is an open-source reference platform for automotive Linux-based systems [31]. It can be used to configure the Yocto build system to create an operating system image with example software that can be used out-of-the-box as a reference for creating informative software components for the automotive environment. The creation of a fully custom Yocto-based Linux distribution would not have been possible to include in the scope of this thesis, therefore the PELUX Yocto image is used as a base platform on top of which the system is built.

With the PELUX base platform also comes a cross-compilation SDK for software development for the target embedded Linux platform [32]. The SDK takes advantage of virtualization, and uses Vagrant, a virtual machine development environment manager [33], to create and maintain a fully configured, Ubuntu-based virtual machine for effortless software development.

6 SOFTWARE DESIGN

At the beginning of a software development process, attention should be paid to the scalability and modularity of a selected design. The various components that go in an application should be identified and visualized at an early stage, to make sure they integrate as expected. An effective way to treat the individual components is to separate them into layers, based on their position in the underlying data flow pipeline found in some form in every application or system.

Figure 6 shows an example of a layered software architecture, where each layer has a role in the underlying flow of data, either by gathering, manipulating, storing, or presenting the data. The layered model encourages defining flexible interfaces between the components even before the development process has begun, which can make integration easier, and enable modularity.

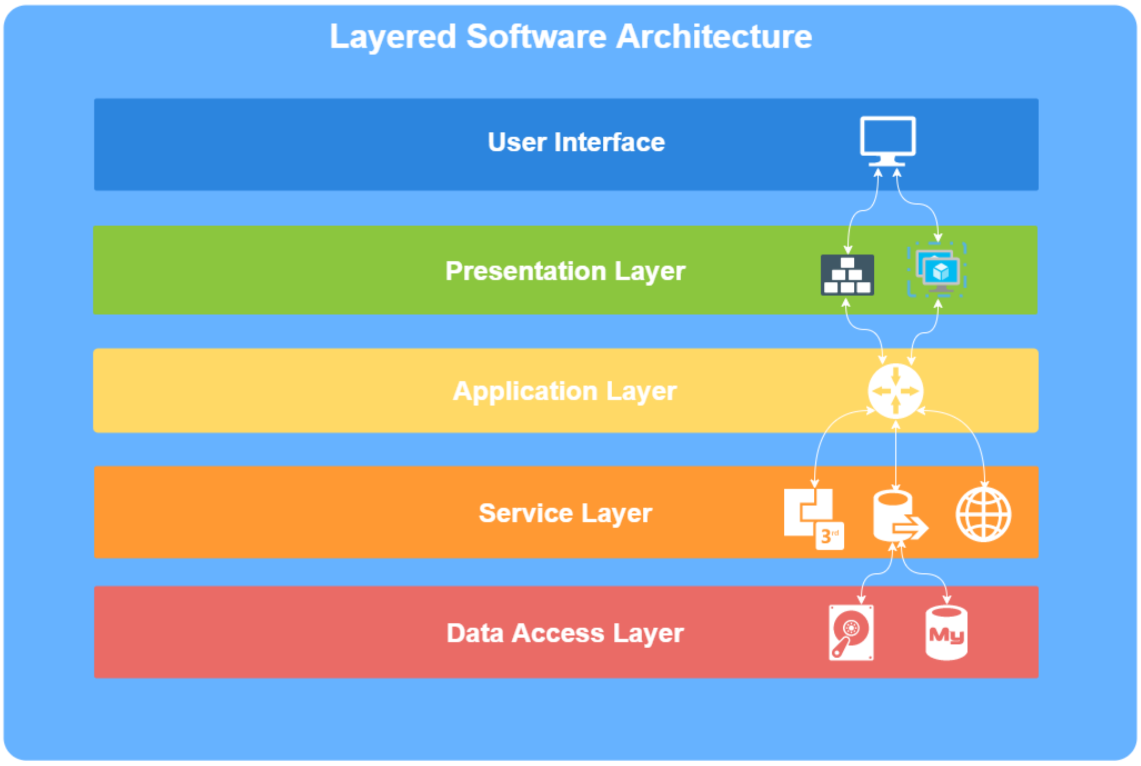


Figure 6. Example Layered Software Architecture

6.1 User Stories and Requirements

One of the first steps in a software development project is determining what to do. Finishing a project is not possible if there is no consensus of the desired outcome, therefore, a list of goals, or, as they're usually called in the software development industry, *requirements,* must be devised. These specifications provide the base for the software design process and help to refine the expected features of the individual components of the device or system.

Despite its demonstrative nature, the project described in this thesis was carried out as much as possible as any real project. Proper project management processes were established and followed along with agile development methods. One well-known method for gathering requirements in agile development paradigm is to produce user stories [34]. This method was also adopted in the design phase of this project. The basic form of a user story follows this pattern:

As a <user role> I want to <goal> [so that <benefit>].

The main purpose of writing user stories to gather requirements is to approach the problem of figuring out the exact needs of the customer in a more natural way, by examining the expected outcome from different perspectives, and writing short descriptions of the outcome from that perspective [35]. This utilization of natural language, breaking down the problem, and its inspection from a higher level can be a powerful tool in initial system design and assessing the outcome of the project, therefore it was applied in this project as well. A full list of requirements for this project can be found in Appendix 1.

If the given resources for a project are limited, it may in some cases be necessary to prioritize the requirements based on their importance for the outcome of the project and specific customer needs. One method for categorizing the requirements is the MoSCoW method, in which the requirements are given labels, such as Must have, Should have, Could have and Won't have. This verbal method can improve the understanding between the parties involved in the development process. [36]

6.2 System architecture

Based on the requirements inferred from the user stories an architecture was designed for the project. The main goal of architecture development was to follow good programming practices to create a fully functioning and maintainable system. These practices include code modularity for easier reuse of components and the establishment of a proper hierarchy between the components. A well-defined hierarchy results in a more rigid architecture, which is easily maintained.

High level overview

The system consists of three individual components, as depicted in Figure 7. Each component has its own role in the system, with the car system performing a major part of the overall functionality. It provides all the necessary Bluetooth interfaces to which the two other components, the remote-control application, and a mobile phone, can connect to exchange information.

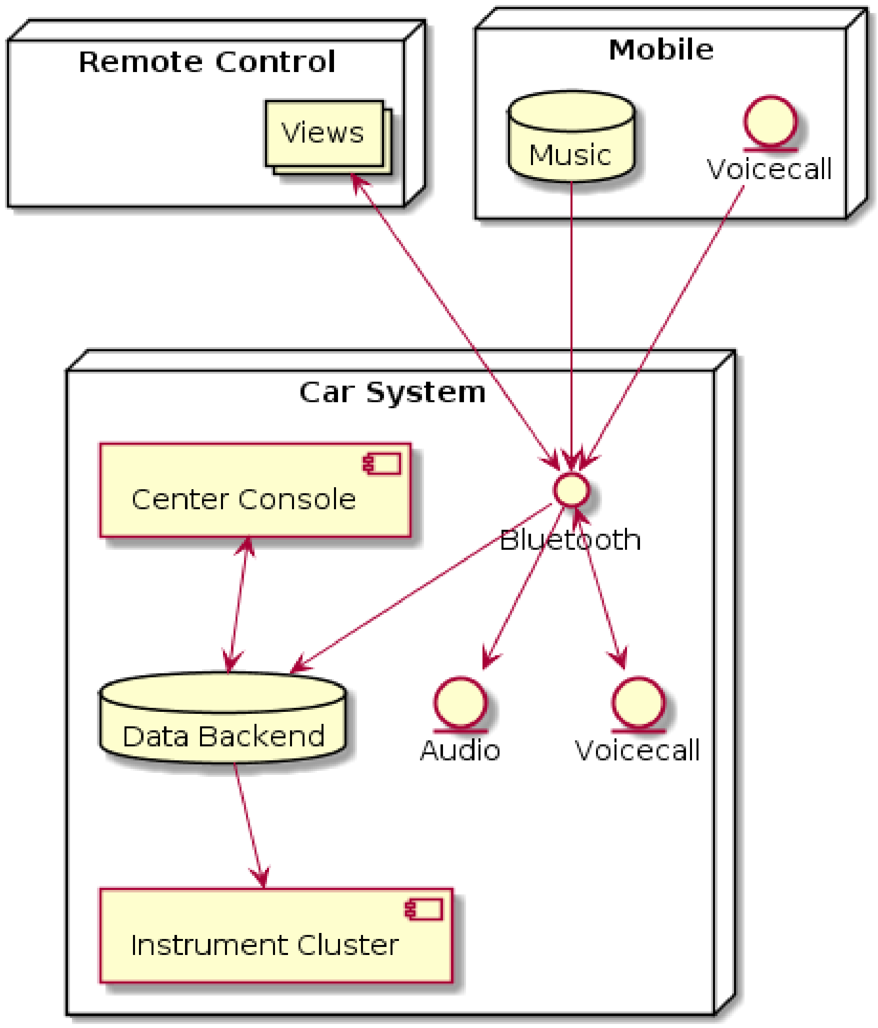


Figure 7. System architecture overview.

The data backend is a singleton class, which means there can only exist one copy of it at any given time. This ensures that each component has a reference to the same version of the class, and thus share the state of the system. The data class consists of a list of member variables that represent the current state of the system. It is wrapped with interface classes that provide getters and setters that can distinguish the sender of the data update signal, and therefore act accordingly depending on who accesses or modifies the data.

The remote-control node shown in Figure 7 is a mobile application that provides a remote viewport to the system status and provides some soft controls for changing the state of predefined components, such as climate settings. The application is also developed using the Qt Framework but runs on an Android device. This further demonstrates the cross-platform development capabilities of the framework. The application communicates with the car system using a Bluetooth serial-like connection profile, RFCOMM, introduced in Chapter 4.4.

The mobile node, on the other hand, represents any modern smartphone with basic Bluetooth voice and media functionalities. It can be connected with the car system using platform-independent and widely implemented Bluetooth features, which enable common functionalities, such as media playback and voice call forwarding.

Both the car system and the remote-control application were designed in such a way, that identifying the different software layers introduced in Figure 6 was trivial. Figure 8 shows how the layered model can be applied to both applications. The modularity of the layered model enabled a high level of code reuse between the two applications, it was only necessary to modify the lower level platform-specific service interfaces, and to scale down the user interface for the mobile application, rest of the components could be shared between the applications.

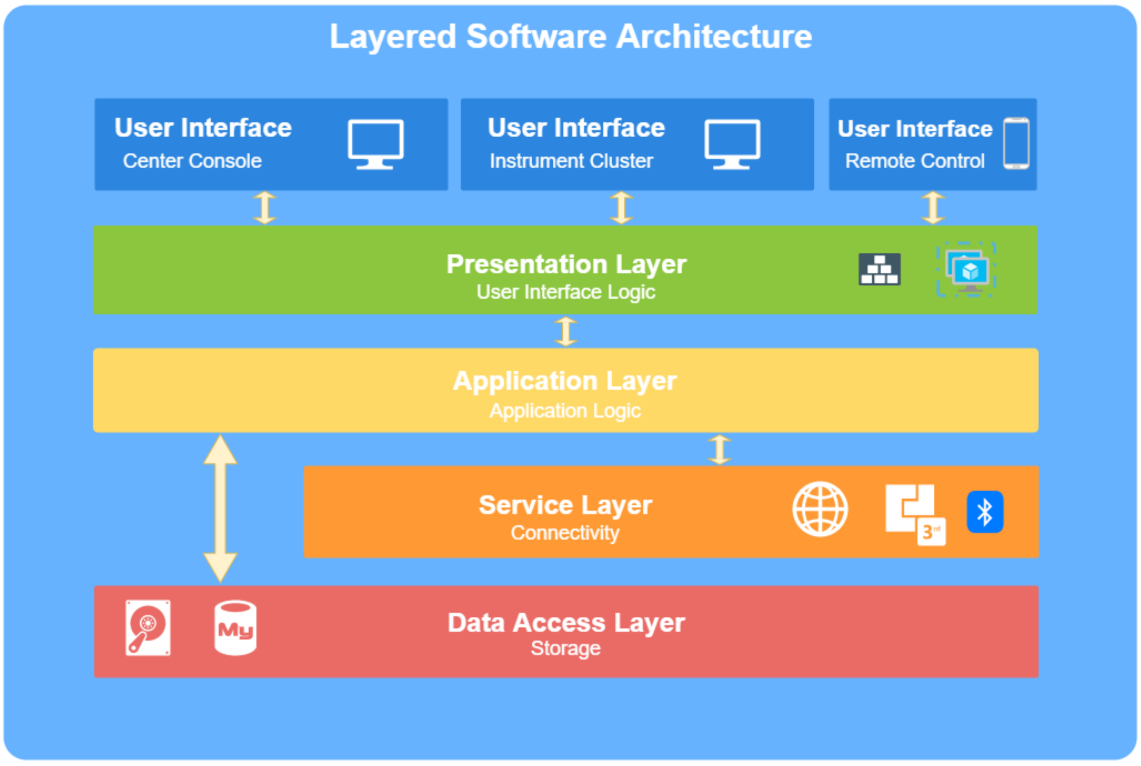


Figure 8. Layered Software Architecture

Car System

The car system consists of two physical screens, both connected to the same computing unit. The instrument cluster screen acts merely as an informative display intended to replace classic analog dashboards, while the center console implements touch control for easy interaction with the system. Both views are rendered by the same application, using OpenGL ES graphics. Specifically, the backend used is EGLFS, a Qt Framework's full-screen version of the popular EGL rendering interface, which enables presenting full-fledged graphics components without the need for a windowing system. This makes it particularly suitable for embedded systems, which seldom implement graphics windowing.

On the application level, the system is separated into two major parts, as seen in Figure 9. The application node depicts the user interface and its logic and includes also the main entry point for the software. The plugins, on the other hand, are shared libraries (.so) that provide the necessary backend functionalities for the application. This separation greatly increases component reuse, since the user interface rarely needs to be adapted for different hardware. The library plugins handle most of the communication with the hardware and in consequence, are subject to modifications when moving onto different platforms.

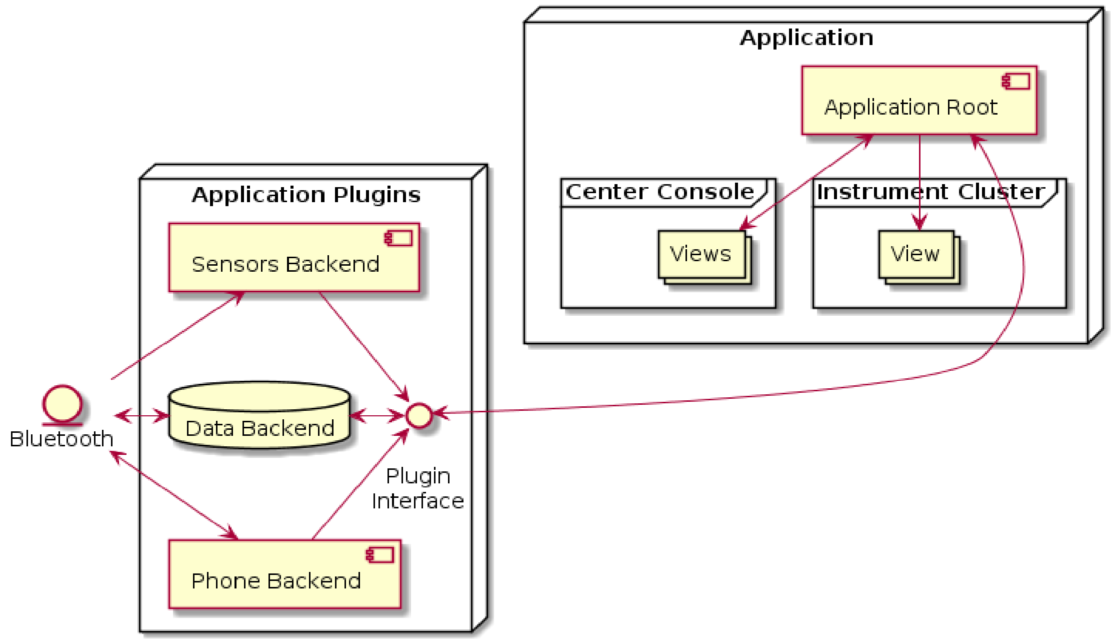


Figure 9. Application architecture.

6.3 User Interface

The final graphical design for the user interface was provided by a professional UI designer, however, the implementation of the graphical and logical components, as well as the functionality and underlying architecture of the user interface did fall within the scope of the project. The following subchapters will introduce the layout and logical architecture of the user interface.

User interface layout

When designing the user interface layout and navigation through it, a great deal of thought was put into ensuring a fluid user experience and maximum usability. The need for ease of use of the system was further amplified by the targeted automotive environment, where interaction with the system should not compromise the safe operation of the vehicle. In other words, using the system should be as intuitive as possible to require a minimal amount of concentration.

In its default state, at the home view, the center console view is designed to give an overview of the state of the system. The homepage, illustrated in Figure 10, consists of several widgets, which are essentially smaller sized views of the full-screen views found on other pages. The widgets provide minimal or no controls compared to their full-screen counterparts, instead, they aim to provide a quick overview with a minimal set of components. The widgets are designed to match the full-screen views in terms of graphical design, to enable maximum component reuse and interface uniformity. When a widget is clicked, the user interface transitions to the corresponding view.

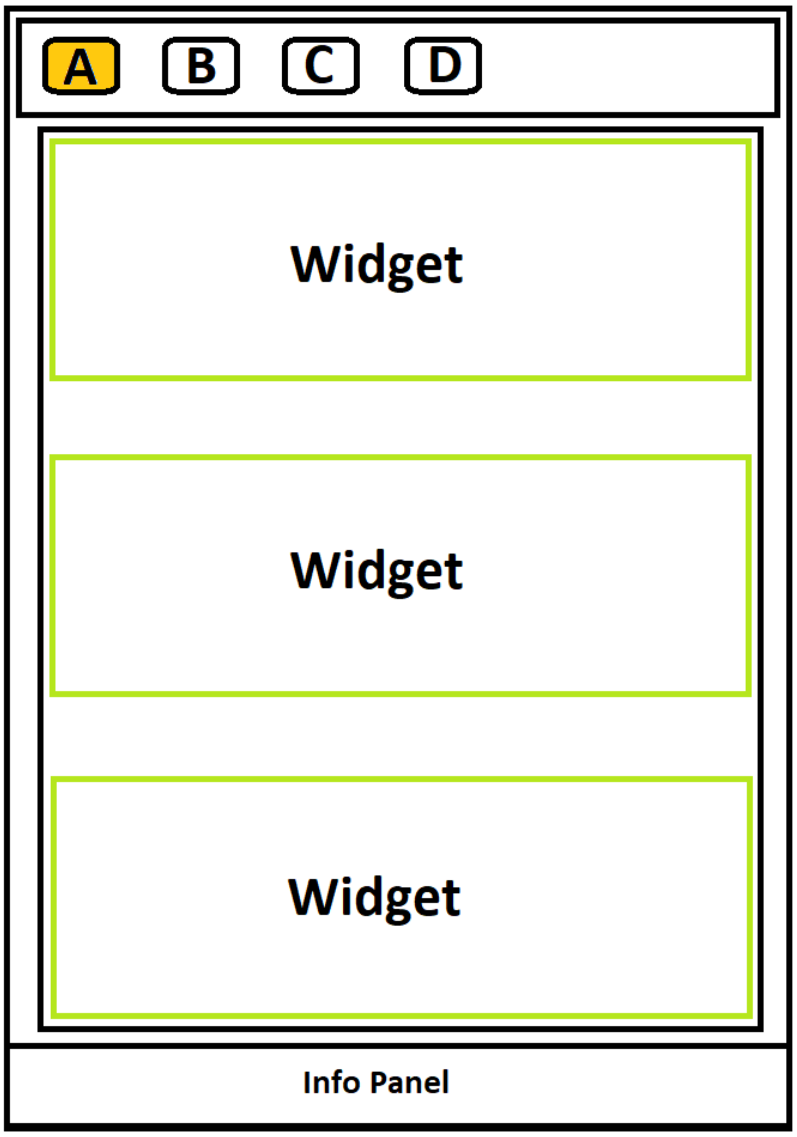


Figure 10. Center console home view layout.

The navigation through the user interface happens either with swiping gestures, or by clicking the navigation bar on top of the screen, as illustrated by Figure 11.

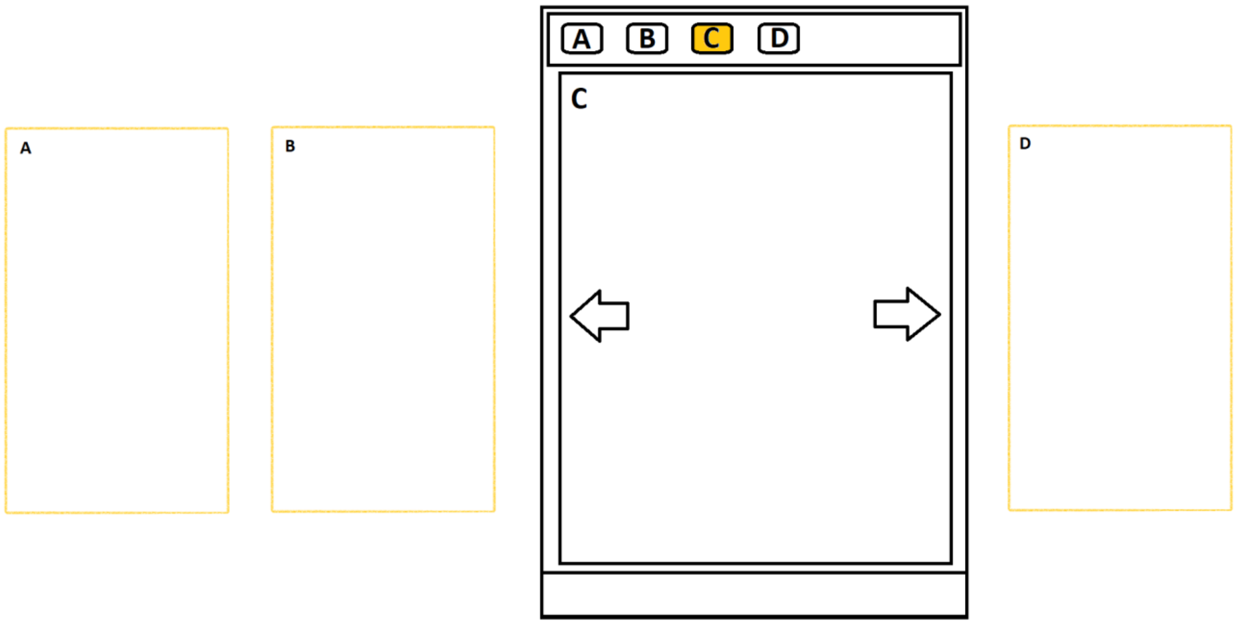


Figure 11. Center console navigation.

The Instrument Cluster view, as illustrated in Figure 12, is a purely informative display mimicking a classical dashboard found in modern cars. Even though the display is not interactive, that is, it cannot be controlled by touch like the Center Console, it consists of multiple active components providing the driver relevant information on the status of the vehicle. The two dials can visualize the current speed and power output of the vehicle, the warning indicators inform the driver of abnormal situations, and the top panel can show the name of the currently playing music track or a caller’s name.

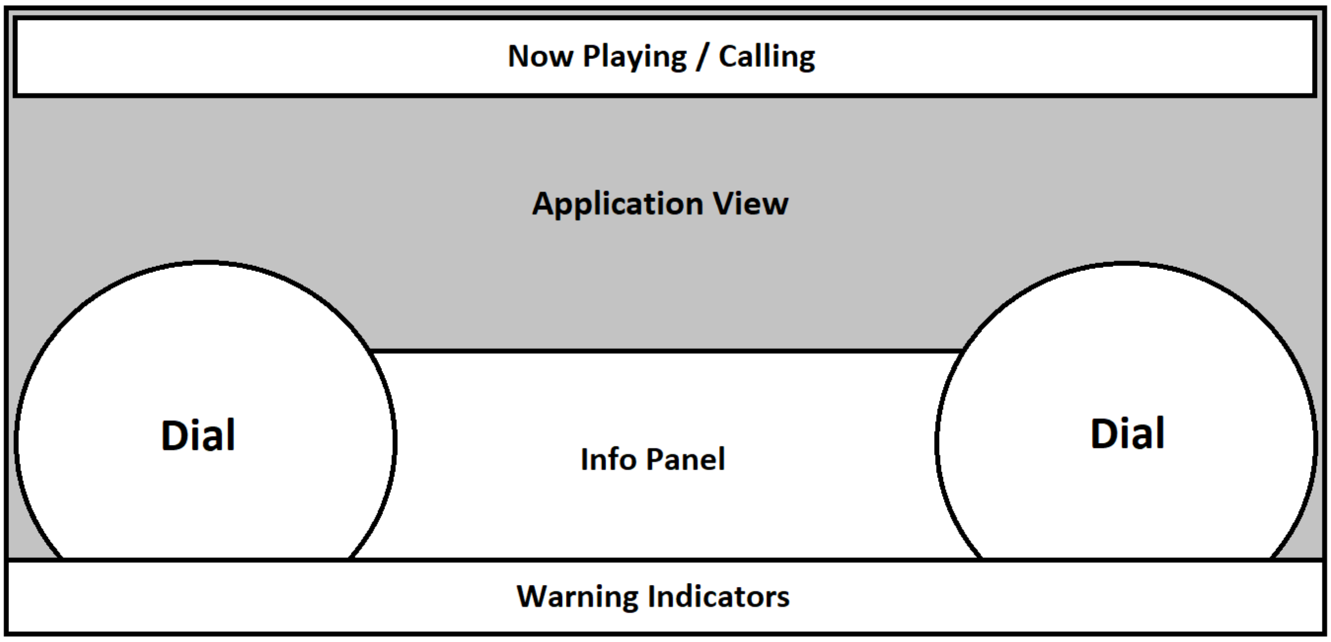


Figure 12. Instrument Cluster layout.

The Application View residing on the background of the instrument cluster is a child of one of the views on the Center Console, providing a widget-like window to the main view. For the purposes of this demonstration, only the navigation widget is viewable on the Instrument Cluster. The following subchapter attempts to shed light on the interconnection of these major components, including the Application View.

The Remote Control Application user interface appearance and layout are designed to match the Center Console user interface as closely as possible. The navigation happens through swiping left or right, or by using the navigation bar on the bottom of the screen, as shown in Figure 13. These unified design and navigation methods ease the use of the different parts of the system since it is necessary to learn only one of the components to use the other. To emphasize the uniform design, the different views on the software are arranged in the same order, apart from the Remote Control application not implementing all the views.

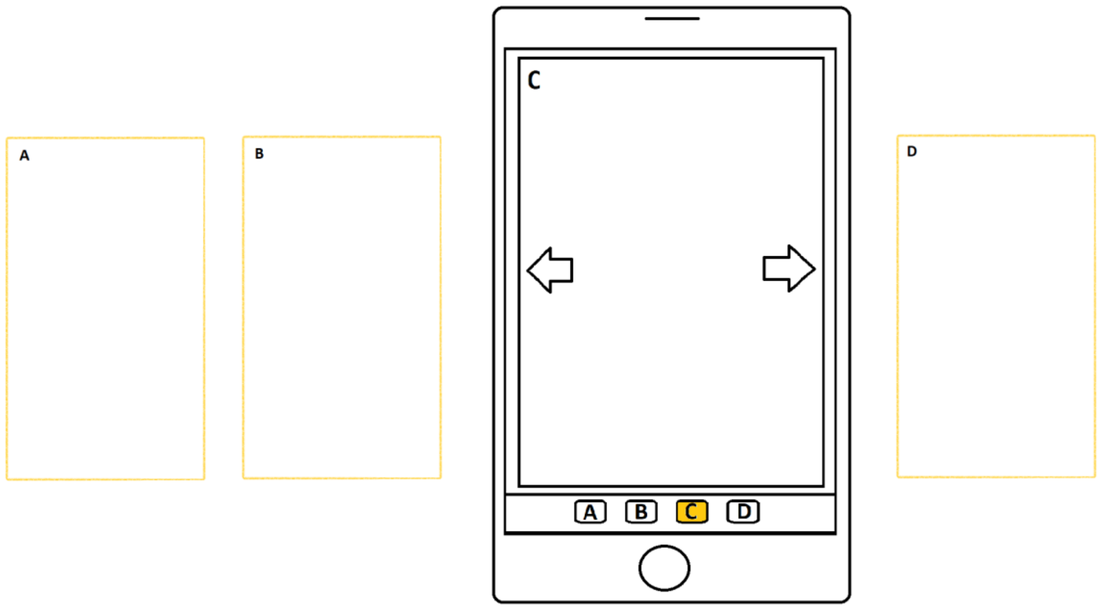


Figure 13. Remote control application layout.

User Interface architecture

Since a great deal of the application logic was implemented on the front-end, a proper architecture had to be designed to ensure maintainability, code reuse, and modularity. The Qt QML language provides a way to nest individual components inside each other to form hierarchies between them in a tree-like structure, with a single root element followed by all of its children. The components can be loaded in or created and added to the tree dynamically, which allows starting up the application rapidly with only a few necessary components.

The main, or application root element, is the most important element of the user interface hierarchy. It acts as a parent for all the other components and has references to all the needed external components, such as the data backend, which it can then pass by reference to all of its children. This hierarchy is visualized in Figure 14. The application root element has a reference to or “owns” both screen elements, the Instrument Cluster and the Center Console, which both in turn have references to their child components, which can also have any number of children.

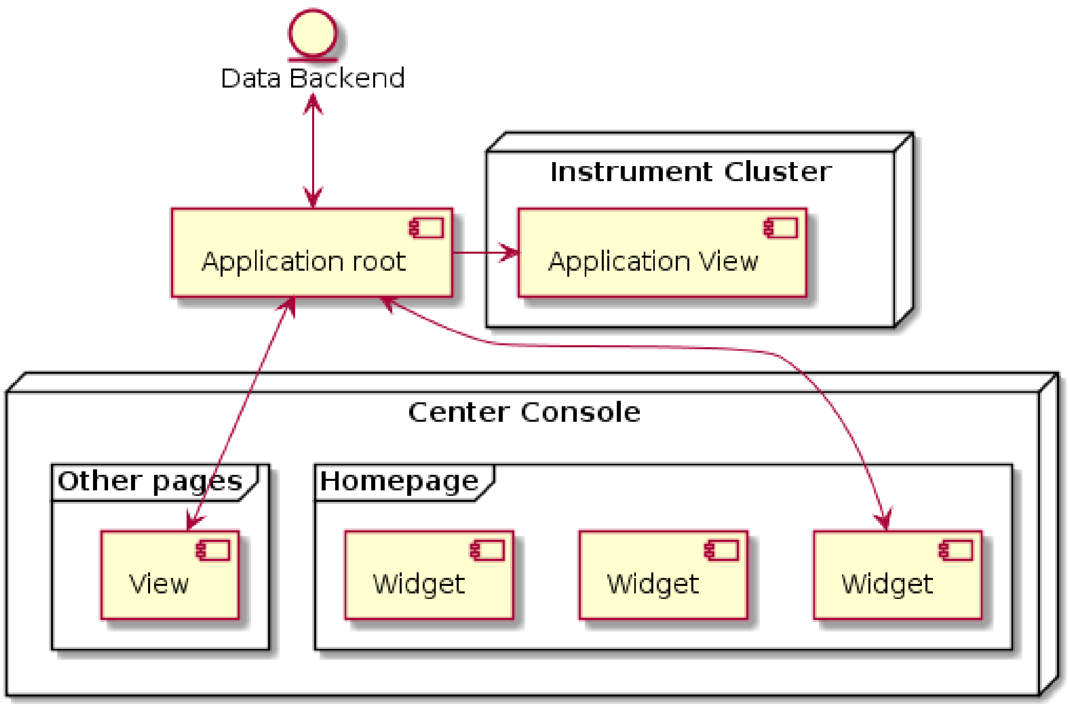


Figure 14. User interface architecture.

7 IMPLEMENTATION

After the software design phase, the realization, also known as the implementation phase, was carried out. This chapter discusses the implementation experience and difficulties faced during the process.

7.1 Operating System Configuration

The first step in creating a working system is the configuration of the platform on which the software will run on. Since the PELUX Yocto operating system image is intended for reference only, it provides no software for playing audio, and the BlueZ Bluetooth protocol stack included in the image is misconfigured for audio playback. This means that a fair amount of research and configuration was needed to play music from the device speakers.

Audio software

In order to play sound from the speakers installed to the system, an audio driver is required. ALSA (Advanced Linux Sound Architecture) is a Linux kernel software framework that provides a generalized API to the different audio drivers used on different hardware [37]. This module, called *alsa-lib* had to be integrated into the operating system image to enable communication with the system audio card.

Bluetooth Audio

The installed BlueZ Bluetooth protocol stack did not, by default, provide Bluetooth audio functionality, instead it had to be configured and built with audio profile support to be able to provide an audio stream to a sound server.

Sound server

In addition to the changes to the BlueZ Bluetooth protocol stack and the introduction of an audio driver, a sound server was required to provide a level of abstraction between the two, for them to communicate with each other. Pulseaudio is well supported with BlueZ and was chosen to act as a sound server in this configuration.

The Pulseaudio sound server default configuration was incapable of handling Bluetooth-based audio streams and had to be reconfigured through a configuration file. The Bluetooth audio functionalities were enabled by including BlueZ device and discovery modules into the configuration, as well as a loopback module, which enabled routing the audio stream to the physical speakers.

7.2 UI design

The user interface appearance, and most of the layout were designed by a professional UI/UX designer. The design process, however, entailed two-way communication between the designer and the development team to properly visualize the possible features, layout, and functionalities of the infotainment system. The user interface designer took part into multiple weekly meetings, and several different drafts and designs were exchanged between the two parties. As the implementation of the user interface carried on, the designer was shown snapshots of the status of the projects, and the fine-tuning of the user interface could continue until the end of the realization phase.

7.3 Resources

After the implementation of the core functionalities of the system and the user interface, as well as the general layout thereof, two software developer interns took part in the project. They were guided to take part in the implementation of the individual graphical components of the user interface. Their guidance brought along some overhead in terms of time needed in guidance and additional project management tasks, however, it proved to be a valuable learning experience in addition to the skills gained during the actual programming and configuration.

7.4 Project Management

The project’s management followed established agile development guidelines, or paradigms, to ensure overall quality of the project, and to help defining the scope of the project. The early adoption of a Kanban board as the task management tool and the clear list of requirements gathered from user stories were especially helpful when more developers joined the project.

8 RESULTS

The purpose of the project described in this thesis was to create a fully functioning system resembling an automotive infotainment system to assess the suitability of the Qt Framework in modern embedded user interface development on a custom embedded Linux platform. In the beginning of the project, a set of requirements, introduced in Chapter 6 and listed in Appendix 1, were devised to create a clear plan of the scope of the project and to eventually evaluate the result of the project. This chapter aims to evaluate the result of the project based on those requirements, as well as summarize the overall experience from the point of view of the author.

8.1 Requirements fulfilment

System architecture and overview

The user stories listed in Table 1 and the requirements inferred from them imposed the physical structure of the system, consisting of three separate screens, working as two separate systems.

Table 1. User stories regarding architecture.

User story no.	**As a** *<type of user / persona>*	**I want to** *<goal / objective>*
7	Driver	See vehicle diagnostics information on the center console
3	Driver	See the instrument cluster
6	Passenger or vehicle owner	See basic vehicle information on my phone

These two components were the car system, and the mobile application, which were both successfully created during the project using the Qt Framework. The car application ran on the targeted Embedded Linux platform as a full-screen application on two physically separate screens, without the need for a windowing system often not found in embedded systems. The system was able to independently boot up and start the application automatically without the need for user interaction. This was made possible with a valid operating system configuration when building the operating system image based on the PELUX platform, built with the Yocto Project.

User interface

The user stories listed in Table 2, in the previous sections, and requirements thereof state the desired properties of the user interface. Additionally, the fluidity, smoothness and user-friendliness were assessed based on feedback from individuals outside the project.

Table 2. User stories regarding user interface.

User story no.	**As a** *<type of user / persona>*	**I want to** *<goal / objective>*
1	Driver	Call a person listed as a contact in my mobile phone using the infotainment system
2	Passenger	Change navigation destination and find places nearby with my phone
4	Driver	Answer calls to my phone using the infotainment system.
5	Driver	Navigate to a certain place
6	Passenger or vehicle owner	See basic vehicle information on my phone
7	Driver	See vehicle diagnostics information on the center console
8	Passenger	Control the vehicle music player with my phone
10	Driver	Adjust the climate control
11	Passenger	Adjust the climate control with my phone

The requirements inferred from the user stories in Table 2 describe a system with two separate components, the vehicle unit, and the controlling mobile application, both equipped with several different views for showing the information in smaller partitions, and both having nearly the same functionality, with the mobile application having soft control over the vehicle system.

The vehicle system included views for diagnostics, climate controls, and media, with simple user control using icons as buttons, and a few sliders. The navigation view was a rather complex combination of an online map with 3D rendering of buildings, and a search and history view capable of changing the destination of the navigation simulation. The two views changed their respective size ratios on the screen with relatively smooth transition animations to react to user interaction, such as expanding the search view. The navigation view simulated navigating to a user-selected destination by artificially creating and following coordinates along a route and updating the map according to the coordinate information.

The mobile application recreated most of the views seen on the vehicle system in a mobile-friendly manner. During development, it was possible to reuse the majority of the program code used in the vehicle system, especially in individual components, to create the mobile application. Some internal layout changes were vital for maintaining the readability of the user interface on the smaller mobile screen, but the visual uniformity of the user interfaces was preserved. As in the vehicle unit, the mobile application had a fluent look and feel, with smooth transitions between the views. Low mobile network bandwidth created notable latency on map tiles loading on the navigation view, however, the perceived refresh rate did not suffer, but the missing tiles appeared white.

The user interface also received positive feedback from outside individuals who interacted with the system.

Connectivity

The user stories listed in Table 3 and requirements inferred from them specify the different connectivity aspects of the system. These requirements describe a system with four communication methods; Bluetooth serial connectivity, Bluetooth media playback and control, Bluetooth voice call and contacts profile connection, and Internet connectivity. Of these four methods, the Internet connection is platform-specific and needed no configuration in this project. The Bluetooth connectivity did require changes to the operating system configuration, described in Chapter 7. However, the final vehicle unit was able to successfully communicate with the mobile phone and the application running on it using the three aforementioned communication methods or forms.

Table 3. User stories regarding connectivity.

User story no.	**As a** *<type of user / persona>*	**I want to** *<goal / objective>*
1	Driver	Call a person listed as a contact in my mobile phone using the infotainment system
2	Passenger	Change the destination and find places nearby with my phone
4	Driver	Answer calls to my phone using the infotainment system
6	Passenger or vehicle owner	See basic vehicle information on my phone
8	Passenger	Control the music player with my phone
9	Driver	Play music from my mobile phone
11	Passenger	Adjust the climate control with my mobile phone

The climate, media, and navigation view controls, as well as the diagnostics information shown on the companion application were conveyed using the Bluetooth Serial Port Profile with no notable latency or errors. It was possible to separate the serial connection from the media and voice call ones, enabling the use of a separate mobile phone, or multiple, for viewing the diagnostics data and controlling the vehicle unit.

The media and voice call profiles provided no direct control over the system other than providing the contact information and the voice call and music player audio. The user interface was able to respond to changes in the connection of these profiles by dynamically creating or destroying the corresponding views. More precisely, the media and voice call views were dynamically removed or added to the grid of widgets on the homepage and were inaccessible through swipe gestures when no connection was established.

9 TESTING

To verify the correct functionality and integration of all the components, a testing plan was created for the project. The testing plan concentrated on the interoperability and integration of the various features, especially regarding connectivity, where multiple devices might or might not be connected to the system at the same time via separate Bluetooth profiles.

The testing plan consisted of uniquely labelled test cases, like those shown in Table 4, giving information on the prerequisites for the test, how to execute the test, and the expected outcome. In addition, the unique labelling of the test cases allowed defining dependencies between the tests, which made fulfilling the test prerequisites and finding the root cause for a possible problem easier.

Table 4. Example test cases

Test id	**View**	**Scenario**	**Precondition**	**Steps**	**Expectation**	**Result**
PHONE-CONN-1	Home	Connecting a media device should show media and phone widgets on primary screen Home view	-	1. Bluetooth icon pressed on home screen 2. Device selected from the list of available devices 3. User accepts pairing on device	1. List of available devices shown properly 2. Devices connect successfully 3. Phone and media widgets shown on Home view	OK
PHONE-WIDG-3	Home	User should be able to call a number from phone widget	PHONE-CONN-1	1. Number grid icon pressed on phone widget 2. Phone number selected by pressing numbers 3. Call button pressed 4. Disconnect button pressed or call ended by recipient	1. Number grid shown properly 2. Typed phone number shown correctly 3. Call started on connected device 4. Call disconnects properly 5. Number and call duration added to history	OK

The role of the testing plan during the development was to function as a checklist when assessing the effect of a change in the software, to make sure everything was still functioning properly. This manual sorting through the test plan and determining which, if not every, test to execute after each minor change was cumbersome, and therefore implementing test automation was considered.

In addition to the need for a test automation pipeline, no systematic method for testing the user interface was implemented in the development process. Based on purely visual inspection, making changes to the user interface took a lot more resources than what an automated routine would have enabled.

10 CONCLUSIONS AND FUTURE WORK

The purpose of this thesis was to create a demonstrative automotive In-Vehicle Infotainment (IVI) system from a gathered set of requirements to assess the suitability of the Qt Framework in the creation of modern embedded user interfaces, using a custom embedded Linux distribution as a platform. Proper project management methods were used to replicate real-world development processes to increase the accuracy of the assessment.

The collection of requirements was fulfilled completely with only minor negative notions found through testing. The Qt Framework's capability to act as a sole development toolkit for a project of this scope proves great maturity and flexibility of the framework and makes it a good option for embedded and mobile user interface development. The cross-platform and especially embedded Linux support further increase its usefulness in the described environment.

From the developer's point of view, the Qt Framework enabled effortless development even in the lower level, hardware-oriented tasks by providing a suitable level of abstraction, while providing full control over the underlying system. Learning to use the framework was straightforward, and even the more complex, powerful features it provides were within the reach of a developer with no prior experience with the technology.

Although the purpose was to create a fully functioning system for demonstration purposes, several features remain to be implemented to bring the system's level of completeness on par with its real-world counterparts. The Qt Company offers a selection of proprietary modules, which could enable taking into account some more stringent requirements for the system such as functional safety, by using modules such as Qt Safe Renderer. The operating system configuration itself would also need support for over-the-air updates and booting directly into the infotainment environment.

Chapter 9 also introduced some considerations for improving the development process, especially testing. The implementation of a fully automated build and testing pipeline fell out of the scope of this project; however, such configuration would possibly have enabled shorter delays from a code change to a new, tested feature by automating and integrating the building and testing routines. Such frameworks exist already and should be implemented in a production-grade system.

REFERENCES

[1] Qt Wiki Contributors, “About Qt - Qt Wiki,” 26 4 2019. [Online]. Available: https://wiki.qt.io/About_Qt. [Accessed 16 06 2019].

[2] The Qt Company Ltd., “User Interfaces,” 2019. [Online]. Available: https://doc.qt.io/qt-5/topics-ui.html. [Accessed 16 6 2019].

[3] The Qt Company Ltd., “Qt Widgets,” 2019. [Online]. Available: https://doc.qt.io/qt-5/qtwidgets-index.html. [Accessed 16 6 2019].

[4] The Qt Company Ltd., “Model/View Programming,” 2019. [Online]. Available: https://doc.qt.io/qt-5/model-view-programming.html. [Accessed 16 6 2019].

[5] The Qt Company Ltd., “Qt Quick,” 2019. [Online]. Available: https://doc.qt.io/qt-5/qtquick-index.html. [Accessed 16 6 2019].

[6] The Qt Company Ltd., “QML Applications,” 2019. [Online]. Available: https://doc.qt.io/qt-5/qmlapplications.html. [Accessed 16 6 2019].

[7] The Qt Company Ltd., “Overview - QML and C++ integration,” 2019. [Online]. Available: https://doc.qt.io/qt-5/qtqml-cppintegration-overview.html. [Accessed 16 6 2019].

[8] The Qt Company Ltd., “qmake Manual,” 25 06 2019. [Online]. Available: https://doc.qt.io/qt-5/qmake-manual.html. [Accessed 25 06 2019].

[9] The Qt Company Ltd., “The Qt Resource System,” 25 06 2019. [Online]. Available: https://doc.qt.io/qt-5/resources.html. [Accessed 25 06 2019].

[10] The Qt Company Ltd., “Signal and Handler Event System,” 25 06 2019. [Online]. Available: https://doc.qt.io/qt-5/qtqml-syntax-signals.html. [Accessed 25 06 2019].

[11] The Qt Company Ltd, “Qt Safe Renderer Overview,” [Online]. Available: https://doc.qt.io/QtSafeRenderer/qtsr-overview.html. [Accessed 23 10 2019].

[12] The Qt Company Ltd., “Signals & Slots,” 25 06 2019. [Online]. Available: https://doc.qt.io/qt-5/signalsandslots.html. [Accessed 25 06 2019].

[13] The Qt Company Ltd., “The Meta-Object System,” 26 06 2019. [Online]. Available: https://doc.qt.io/qt-5/metaobjects.html. [Accessed 26 06 2019].

[14] The Qt Company Ltd., “Using meta-qt5,” 26 06 2019. [Online]. Available: https://doc.qt.io/QtForDeviceCreation/qtee-meta-qt5.html. [Accessed 26 06 2019].

[15] The Qt Company, “GitHub - meta-qt5,” 26 06 2019. [Online]. Available: https://github.com/meta-qt5/meta-qt5. [Accessed 26 06 2019].

[16] The Linux Foundation, “Automotive Grade Linux Releases Open Infotainment Platform Built by Automakers and Suppliers,” 28 12 2016. [Online]. Available: https://www.automotivelinux.org/announcements/2016/12/28/automotive-grade-linux-releases-open-infotainment-platform-built-by-automakers-and-suppliers. [Accessed 24 04 2019].

[17] Linux Foundation, “Industry Momentum for Automotive Grade Linux Continues to Grow with Five New Members,” 17 12 2018. [Online]. Available: https://www.linuxfoundation.org/press-release/2018/12/industry-momentum-for-automotive-grade-linux-continues-to-grow-with-five-new-members/. [Accessed 24 04 2019].

[18] Linux Foundation, “Is Yocto Project for you,” 2018. [Online]. Available: https://www.yoctoproject.org/is-yocto-project-for-you/. [Accessed 24 04 2019].

[19] Linux Foundation, “Yocto Project Overview and Concepts Manual,” 2019. [Online]. Available: https://www.yoctoproject.org/docs/2.6.2/overview-manual/overview-manual.html. [Accessed 24 04 2019].

[20] I. Poole, “What is Bluetooth Technology: basics & overview,” Electronics Notes, [Online]. Available: https://www.electronics-notes.com/articles/connectivity/bluetooth/what-is-bluetooth-technology-basics-summary.php. [Accessed 09 09 2019].

[21] Bluetooth SIG, “Bluetooth Market Update 2019,” 04 2018. [Online]. Available: www.bluetooth.com. [Accessed 10 11 2019].

[22] I. Poole, “Bluetooth radio interface, modulation, & channels,” Electronics Notes, [Online]. Available: https://www.electronics-notes.com/articles/connectivity/bluetooth/radio-interface-modulation-channels.php. [Accessed 2019 09 09].

[23] Wikipedia Contributors, “List of Bluetooth protocols,” 01 08 2019. [Online]. Available: https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols. [Accessed 2019 09 09].

[24] Wikipedia Contributors, “List of Bluetooth profiles,” 01 08 2019. [Online]. Available: https://en.wikipedia.org/wiki/List_of_Bluetooth_profiles. [Accessed 2019 09 09].

[25] Bluetooth SIG, Inc., “Bluetooth Core Specification v5.1,” 21 January 2019. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification/. [Accessed 10 11 2019].

[26] I. Poole, “Bluetooth profiles,” [Online]. Available: https://www.electronics-notes.com/articles/connectivity/bluetooth/profiles.php. [Accessed 2019 09 09].

[27] freedesktop.org, “About PuldeAudio,” 29 12 2014. [Online]. Available: https://www.freedesktop.org/wiki/Software/PulseAudio/About/. [Accessed 16 11 2019].

[28] Wikipedia Contributors, “OBject EXchange,” 05 04 2019. [Online]. Available: https://en.wikipedia.org/wiki/OBject_EXchange. [Accessed 2019 09 14].

[29] BlueZ Project, “BlueZ - About,” 2016. [Online]. Available: http://www.bluez.org/about/. [Accessed 16 11 2019].

[30] Linux Foundation, “Yocto Project Mega Manual,” 10 2019. [Online]. Available: https://www.yoctoproject.org/docs/3.0/mega-manual/mega-manual.html. [Accessed 10 11 2019].

[31] Luxoft Sweden AB, “PELUX,” [Online]. Available: https://pelux.io. [Accessed 21 04 2019].

[32] Luxoft Sweden AB, “PELUX Development Handbook,” 18 04 2019. [Online]. Available: https://pelux.io/software-factory/master/. [Accessed 21 04 2019].

[33] HashiCorp, “Introduction to Vagrant,” [Online]. Available: https://www.vagrantup.com/intro/index.html. [Accessed 10 11 2019].

[34] R. Alt-Simmons, “Gathering Analytic User Stories,” in *Agile by Design: A Project Manager's Guide to Analytic Lifecycle Management*, John Wiley & Sons, Incorporated, 2015, pp. 90-98.

[35] Wikipedia contributors, “User Story,” Wikipedia, The Free Encyclopedia, 2 5 2019. [Online]. Available: https://en.wikipedia.org/wiki/User_story. [Accessed 6 5 2019].

[36] Wikipedia Contributors, “MoSCoW method,” Wikimedia Foundation, Inc., 3 11 2019. [Online]. Available: https://en.wikipedia.org/wiki/MoSCoW_method. [Accessed 23 11 2019].

[37] ALSA Team, “Advanced Linux Sound Architecture (ALSA) project homepage,” 30 10 2013. [Online]. Available: https://www.alsa-project.org/wiki/Main_Page. [Accessed 21 04 2019].

[38] The Qt Company Ltd., “Property Binding,” 25 06 2019. [Online]. Available: https://doc.qt.io/qt-5/qtqml-syntax-propertybinding.html. [Accessed 25 06 2019].

[39] L. R. Albert Huang, “Bluetooth for Programmers,” 2005. [Online]. Available: http://people.csail.mit.edu/rudolph/Teaching/Articles/BTBook.pdf. [Accessed 27 August 2019].

User stories and requirements

Below is a list of all the user stories and the gathered requirements.

As driver I want to be able to call a person listed as contact in my mobile phone using the infotainment system.

- *Get phonebook from a mobile phone connected via Bluetooth*
- *Create user interface for voice call control*
- *Implement voice call functionality*

As a passenger I want to be able to change navigation destination and find places nearby with my phone.

- *Implement serial connection between phone and car systems*
- *Implement mobile navigation view with place search functionality*
- *Update navigation status to mobile phone*

As driver I want to see the instrument cluster.

- *Create a separate view on a separate screen for the instrument cluster*

As driver I want to be able to answer calls to my phone using the infotainment system.

- *Inform the driver of incoming calls on instrument cluster*
- *Inform the driver of incoming calls on the center console*
- *Implement voice call functionality*
- *Implement controls for answering a phone call*

As driver I want to navigate to a certain place.

- *Create a navigation view for the center console*
- *Create a navigation view for the instrument cluster*
- *Implement navigation functionality*
- *Implement place search functionality*

As a passenger or a vehicle owner I want to be able to see basic vehicle information on my phone.

- *Implement serial connection between phone and car systems*
- *Implement mobile diagnostics view*

As driver I want to see vehicle diagnostics information on the center console.

- *Create a vehicle diagnostic view for the center console*
- *Implement diagnostics data backend*

As a passenger I want to be able to control the music player with my phone.

- *Create a media view for the center console*
- *Implement serial connection between phone and car systems*
- *Implement mobile media view*
- *Implement media playback functionality*

As driver I want to play music from my phone.

- *Implement media playback functionality*
- *Create a media view for the center console*
- *Create a media view for the instrument cluster*

As a driver I want to adjust the climate control.

- *Create a climate control view for the center console*
- *Implement climate control data backend*

As a passenger I want to be able to adjust the climate control with my phone.

- *Implement climate control data backend*
- *Implement serial connection between phone and car systems*