CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

LINUX-BASED AUTOMOTIVE AUDIO SYSTEM

A graduate project submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

By

Jordan Yelloz

December 2016

The graduate project of Jordan Yelloz is approved:

Yevgeniya Mushkatblat	Date
Dr. John Noga	Date
Dr. Jeffrey Wiegley, Chair	Date

California State University, Northridge

Table of Contents

List of Figures

ABSTRACT

LINUX-BASED AUTOMOTIVE AUDIO SYSTEM

By

Jordan Yelloz

Master of Science in Computer Science

In recent years, the amount of personal electronic devices has greatly increased. With a host of different manufacturers and standards, there has been a difficulty to properly integrate these devices. An product category which suffers noticeably from this shortcoming is the automotive field. While car companies and consumer electronics companies have been making progress recently and introducing new products to the market, their solutions are suboptimal for many consumers. High cost, lack of features, and lack of customizability are all issues that diminish the efforts of the industry making these devices. This paper mentions some existing products and describes their inadequacies. The solution to this problem is a new platform based on the latest advances free software and commodity hardware. The final product includes a set of software with a graphical user interface capable of displaying necessary information as well as a reference hardware platform from which custom purpose-fit hardware can be built. This computer will be able to connect directly to the amplifier and electrical power supply of an automobile to operate as expected inside the car. The result should be a reasonably-priced solution for knowledgeable users who want a simplified, but customizable solution for their automotive audio needs.

Chapter 1

Introduction

This paper documents a solution intended for automotive audio applications. What is proposed uses a computer and some relatively new, free software technologies to achieve the desired level of functionality. Chapter 1 describes the state of after-market car audio systems at the time of this writing. Chapter 1 also details inadequacies of other attempts to install a computer in an automobile. Chapter 2 provides information on related technologies in the areas of operating systems, peripherals, hardware emulation technologies, and some existing solutions. Chapter 3 contains the entirety of the solution put forward by me. This includes the requirements, design, and implementation performed through this project's duration. Chapter 4 contains the results of this project. Included are some ideas for extending the solution by adding some experimental and industry-standard features.

1.1 Problem

With all the devices available in the market, none of the offerings supports the desired feature set and overall quality.

At the time this project started, low-cost models only supported analog connections to personal media devices (iPods, etc.), if any. This type of connection has several drawbacks and few advantages. By nature, analog connections are lossy with respect to audio quality. Furthermore, an analog connection limits the level of communication between the media device and the audio system.

The implication of this limit are that there is be no effective method for the audio system to control the media device or access any of its content (other than the emitted audio signal).

One key feature lacking in all commercial offerings is customizability. Since these devices are marketed and packaged as consumer electronics, they are not expected to provide the level of customizability that some users would prefer. From a developer's standpoint, increasing the level of customizability can reduce the reliability of the device and increase support costs. While this is understandable, my solution is designed for a specific set of users where support is less of an issue.

1.2 Objectives

The objectives of this project are to produce a working product that meets the requirements described in section 3.2. Inside the computer will be a set of software packages capable of providing several features that integrate personal electronic devices such as mobile phones and personal media player with the car audio system. Also, the device should support GPS geolocation.

Chapter 2

Related Works

In 2007, at the start of this project, there were few options for advanced automotive audio systems. In 2016 there are several options to choose from and large commercial entities have deployed their software in new cars. It is also possible to purchase full-featured hardware running advanced operating systems such as Android and install them into any compatible automotive dashboard.

2.1 Complete Solutions

There are a few existing solutions to the problem described in this project. They are fairly new and some are backed by large entities as well as small independent developers.

2.1.1 Harman Connected Car

Harman Connected Car [1] is a large software project claiming to support bringing a wide range of automotive subsystems within a multimedia environment. This project appears to an extensive and professional solution and is currently deployed or being deployed in automobiles of at least 24 major international car brands [2]. Features such as audio, communications, air conditioning integration, navigation, voice recognition, cameras, and sensors are supported. Furthermore, the solution also includes a remote management platform to handle over-the-air software updates.

2.1.2 iCarus

The iCarus Car PC [3] is a commercially available independent hardware and software solution that can be inserted into a car dashboard and used for similar purposes as this project. The software runs Linux with some proprietary application code. The hardware consists of a custom board with several peripherals such as 802.11 and Bluetooth wireless networking connected to an off-the-shelf Raspberry Pi single-board PC. At the time of this writing, a fully functioning unit can be purchased directly from the maker for 284 EUR [4].

2.1.3 Wind River Helix Chassis

Helix Chassis [5], by Wind River, is another full software solution by a major technology company. It offers a similar set of capabilities as the Harman Connected Car solution. Wind River claims over 70 customers implementing solutions with the Helix Chassis suite. A sub-project of the Helix Chassis suite, Cockpit offers a Linux-based platform based on the to-be-mentioned GENIVI framework with a custom JavaScript-based user interface and application system.

2.2 Frameworks and Standards

There are also efforts to provide open standards as well as a base platform for automotive information and audio systems.

2.2.1 Android and Android Auto

Android is a Linux-based operating system targeted toward mobile and entertainment devices such as phones, tablets, and television set top boxes. It is possible for independent developers to leverage the Android Open Source Project in building a custom operating system for automotive systems on top of existing Android components. Android has also released its own automotive variant, Android Auto, which is deployed in several 2016 and 2017 car dashboards as well as available in Android phones docked inside a car. Application development in Android is typically carried out with Java or other languages compatible with the Java Virtual Machine.

2.2.2 GENIVI

GENIVI is an alliance of several automotive manufacturers and automotive audio suppliers coordinating on several open source components to provide the core functionalities necessary to provide the foundation of a custom automotive operating system [6].

2.2.3 Freesmartphone.org

Freesmartphone.org is an open source project to provide a similar set of middleware interfaces and implementations for independent development of fully user-customizable mobile phones and tablet devices [7]. While this project is not focused on automotive audio, many of its concerns are shared with an automotive audio project.

Chapter 3

My Solution

3.1 Overview

This chapter details my solution to the problem. The traditional software engineering task-based organization is used. Section 1 describes the requirements of the solution, focusing on features. Section 2 describes the architecture of the solution. Section 3 describes the implementation of the solution. Hardware and software topics are covered in section 3, from physical installation to the graphical user interface. Section 4 describes the testing regime undertaken during the development of this project.

3.2 Requirements

This project's goal is to produce a software platform that is capable of powering a reference hardware platform based on retail components providing audio, some mobile phone integration, and navigation functionality to an automobile.

3.2.1 Graphical Display

The project should display its system state via a Graphical User Interface.

3.2.2 4-channel Audio Support

The project should be able to output digital audio in four independent channels, all configurable by the operating system.

3.2.3 Audio Playback Support

The project should be able to decode for playback the most common formats of multimedia documents, especially MP3, m4a (Unprotected MPEG-4 AAC audio), and FLAC (Free Lossless Audio). The project should also include the capacity to add decode support for new formats in the future, as long as the underlying hardware can decode the information in real time.

3.2.4 Arbitrary Execution

The device should be capable of executing any properly built and installed programs desired by the user. The user will only need a software development kit (SDK) including a compiler for the specific device's architecture and the development resources such as libraries and headers.

3.2.5 Remote Administration

The project should be capable of remote administration on any desired, capable network interface. No physical manipulation of the target device should be required in order to change most internal settings and perform over-the-air software updates.

3.2.6 Personal Media Device Support

The project should interface with personal media devices available to the underlying operating system. The solution presented in this project will support USB-connected devices which provide a USB Mass Storage device and a readable filesystem to the host system.

3.2.7 Bluetooth Integration

The project should support wireless networking over the Bluetooth protocol. This protocol will allow users to link the device with mobile phones in order to provide call and messaging notification.

3.2.8 GPS Navigation

The project should support geolocation if a GPS device is installed.

3.3 Metrics

3.3.1 Monetary Cost

This project relies entirely upon free software. The hardware reference platform's bill of materials is the only monetary cost associated with the project. The intent of this project is to undercut the cost of existing solutions that are available to retail customers. The Monetary

Cost metric is measured in US Dollars including all taxes and shipping fees paid.

3.3.2 Boot Time

The time for a device to start up from a powered-off state will be measured and compared to other comparable systems such as mobile phones and personal computers. The goal is to achieve a quick boot time that is acceptable to automotive end-users and effort will be taken to minimize this value. The Boot Time metric is measured in seconds by the time from the exit of the bootloader until the availability of a serial terminal login.

3.3.3 Filesystem Size

While storage is inexpensive, this project doesn't intend to store a large amount of data. Furthermore, it intends to include as little as possible code to maximize performance and maintainability. Efforts will be made to minimize the filesystem size and occupy much less storage space than existing mobile operating systems. The Filesystem Size metric is measured by the size of the root filesystem in bytes.

3.4 Hardware Architecture

The hardware platform chosen for this project is based on the Texas Instruments (TI) Sitara AM335x platform. The AM335x series of System on a Chip (SoC) platform consists of an ARM Cortex-A8 processor running at up to 1 GHz as well as several other components [8]. The AM335x chip includes several built-in subsystems to meet the requirements of this project. First, the platform is fully supported by the chosen bootloader and operating system. Second, the platform contains the necessary interfaces and components to integrate and support any external peripheral devices required by the project. Finally, the SoC is community-backed with cheap single-board computer implementations such as the BeagleBone Black (BBB). The BBB was chosen as the basis of this project's hardware reference platform due to its reasonable cost and versatility.

3.4.1 Computing

The AM335x includes an ARM CPU with adequate processing power for this project. The CPU can run as fast as 1 GHz and has mature support within u-boot and Linux. Since Linux is not a Real Time Operating System (RTOS), there are use cases where Linux cannot be relied upon. For these cases, Linux can work cooperatively with two additional built-in processors, the TI PRUSS units which are RISC processors that run at 200MHz and are fully programmable.

3.4.2 Graphics

The AM335x platform contains its own Graphics Processing Unit (GPU). The main purposes of this GPU are for hardware-accelerated 3D graphics and video playback. These features are not necessary for this project but may be useful in the future. The project still requires the display of two-dimensional graphics, however. The AM335x contains a framebuffer unit which supports LCD as well as HDMI video output. This framebuffer is supported by the X11 window manager as well as other Linux-compatible graphics systems. For the display component, a 3.5" TFT display was chosen.

3.4.3 Audio

The AM335x platform contains 2 dedicated TI McASP (Multi-channel Audio Serial Port) audio processing units which are capable of supplying audio data from the system to connected audio codec chips. The McASP supports transmitting and receiving digital audio signals in I2S as well as PCM and S/PDIF formats to codec chips. The TI PCM5102 I2S DAC codec was chosen for this project, simply because it requires little configuration and has several break-out boards that are easy to obtain and inexpensive. Each codec board supports two channel output so it is necessary to integrate two codecs with the McASP. The I2S audio protocol packs two channels of audio into a single data line, switching between them using a frame-sync clock. Each McASP unit has four data lines (serializers) so the first two are connected to each codec's data input port. Each codec will also receive two

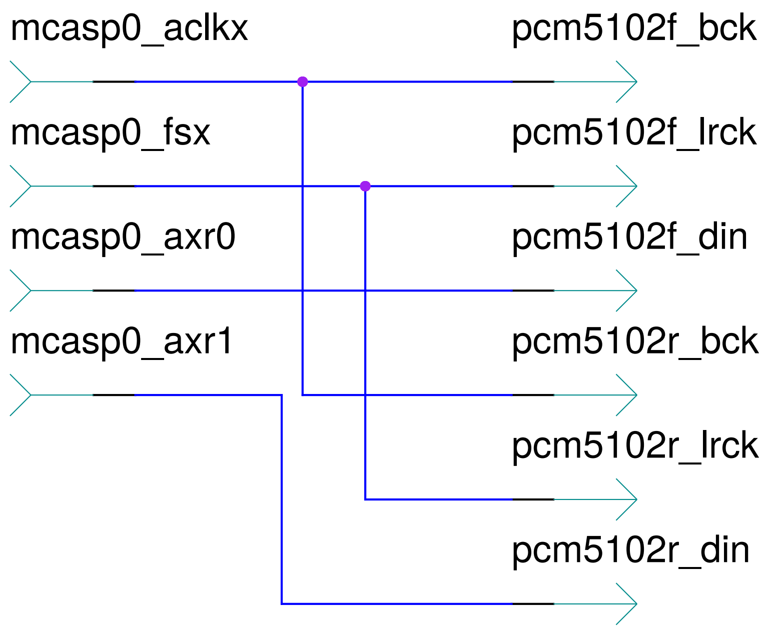

Figure 3.1: I2S audio configuration (power not shown)

clock signals from the McASP, the bit clock and the frame-sync clock. The bit clock is a clock signal which is used to time the audio data emitted by the serial port and has a timing always a multiple of the frame sync clock. These signals are shared across the entire I2S bus controlled by the McASP. In this configuration (Figure 3.1), the McASP is considered the master and both codecs are considered slaves. The I2S specification also calls for a high-speed clock, also called the master clock, which typically is divided to derive the other clocks, though the PCM5102 does not need this signal.

3.4.4 Storage

The AM335x platform allows several options for storage. It has built-in controllers for MMC, NAND, and NOR flash memory. Other storage can also be added over its USB interfaces. The BBB includes an embedded MMC (eMMC) storage unit which will be used to store the operating system including the bootloader, kernel, and root filesystem. The USB interface support will allow external storage devices to supply data to the system such as music files and geographical information.

3.4.5 Peripheral Buses

In addition to the previously mentioned buses, the AM335x platform contains controllers for several other, more general purpose communications buses. It has controllers for I2C, SPI, and UART serial interfaces. These controllers can be used to connect the host system to a range of peripherals such as GPS receivers, terrestrial radio receivers, as well as wireless network interfaces such as 802.11 and Bluetooth. As previously mentioned, USB is also available as a general purpose peripheral bus.

3.4.6 Other Pins

The AM335x platform also contains several general-purpose (GPIO) pins which can be used for less complex operations than typical peripheral bus protocols such as indicator circuits and as a host interrupt source. These extra pins are also used to supplement functionality of devices connected over another bus such as SPI or MMC. In addition, the AM335x platform also includes support for pulse-width modulation (PWM) input and output, which can be used to drive clock signals, some types of motors, and dimmable LED light sources as well as the capture of the input source intensity.

3.4.7 Bluetooth Networking

For Bluetooth networking, a commodity USB bluetooth adapter has been selected for this project based on the CSR8510 chipset.

3.4.8 802.11 Networking

For 802.11 wireless networking, a commodity USB 802.11n adapter bas been selected for this project based on the MT7601 chipset.

3.4.9 Touchscreen

For the touchscreen input, a screen with the XPT2046 controller was chosen. This controller communicates with the host over SPI and is packaged with the chosen TFT display.

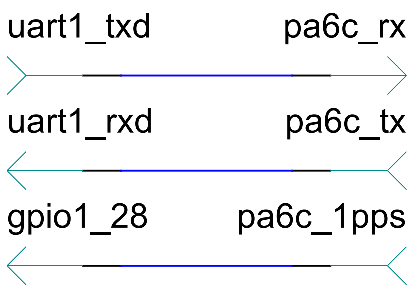

Figure 3.2: GPS serial connection (power not shown)

3.4.10 GPS

In order to provide location information, the hardware reference platform includes a GPS receiver module. This module is connected to the host over the UART serial interface, using 3 signalling pins and a power connection (Figure 3.2). GPS data is emitted by the transmit pin of the GPS module into the receive pin of the host and a clock signal is emitted by the PPS (Pulse Per Second) signal of the GPS module, typically into the carrier detect pin of the host. For this application, the PPS signal was actually wired to a separate GPIO pin on the host. The GPS module chosen for this project is a MT3339 chipset inside a GlobalTop FGPMMOPA6C self-contained module.

3.4.11 Bill of Materials

This project's Hardware Cost metric is determined by the choices made in this section. Excluding cables, the hardware reference platform for this project consists of the parts listed in Figure 3.3. All the hardware necessary for this project was purchased from retailers in single or double quantities for less than $112.

3.5 Software Architecture

3.5.1 Platform

The project was designed with simplicity, performance, and modularity in mind. Its target hardware is cheap single-board computers. These motivations influenced the choice of a development platform.

Part	Quantity	Unit Price	Total
BeagleBone Black	1	56.69	56.69
PA6C GPS Module	1	18.50	18.50
3.5” Touchscreen Display	1	12.81	12.81
PCM5102a DAC	2	7.59	15.18
MT7601 802.11n	1	4.98	4.98
CSR8510 Bluetooth	1	2.99	2.99
			111.15

Figure 3.3: Bill of Materials, all prices in US Dollars

3.5.2 Organization

The project is designed into several service components, each with a well-defined role and programming interface. This type of organization allows for independent development of components, a clear separation of concerns, and autonomous behavior of each component. If the tools are also given their own process space in a modern multi-process operating system such as Linux, there is also the potential for increased security and robustness against failures in one component. In fact, the application code of this project was designed as a set of interfaces for the D-Bus interprocess messaging system [9].

3.6 Implementation

This section describes the implementation of the project’s software.

3.6.1 Platform

This section covers all concrete choices made regarding third-party software libraries and systems in addition to any third-party project infrastructure included in the project.

3.6.1.1 Bootloader

The bootloader chosen for this project is Das U-Boot [10], an open source bootloader supporting several embedded architectures. This software exposes all the necessary facilities to locate, load and execute the real operating system kernel which will host the project’s software.

3.6.1.2 Operating System

The operating system chosen for this project is Linux, an open source operating system kernel that runs on many embedded computer systems as well as larger ones [11]. Much of the software written for this project will run on any UNIX-like operating system, however.

3.6.1.3 Frameworks

When developing applications, the choice of frameworks can be the most significant driving influence in the implementations of a project. While this project has chosen Linux as its operating system, it has been developed heavily with portable libraries that can be built (and run) on other operating systems and architectures. This project has chosen to only build upon open source libraries.

At the lowest level, all the application code of this project uses GLib [12], a portable utility library providing many data structures and support functionality to assist application development. GLib also includes GObject, a C-based object-oriented type system, and GIO, an asynchronous Input/Output (I/O) library with many more additional application support features. Importantly, GIO provides a client and server implementation of the D-Bus protocol using GIO's asynchronous I/O model.

3.6.1.4 Programming Language

Initially, this project was programmed in Vala, a language with a similar syntactical appearance to C# but targeting the GObject type system. This is achieved by the Vala compiler generating GObject-style C code and mapping documentation annotations as well as custom metadata to Vala constructs. Since Vala is just a code generator, its use does not require a virtual machine, runtime interpreter, or garbage collector. These properties are advantageous to embedded software development where processing speed, memory usage, and power consumption are less abundant than desktop or server environments. Vala allows for fairly rapid prototype development due to its significantly smaller amount of boilerplate code necessary to define and implement object-oriented constructs such as interfaces and classes (as well as subclassing). Through the development of the project, many difficulties were experienced with this language that required the author to fix bugs within the Vala language or its bindings to external libraries. Furthermore, the nature of the Vala language and libraries implemented with the GObject type system introduce a maintenance requirement for the Vala bindings to a specific library. Changes to APIs and incorrect transcription or scanning will lead to discrepancies between the original C API of a library and the generated or manually maintained Vala bindings. These deficiencies, combined with the declining activity of the Vala project have disqualified the language from being the primary language of this project.

Due to the project GObject Introspection [13], many more programming languages are available to the GObject programmer. GObject Introspection provides automatic bindings from GObject libraries to JavaScript, Python, Lua, Perl, Ruby, Guile, and other languages [14]. Similar to Vala, which is partially compatible with GObject Introspection, projects employing GObject Introspection are subject to mismatched bindings. Furthermore, attempts to map GObject semantics into a specific programming language's idioms are not guaranteed.

The compound effects of developing with non-native programming languages during the development process eventually led to their abandonment in favor of GObject-style C. The C11 revision of C was eventually chosen for this project since it is fully supported by the compilers GCC [15] and Clang [16]. This choice provides full control of all library APIs with the slightly inconvenient syntax of the GObject type system's C macros.

3.6.1.5 Build System

Build systems are an important part of any sufficiently large software project. They are responsible for configuring any target-specific options and producing target artifacts, the part of the project that will be consumed by the end-user. The most common build system for UNIX-style projects is make, part of the POSIX standard. Limitations in make have repeatedly influenced efforts in producing new build systems over the years. Among the replacements, GNU Autotools and CMake are the most popular for UNIX-style projects. Waf and Meson were also evaluated during the development of this project. After evaluation, Meson was eventually chosen as the build system for this project. Ninja is a newer tool that is very similar to Make in that it only supports scheduling and running tasks based on recipes and it claims to perform those tasks with a higher degree of parallelism.

GNU Autotools, also known as the GNU Build System, is a combination of GNU Autoconf [17], GNU Automake, [18], GNU Libtool [19], and GNU Make [20]. These tools are orchestrated in order to provide a fully portable solution for building software projects. While Autotools is a well-tested product with many users, it suffers from a few of its own deficiencies. First, to use Autotools in any non-trivial capacity requires the basic knowledge of several languages including m4, POSIX shell, as well as the GNU makefile syntax. Furthermore, since it eventually relies on make, Autotools has limitations in its ability to parallelize the compilation of object files from C source, which is a highly parallelizable process. The main steps of building an Autotools project are generating the configuration script (autoconf), configuring a specific build instance (configure), and performing the actual build (make).

CMake is a newer project which follows part of the Autotools strategy but unifies the entire build description into a single language, which is new and specific to CMake. Its principal similarity to Autotools is that it generates Makefiles during a configuration stage. CMake can also generate build files for the Ninja build tool as well as many IDE-specific project files.

Waf is a complete replacement attempt of a configuration and build tool into a single self-contained tool. It provides similar features to Autotools with claimed performance gains as well as simplicity of the build description files. Similar to other existing systems, Waf basically follows a configure-then-build pattern. Its build descriptor files are written in Python, eliminating the need to learn a new language that will not be useful for any other purpose. Its internal build tool also has a job scheduler which allows it to achieve higher parallelism rates than a project using a recursive set of Makefiles.

Finally, the newest addition evaluated for the project is Meson, which is an experience somewhere between that of CMake and Waf. It has a custom build descriptor language which is similar to Python and is implemented in Python. It does not contain its own build tool but it generates only Ninja build files and some IDE-specific project files. Meson has other strengths, though, that qualify it better for this project than the other tools. First, it has built-in functionality to integrate with parts of the GLib platform infrastructure, providing build targets for GResource bundles, GSettings schemas, GDBus code generation, as well as other tools. Finally, while cross-compilation is supported at some level all the previously mentioned solutions – with Autotools being the most extensive – Meson completely encapsulates the cross-compilation toolchain functionality in a single file that can be shared across any number of projects that are built with Meson.

3.6.1.6 Filesystem Build System

In addition to building the components authored during this project, it is also necessary to build operating system images for the storage media. This is a complicated task and requires gathering all the dependencies of the project and building them in order. There are several systems that intend to ease embedded development with regard to building root filesystems. The two systems evaluated for this project are OpenEmbedded [21] and Buildroot [22]. Buildroot was eventually chosen as the filesystem builder for this project.

Buildroot is composed mainly of Makefiles with some additional support tools. These Makefile rules factorize and abstract the commonly duplicated rules necessary for cross-compiling packages built with several common build systems such as the ones mentioned previously. It is capable of building an entire Linux distribution from nothing including the cross-development toolchain, the Linux kernel, all software packages needed for a target platform, and the actual filesystem images for various types of storage. Furthermore, for independent projects, it supports overlaying its existing supported packages with a user-specified set of external packages. This feature provides a well-defined path for developers to add independently maintained board support packages, experimental, or special-interest functionality to the Buildroot system.

3.6.2 Components

This section describes all the components implemented as the software portion of this project.

3.6.2.1 Linux Kernel Additions

In order to ease the process of connecting multiple identical PCM5102a audio codecs to a single TI McASP audio controller, a small Linux kernel driver was developed. It simply parses a list of references to codecs in a Device Tree file and arranges them under a single interface link within the Linux ASoC framework. The end result is a device that appears to be a single sound card supporting 4-channel surround sound when accessed by audio applications. Since the PCM5102a codec itself does not have a control interface, all control features such as volume control and balance are handled in the userspace.

A custom Device Tree file was developed to inform the Linux kernel of all devices and hardware properties associated with the harware reference platform. Since Linux already supports the BeagleBone Black robustly, it includes a set of Device Tree definitions for implementations based the TI AM335x. All the custom hardware added to the device, with the exception of USB devices (which are automatically discovered), were described in this file in addition to the information included in the definitions that are already part of Linux.

3.6.2.2 Media Data Service

The Media Data Service is an implementation of the media service interface. It is responsible for handling discovery of media sources, finding media content in those sources, and providing a searching and browsing D-Bus API for found media. The service also performs analysis on media files in order to supply textual and graphical metadata to consumers.

3.6.2.3 Media Player Service

The Media Player service is simply a media player and an implementation of the media player interface. Its only purpose is to manage the playback and queueing of media content cataloged by the Media Data Service. It exposes a D-Bus API to allow a mediating consumer of itself and the Media Data service. It is implemented as a subclass of the GIO GApplication class and encapsulates a GStreamer pipeline to play back audio and a custom data structure for playlists.

3.6.2.4 Geolocation Service

The Geolocation Service is an implementation of the geolocation interface. It is responsible for communicating with a GPS receiver device and exporting its geolocation data in a D-Bus API. It is implemented as a GIO GApplication subclass and emits signals whenever the status of the GPS receiver is updated.

3.6.2.5 Messaging Service

The Messaging Service is an implementation of the messaging service interface. It is responsible for communicating with compatible mobile communications devices over the Bluetooth protocol. Furthermore, this service is responsible for integrating with the underlying operating system's Bluetooth framework to expose the pairing workflow to the Graphical User Interface. It is implemented as a GIO GApplication subclass.

3.6.2.6 Graphical User Interface

Combining all the data available through the previously described D-Bus interfaces, this project's Graphical User Interface (GUI) presents a contextual view of the overall system state and react to out-of-band events. The GUI operates in its own process space and does not directly manage any of the underlying technology. The GUI is a full-screen application implemented with the GTK+ 3.0 user interface toolkit. The project contains several views are necessary for a usable product.

Top-Level Views The user interface contains two top-level views, the idle view and the home view.

The idle view (Figure 3.4) is displayed when no activities are running and no communications with the user have been detected within a set period of time. It will overlay any existing view and expose the previous view when the interface is re-activated.

The home view (Figure 3.5) is a basic menu of buttons which allows the user to begin their chosen activity.

Music Views The music views are the primary focus of this project and the most complex. They mainly synthesize information from the music service to allow navigating a collection

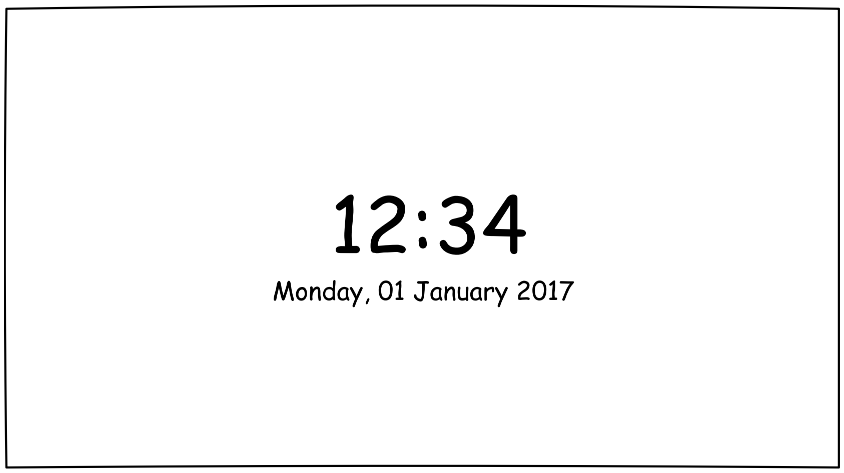

Figure 3.4: Idle View

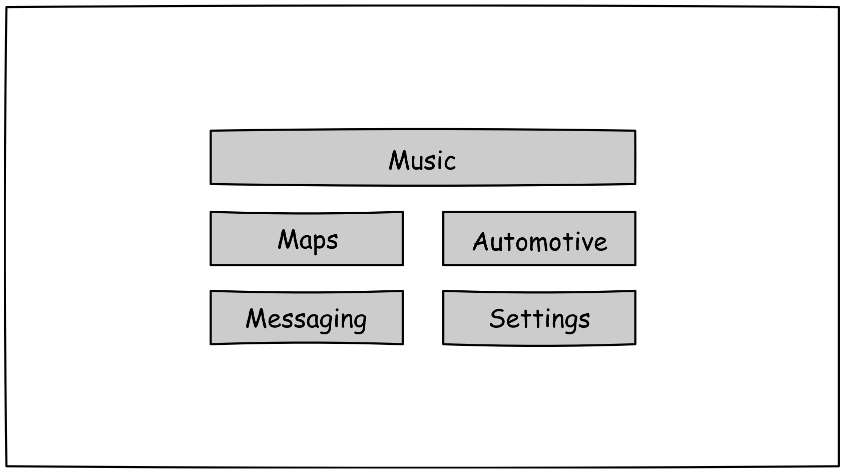

Figure 3.5: Home View

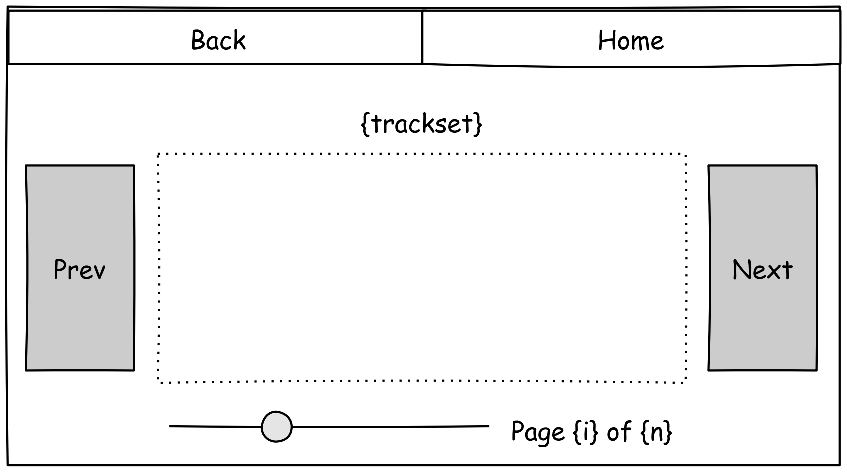

Figure 3.6: Abstract Track Set Browser

of media on connected devices and managing the playback of media. These views were designed and developed with a focus on minimal amount of interaction with the user in order to minimize distractions.

The first music view (Figure 3.6) is a abstract browser that is capable of navigating through a single-level list of textual information one page at a time. This view is realized and slightly refined as an Artists view, an Artist Detail view (Figure 3.7), an Albums view, Playlists view, an Album view, a Playlist view, and a Track List view. Each instance of one of these views are stacked on top of each other during navigation and the top view is discarded when the user navigates backward.

Maps Views When a Geolocation source is available along with mapping data, The GUI contains contain a basic map view, displaying the user's location at a modifiable magnification level.

Messaging Views The messaging aspect of the GUI simply display temporary notifications of telephone call as well as text messaging information. During a hands-free phone

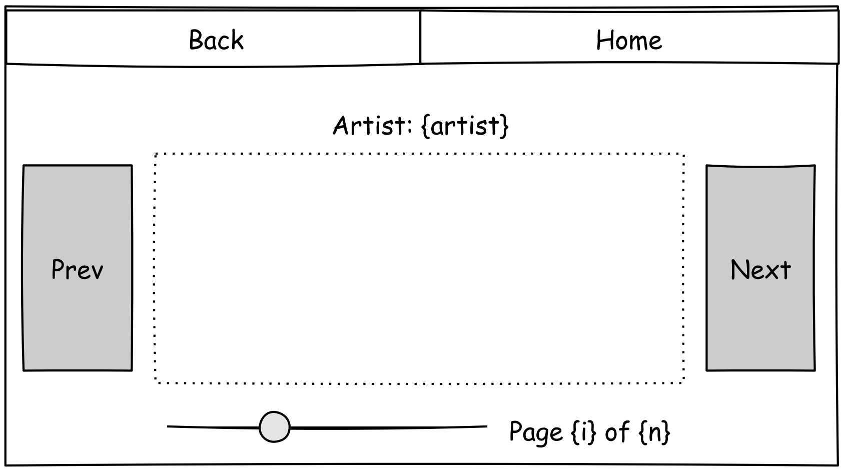

Figure 3.7: Artist Detail View

call, the user interface view is interrupted to provide call status on the screen. The messaging subsystem of the GUI is also tasked with implementing the visual side of the Bluetooth device pairing process.

Settings Views In order to provide some customizability during runtime, each configurable feature of this project can expose its configurable properties into the application's settings view. There are also global settings that are configured with this view.

3.6.3 Software Metrics

3.6.3.1 Filesystem Size

The Root Filesystem size of this project when stored inside a standard, uncompressed ext2 filesystem is 124179456 bytes long (approximately 103 MiB). Since Buildroot provides functionality to build compressed filesystems, the build system is also configured to produce Squashfs root filesystems from the same data as the ext2 filesystem. Using the Gzip compression option, a Squashfs filesystem is 42180608 bytes long (approximately 41MiB).

3.6.3.2 Boot Time

The typical Boot time of the uncompressed filesystem from the exit of the bootloader to the start of a serial terminal login is approximately 7 seconds. When booting with the compressed SquashFS filesystem, the typical value for the Boot Time metric is 8 seconds, a negligible performance regression compared to the filesystem size savings.

Chapter 4

Conclusion and Further Research

Given the wide-ranging solutions provided by some of the newer commercially backed systems that are shipping in newer automobiles as well as the dominance of the Android operating system for mobile devices, this project would need a significant amount of work to be performed to achieve competitive market share, though that was not a goal of this project. This section outlines several future areas which this project could be expanded in scope and depth. One area this project should not receive any development is any ”Cloud” integration.

4.1 Automotive Integration

A clear area of functionality that could benefit this project is the addition of features that integrate with automotive hardware such as On-Board Diagnostics (OBD), Tire Pressure Monitoring System (TPMS), and climate control.

4.2 Navigation

While this project can supply maps and geolocation, it does not supply any navigation directions from one location to another. Many existing solutions provide navigation instructions and traffic integration.

4.3 Mesh Networking

An ambitious, largely untapped, and potentially dangerous power of every car on the road containing wireless networking capabilities introduces many possibilities for the future of transportation. For example, automobiles could potentially broadcast important information to each other in order to inform drivers of upcoming hazards and other conditions such as average speed. This type of network can provide a non-commercial, peer-to-peer replacement of proprietary live traffic data offered by paid services.

4.4 Security

This project, as-is, does not suffer from many security threats since there is not much at stake, a limited surface to attack, and no valuable information stored. However, any networking features and integration with automotive components will most likely place a system such as this project under a much larger threat.

Bibliography

[1] Connected Car! Telematics! Infotainment! ADAS. http://www.harman.com/connected-car. Accessed Nov 2016.

[2] CONNECTED BY HARMAN. http://car.harman.com/connected-vehicles.html. Accessed Nov 2016.

[3] Raspberry Pi Car PC: iCarus project. http://i-carus.com/. Accessed Nov 2016.

[4] Shop. http://i-carus.com/shop/. Accessed Nov 2016.

[5] Wind River Helix Chassis. http://www.windriver.com/products/chassis/. Accessed Nov 2016.

[6] About GENIVI. https://www.genivi.org/about-genivi. Accessed Nov 2016.

[7] freesmartphone.org. http://www.freesmartphone.org/. Accessed Nov 2016.

[8] AM335x. http://www.ti.com/lsds/ti/processors/sitara/arm_cortex-a8/am335x/overview.page. Accessed Nov 2016.

[9] dbus. https://www.freedesktop.org/wiki/Software/dbus/. Accessed Nov 2016.

[10] Das U-Boot. http://git.denx.de/?p=u-boot.git;a=blob_plain;f=README;hb=HEAD. Accessed Nov 2016.

[11] Linux. https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/README. Accessed Nov 2016.

[12] GLib Overview. https://developer.gnome.org/glib/stable/glib.html. Accessed Nov 2016.

[13] GObject Introspection. https://wiki.gnome.org/Projects/GObjectIntrospection. Accessed Nov 2016.

[14] Bindings based on GObject Introspection. https://wiki.gnome.org/action/show/Projects/GObjectIntrospection/Users. Accessed Nov 2016.

[15] C11Status. https://gcc.gnu.org/wiki/C11Status. Accessed Nov 2016.

[16] Language Compatibility. http://clang.llvm.org/compatibility.html. Accessed Nov 2016.

[17] Autoconf. https://www.gnu.org/software/autoconf/manual/autoconf.html. Accessed Nov 2016.

[18] Automake. https://www.gnu.org/software/automake/manual/automake.html. Accessed Nov 2016.

[19] Libtool. https://www.gnu.org/software/libtool/manual/libtool.html. Accessed Nov 2016.

[20] GNU Make. https://www.gnu.org/software/make/manual/make.html. Accessed Nov 2016.

[21] Openembedded.org. http://openembedded.org/wiki/Main_Page. Accessed Nov 2016.

[22] Buildroot - Making Embedded Linux Easy. https://buildroot.org/. Accessed Nov 2016.