CHALMERS

Study on Open Source In-Vehicle Infotainment (IVI) Software Platforms

Master of Science Thesis in Embedded Electronic System Design

ANDERS KLAVMARK
TERJE VIKINGSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science & Engineering
Gothenburg, Sweden June 2015

Study on Open Source In-Vehicle Infotainment (IVI) Software Platforms

Anders Klavmark,
Terje Vikingsson,

Examiner: Sven Knutsson
Supervisor: Per Larsson-Edefors

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2015

Acknowledgements

This report is the result of a master thesis of 30 credits and was carried out by Anders Klavmark and Terje Vikingsson from Chalmers University of Technology. The thesis was conducted during 20 weeks in the spring of 2015 at ÅF in Lindholmen, Gothenburg.

We would like to give a special thanks to our supervisors at ÅF, Anders Rylander, Mats Christiansson and Johan Almquist, for all the help and guidance throughout the project.

We would also like to express our gratitude to Ola Wennberg, Manager infotainment, SW & HMI, at ÅF Automotive, who believed in us and gave us the opportunity to perform this master thesis. Thanks also to Per Larsson-Edefors, our supervisor at Chalmers, for the continuous support and feedback during the project.

Last but not least, thank you Jeremiah Foster and Johan Thelin at Pelagicore, and Gunnar Andersson at Volvo Car Corporation for providing us with so much valuable information.

Abstract

On ÅF's behalf, this report addresses three open source initiatives for use in automotive infotainment, namely GENIVI, Automotive Grade Linux (AGL) and Automotive Grade Android (AGA). This was carried out in form of a study that presents and analyses the three named open source initiatives with respect to their respective objectives, backers, technology, openness, licensing, maturity, etc. followed by a practical implementation of one selected candidate of these initiatives. As the root cause for all of this was ÅF's willingness of facilitating the identification of a suitable platform for an In-Vehicle Infotainment (IVI) project and to demonstrate open source as a successful concept for IVI development.

For the study, on-line and literature reviews have been essential in our information gathering efforts and have been used extensively throughout the project. With these, a general understanding of each initiative was obtained and many of our initial questions got answered. To obtain further knowledge, interviews were conducted with people having insight and/or experience in IVI and open source development.

In the final stage of the study, practical work commenced on the development of an IVI prototype aimed at demonstrating functionality of one investigated initiative. Our main focus was the software, so the open source IVI platform to be used for the prototype was selected prior to the hardware. For this design implementation, we chose GENIVI since it had the largest numbers of backers, an active community, and the most mature and comprehensive solution. Hardware was purchased subsequently and the implementation of the prototype took off.

The results of the study have led us to believe that GENIVI possess the greatest chances to be successful and achieve broad adoption from the industry since it has a relatively high level of maturity, activity and large number of backers. Furthermore, the prototype complies with the idea of working as a demonstrator for the GENIVI software and design.

Acronyms

ABI Application Binary Interface

AGA Automotive Grade Android

AGL Automotive Grade Linux

AMB Automotive Message Broker

API Application Program Interface

ASF Apache Software Foundation

BSP Board Support Package

CAN Controller Area Network

ECU Electronic Control Units

FMS Fleet Management System

FOSS Free and Open-Source Software

HMI Human Machine Interface

HVAC Heating Ventilation and Air Conditioning

IP Intellectual Property

IPR Intellectual Property Rights

ISV Independent Software Vendors

IVI In-Vehicle Infotainment

MOST Media Oriented System Transport

MPL 2.0 Mozilla Public License 2.0

NFC Near Field Communication

OEM Original Equipment Manufacturer

OGI Open Governance Index

OS Operating System

OSI Open Source Initiative

OSS Open Source Software

OSV	Operating System Vendors
PDK	Platform Development Kit
PMC	Project Management Committee
SDK	Software Development Kit
SPDX	Software Package Data Exchange
VIL	Vehicle Interface Layers

Dictionary

Artifact (Software) A tangible by-product of software development (e.g. code, use cases, diagrams, requirement and design documents, etc.)[1].

Copyleft (License) Refers to licenses that use copyright law to offer the right to distribute copies and modified versions of a work, and requiring the same rights to be preserved in modified versions of the work. Most copyleft licenses are open source licenses[2].

Head Unit Commonly used name for IVI system, including hardware and software[3].

Permissive (License) A non copyleft open source license that guarantees freedom to use, modify, and redistribute, but that also permits proprietary derivative works[4].

Runtime (System) A software designed to support the execution of computer programs written in some computer language[5].

GStreamer (Software) Pipeline-based multimedia framework that allows programmers to create media-handling components[6].

Wayland (Software) A protocol that specifies the communication between a display server and its clients[7].

SystemD (Software) A suite of system management assets designed as a central management and configuration platform for Linux. Used in system startup process[8].

Contents

1

Introduction

The past few years, a number of commercial and open source initiatives for commoditizing software and hardware for use in automotive infotainment, have emerged. Choosing a suitable platform for an In-Vehicle Infotainment (IVI) project is challenging given the diversity between these initiatives in objectives, backers, technology, openness, licensing, maturity, etc.

Since these initiatives are all relatively new, no comprehensive study of said initiatives exist and a major challenge when choosing platform is to determine which initiative best conforms to one's needs and requirements.

Moreover, In-Vehicle Infotainment (IVI) system development is increasingly focused on efforts to commoditize and integrate additional software systems and hardware to the infotainment domain. This includes software architectures such as the Automotive Open Software Architecture (AUTOSAR), which work, inter alia, to create a development base for industry collaboration by standardizing basic software functionality of automotive Electronic Control Units (ECU) [9]. It also includes additional hardware components, which relates to the platform's customizability towards different hardware and adaptability to new components.

In this initial study, we seek to analyze and evaluate three main candidates for IVI system development, namely GENIVI, Automotive Grade Linux (AGL) and Automotive Grade Android (AGA). A practical implementation of one candidate on a development board demonstrates basic functionality and enables an evaluation of the effort required to input an additional external data signal.

1.1 Background

The components of current-day IVI systems are generally based on different operating systems, with various proprietary software solutions on top. As such, for every update of the system, IVI development must start over. In combination with the customer demand to accommodate mobile devices, the industry is now increasingly considering open source software solutions. A system based on Android or Linux would effectively build on prior advances and enable incorporation of existing functionality [10]. This would enable developers and manufacturers to increase their production cycle by creating a platform with core services, middle-ware and application layer that is reusable [11].

Pioneering this field are the GENIVI Alliance, the Automotive Grade Linux working group and the Automotive Grade Android initiative.

1.1.1 ÅF

This master thesis was performed on behalf of ÅF, an engineering and consulting company with missions in the areas; energy, industry and infrastructure. It was founded in Sweden in 1895 and today has about 7000 employees. The company is based in Europe, but has its business and clients worldwide.

1.2 Purpose

The purpose of this report is to provide a detailed description and comparison of three main open source platform initiatives to IVI systems and to demonstrate how an implementation setup of one of these platforms can be made.

1.3 Problem description and Scope

ÅF is interested in a study which presents and analyzes open source IVI initiatives based on their respective objectives, backers, technology, openness, licensing, maturity, etc. Determining which best conforms to one's needs and requirements is a major challenge as no comprehensive study of these initiatives exist.

The study is focused on three principal open source initiatives; GENIVI [12], Automotive Grade Linux (AGL) [13] and Automotive Grade Android (AGA) [14], with the aim of easing identification of a suitable platform for IVI developers. Actors co-operating in the same industrial sector, whether it be car manufacturers, original equipment manufacturers (OEM's) or software developers, may, while sharing the same goal, have different requirements on an IVI platform. As such, the study will also aim to conclude where priorities of various actors are considered and which of the initiatives that best meet the requirements from the actors respectively.

One platform will subsequently be selected for practical studies. It will be realized on a development board as a demonstration of a possible implementation setup of one of the open source initiatives and to enable an evaluation of its adaptability to additional hardware, external sensor data will be inputted. The software's potential to input and process data originating from existing vehicular systems (e.g. sensor data) is of interest for IVI solutions as these signals may provide valuable information about the state of the vehicle. Presenting data on the IVI prototype through basic platform functionality (e.g. GUI) is considered an optional goal.

1.4 Limitations

From the start of the project, we solely had one hard limitation. It was that we would select only one of the open source platform alternatives to use for our prototype, even if the study would show that more than one of the alternatives could be viable to use.

1.5 Report layout

An overview of the different chapters in the report is presented here. In order to facilitate the understanding of the results in the study and provide additional background information, a chapter with relevant theory is presented in Chapter 2. The method that the study and prototype development have followed is presented in Chapter 3, together with a brief description of the main tools that were utilized. In Chapter 4, the three investigated open source IVI initiatives are presented and compared. The design and implementation of the prototype is presented in Chapter 5. Final conclusions are provided in Chapter 6.

2

Background

This chapter provides background theory behind some of the concepts/areas addressed in the rest of this report.

2.1 In-Vehicle Infotainment (IVI)

In-Vehicle Infotainment (IVI) is a term from the automotive industry which refers to vehicle systems that combine the delivery of information with the delivery of entertainment to passengers and drivers. To provide and manage these kind of services, IVI systems are equipped with a Human Machine Interface (HMI) consisting of audio/video interfaces, keypads, touchscreens, etc. Additional tools and features that are typical for these systems include audio/video playback and two-way communication tools (CD/radio, navigation, voice commands, rear seat entertainment, etc.). Furthermore, mobile device connectivity has in recent years become a major component in IVI systems in an effort to meet the increasing needs of accessing internet and common smartphone content, such as music streaming, traffic information and weather forecasts. Safe use of IVI content is another area that has received much attention lately, which has lead to that many IVI systems have security features that are intended to prevent driver distraction [15].

An IVI system consists of many interconnected hardware and software components, and the general architecture or framework of an arbitrary system can be described with a set of layers, ranging from the hardware layer in the bottom, to the HMI at the top. Between every layer are well defined interfaces and each layer includes a series of key technology blocks in software and/or hardware to provide the desired functionality of the system, see Figure 2.1. Below is a brief description of each layer and their principal building blocks [16].

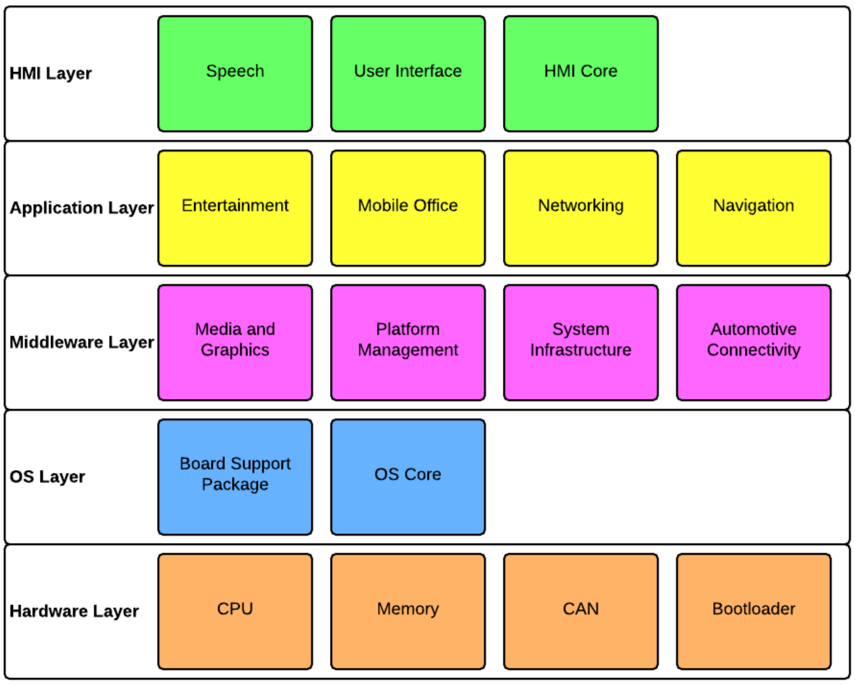


Figure 2.1: System architecture overview for an IVI platform, reproduced from [16]

- `Human Machine Interface (HMI) Layer:` The HMI is the interface to the user of the IVI system and controls the display of the IVI system's head unit. It is responsible for processing and responding to user inputs like touch screen input, speech recognition input and knob/button-based input [16].

- `Application Layer:` The application layer contains a mix of applications, all designed to provide a specific function to the benefit of the user. Applications are dependent on other software referred to as system software to be able to execute. System software differentiates from application software in the sense that the it serves the latter (which in turn serves the user) [17].

- `Middleware Layer:` The middleware layer consists of components and interfaces in software that supplies services to the application that are not available from the operating system layer, so that the functional areas of the application layer can be realized. Consequently, the middleware layer simplifies the communication and input/output of data between the application layer and the operating system layer, as a result application developers can focus on the particular purpose and functionality of their application [18].

- `Operating System (OS) Layer:` The operating system layer generally constitute the operating system along with a Board Support Package (BSP) and drivers. The operating system itself typically manages hardware and software resources and provides applications with common services [19]. The BSP provides essential support code that facilitates the porting of an OS to a new hardware environment [20]. The drivers operates and manages attached hardware devices by providing software interfaces to hardware devices. Thus, the operating system can access hardware functions without having to know any detailed information about the hardware currently in use [21].

- `Hardware Layer:` The hardware layer is composed of a processor with additional essential hardware and firmware to boot the OS. Additionally, this layer is often equipped with a set of automotive I/O devices like CAN/MOST interfaces [16].

In recent years, the amount of software found in modern vehicles has increased dramatically and a typical car may have as many as 100 million lines of code in its systems [22]. As IVI systems have gone from luxury to commodity, the importance of having a competitive advantage over one's opponents on this frontier is vital. In the traditional approach to IVI system development, the automotive manufacturer typically has very few software engineers involved in actual development. Instead, tier one suppliers develop all software for a set of requirements provided by the manufacturer [23]. This approach may offer the delivery of a very mature product (e.g. Windows embedded), but holds the manufacturer in complete dependency of the tools and features supplied by that tier one.

A solution based on open source code on the other hand, enables OEM's and tier one developers to directly write features into that solution. Furthermore, open software may have thousands of providers and contributors, mitigating the risk associated with relying on one supplier for a system [24].

Many customers have also grown accustomed to the fast technological turnover found in many of today's consumer products (e.g. mobile phones) and expect the same from the hardware and software found in their cars. As the automotive development/delivery cycle for vehicles is close to three times that of a typical consumer electronic device, this is a major challenge for manufacturers. Another thing an open source solution has going for it is that the features and applications associated with IVI systems are rarely unique to the automotive domain. Rather, most are heavily influenced by the consumer sector, a connection which an open source software development model could exploit to enable transfer of innovation between the two industries [3].

2.2 Open source software

According to the non-profit corporation Open Source Initiative (OSI), which educates about and advocates for open source, the term "open source software" refers to software that can be freely used, changed and shared (modified or unmodified) by anyone. Even so, open source does not only concern means of access to the source code, but also distribution of the software [4]. See Appendix A for a complete list of criteria needed to be fulfilled in order for a software to be labelled open source.

Contrasting open source software is closed source or proprietary software. This is software which source code is not publicly available, rather the rights to use it is licensed for a fee by the owner/developer. The application of such software is generally surrounded by great restrictions and the source code kept secret [25].

Furthermore, a common misconception about open source software is the notion that it is the same as free[1] software. Even though the terms are, by many, used interchangeably, there is a fundamental difference between the two concepts; free software is *always* open source, but open source software does not entail that it is also free software. The term Free and Open-Source Software (FOSS) is sometimes used to emphasize that a piece of software is both [26].

In a typical open source project, the content creator is also the maintainer and can approve patches made by contributors to be incorporated into the original build. However, before they are, this process of modifying the original work is known as branching. As these patches are approved and applied towards the source (content creator), it is also known as ”upstreaming”. Sometimes changes are made in a branch that renders it unfit for merging with the original build, this kind of permanent split is known as a fork.

The employment, contribution and distribution of open source software is today widespread and in many business areas its use is considered common practice. Both great successes such as Android, and less successful projects such as Symbian have utilized open source software. Despite this, the aspect of openness and how to measure it is not always clear and it can be difficult to see how open or closed an open source project really is.

The Open Governance Index (OGI) introduces and quantifies governance as a measure of openness. It further expands on the open source licenses specifications of rights to use, copy and modify by determining the rights to gain visibility, to influence and to create derivatives of a project. As such, to properly differentiate between an open and a closed project, one would have to look at its employment of governance; its governance model [27].

2.2.1 Governance

Open source code is controlled by licenses, there are many, but some of the more popular open source licenses include;

- Apache License 2.0
- BSD 3-Clause ”New” or ”Revised” license
- BSD 2-Clause ”Simplified” or ”FreeBSD” license
- GNU General Public License (GPL)
- GNU Library or ”Lesser” General Public License (LGPL)
- MIT license
- Mozilla Public License 2.0
- Common Development and Distribution License
- Eclipse Public License

[1]Free as in freedom (libre), not its monetary equivalent

However, these only apply to the source code, in contrast to governance which determines who has influence and control over the actual project or platform.

In research conducted by the market analysis and strategy firm, VisionMobile, the Best Practices for governance are outlined across four key areas [27].

Access

The primary concern of any open source project is the access to source code, it should be easily read, downloaded, modified and run.

Any developer should have access to the code without restrictions and there should be no bias/discrimination towards specific developers. Furthermore, all projects should use an OSI approved license.

Development tools, mailing lists and forums should also be accessible, with minimum effort, for developers.

Development

Copyright concerns and licenses that protects Intellectual Property (IP) should not impede the code contribution process and, provided that an appropriate open source license has been adopted in the first place, a broad copyright license should suffice. For the long term success of any open source project, transparency of decision making and equitable treatment of all developers, so that they can become committers, are identified as critical. Recurring rejections or blatant ignorance of contributions will most certainly result in loss of developer support.

Developers should also be granted access to information regarding the direction the project is headed and the ability to influence this themselves. As such, any developer who has proved having sufficient knowledge about the code should be able/allowed to commit code to the project[2]. For all commits there should also be metrics, regarding where they came from, available.

Finally, an ”upstream first” policy for contributions should be employed so that any changes benefit upstream and downstream projects adequately.

Derivatives

An increasing trend for open source projects is the adoption of compliance frameworks. This is a measure employed to reduce fragmentation between versions and guarantee that applications are transferable across platforms and Operating System (OS) distributions.

In order to avoid the compliance process being deliberately tampered with by a developer or organization it should be kept as independent and transparent as possible.

[2]A concept known as ”Meritocracy”

Community Structure

The community structure of an open source project can have a big impact on its openness. A non-profit model often provide independence, in the sense that control of the product is not in the hands of any one organization.

A formal association with the Linux Foundation also strengthens the open source credibility of a project.

How authority is exercised within the community is another aspect worth considering, as it many times is rather centralized and does not create an environment where it is easy for others to permeate the decision making process [27].

3

Method

The project started with a planning phase followed by on-line and literature studies. This provided sufficient background information within the given area to lay a foundation for the subsequent study and initial prototype design-direction. The project in itself has relied on extensive, continuous on-line studies of various In-Vehicle Infotainment (IVI) systems, coupled with interviews of people in the industry.

Evaluations regarding the results of the study and design of the prototype were held continuously throughout the project to ensure both internal and external requirements were met. A detailed description of the study method and prototype development is presented below.

3.1 Study method

The study can be divided into two principal analysis stages. First, a general analysis which considers the different initiatives with respect to their objectives, backers, technology, openness, licensing, maturity, etc. In the second stage the initiative's similarities and differences are analysed, as well as some of their associated advantages/disadvantages. Both analyses aim to facilitate the identification of a suitable platform for IVI development.

On-line and literature reviews have played an integral role in our information gathering efforts and have been used extensively throughout the project. Results from these studies have contributed to many of our initial questions being answered and provided us with a general understanding of each initiative. To further expand on this, we conducted interviews with people having insight and/or experience in IVI and open source development. These sessions were extremely valuable in the sense that they provided us with information that would otherwise have been difficult to procure.

3.2 Prototype method

In the final stage of the study, practical work commenced on the development of an IVI prototype aimed at demonstrating functionality of one investigated initiative. In order to minimize the risk of falling behind schedule, different software and hardware platforms were considered early. As such, partial results from the study were of help in deciding what hardware would be suitable for which software.

Since our main focus was the software, the open source IVI platform was selected prior to that of the hardware. For this, several parameters were considered, such as which initiative had the largest numbers of backers, if there was an active community, the level of maturity and which had the most comprehensive solution. See Section 5.1 for additional information.

Regarding the selection of hardware, a few requirements had to be fulfilled, namely the availability of a Board Support Package (BSP), community support channels and having compatibility with the selected open source platform. The former was a high priority as it is needed in order to get software up and running on hardware.

After having selected an appropriate hardware platform for implementation of the chosen open source platform, the following steps were taken in order to integrate them.

To begin with, it is important to have a suitable build environment running on the host system in order to effectively build the software for the target. This includes having the right operating system version, support packages and software build tools. This was set up in collaboration with our supervisors at ÅF.

For the actual build, a number of different building blocks in form of software layers were imported via Git[1] and a number of build parameters set, specifying which layers should be included for which target. A successful build resulted in an image of the software ready to be run on the specified target.

In order to facilitate development, we decided to first try to build the software for an emulated machine using QEMU[2]. This allowed us to get an idea of how the build environment and software worked before targeting the hardware platform with its associated BSP.

3.3 Tools and resources

The project, including the study and subsequent prototype development was aided by both physical tools and software resources. These are presented below.

3.3.1 Software

In the following list, the main software utilities used during the project are briefly presented.

[1]A distributed revision control system for software development [28]
[2]QEMU is an open source machine emulator and virtualizer [29]

- `GENIVI Demo Platform (GDP)`
  Used for the prototype as a technology demonstrator of the GENIVI software and design.

- `Yocto Project`
  The Yocto Project was used by GENIVI and consequently also by us to create a custom Linux-based system of the GENIVI Demo Platform.

- `Freescale Board Support Package`
  Used for customization of the GENIVI Demo Platform to the Freescale i.MX53 Quick Start Board.

3.3.2 Hardware

The hardware utilities utilized in the project are stated in the list below.

- `2 Windows PC:s`
  Mainly used for literature studies and documentation.

- `Windows PC running Ubuntu 14.04 LTS in Oracle VM VirtualBox`
  Used for building the GENIVI Demo Platform together with the Yocto Project.

- `Freescale i.MX53 Quick Start Board`
  The development board used for the prototype.

3.3.3 Resources

Throughout the project, we have had access to ÅF's office and their development facilities at Lindholmen Science Park in Gothenburg, as well as continuous supervision from employees at ÅF. Furthermore, economical resources for purchasing of needed hardware has also been covered and provided by ÅF.

4

IVI initiatives

In this chapter, we carry out an investigative study on three open source initiatives; GENIVI, Automotive Grade Linux (AGL) and Automotive Grade Android (AGA).

4.1 GENIVI

"GENIVI Alliance is a non-profit consortium of over 180 automotive industry companies promoting the collaboration and deployment of open source software in the automotive electronics business, specifically infotainment." [30]

The GENIVI Alliance was formed in the beginning of 2009 [23] in an effort to counter the increasingly complex and expensive process of developing, testing, deploying and supporting In-Vehicle Infotainment (IVI) products and services [31]. Its founding members consisted of eight companies from the automotive sector and included manufacturers, first tier suppliers, silicon vendors and operating system vendors. Since then a multitude of Independent Software Vendors (ISV), middleware vendors and software services companies have joined the ranks, making the total number number of members reach more than 180 [23].

There is a paradigm shift in the development of IVI systems and up until recently, automotive manufacturers relied exclusively on suppliers to build and provide each system independently for a given list of requirements, resulting in long development cycles and costly licensing fees. GENIVI is a driving force in the shift to a non-proprietary platform, where community-developed open source software lays the foundation [32]. Figure 4.1 offers an illustration of some of the different components phased out of the proprietary domain with GENIVI or an equivalent open source solution.

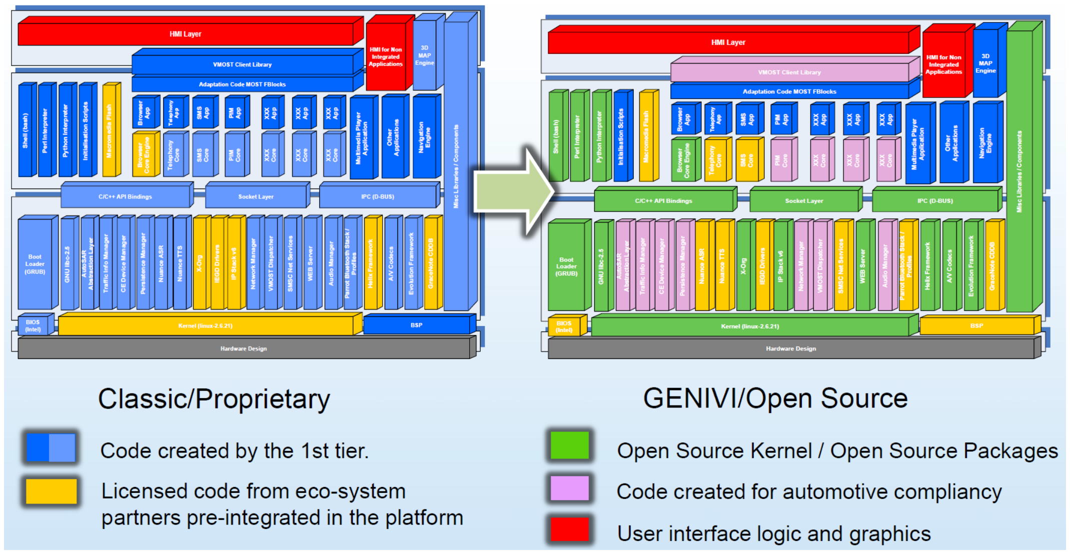

HMI Layer
HMI for Non Integrated Applications
3D MAP Engine
VMOST Client Library
Adaptation Code MOST FBlocks
C/C++ API Bindings
Socket Layer
IPC (D-BUS)
Boot Loader (GRUB)
BIOS (Intel)
Kernel (linux-2.6.21)
BSP
Hardware Design
Misc Libraries / Components
Classic/Proprietary
GENIVI/Open Source
Code created by the 1st tier.
Licensed code from eco-system partners pre-integrated in the platform
Open Source Kernel / Open Source Packages
Code created for automotive compliancy
User interface logic and graphics

Figure 4.1: Proprietary vs open source [33]

One of GENIVI's more outstanding goal is to provide a platform consisting of only about 5 % self developed code created from scratch, the rest would be adopted or adapted from existing Open Source Software (OSS) projects [33].

4.1.1 Objectives

While the GENIVI alliance is engaged in software development, they are not trying to solve a technical problem as much as a business one [23]. Driven by today's high repeated effort and cost associated with the development and maintenance of infotainment in vehicles [3], GENIVI is working for an industry wide adoption of OSS and aims at delivering a specification of a reusable, open source platform consisting of Linux-based core services, middleware and open application layer interfaces [31].

With the objective to offer the industry an environment for faster innovation and lower software development costs, the alliance also organizes technical work groups and composes recommendations designed to minimize differences between implementations [30].

More than simply focusing on pushing Linux to the automotive domain, GENIVI's main goal is to define a standardized common software platform to be used for developing IVI systems. By identifying and implementing non-differentiating functionalities [1] that all IVI systems require, the GENIVI platform is a packaging of operating system and middleware components, not a complete IVI environment. For manufacturers, this solution enables resources to be focused on development of higher level components, such as the application layer and Human Machine Interface (HMI) instead of the base system [23].

[1]Basic functionality not subject to variation between implementations

Their strategic initiatives (as specified by GENIVI) are as follows [12]:

1. Accelerate the development and implementation of the fully connected vehicle for infotainment applications
2. Deliver a platform consisting of standardized middleware, application layer interfaces and frameworks
3. Extend Open Source community innovations to support the automotive domain
4. Engage developers to deliver compliant implementations
5. Sponsor technical, marketing, and compliance programs

In order to reach the above, the alliance produces three principal outputs; a compliance specification, code projects and baseline software releases. The objective of the former is to ensure that the integration process of third-party software components is alleviated by GENIVI compliant products (as opposed to providing full Application Program Interface (API) or Application Binary Interface (ABI) compatibility across implementations) [23].

The specification, which provides a clear definition of middleware components needed to be included to achieve GENIVI compliance, is a document derived from the collaborate work of Expert Groups (EG) and subsequently reviewed by a System Architecture Team (SAT). For every new iteration of this compliance specification, there is also a software baseline release. This is essentially a packaging of components, as required by the specification, on top of a Linux distribution, designed to reflect a GENIVI compliant platform. The code projects hosted by GENIVI make out some of the content of these baselines and is an effort by the alliance to implement IVI functionality that does not already exist in upstream projects [34].

4.1.2 Backers and Organizational Structure

As mentioned above, the GENIVI alliance is a consortium of over 180 companies constituting Original Equipment Manufacturers (OEM's), Operating System Vendors (OSV)'s and various suppliers. Its charter members are the BMW Group, Jaguar/Land Rover, Magneti Marelli, PSA Peugeot Citroën and XSe Automotive[2]. Additional major backers include Intel, ARM, Hyundai and Volvo Cars, to name a few. A complete list of members can be found in Appendix B. While any organization interested in the success of IVI systems and related products can join the alliance, it is governed by a board of directors comprised of top-tier members and a selected number of lower-tier members. Supporting members and teams that, in collaboration with the board, oversee strategical, technical and commercial components are presented in Figure 4.2 [12].

[2]Now part of Mentor Graphics

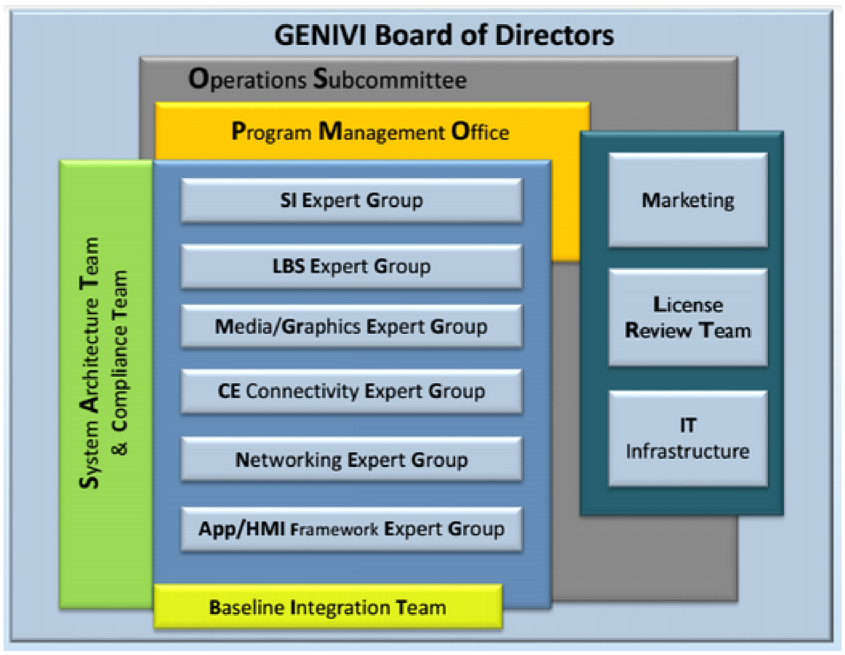

GENIVI Board of Directors
Operations Subcommittee
Program Management Office
System Architecture Team & Compliance Team
SI Expert Group
LBS Expert Group
Media/Graphics Expert Group
CE Connectivity Expert Group
Networking Expert Group
App/HMI Framework Expert Group
Baseline Integration Team
Marketing
License Review Team
IT Infrastructure

Figure 4.2: Functional Organization [12]

The operations subcommittee (OPS) has a supportive function to the Board of Directors by assisting on operational issues and is in close contact with the rest of the groups and teams. Its primary role is to streamline the alliance in terms of resources used to meet stakeholder expectations. Requirement collection, specification development, adherence to Intellectual Property Rights (IPR) policies and processes, testing and release of reference implementations, and adoption and compliance activities are activities undertaken by the Program Management Office (PMO) in its work to develop and monitor the technical working plan of the alliance [12]. There are six principal expert groups (EG), whose main area of responsibility is adapting available open source code from upstream projects to meet requirements imposed by Original Equipment Manufacturer (OEM)'s and tier one suppliers. Occasionally, GENIVI also acts as a sponsor and launches new projects to develop software (code) for which it currently does not exist any viable base candidates [30]. The System Architecture Team (SAT) works closely with the different expert groups and defines the overall GENIVI platform and its boundaries. Part of SAT is also the Compliance Team, who is responsible for the compliance statement. The processes, policies, and tools for the development of GENIVI software is overseen by the Baseline Integration Team (BIT). Finally, the License Review Team (LRT) ensures the technical work of the alliance uses a legally viable approach, by managing all licenses and legal activities [12].

4.1.3 Technology

From the Linux kernel up to the middleware, GENIVI specifies a common software infrastructure by implementing essential and non-differentiating functionality of an overall IVI system, which allows developers to focus on the customization of upper layers, such as the HMI.

In essence, it provides an entire spectrum of manufacturers and their suppliers with an underlying framework upon which they can then add their proprietary, differentiating software products and services, as illustrated in Figure 4.3 [30].

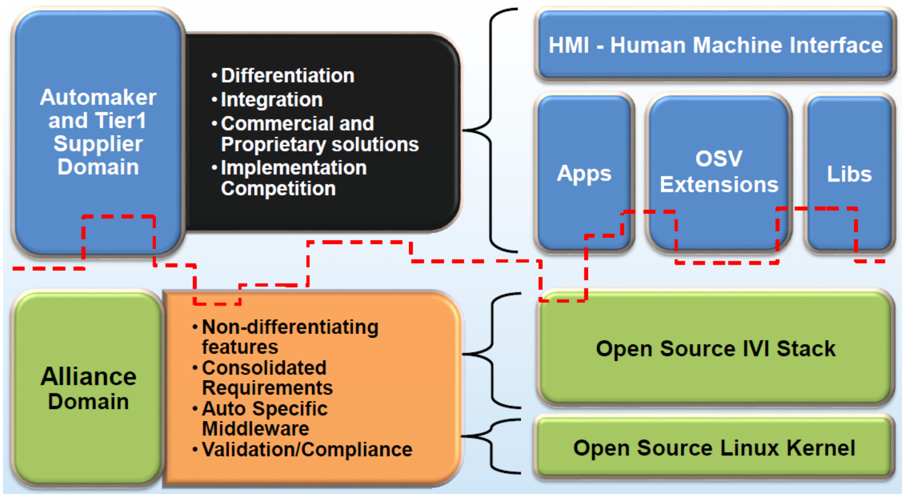

Automaker and Tier1 Supplier Domain
• Differentiation
• Integration
• Commercial and Proprietary solutions
• Implementation Competition
HMI - Human Machine Interface
Apps
OSV Extensions
Libs
Alliance Domain
• Non-differentiating features
• Consolidated Requirements
• Auto Specific Middleware
• Validation/Compliance
Open Source IVI Stack
Open Source Linux Kernel

Figure 4.3: GENIVI Domain [12]

Leveraging the potential from its open source origin (Linux), a great deal of GENIVI's work takes place in existing open source software projects located upstream[3].

Additional projects are initiated by GENIVI for which no suitable alternative already exists in the open source community. The components of these projects then make up so called baselines, which are released with each new iteration of the compliance specification [32].

4.1.3.1 Compliance Specification

GENIVI's main deliverable is its compliance specification which aligns IVI software requirements from multiple automakers, OEM's, suppliers and other members [32]. It is a detailed documentation which defines the core services, middleware and open application layer interfaces required by an IVI software stack to achieve compliance with GENIVI. The underlying idea is that if a level of standardization can be introduced to the software stack, both the development costs and cycle times can be reduced. This is primarily achieved by identification and specification of non-differentiating components and features [3].

[3]Direction toward the original authors or maintainers of software

Although limited to members only, OEM's, integrators and software suppliers may register their products to be certified against all functionality described by the specification in so called compliance programs. This is either a platform compliance, which relates to the middleware and kernel or a component compliance, which relates to additional functionality located in the HMI and application layers. Updates to the compliance specification, which is created and maintained by the expert groups, are released bi-annually [12].

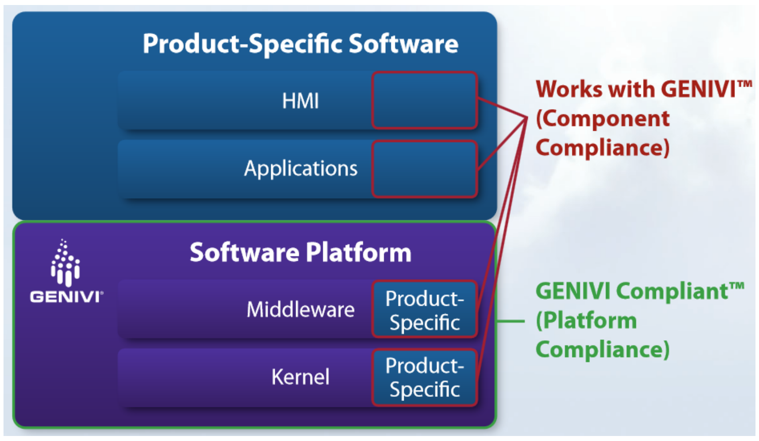

Product-Specific Software
HMI
Applications
Works with GENIVI™ (Component Compliance)
Software Platform
GENIVI
Middleware
Product-Specific
Kernel
Product-Specific
GENIVI Compliant™ (Platform Compliance)

Figure 4.4: GENIVI Component/Platform compliance [12]

Figure 4.4 illustrates the two compliance programs currently run by GENIVI. These programs offer enough standardization to allow developers to deliver implementations that will run on all compliant distributions and many OEM's have now made GENIVI compliance a requirement in their requests for proposals for new IVI systems [33]. To extend the platform compliance to include the HMI and applications, the "Works with GENIVI" label enables developers to register their software applications for compliance.

With every new version of the compliance specification, GENIVI also release public software baselines which constitute GENIVI middleware, a Linux kernel and various open source projects. Such a collection of components is presented in Figure 4.5 for the GENIVI 5.0 "Gemini" release [35].

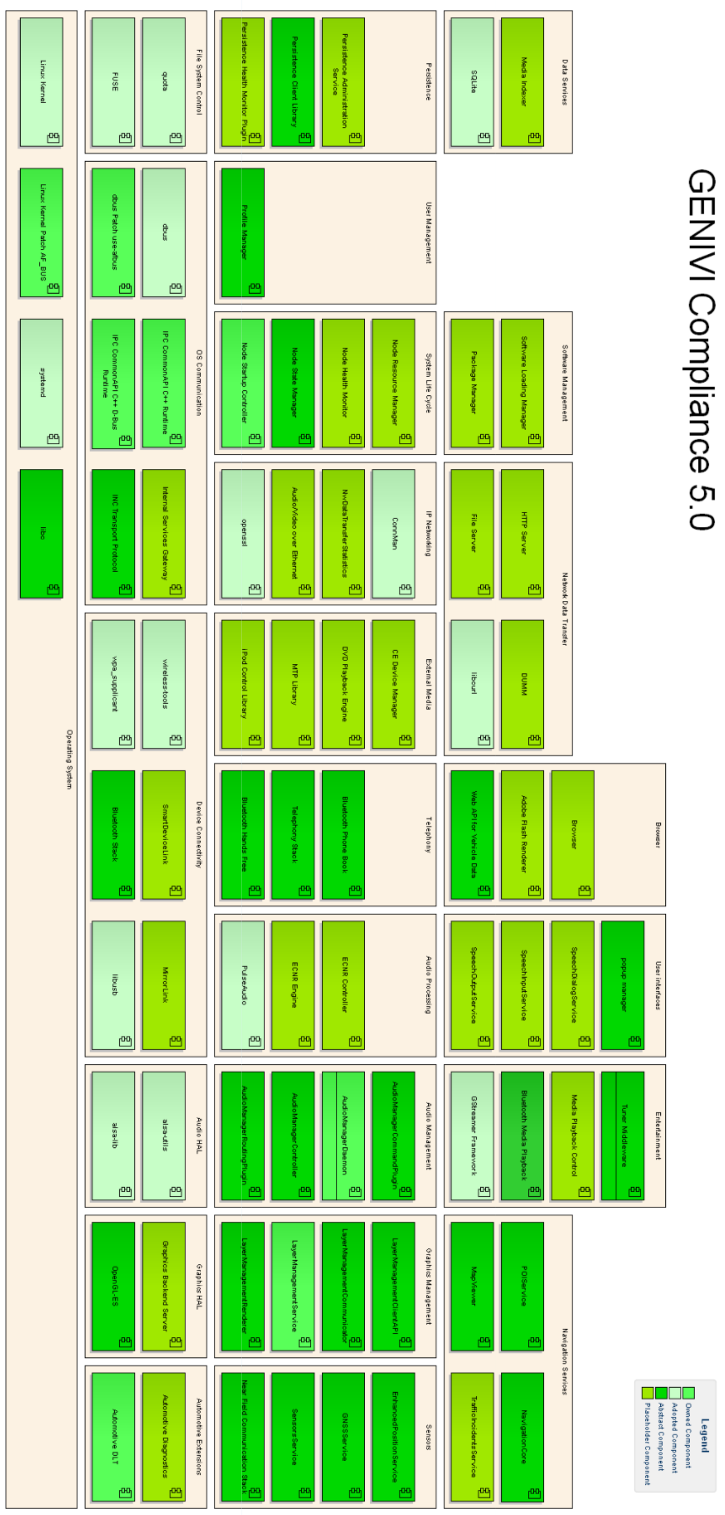


Figure 4.5: GENIVI component cluster [35]

The compliance specification distinguishes between three basic component types; *Placeholder*, *Abstract* and *Specific (owned/adopted)* components.

	Placeholder	Abstract	Specific	
P1	A	B	C	Mandatory
P2	D	E	F	Optional

Table 4.1: Component compliance

The former represents components that are currently unimplemented and are defined only by a set of requirements that their specific implementations has to fulfil. An example would be the browser component which has requirements on rendering, caching, etc.

The abstract components too do not refer to specific implementations, instead they are defined by their provided and required interfaces as well as behaviour. As such, the above two component types offer freedom as to how the capability is implemented, provided it meets all imposed implementation or functionality and interface requirements. An example of an abstract component would be the SensorsService component which requires an API to access vehicle sensor data.

Specific components on the other hand are owned or adopted components, defined by their implementation available in the form of source code. An example would be the Linux Kernel or the GENIVI hosted Automotive Diagnostic Log and Trace (DLT) component.

Furthermore, all components are assigned a priority level; *P1* or *P2*. The first level means the component is mandatory, whereas the second level signifies that it is optional but desired. In order to reach compliance, all mandatory components must be included.

In table 4.1 above, the component marked ”A” is a component that is specified by its imposed requirements and must be included, however, since it is a placeholder, a developer may chose whichever component it wants, as long as the component fulfils the requirements. At the other end of the spectrum is the component marked ”F”. This is optional to implement, however, if a developer choose to include it, a specific component must be used (you are not free to use whichever component you want). As such, even though GENIVI is completely open source, there is room for adopters to include proprietary solutions into the build and still reach compliance [35].

4.1.3.2 Projects

GENIVI’s approach to software development follows an ”upstream first” policy, meaning that prior to creating a solution of their own, any viable component already available in the FOSS community will first be adopted or adapted to their needs. However, suitable software which meet the defined requirements for their IVI platform does not always exist in upstream projects. As such, additional project hosting is necessary to facilitate some of the development of the sought-after functionality. Even so, only around 5 percent of the platforms code base consists of automotive specific code produced by GENIVI themselves [32].

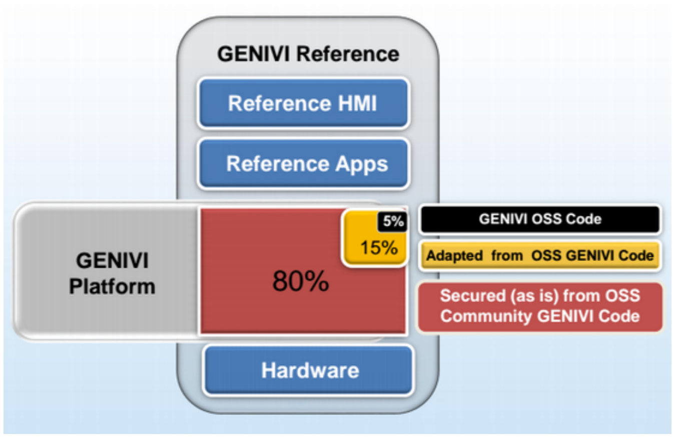


Figure 4.6: Code distribution for the GENIVI platform [12]

In order to make the features and functions of the IVI software platform acceptable to a broad range of automotive companies, imposed software requirements are contributed, discussed and aligned in the previously mentioned expert groups. These requirements are then made available to developers within GENIVI hosted projects.

Projects listed in the compliance specification are considered ”core” projects and are assigned a ”topic expert” dedicated to manage the communication between the project maintainer and the sponsoring expert group. The maintainer leads the project and is responsible for considering new features, accepting and committing patches and responding to questions related to the project. As such, any project participant is free to submit a patch, but it is ultimately up to the maintainer to commit it to the code repository.

Although not part of the compliance specification, additional project hosting is common. These projects are called ”incubator” projects and generally constitute tools, utilities or proof-of-concept solutions which implement functionality not yet defined by GENIVI. As they may eventually be added to the compliance specification, these projects are just as important as ”core” projects, but do not have a ”topic expert” assigned to them.

The alliance currently hosts more than 20 projects, including an audio manager, a graphical layer manager and a diagnostics log-and-trace component [32]. For the full list of projects, see Appendix C.

4.1.3.3 Baselines and Demo platform

Another major deliverable for the GENIVI alliance is the release of baseline software meant to provide a minimal software platform aligned with the GENIVI compliance. A selected work group of the alliance, the Baseline Integration Team (BIT), congregates software components specified in the GENIVI compliance specification and builds so called ”Baselines” [30]. These are assemblages of GENIVI middleware on top of a Linux kernel together with various open source resources [34]. The aim is that these baselines will verify that the compliance specification is working and that it is possible to integrate and assemble the software components to provide the expected functionality.

As the baseline releases follows that of the compliance specification, the content of a baseline will always match a version of the compliance specification (they are always GENIVI compliant). Furthermore all baseline releases come with a Platform Development Kit (PDK), which means they are open to customization at Kernel and middleware level.

There are currently two publicly available GENIVI Baseline incarnations, the Yocto and the Baserock baselines, each composed of approximately 150 different components [30]. Not only do they provide the build tools that enable the build-capability for the baselines, but also the means to add additional components to the project (i.e. add software components to a GENIVI baseline).

These are released with every new version of the compliance specification and are designed to reflect a GENIVI compliant software platform. OEM's and tier one suppliers then add their own software (usually a combination of open and closed code) to meet additional OEM system requirements [34].

The Yocto GENIVI Baseline

Hosted by the Linux Foundation, the Yocto Project is an open source project that provides tools and processes to allow developers to create their own custom Linux distributions [36].

The Yocto GENIVI Baseline is based on a software layer called ”meta-ivi”. This meta-data layer aligns the Yocto Project's reference system ”Poky” with the GENIVI compliance specification and contains information that defines build tasks for the Baseline. Meta-ivi is fully Yocto compatible which means it can be mixed with other Yocto compatible layers, including BSP layers [37]. With Poky, the meta-ivi layer provides the recipes to build free OSS, and to combine the results into binary images. It is also possible to generate an installable Software Development Kit (SDK), optimized for that particular binary image [37].

Both meta-ivi and Poky source code is available in git-repositories hosted by the Yocto Project.

The Baserock GENIVI Baseline

Baserock is maintained by Codethink, an associate member of GENIVI and similar to the Yocto Project, Baserock is an open source project aimed at delivering custom software systems by providing tools to build open source software components into complete operating systems. The Baserock GENIVI Baseline uses these build tools to assemble an optimized GENIVI software stack [32].

GENIVI Demo Platform (GDP)

Launched by the BIT is also a GENIVI demo platform meant to offer a higher experience of the GENIVI software. This builds on the baselines by adding an HMI and some proof-of-concept applications. It is not meant to represent a complete IVI solution, rather show off how a GENIVI IVI may look like [34].

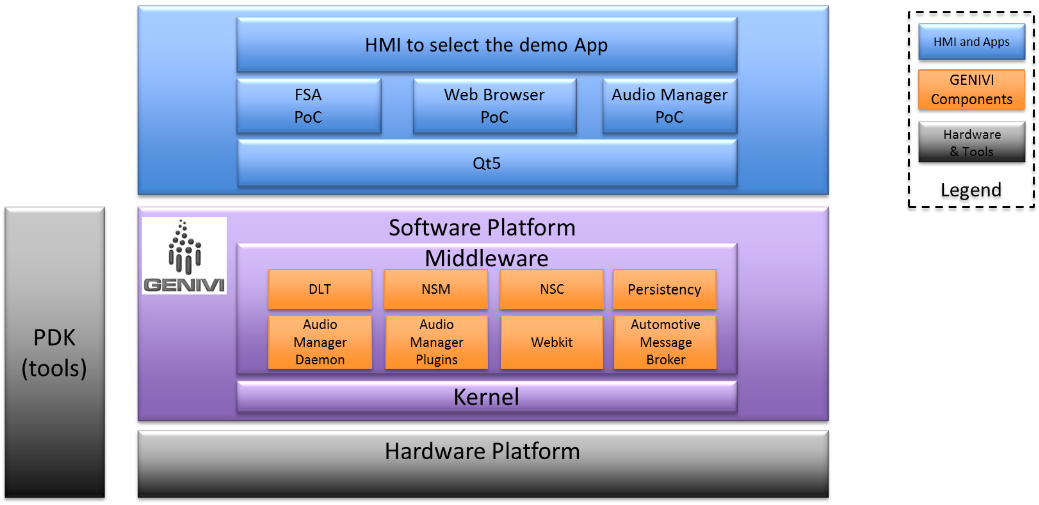


Figure 4.7: High-level block diagram for the GENIVI Demo Platform [34]

Figure 4.7 illustrates the hardware and software blocks in the GENIVI demo platform. The only thing differentiating this from the baselines is the top block, as the baselines do not offer any HMI or application implementations [34].

4.1.4 Openness and Licensing

So is GENIVI open? Predominantly so. The majority of the previously closed projects hosted by the alliance have now been pushed into the open source domain, with component source code available in GENIVI git-repositories. The default license for GENIVI hosted code artifacts is Mozilla Public License 2.0 (MPL 2.0) [38] and access is non restrictive; anyone can download the code and start developing, and GENIVI provides open support mechanisms in the form of mailing lists, bug-tracking databases, documentation servers and wiki-pages [32]. The same applies for the public Yocto GENIVI baseline which is based on meta-ivi, the Yocto layer for IVI. Additionally, access to the project plan is here provided [39].

As such, GENIVI follows up on its commitment to utilize software available under open source licenses, and what has been made public and pushed to the open source domain conforms with the practices of OSS. However, not all software is available to non-members and some of its work is conducted behind closed doors. Access to software test tools and the compliance specification are two important things that remains out of reach for non-members.

Although any membership level grants you access to all the above, the alliance employs a three-tiered membership system where all tiers are accompanied by an annual fee [38].

4.1.4.1 Licenses

GENIVI has requirements on the licenses used for their projects;

- The license shall allow combining the GENIVI hosted code with proprietary software and GPLv2 software alternatively.
- The GENIVI Alliance prefers a copyleft license to avoid privatization of the GENIVI hosted code.
- The GENIVI Alliance would like to use a license with a clear scope of the patent license regime.
- Where possible, the GENIVI Alliance would prefer a single license for all GENIVI hosted code instead of several licenses for different components.
- The GENIVI Alliance wants to avoid an uncommon license.
- The license should be valid worldwide.
- A GENIVI member must be able to sell products based on this to non-members.

As such, GENIVI has green-lit[4] the use of a number of OSS licenses for code and document artifacts. In the public policy for GENIVI licensing and copyright version 1.6 are specified the default licenses for code and document artifacts respectively, together with some additional exception licenses [38].

Code artifacts

- `Mozilla Public License, Version 2.0:` Weak copyleft. Preferred OSS license, default for GENIVI hosted code.
- `Apache 2.0 License:` Default permissive license for GENIVI hosted code. Incompatible with GPLv2.
- `MIT/X11 License:` Permissive license that is GPL compatible. Used for software components that need to be permissively licensed and combinable with GPL licensed software.

[4]Reviewed and accepted as suitable, without restrictions

- `GPLv2 and LGPLv2.1`: Strong copyleft. Acceptable when there is a risk for potential privatization. The former may only be used for programs and the latter only for software libraries.

Document Artifacts

- `Creative Commons Attribution-ShareAlike 4.0 License (CC-BY-SA)`: Copyleft. Disallows proprietary forks. GENIVI's preferred and default license for documentation artifacts.

- `Creative Commons Attribution 4.0 License (CC-BY)`: Permissive. Used for informative material.

- `Creative Commons Attribution-NoDerivatives 4.0 License (CC BY-ND)`: Accepts no derivatives. Used for documentation that GENIVI wants to ensure remains unaltered unless changed by GENIVI themselves. Applied to public parts of compliance specifications.

These are the principal licenses used by GENIVI. The MPL 2.0 and CC-BY-SA are the default licenses for code and document artifacts respectively, and must be used unless there is significant technical or business reasons to use a different license.

In terms of copyright for code hosted by GENIVI, it remains with the original author so long as it was not developed under a contract that requires ownership rights to be granted to GENIVI [38].

4.1.5 Maturity

In the context of open source GENIVI has come a long way since it first set out in 2009. Even though the lions share of GENIVI's members are companies in the automotive sector, with deep roots in proprietary software development, significant progress to openness has been made. Moreover, since its inception the GENIVI alliance has grown to include more than 180 companies ranging from OEM's to first tier suppliers and silicon vendors.

With a platform building on Linux, GENIVI has not only positioned itself in line with other organizations/alliances/work-groups (most notably AGL), but can also benefit from some of the perks that come with a mature, engaged and experienced community. In effect, everything they choose to adopt and/or adapt will build on prior advances made by a strong OSS community. As the evolution of IVI systems and their development goes forward, a community-based development model, where you give a little but get a lot, may prove to be the right way.

Additionally, in late 2013, as first automaker, BMW began production of vehicles containing GENIVI-based head units, proving that this new approach to IVI development is adequate in terms of functionality and quality [3].

With more and more companies taking interest in a Linux-based IVI system GENIVI's future is looking bright. While it is hard to say for sure if GENIVI is the solution to all things in IVI, the transition to the open source domain is undoubtedly a step in the right direction as the problems associated with an ever increasing amount of software, recurring costs of each program and long development cycles are not likely to solve themselves.

4.2 Automotive Grade Linux (AGL)

Automotive Grade Linux (AGL) was launched by The Linux foundation in 2012 as an open source project for the development and delivery of a common Linux-based software platform for the automotive industry. As such, its scope goes beyond IVI and in the long run the project aspires to include telematics and instrument clusters as well. The idea is that the platform should work as a reference platform that OEM's and suppliers can contribute to and utilize to produce their own commercial product by implementing additional technologies and creating a customized branded user interface on top of it [40].

Since its inception AGL has had the support of a wide range of companies, including automotive manufacturers, tier 1 suppliers and electronics and silicon vendors. The first automotive manufacturers to receive membership were Jaguar Land Rover, Nissan and Toyota. Additional members include Renesas, Fujitsu, Samsung, Intel and Texas Instruments. Today the AGL workgroup has over 50 participating members [41].

4.2.1 Objectives

The aim of AGL is to speed up the development and adoption of an entirely open source IVI software platform for vehicles. To accomplish this, AGL is striving to unite automotive manufacturers and technology companies to develop and maintain a common platform with Linux at its core that provides OEM's with full control of the user interface [40].

Much like GENIVI's approach, this would offer OEM's a base platform, on top of which they can focus their innovative efforts, rather than spending resources on development of separate, individual solutions [42].

Their goals, as described by themselves, are [13];

- Provide an automotive open source Linux platform that fulfils joint and shared requirements of the automotive industry.
- Become an upstream distribution intended to be adapted and optimized into commercial products.
- Provide a reference distribution to show and emphasize the benefits and abilities of the technology.

- Provide a development distribution aimed to enable a quick getting started experience for engineering projects as well as rapid prototyping.
- Offer a wide community of support composed of individual developers, academic associations and companies.

4.2.2 Backers and Organizational Structure

As mentioned above, an excess of 50 companies are today participating in the AGL workgroup. This includes four major automotive manufacturers and a variety of suppliers, namely Jaguar Land Rover, Nissan, Toyota and Hyundai, and Intel, Samsung, Fujitsu, Renesas and Texas Instruments respectively [43]. For a complete list of participants, see Appendix D.

The organizational structure of the AGL project is depicted in Figure 4.8. The Linux Foundation is hosting the project, essentially a workgroup consisting of a steering committee, its coordinator and a number of expert groups. The latter are responsible for carrying out development work, whereas the steering committee sets the project direction. The development process follows an ”upstream first” policy and anyone can become a contributor to the project, whether it be individuals, companies or academic associations [13].

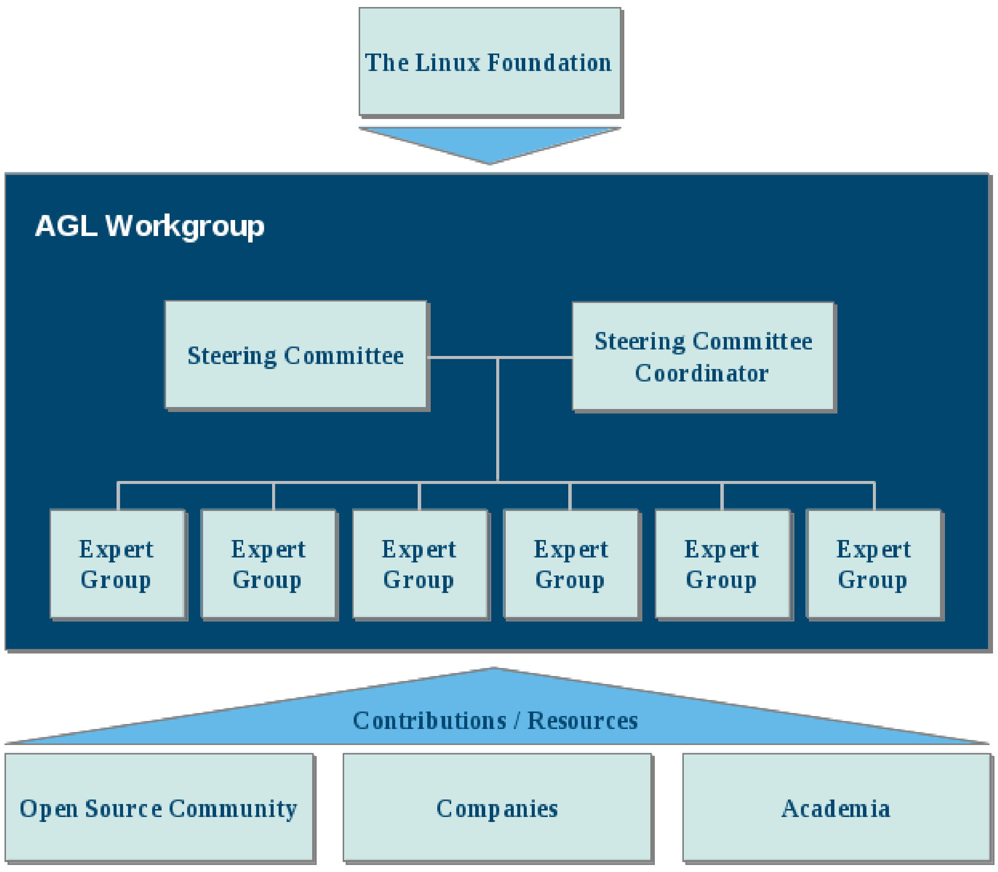


Figure 4.8: The structure of the AGL project [13]

The Linux Foundation's principal role in the project is to promote the vision of AGL by providing a neutral environment for collaboration and encouragement of working relationships between OSS communities and the automotive industry. In addition to this, the foundation is also offering training in technical, legal, open source and Software Package Data Exchange (SPDX) areas.

The steering committee is represented by a mix of companies and actively drives the day-to-day activities of the project. It promotes the overall adoption and technical success of AGL by providing supportive resources including code, specifications, tests and documentations.

The development work of AGL is managed by expert groups who provide technical expertise and leadership by designing, implementing, testing and documenting specific features of AGL. These are generally formed by the steering committee and are either responsible for a specific function (i.e. integration, quality assurance, management, etc.) or a specific part of the software (i.e. system maintainers for a particular code base). The expert groups are also responsible for collaborating with other open source projects linked to AGL and to integrate appropriate solutions, as well as specification and evaluation of suitable hardware for the platform [13].

4.2.3 Technology

In all development, economization of resources is of great importance and reuse of existing work is common. This is true also for open source software development, where improving and working upstream first often provides a more stable and well-tested platform, something that AGL reflects in how it is built [13].

As such, the initiative chose the Tizen IVI profile, see Section 4.2.3.1, as the base for their AGL IVI distribution. For other applications (e.g. Telematics, instrument clusters, etc.) AGL may work off a different platform, or it will start development from scratch [44].

The basic building blocks of the AGL platform is quite similar to other embedded Linux platforms, as it is possible to use the same kernel and a lot of the same middleware and open source components. However, AGL has a higher interest in fast boot-time, security and getting access to vehicle-specific buses such as the Controller Area Network (CAN). The AGL platform supports a variety of different hardware architectures and is to some extent compliant with the GENIVI compliance specification (likely to be fully compliant in the near future) [45].

Since the AGL IVI platform is based on Tizen IVI, which is an IVI layer/profile for the Tizen OS originally created for mobile and embedded devices, it builds on an extensively tested and successful OS platform by adding additional user interface and middleware components to it. Furthermore, the majority of what is developed by AGL is submitted back upstream to Tizen [13].

The demarcation line between AGL and Tizen resides somewhere at the middleware layer [45], see Figure 4.9.

AGL's software architecture can be described with four layers, starting with an application and HMI layer on top, followed by a framework layer that houses API's for creating and communicating with applications. Underneath it is a services layer that contains user-space services accessible by all applications. At the bottom is an operating system layer which provides the kernel, various device drivers and common OS utilities [46].

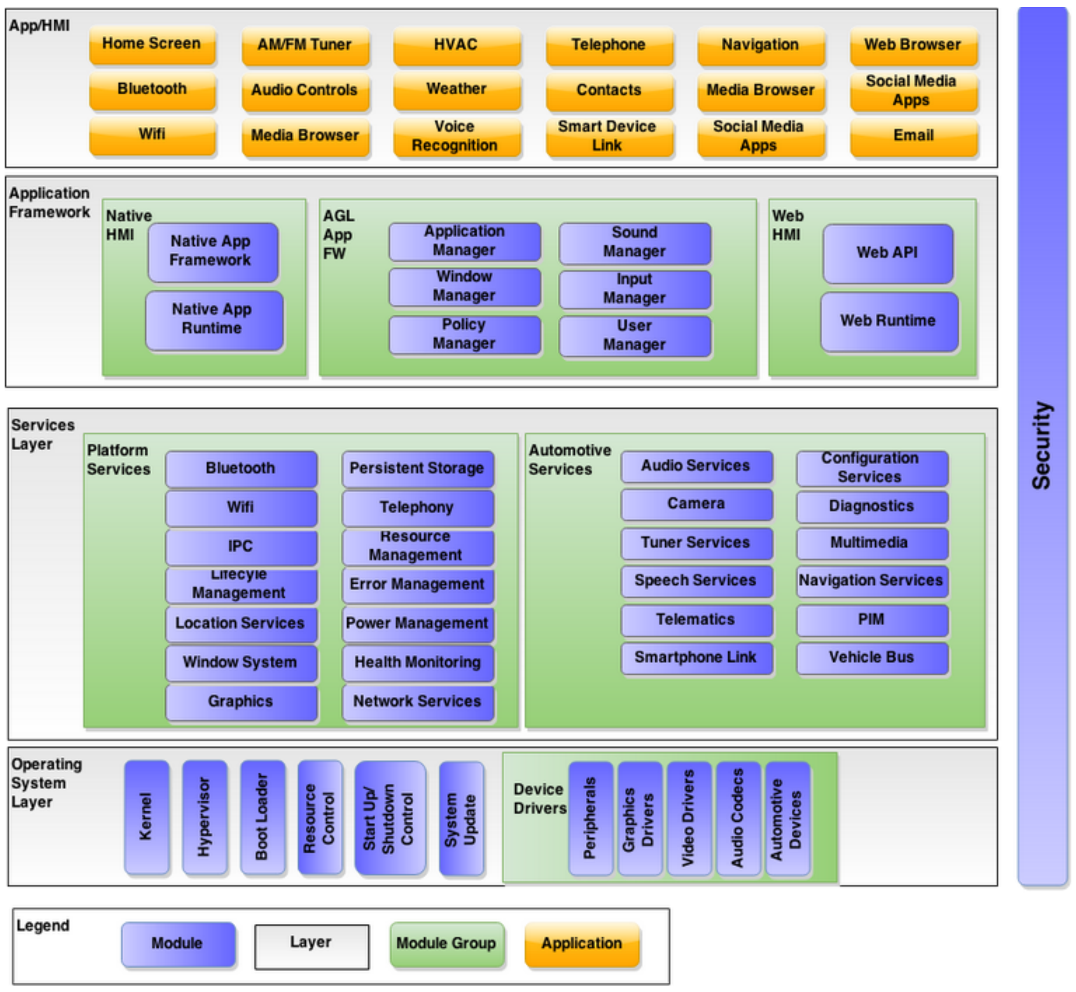


Figure 4.9: AGL Architecture Block Diagram [46]

The AGL platform's user interface and features are completely made in JavaScript and HTML5, and to allow applications to transmit data to and from the vehicle, the platform communicates with the vehicle via an Automotive Message Broker (AMB) with the use of Tizen IVI web runtime. This is in turn based on Crosswalk, an HTML5 application runtime which extends the web platform to enable the deployment of applications with dedicated runtimes. Some of the features currently included in the user experience for AGL are presented in the list below [47].

- Home Screen
- Dashboard
- Heating Ventilation and Air Conditioning (HVAC)
- Google Maps
- Media & News Service
- Bluetooth Support
- Media Oriented System Transport

(MOST) Audio Controls
- Smart Device Link[5]
- Near Field Communication (NFC)
- Browser
- Wi-Fi
- Fingerprint Recognition
- Voiceprint Recognition
- Weather information
- Email Service

4.2.3.1 Tizen

Tizen is a flexible open source Operating System (OS) aimed at many different device areas. It is built with a bottom-up approach and tries to address stakeholder needs in the mobile and connected devices ecosystem. As such, it is available in several tailor-made profiles; *Tizen IVI, Tizen Mobile, Tizen TV* and *Tizen Wearable.* These profiles all share the same bottom infrastructure, called *Tizen Common*, on top of which they employ custom features to meet the needs of the given business sector.

The Tizen IVI profile is driving the evolution of IVI systems by offering a free and open source development platform. As IVI solution it is designed to provide modern portable applications the ability to deliver rich multimedia and internet experiences in the vehicle. The latest version of the platform is Tizen IVI 3.0 and is used by AGL [49].

Tizen IVI 3.0

Tizen IVI 3.0 is based on Linux and is one of the most advanced open source platforms available to the automotive industry. It is a platform that can be used to build commercial automotive IVI solutions, and can be extended and modified by OEM's to meet any specific requirements. Tizen IVI is compliant with the GENIVI 7.0 specification on two hardware platforms; the Minnowboard MAX [50] and the Nexcom VTC 1010-IVI [51], and has been chosen by AGL as upstream reference OS platform.

To ensure device compatibility Tizen IVI 3.0 has been equipped with a compliance specification and an equivalent/corresponding test suite. There is also an SDK available for web application development called Tizen IVI SDK, which can give access to vehicle Web API's that are compliant with the W3C Web API standard. As a supplement, support for the Yocto build tool has been added and can be used to adapt and enhance the Tizen IVI 3.0 images to meet specific requirements [52].

Tizen IVI 3.0 supports various hardware architectures and platforms. It has a good build-up structure, with a quite extensive middleware layer and an HTML5 application development framework. It also includes many essential and practical features like multimedia using GStreamer, support for multiple display windows using Wayland, fast boot using SystemD, connectivity and telephony stacks [53].

[5]A GENIVI project that standardizes in-vehicle interfaces to provide a framework for integration of brought-in applications [48]

4.2.4 Openness and Licensing

In terms of licensing, the AGL project has a policy of allowing contributing projects to be free to choose their own OSS license. Each individual package thus contain its own license information. At the moment, the most commonly employed license is Apache 2.0, but many packages make use of MPL 2.0 to allow others to build proprietary extensions to the core system without having to open source them. As such, the release in its entirety does not have an overall license, but is governed by the licenses of the individual components.

The majority of the projects hosted by the Linux Foundation follow this policy of offering flexibility to which license a code is submitted under, as long as it fits the project (compatible to other code's licenses) and the contribution adheres to the overall compliance requirement [54].

For code contributed to or taken from upstream projects, the license is usually retained under the upstream project's license. All licenses used by AGL are approved by the OSI and newly developed code is increasingly licensed under MPL 2.0, whereas documentation artifacts use Creative Commons Attribution International 4.0 license.

When it comes to openness, the AGL project is aligned with the *best practices*, described in Section 2.2.1. It has an open community of developers, manufacturers and vendors, where anyone is welcome to participate and contribute. As such, code and documentation artifacts are free for anyone to access and use. Furthermore, open source mechanisms such as, mailing lists, documentation and wiki-pages are provided, although not all are publicly available. One example is the steering committee mailing list, which is only accessible to members of the committee.

The AGL project employs a three-tiered membership system, where members are listed as gold, silver or bronze depending on their level of engagement in the project [13].

4.2.5 Maturity

AGL was launched in 2012 with the aim of delivering a continuously maintained common Linux-based software platform for automotive purposes, primarily IVI. With the Tizen IVI profile as upstream distribution, the AGL workgroup are successfully introducing open source Linux to the automotive domain and has enjoyed a steady growth of members since it set of. Minimizing replication of development efforts where possible and working upstream is a mentality that aligns itself not only with other initiatives (e.g. GENIVI), but the industry as a whole. There is also a high degree of open collaboration within the AGL community and the majority of the work is taking place in an open environment.

Furthermore, as AGL is based on Linux, automakers will have access to a continuously growing software stack with support for multiple hardware architectures and an ability to create an explicit, unique user experience. Being a Linux Foundation project also adds merit to the overall credibility of the project. A first version of AGL was released in late June 2014 and some of the goals for 2015 include; completing and publishing the AGL specification, create a distribution for the entire industry, and complete a prototype with reference applications [55].

4.3 Automotive Grade Android (AGA)

Automotive Grade Android (AGA) is an open source project hosted by Vehicle ICT Arena, which is operated by Lindholmen Science Park in Gothenburg. Vehicle ICT Arena currently has over 30 partners and a couple of financiers, who in collaboration with the AGA project, strives to enable the integration of Android applications to the IVI domain. The project is especially geared towards developers who want to build applications for the automotive industry, but also system builders for building complete IVI systems [56].

The idea behind AGA is to enable input/output of data to/from the vehicle, so that both new and existing applications can extend their capabilities to those of the vehicle. In addition, AGA has developed supplementary design principles for safe use of applications in vehicles (driver distraction), following the already present Android principles. As such, AGA opens up possibilities for creation of custom applications that take advantage of the vehicle data and ensures these interact with the driver in a safe manner [14].

4.3.1 Objectives

AGA aims at leveraging Android's large and dynamic ecosystem by managing and evolving the use of Android as an IVI platform. The goal is to create and foster an open-innovation environment for in-vehicle software by providing resources and services for developing IVI solutions based on Android.

Additionally, AGA has put a lot of effort on the implementation of guidelines for how applications could safely interact with the driver in order to minimize driver distraction [14].

With the main target audiences for AGA being application developers and system builders, the services and utilities provided are as follows [57];

Application Developers

- `Developer Zone`
  An open community and portal that enables downloads of artifacts and information
- `Software Development Kit (SDK)`
  Software libraries and interfaces enabling infotainment application development for vehicles
- `Developer Guidelines`
  A collection of guidelines that eases the write-process for safe and automotive adapted applications

- `Simulation environment`
  Tools for developers enabling test without having access to a real target environment
- `Reference Platform`
  AGA is deployable on hardware

System Builders

- `Software Stack/Framework`
  A stack/framework that is maintained and further developed by a community
- `Large and Dynamic ecosystem`
  Compatibility with a large set of applications developed for AGA
- `Hardware Integration`
  Well-defined interface for easy integration with hardware
- `Security/policy Enforcement`
  Comprises security mechanisms to ensure only safe operations are permitted
- `Reference implementation`
  Reference implementation of AGA on some specific hardware

4.3.2 Backers and Organizational Structure

As mentioned above, Lindholmen Science Park in Gothenburg operates Vehicle ICT Arena, which in turn is the host of the AGA project. The Vehicle ICT arena has 34 partners, including the core members; AB Volvo and Volvo Car Corporation. Other partners consists of other companies, universities and organizations. The principal financiers of the Vehicle ICT arena are Västra Götalands Regionen, Business Region Göteborg and Vinnova. The main code contributors to the AGA project are Combitech and Swedspot [58]. For a complete list of participants, see Appendix E.

The organizational structure of the AGA project is based on a governance model inspired by the Apache Software Foundation (ASF). It is steered by the Vehicle ICT arena board in collaboration with a Project Management Committee (PMC), where the former supervises and manages the sponsorship, and the latter governs the project's progress. In essence, board members are involved in decisions regarding vision, objectives and financing, and committee members are involved in decisions regarding direction and progress of the project [14].

4.3.3 Technology

The AGA project is an Android-based solution that extends the OS and its API's to be automotive tailored, by including access to vehicle data and driver distraction information.

The AGA API makes it possible for applications to both read and to inject data to the vehicle, and includes support for a large set of Fleet Management System (FMS)[6] signals.

As opposed to for example GENIVI, AGA uses a top-down approach to software development and focuses primarily on application developers. As such, apart from the stack itself, there is an SDK available for developers. The SDK and the stack covers functions and features such as; access to vehicle signals, hardware buttons, driver distraction and policy management.

Intended to ease the development of AGA applications, the SDK allows developers to try out their application using simulated data, rather than having to connect it to a real vehicle. It is currently available on both Linux and Windows.

Figure 4.10 illustrates of how the AGA stack is integrated with the rest of the IVI system layers. The Northbound interface borders the applications layer and manages reading/writing from/to the vehicle, and reacts on driver distraction changes. In the other end is the Southbound interface, which borders the OEM integration layer. This interface integrates vehicle data and hardware buttons, sets access policies, and change the driver distraction level [14].

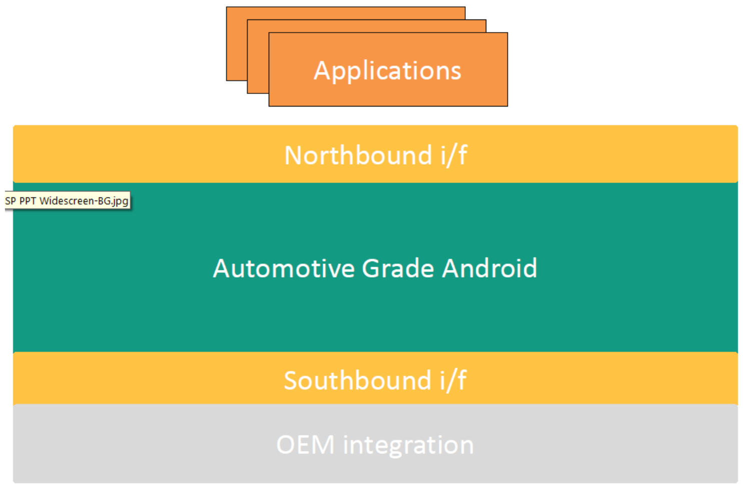


Figure 4.10: An overview of the integration of AGA into the residual IVI system [57]

A technical overview of the AGA stack is presented in Figure 4.11 and its specific components are described below [14].

[6]A standard interface to vehicle data [59]

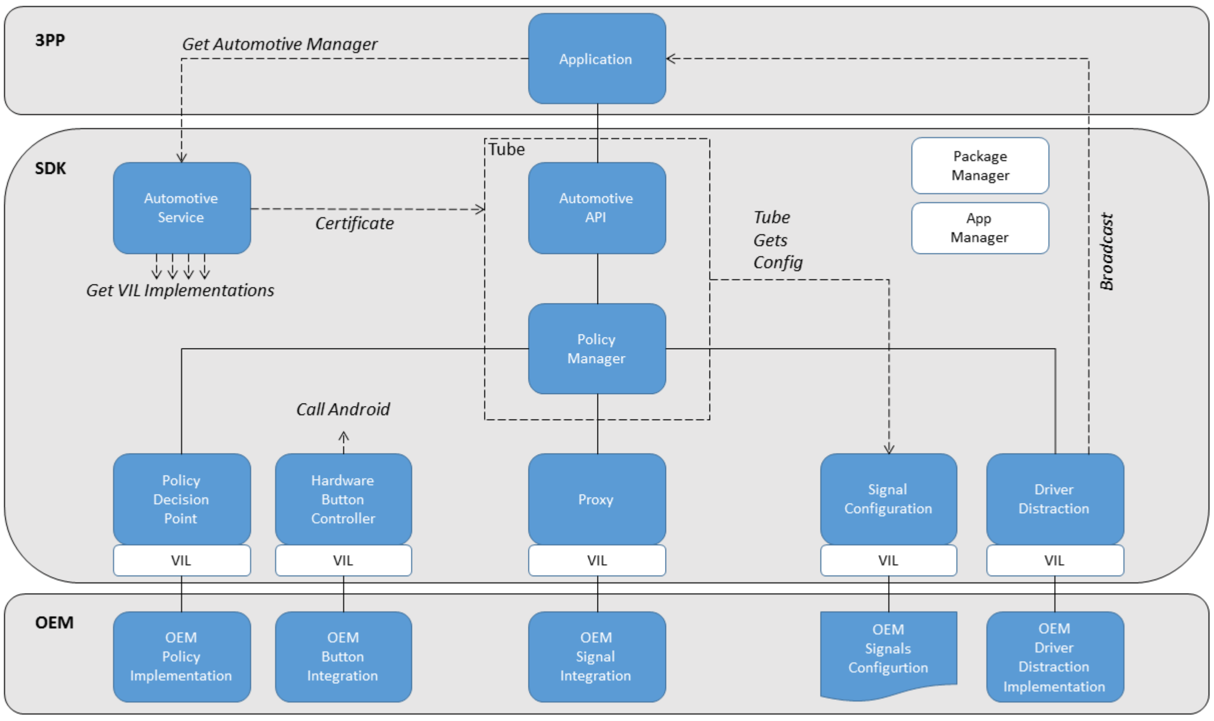


Figure 4.11: An overview of the AGA's architecture [14]

Automotive Service

The automotive service component initiates all the Vehicle Interface Layers (VIL) and manages the applications requests to obtain a new tube interface. The tube interface is provided by an automotive manager, which works like an interface between the application and the created tube. All Android specific code in the AGA stack is encapsulated in this component.

Tubes

Applications use separate tubes in order to ensure that the policy is followed by the different applications. Applications are dropped if they do not follow the policy. The tubes have access to and uses the AGA API and the policy manager.

Policy decision point

The policy decision point component is used to set the policies for the tubes. OEM's have the possibility to implement their own policy settings, as these can often differ for different OEM's.

Hardware button controller

The hardware button controller component is launched by the automotive service and has a pre-defined interface that makes it possible for an OEM to integrate and map hardware buttons in the vehicle, e.g. steering wheel buttons, to standard Android key-presses.

Proxy
The proxy component enables the actual vehicle signal integration by providing a node that vehicle signals can connect to. Furthermore, the proxy component handles the communication between the tubes and the vehicle through various functions, which are called from the individual tubes.

Signal Configuration
The signal configuration component is used to configure available vehicle signals. If desired, OEM specific signals can be added to the already existing Fleet Management System (FMS) signals.

Driver distraction
The driver distraction component manages the driver distraction information. The distraction state of the driver is evaluated according to defined levels of distraction. The computation of this level is performed by OEM's and the AGA stack enforces the level up to the applications.

Vehicle Interface Layer (VIL)
The VIL is located between the AGA stack and its Android automotive services, and the vehicle bus hardware. The VIL can be divided into two components, a VIL Daemon and a Vendor VIL. The VIL Daemon is responsible for initializing the Vendor VIL as well as processing communication from the Android automotive services to the Vendor VIL by solicited commands. The Vendor VIL is vehicle-specific and is responsible for process communication from the vehicle bus hardware to the VIL Daemon by unsolicited commands [14].

The internal communication in the AGA stack is realized in several different ways, as shown in Figure 4.12. In order to send and receive signals between the applications and the automotive service, Inter-Process Communication (IPC) with Binder is used. Subsequently, standard Java method calls are used both between the automotive service and the automotive API and between the automotive API and the policy manager. The policy manager uses System Data Protocol (SDP) to communicate with the proxy, which in turn externally provides connectivity to the OEM in form of a SDP node to enable the actual vehicle signal integration. In addition, Android broadcast signaling is used between the driver distraction component and the applications to enable communication regarding state changes [14].

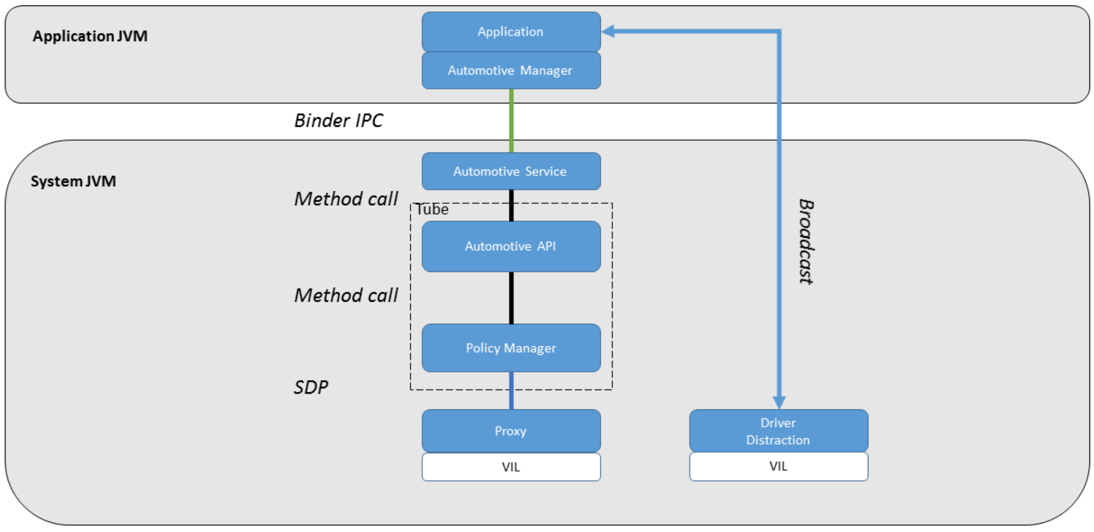


Figure 4.12: An overview of the internal communication within the AGA stack [14]

4.3.4 Openness and Licensing

As mentioned in Section 2.2, how open an open source project/platform is depends not only on the license that is used, but also on how the project is governed in terms of decision making being open, transparent and accessible to all users. Thus, it is of interest to evaluate the project's governance model in addition to the licenses that are used. The OGI introduces governance of a project as a measure of openness by targeting and evaluating four areas; access, development, derivatives and community structure [27].

The AGA project's model of governance is inspired by the ASF and it aims to address and meet the OGI criteria of each area to as great extent as possible. In access, AGA show positive characteristics for openness, such as free source code available to all developers at the same time, under a permissive OSI approved license. There is also transparency in the decision mechanisms in the form of publicly available meeting protocols/discussions, project mailing lists, source code repositories, forums and developer documentation.

When it comes to the development area, there are many properties that contribute to greater openness provided. Identification of all project committers and the possibility for any developer to become one, source code contributions can be tracked back to the original committer, and progress updates of the code and the acceptance process carried out by the maintainer are also available.

For the derivatives area, the general attitude is to not control and constrain the use of the platform or the go-to market channels for application derivatives. Finally, the AGA project employs a flat community structure, which means that different membership statuses do not entail tiered rights [14].

Licenses
The source code of AGA is released under Apache License 2.0 [14], which is a permissive free software license from the ASF. The Apache License allows a user to freely use, modify and distribute the software, under the terms of the license, without having to worry about royalties. Furthermore, the Apache License 2.0 is compatible with GNU General Public License (GPL) version 3. This allows code released under Apache License 2.0 and GNU GPL version 3 to be combined, as long as the GNU GPL version 3 is used for the resulting software [60]. The default license for AGA documentation artifacts is Creative Commons Attribution International 4.0 License [14].

4.3.5 Maturity

The AGA project is attempting to fill a void by introducing an open Android platform to the vehicle integrated device domain, similar to what AGL is doing with Linux. Despite having the lowest number of backers of the three initiatives in this study, the project maintainers have managed to release a first version of AGA containing simulation tools, documentation, SDK and ROM build scripts for selected platforms. The main contributors and maintainers are Combitech and Swedspot, who are also partners of the AGA host, Vehicle ICT arena. Even though the arena has over 30 partners, there is currently only one car manufacturer (Volvo Cars) aboard. This may not be critical in the short run, but for a broad adoption of any initiative, manufacturer support is likely to be key (after all, they are making the vehicles for which the technology is developed). In terms of openness, AGA has aligned itself with the best practices of FOSS and use a permissive license for all source code. The first official (and current) release of the AGA stack was in the autumn of 2014 [14].

4.4 Evaluation

The performed study shows that open source software solutions are on the advance, with members of the industry joining ranks to develop the next generation of IVI systems. In a study commissioned by GENIVI in 2010, a number of companies (both members and non-members) were asked to identify the major market factors that will drive the future of IVI architecture. Disclosed therein was that the average royalty price per unit for proprietary IVI stacks ranged from \$5 to \$15 and the total cost for development ranged between \$20 million and \$50 million on average. Assuming a mature and stable platform, the participants felt that a 50% cost reduction could be achieved by utilizing open source as opposed to proprietary solutions. Even though the royalty price model was likely to persist, it too was expected to be driven down (-25-30%) [61].

However, while the automotive industry has a lot to gain from stepping into the open source domain, one should realize that such a shift would not render proprietary solutions obsolete or unwanted. The previously mentioned differentiating features constitute an area where these solutions will offer manufacturers the means to create and tailor highly competitive custom, branded user experiences.

4.4.1 Initiatives

Despite the different approaches between the initiatives described above, they are attempting to solve a common problem faced by the entire industry - an increasing repeated effort and cost of development and maintenance of software for automotive IVI purposes. The solution is spelled open source and while it at first glance may seem as AGL and GENIVI are competing for the same price, the truth is they are closer to complementary entities than anything else. Where GENIVI is focusing on middleware standardization, API specification, and supply chain management, AGL goes further, all the way up to the HMI and focuses more on actual integration and providing a technology platform for people to use.

As such, the GENIVI compliance specification points out certain Linux components, their versions and which API's that needs exposure to reach compliance. This corresponds to about 80% of the software stack, which for an OEM means that they can then add their differentiating features on top, reducing the overall development cost. AGL, which will eventually be GENIVI compliant, will be a distribution that people can use and derive their product development from. Rudolf Streif, the Linux Foundation's director of embedded solutions informs Rory MacDonald of Linux User & Developer magazine that, "We see AGL as the Debian or the Fedora of automotive... cutting-edge developer distributions" [44]. In other words, a technologically advanced platform that you work off of to develop your end product, not something you put directly in production.

Both GENIVI and AGL have Linux in their cores and employ an upstreams first policy to software development. However, the GENIVI compliance specification stops short of the upper software layers and focuses on defining non-differentiating features, i.e. features that all IVI systems are expected to have, regardless of make or brand. As such, where GENIVI uses a bottom-up approach aimed at describing middleware and specifying API's, AGL offers a solution with a higher focus on features located in the HMI and application layers.

With AGL aiming to get GENIVI compliant, their target audience is largely the same, in fact, many companies/backers are equally involved in both projects, however, as their scopes are not the same, the projects' objectives differ somewhat. Where GENIVI's objectives in broad strokes can be described as defining a standardized base-platform for IVI solutions, those of AGL would be the development and maintenance of a common IVI platform based on Linux.

The AGA initiative on the other hand, is as much of a response to the increased cost of developing IVI systems, as it is a product of the successful mobile device application market and the interest in a similar system for automotive applications. With an objective that aims at evolving the Android platform to integrate with the automotive domain, AGA provides resources and services for developing IVI solutions based on Android. Even though no initiative investigated in this study offer a complete, ”off-the-shelf” IVI system solution, the AGA project is seen as being furthest away from this with its extension of the Android operating system. However, Android have a large applications development community and potential to become a complimentary OS for application support. At the very least, AGA can prove that the Android platform can be extended to the automotive domain.

4.4.2 Future

As automotive manufacturers, OEM's and suppliers are developing a new generation of head units, an increasing amount of work that previously took place in a strictly closed environment is being pushed into an open domain.

The notion that a collaborative transition to open source software will offer a lot in the field of automotive software systems is now widely accepted. However, as many companies are likely to struggle with this transition, GENIVI is identified as a suitable intermediate as it does not exclude proprietary components altogether. In fact, the development of the compliance specification has been made with proprietary components in mind, which is why the abstract and placeholder components may offer a lot in terms of customization.

While future IVI systems are likely to be open source, or at the very least use a mix of open and closed code, it is not entirely sure which initiative will gain the most ground in the battle for the next generation automotive head unit. We identify the involvement of manufacturers in the development process as a key ingredient to a successful adoption of any automotive system and initiatives that lack a consortium (in the style of GENIVI) may loose ground with nothing to drive the adoption.

However, the initiatives based on Linux are identified as having the best conditions for success, as a study by IHS Automotive concludes that Linux will push past both QNX and Microsoft Embedded Windows and take the lead with 41.3 % of the market shares by 2020 [62].

5

Design and Implementation

This chapter describes the implementation of the prototype, starting with the selection of software and hardware, including how and why these were chosen. The hardware setup and initialization of software are also steps that are explained here together with an evaluation of the prototype.

5.1 Software

The knowledge gained from the above study formed the basis for the selection of software platform for the prototype and was made prior to that of the hardware. We were first and foremost interested in a platform that was stable enough to satisfy our need for a relatively simple implementation process, while at the same time being advanced enough to offer additional customization options and demonstrative functions. A mature solution was as such preferable.

After having considering the different alternatives we ultimately decided to base our prototype on the GENIVI Demo Platform (GDP). One of the major factors that influenced this choice was the fact that the GDP is aligned with the Yocto Project and uses its templates, tools and methods for creating a custom Linux-based distribution. This meant we would be able to use a stable build system with broad support that offers many possible configuration options for a range of different software layers. As GENIVI's baseline software also offers additional freedom to certain component implementations, it was seen as an appropriate platform also in terms of software versatility. Furthermore, the software specification proposed by the GENIVI Alliance is already adopted into some head units today (BMW), and with the large number of backers and activity showed by the initiative this adoption is likely to increase in the coming years.

The combination of these factors together with the fact that our supervisors had some prior experience with the Yocto Project and that the GDP was specified to run on commercially available hardware made it our choice as IVI demonstrator.

The platform is based on the Yocto GENIVI Baseline and adds an HMI and a few Proof of Concept (PoC) applications together with P1 middleware components of the compliance specification [12]. The current version is based on the Intrepid release of the Yocto GENIVI Baseline and is aligned with both the GENIVI compliance specification (v7.0) and the Yocto Project 1.7 (dizzy) [34].

5.2 Hardware

Since the GDP was compatible with the Yocto Project 1.7 (Dizzy), the hardware would require an associated Board Support Package (BSP) equally compatible with the same release version. We also wanted to be able to utilize the Human Machine Interface (HMI) that comes with the GDP together with some kind of touch-interface.

Initially, there were several possible candidates, but a limited budget narrowed it down somewhat. Out of two hardware boards recommended by GENIVI as suitable for use with the GDP, one was out of stock and the second not available for the general public.

As a result we looked at a third alternative, the Freescale i.MX53 Quick Start Board (QSB), which, although it was listed as generally suitable for GENIVI applications, was not specifically recommended for the GDP. However, we decided to go with said hardware since it had an associated BSP compatible with the Yocto Project 1.7 (Dizzy). The board itself is specifically aimed at infotainment solutions and uses an ARM architecture, has a display controller, hardware accelerated graphics, 720p video encoder, 1080p video decoder and several different connectivity options. In addition to the board we also purchased a compatible LCD touchscreen [12]. For additional information about the Freescale i.MX53 QSB, see Appendix F.

5.3 Hardware setup and software integration

Connecting the board to the host machine and setting up its peripherals was a fairly straight-forward process that enabled us access to power, internet, terminal and display. The host system was running Ubuntu 14.04 LTS in an Oracle VM VirtualBox on a Windows PC.

As mentioned above, the GDP is based on the Yocto GENIVI Baseline which utilizes the Yocto Project's tools and aligns its reference system "Poky" with the GENIVI compliance specification. The Yocto Project, providing the essentials for building customized embedded Linux distributions, imports and adds meta-data layers to build-recipes in order to create bootable images for a desired target platform [36].

The layers and an outline of their dependencies are for our build presented in Figure 5.1.

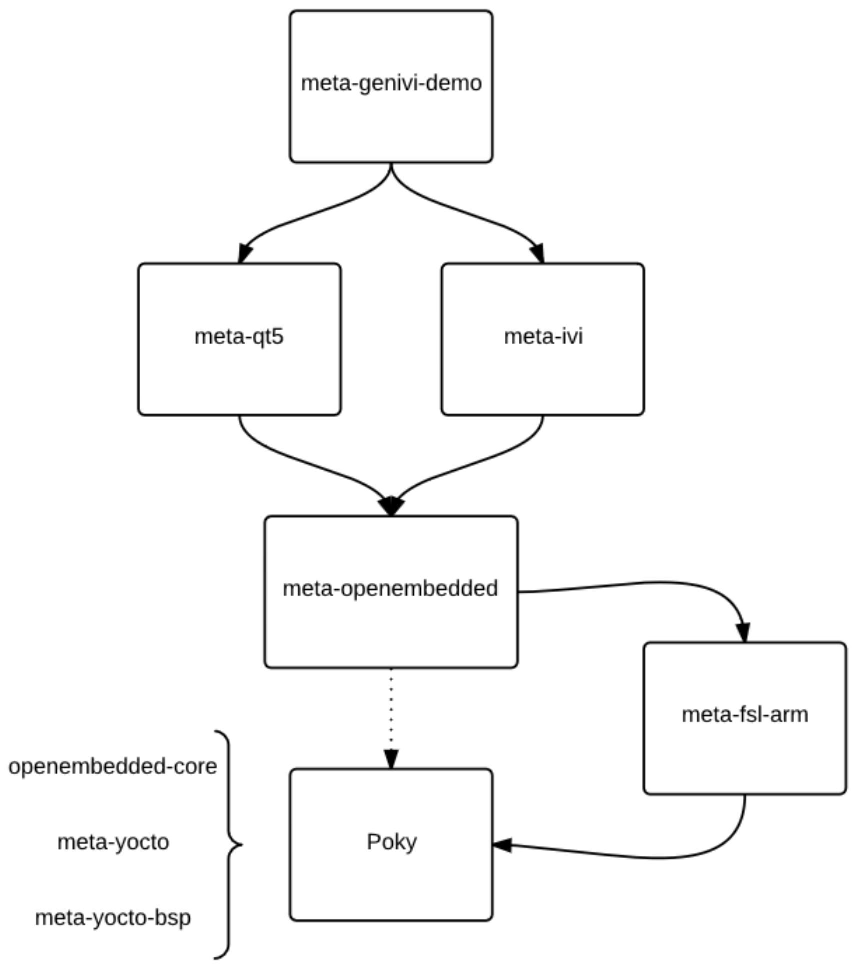


Figure 5.1: Dependency Tree

Starting at the bottom of this illustration, Poky constitutes a collection of tools and meta-data layers that form a cross-compiling integration layer. Included in this is a BSP layer that is used when building for a virtual target (QEMU). As such, the meta-fsl-arm is only ever used when building for the actual hardware (Freescale i.MX53 QSB). The next layer, meta-openembedded, essentially contains additional shared Open Embedded meta-data resources required by openembedded-core.

Meta-qt5 contains recipes for qt5 modules, which are commonly used in application software with graphical user interfaces, and meta-ivi constitute the Yocto GENIVI Baseline which adds In-Vehicle Infotainment (IVI) support when used with Poky. At the top is the meta-genivi-demo which adds an HMI and PoC applications to the baseline. As such, the GDP-build effectively uses all layers depicted in the figure when built for the hardware target, whereas the baseline-build only uses meta-ivi and down.

These layers together with supportive peripheral software were cloned from Git repositories down to our host system. One such supportive software was the Yocto project's build-engine "BitBake" which was used for all image builds. Separate directories for each build and target were also created to facilitate debugging and decrease execution times. All build parameters were in these directories set by two configuration files, *local.conf* and *bblayers.conf*. These contained information relating to which target the build was for (QEMU/Hardware) and which layers should be included (GDP/Baseline). Some more detail about the build configurations can be found in Appendix G.

Provided a successful build, a bootable image would be created for the specified target composed of all included software layers comprising the custom Linux kernel and root file system.

During the software setup and integration of the GDP for the hardware board, we had numerous issues with the meta-genivi-demo layer, which turned out to be somewhat unstable, and encountered compatibility conflicts between the baseline's meta-ivi and the BSP's meta-fsl-arm layer. The former issues, albeit time-consuming, were all resolved either by GENIVI or by our selves, but the compatibility conflicts remained unsolved despite extensive debugging and repetitive attempts to solve it. As a result, neither the GDP or Baseline could be built for the hardware board.

However, the builds for the QEMU target proved more successful, enabling us to run both the GDP and the baseline. This shows that the GENIVI-specific meta layers work fine with the Yocto project in general and Poky in particular. The baseline, lacking a proper HMI solely offered a shell, whereas the GDP offered a richer experience with its User Interface (UI) and proof-of-concept applications. Unfortunately, these were quite slow since the QEMU environment did not offer hardware accelerated graphics.

Additionally, a couple of test images provided by the Yocto project were built for the hardware target, with the BSP's meta-fsl-arm layer. This was to exclude the risk of a mismatch between Yocto/Poky and the BSP. As these builds were successful and no mismatches were found, it was determined that the compatibility conflicts mentioned above reside in between the baseline's meta-ivi layer and the BSP's meta-fsl-arm layer.

The root-cause of these compatibility conflicts has been traced back to an open source graphics-package, the *Mesa 3D graphics library*. This is a library used for rendering interactive 3D graphics and the issue relates to the EGL components therein [63]. EGL, which is enabled by the meta-ivi layer, is an interface between rendering Application Programming Interface's (API) (e.g. OpenGL) and the native windowing system (e.g. Wayland). However, the meta-fsl-arm of the BSP effectively disables EGL, manifesting a conflict of interest.

All attempts to modify the BSP and/or baseline to either include Mesa EGL or strictly exclude it, have so far been unsuccessful, resulting in a number of consequential errors. These errors seemed to be more severe than the one they originated from and with little additional time, no further resources were spent on solving them.

5.4 Evaluation

For the prototype, the design and implementation choices, including the selection of software and hardware, all had an impact on the final result.

Regarding the selection of software, the GDP with the Yocto GENIVI Baseline was selected. According to us, the GDP is a good complement to the Yocto GENIVI Baseline and a suitable platform for demonstration purposes of the GENIVI software and design, while the Yocto GENIVI Baseline serves as a proof of concept of the GENIVI compliance specification and could also be useful as a jump start for IVI developers who want to develop their own GENIVI compliant IVI system. However, since GENIVI's work on these platforms is an ongoing process they were not completely stable and we encountered numerous bugs which lead to a not so straight forward build process. This was alleviated somewhat by the good supportive efforts showed by its members in mailing lists concerning the GDP and Yocto GENIVI Baseline.

In our work with the prototype we also got acquainted with the Yocto Project, something we value highly, as its build tools and methods are used in many similar software systems today and are likely to be in the future as well.

As for the selection of hardware, we can in retrospect be somewhat critical of our choice since the associated BSP was not fully compatible with the GDP without adjustments, something we did not succeed in doing. Regardless, the hardware by itself is aimed at IVI implementations and has a lot of useful features with extensive documentation and community support channels. The issue is rather related to that this specific board + BSP combination was not tried and tested for this specific software.

However, at the time of writing, no suitable alternative could be acquired since these were either out of stock or, as was the case for the most promising option, only distributed to close partners (for a considerable amount of money).

The lesson we have taken with us from the choice of hardware is that it is often important and sometimes absolutely crucial to use tried and tested hardware for a specific software target and not to underestimate the value of having access to adequate support channels for help/feedback.

Regarding the results from the prototype development with the GDP (and the Yocto GENIVI Baseline), the QEMU implementation does indeed serve as a demonstrator for the GENIVI software platform and design. Despite our aspiration to implement it on hardware and evaluate its potential to input and process additional sensor data we believe the QEMU implementation offers ÅF valuable information about the GENIVI initiative's software implementations and current limitations.

6

Conclusion

Within the scope of this project we have investigated three open source initiatives for In-Vehicle Infotainment (IVI) software platforms. Additionally, the GENIVI Demo Platform was chosen for further practical prototyping. Given that the studied initiatives are all relatively new, this report aims at facilitating the evaluation process for developers interested in open source IVI software platforms.

Since our prior knowledge on the subjects of open source and IVI was rather limited, a great deal of resources was in the early stages of the project spent on online studies of these topics. However, shortly after having begun collecting information about the initiatives it was clear that it in some cases was rather sparse. As such, additional interviews were held with people who were either directly involved in these initiatives, or familiar with automotive software in general. These sources of information constitute the bulk of data collected during the course of this project. Apart from describing the three initiatives based on their objectives, backers, technology, openness, licensing and maturity, the report also provides background information related to IVI and open source software by briefly describing a common IVI architecture, and concepts associated with openness of software.

Out of the three initiatives investigated in this study, we believe the GENIVI Alliance has the best prospects for success in the near future. It has a large number of backers, a relatively high level of maturity and very active support channels. In combination with a detailed compliance specification that was made with proprietary software solutions in mind, the alliance aligns its work in the open source domain with that of its proprietary origin in a way that offers OEM's and suppliers freedom as to how certain software components are implemented, leaving room for highly competitive and customized solutions. The approach taken by GENIVI, to work towards a standardization of middleware components and API's, seems like the right way to go and the fact that BMW in 2013, as first automaker, shipped one of their models with a GENIVI-based head unit further strengthens the validity of the specification in terms of functionality and possible cost reduction.

Automotive Grade Linux (AGL) is similar to GENIVI in the sense that they both are working for an industry-wide adoption of an open source Linux IVI solution. However, their focus lies on the development of an automotive grade Linux stack, not a compliance specification. Based on Tizen IVI, it concerns all software layers associated with IVI, including the Human Machine Interface (HMI) and application layers. With a sizeable number of backers and a desire to become GENIVI compliant, AGL now have the potential to become *the* Linux stack for automotive applications and IVI. However, it does not seem to be as active as GENIVI and, open source recycling aside, it is unclear how much they have actually developed themselves.

When it comes to Automotive Grade Android (AGA), its role in the automotive domain is different. By providing the API's and Software Development Kit's (SDK)'s to extend Android to vehicles and enable the development of associated applications, AGA by itself does not offer an IVI solution. Rather, it offers an interface to access vehicle signals and the tools to create vehicle grade android applications. As such, we believe the success of AGA relies more on providing special purpose applications (in the style of fleet management systems for trucks) than infotainment as we know it. The fact that Google, with its Android Auto software have moved into the automotive domain brings additional uncertainty regarding the future availability of Google-specific apps (Play Store, Maps etc.) for other Android-based automotive solutions.

Regarding the practical work on a demonstrator of the GENIVI Demo Platform, we had the aspiration to both demonstrate basic functionality and evaluate its adaptability to additional hardware (sensor data). Even though we only managed to get a prototype up and running in a QEMU environment we believe it offers a good demonstration of a GENIVI-based IVI platform and that the components included in the compliance specification work as intended.

Bibliography

[1] Artifact (Software Development). Wikipedia. Accessed: 2015-03-19. [Online]. Available: http://en.wikipedia.org/wiki/Artifact_%28software_development%29

[2] Copyleft. Wikipedia. Accessed: 2015-03-18. [Online]. Available: http://en.wikipedia.org/wiki/Copyleft

[3] BMW Case Study. GENIVI Alliance. Accessed: 2015-02-19. [Online]. Available: http://www.genivi.org/sites/default/files/BMW_Case_Study_Download_040914.pdf

[4] Open Source Initiative. Open Source Initiative. Accessed: 2015-03-05. [Online]. Available: http://opensource.org

[5] Runtime System. Wikipedia. Accessed: 2015-03-23. [Online]. Available: http://en.wikipedia.org/wiki/Runtime_system

[6] GStreamer. Wikipedia. Accessed: 2015-04-13. [Online]. Available: http://en.wikipedia.org/wiki/GStreamer

[7] Wayland (display server protocol). Wikipedia. Accessed: 2015-04-13. [Online]. Available: http://en.wikipedia.org/wiki/Wayland_(display_server_protocol)

[8] Systemd. Wikipedia. Accessed: 2015-04-13. [Online]. Available: http://en.wikipedia.org/wiki/Systemd

[9] Autosar development partnership. Autosar. Accessed: 2015-01-17. [Online]. Available: http://www.autosar.org/about/basics

[10] Clark Libby. 5 Reasons Infotainment is the First Target for Open Source Software in Cars. Accessed: 2015-01-17. [Online]. Available: http://www.linux.com/news/featured-blogs/200-libby-clark/719560-5-reasons-infotainment-is-the-first-target-for-open-source-software-in-cars/

[11] Lucas Mearian. Your car is about to go open source. Accessed: 2015-01-17. [Online]. Available: http://www.computerworld.com/article/2485817/emerging-technology/your-car-is-about-to-go-open-source.html

[12] GENIVI. GENIVI Alliance. Accessed: 2015-02-11. [Online]. Available: http://www.genivi.org

[13] AGL. Automotive Grade Linux (AGL). Accessed: 2015-02-20. [Online]. Available: https://www.automotivelinux.org/

[14] AGA Wiki. Automotive Grade Android (AGA). Accessed: 2015-03-15. [Online]. Available: https://developer.lindholmen.se/redmine/projects/aga/wiki

[15] In-Vehicle Infotainment (IVI). Techopedia. Accessed: 2015-03-22. [Online]. Available: http://www.techopedia.com/definition/27778/in-vehicle-infotainment-ivi

[16] An Architecture for In-Vehicle Infotainment Systems. Dr. Dobb's. Accessed: 2015-03-23. [Online]. Available: http://www.drdobbs.com/embedded-systems/an-architecture-for-in-vehicle-infotainm/222600438

[17] Application software. Wikipedia. Accessed: 2015-03-23. [Online]. Available: http://en.wikipedia.org/wiki/Application_software

[18] Middleware. Wikipedia. Accessed: 2015-03-23. [Online]. Available: http://en.wikipedia.org/wiki/Middleware

[19] Operating system. Wikipedia. Accessed: 2015-03-23. [Online]. Available: http://en.wikipedia.org/wiki/Operating_system

[20] Board support package. Wikipedia. Accessed: 2015-03-23. [Online]. Available: http://en.wikipedia.org/wiki/Board_support_package

[21] Device driver. Wikipedia. Accessed: 2015-03-23. [Online]. Available: http://en.wikipedia.org/wiki/Device_driver

[22] Andrew Patterson. Automotive infotainment systems: Open source drives innovation. Embedded Computing. Accessed: 2015-03-12. [Online]. Available: http://embedded-computing.com/articles/automotive-source-drives-innovation/

[23] Michael Kerrisk (2012-08-8). GENIVI: Moving an industry to open source. Accessed: 2015-02-11. [Online]. Available: http://www.genivi.org/sites/default/files/in-the-news/2012_08_08_GENIVI%20Moving%20to%20Open%20Source%20-%20Michael%20Kerrisk.pdf

[24] Doug Newcomb. The next big OS war is in your dashboard. Wired. Accessed: 2015-03-12. [Online]. Available: http://www.wired.com/2012/12/automotive-os-war/all/

[25] Proprietary Software Definition. The Linux Information Project. Accessed: 2015-03-09. [Online]. Available: http://www.linfo.org/proprietary.html

[26] Open Source Definition. The Linux Information Project. Accessed: 2015-03-09. [Online]. Available: http://www.linfo.org/open_source.html

[27] Open Governance Index. VisionMobile Ltd. Accessed: 2015-03-05. [Online]. Available: http://www.visionmobile.com/product/open-governance-index/

[28] Git (software). Wikipedia. Accessed: 2015-05-20. [Online]. Available: http://en.wikipedia.org/wiki/Git_%28software%29

[29] Qemu. Wikipedia. Accessed: 2015-05-20. [Online]. Available: http://en.wikipedia.org/wiki/QEMU

[30] GENIVI FAQ. GENIVI Alliance. Accessed: 2015-02-11. [Online]. Available: http://www.genivi.org/sites/default/files/GENIVI%20FAQ%2012202013.pdf

[31] About GENIVI. GENIVI Alliance. Accessed: 2015-02-11. [Online]. Available: http://genivi.org/about-genivi

[32] GENIVI Open Source Projects. GENIVI Alliance. Accessed: 2015-02-11. [Online]. Available: http://projects.genivi.org

[33] Joel Hoffmann. The road ahead, Open Source IVI. GENIVI. Accessed: 2015-03-12. [Online]. Available: http://www.omaevents.org/wp-content/uploads/2014/04/05-Joel-Hoffmann.pdf

[34] GENIVI Wiki. GENIVI Alliance. Accessed: 2015-02-11. [Online]. Available: http://wiki.projects.genivi.org/index.php/Main_Page

[35] John Lehmann and Pavel Konopelko. The road ahead, Open Source IVI. GENIVI. Accessed: 2015-03-16. [Online]. Available: http://www.genivi.org/sites/default/files/GENIVI_Korea_Summit_Compliance_And_WwG.pdf

[36] Yocto. Yocto Project. Accessed: 2015-03-18. [Online]. Available: https://www.yoctoproject.org

[37] Webinar - The GENIVI Baseline. Video. The Linux Foundation. Accessed: 2015-03-18. [Online]. Available: https://www.automotivelinux.org/webinar-genivi-baselines

[38] Public Policy for GENIVI Licensing and Copyright Version 1.6. GENIVI Alliance. Accessed: 2015-03-10. [Online]. Available: http://docs.projects.genivi.org/License/Public_Policy_for_GENIVI_Licensing_and_Copyright_v1.6.pdf

[39] Meta-IVI the Yocto layer for In-Vehicle Infotainment. Yocto Project. Accessed: 2015-03-10. [Online]. Available: http://git.yoctoproject.org/cgit/cgit.cgi/meta-ivi/about/

[40] Melissa Logan. JVC Kenwood, Linaro, and Opensynergy Join Automotive Grade Linux. Automotive Grade Linux (AGL). Accessed: 2015-02-19. [Online]. Available: https://www.automotivelinux.org/news/announcement/2014/11/jvc-kenwood-linaro-and-opensynergy-join-automotive-grade-linux

[41] The Linux Foundation Announces Automotive Grade Linux Workgroup. Linux Foundation. Accessed: 2015-02-19. [Online]. Available: http://www.linuxfoundation.org/news-media/announcements/2012/09/linux-foundation-announces-automotive-grade-linux-workgroup

[42] The Linux Foundation explains the benefits of open-source collaboration with Automotive Grade Linux. TelematicsWire. Accessed: 2015-02-22. [Online]. Available: http://telematicswire.net/the-linux-foundation-explains-the-benefits-of-open-source-collaboration-with-automotive-grade-linux/

[43] Nathan Willis. ALS: Automotive Grade Linux. LWN.net. Accessed: 2015-02-22. [Online]. Available: http://lwn.net/Articles/517424/

[44] Rory MacDonald. Automotive Grade Linux. Accessed: 2015-04-08. [Online]. Available: http://www.linuxuser.co.uk/news/automotive-grade-linux

[45] Eric Brown. An Interview with Dan Cauchy. Linux.com. Accessed: 2015-02-28. [Online]. Available: http://www.linux.com/news/embedded-mobile/mobile-linux/780030-automotive-grade-linux-released-an-interview-with-dan-cauchy

[46] Automotive Grade Linux Requirements Definition. Automotive Grade Linux (AGL). Accessed: 2015-03-21. [Online]. Available: https://download.automotivelinux.org/POC/PoC_Spec/MASTER_COPY_AGL_Spec_v0.82.pdf

[47] AGL Wiki. Automotive Grade Linux (AGL). Accessed: 2015-02-21. [Online]. Available: https://wiki.automotivelinux.org/

[48] Smart Device Link. GENIVI. Accessed: 2015-04-13. [Online]. Available: http://projects.genivi.org/smartdevicelink/home

[49] Tizen. Tizen. Accessed: 2015-03-17. [Online]. Available: https://www.tizen.org/

[50] Meet MinnowBoard MAX. minnowboard.org. Accessed: 2015-05-25. [Online]. Available: http://www.minnowboard.org/meet-minnowboard-max/

[51] NEXCOM VTC 1010-IVI. NEXCOM. Accessed: 2015-05-25. [Online]. Available: http://www.nexcom.com/Products/mobile-computing-solutions/tizen-ivi-platform/tizen-ivi-platform/vtc-1010-ivi

[52] Tizen Wiki. Tizen. Accessed: 2015-03-17. [Online]. Available: https://wiki.tizen.org/wiki/Main_Page

[53] GENIVI Compliant Products. GENIVI Alliance. Accessed: 2015-03-18. [Online]. Available: http://www.genivi.org/compliant-products

[54] “AGL License Specification,” E-mail conversation, Automotive Grade Linux, accessed: 2015-03-27.

[55] Webinar - On the road with FOSS. Video. The Linux Foundation. Accessed: 2015-04-08. [Online]. Available: https://www.automotivelinux.org/webinar-road-foss-will-gpl-v3-make-it-board

[56] Automotive Grade Android. Swedspot. Accessed: 2015-03-15. [Online]. Available: http://www.swedspot.com/newave_portfolio/automotive-grade-android/

[57] Automotive Grade Android (AGA). Swedspot. Accessed: 2015-03-16. [Online]. Available: http://vehicle.lindholmen.se/sites/default/files/content/PDF/Innovation_bazaar_4sep2014/3._automotive_grade_android_developers_zone_niclas_lindmark_innovation_bazaar_2014-09-04.pdf

[58] Öppen plattform förenklar för apputvecklare inom fordonsindustrin. Combitech. Accessed: 2015-03-16. [Online]. Available: http://www.combitech.se/Om-Combitech/Nyheter-press-och-media/Nyheter-och-pressmeddelanden/2014---9/Oppen-plattform-forenklar-for-apputvecklare-inom-fordonsindustrin/

[59] Fleet Management System. Wikipedia. Accessed: 2015-05-20. [Online]. Available: http://en.wikipedia.org/wiki/Fleet_Management_System

[60] Apache License. Wikipedia. Accessed: 2015-03-20. [Online]. Available: http://en.wikipedia.org/wiki/Apache_License

[61] Automotive Infotainment Software Architecture Report. GENIVI. Accessed: 2015-04-10. [Online]. Available: http://genivi.org/sites/default/files/GENIVI_IVI_Software_Architecture_Report.pdf

[62] Linux to be top IVI platform by 2020, says study. Linux Gizmos. Accessed: 2015-05-07. [Online]. Available: http://linuxgizmos.com/linux-to-be-top-ivi-platform-by-2020-says-study/

[63] Mesa (computer graphics). Wikipedia. Accessed: 2015-05-25. [Online]. Available: http://en.wikipedia.org/wiki/Mesa_%28computer_graphics%29

[64] i.MX53 Quick Start Development Board. Freescale. Accessed: 2015-05-25. [Online]. Available: http://cache.freescale.com/files/32bit/doc/fact_sheet/IMX53RQKSTRTFS.pdf

A

The Open Source Definition

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of ”patch files” with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.

Source: Open Source Initiative [4]

B

GENIVI Members List

Original Equipment Manufacturers

- BMW Group
- China FAW Group Corporation R&D Center
- China Motor Corporation
- Honda
- Hyundai Motor Group
- Jaguar/Land Rover
- John Deere
- Daimler (Mercedes-Benz Research & Development)
- Nissan Motor Co. Ltd.
- PSA Peugeot Citroen
- Renault SAS
- SAIC Motor Passenger Vehicle Co.
- Volvo Car Corporation

First Tiers

- AISIN AW Co. Ltd.
- Alpine Electronics R&D Europe GmbH
- ALPS Electric Europe
- Clarion Co. Ltd.
- Continental Automotive GmbH
- Delphi
- Denso Corporation
- Harman International Industries Inc.
- Huizhou Desay SV Automotive Co. Ltd.
- Hyundai Mobis Co. Ltd.
- Lear Corporation GmbH
- LG Electronics Inc.
- Magneti Marelli
- Mitsubishi Electric Corporation

- Peiker Acustic GmbH & Co. KG
- Pioneer Corporation
- Robert Bosch Car Multimedia GmbH
- Valeo Interior Controls
- Visteon Corporation

OSV, Middleware, Hardware, and Service Suppliers

- A-Key S.r.L.
- Abinsula
- Accenture
- Access Europe GmbH
- Advanced Driver Information Technology
- Aicas Realtime GmbH
- Airbiquity Inc.
- Allgo Embedded Systems Pvt. Ltd.
- Alten SA
- Altera Corp.
- Aricent Group
- Arkamys
- Arynga Inc.
- ATP Electronics Inc.
- ATS Advanced Telematic Systems GmbH
- AutoNavi Software Co. Ltd.
- BearingPoint GmbH
- Black Duck Software
- BrightONE GmbH
- Cinemo GmbH
- Codethink Ltd.
- Cogent Embedded Inc.
- Collabora Limited
- Comarch SA
- CTAG
- Cybercom
- Digia USA Inc.
- DTS Inc.
- Electronics and Telecommunications Research Institute
- Elektrobit Automotive GmbH
- Ericpol Telecom
- Ericsson AB
- FPT Software Co. Ltd.
- Franuhofer ESk
- Fujitsu Semiconductor Europe GmbH
- Garmin Switzerland GmbH
- GlobalLogic
- GNSD Co. Ltd.
- Green Hills Software Inc.
- HCL Technologies Limited
- HiQ Gothenburg AB
- Hitec Micro System Inc.
- Hortonworks Inc.

- Huizhou Foryou General Electronics Co. Ltd.
- IAV GmbH
- IBM
- Igalia S.L.
- iGATE Patni
- Infobank Corporation
- Integrated Computer Solutions
- Itemis AG
- Ittiam Systems Private Ltd.
- IVIS Co. Ltd.
- Klocwork
- KPIT Technologies
- Lixar IT
- Luxoft
- Magellan
- Mobica Limited
- MTA SpA
- Myine Electronics Inc. (dba Livio)
- Navis Automotive Systems Inc.
- Neusoft Technology Solutions GmbH
- Nexcom
- Nielsen
- NNG Kft.
- NTT Data MSE Corporation
- OBIGO Inc.
- Open Car Inc.
- OpenMobile World Wide Inc.
- OpenSynergy GmbH
- Palamida Inc.
- Pathpartner Technology Consulting Pvt. Ltd.
- Pelagicore AB
- QuEST Global Engineering Services Pvt. Ltd.
- Red Bend Software
- Research & Engineering Center LLC
- S1nn GmbH Co. KG
- Sasken Communication Technologies Ltd.
- Shenyang MXNavi Co. Ltd.
- Sirius XM
- SMART Modular Technologies Inc.
- SmartPlay Technologies India Pvt. Ltd.
- Suntec Software (Shanghai) Co. Ltd.
- Suresoft Technologies Inc.
- Symbio
- Symphony Teleca Corp.
- T-Systems International GmbH
- Tata Consultancy Services
- TechniSat Digital GmbH
- Telechips Inc.
- Telemotive AG
- Tom Tom International B.V.
- Tuxera Inc.

- UIEvolution Inc.
- Videon Central Inc.
- Wind River
- Wipro Technologies
- WiseThan
- XSe (Mentor Graphics)
- ZENRIN DataCom Co. Ltd.

Silicon

- Analog Devices
- ARM
- Broadcom Corporation
- CSR Technology Inc.
- Freescale Semiconductor Inc.
- Intel
- ISSI
- Marvell International Ltd.
- MediaTek Inc.
- Micrel Inc.
- Micron Technology Inc.
- NVIDIA
- NXP Semiconductors Netherlands B.V.
- Qualcomm Incorporated
- Renesas Electronics
- ROHM Co. Ltd.
- Texas Instruments Incorporated
- Vivante Corporation

Other

- Battelle Memorial Institute
- Murata Manufacturing Co. Ltd.

Source: GENIVI Alliance [12]

C

GENIVI Open Source Projects and Baselines

Open Source Projects

- AF_DBUS D-Bus Optimization
- Audio Manager
- Browser Proof-of-Concept
- Diagnostic Log and Trace (DLT)
- Diagnostic Log and Trace - Transport (DLT-T)
- GENIVI Demo Platform
- IPC CommonAPI C++
- IVI Layer Management
- IVI Navigation
- IVI Radio
- LXCBENCH
- Lifecycle Management: Node Startup Controller (NSC)
- Lifecycle Management: Node State Manager (NSM)
- Media Manager
- Persistent Client Library
- SmartDeviceLink
- Wayland IVI Extention
- Web API Vehicle
- YAMAICA
- Pop-Up Manager
- Driver Workload Assessor

Baselines

- Yocto GENIVI Baseline
- Baserock GENIVI Baseline

Source: GENIVI Alliance [32]

D

Automotive Grade Linux Members List

Gold Members

- Intel
- Jaguar/Land Rover
- Panasonic
- Renesas
- Symphony Teleca
- Toyota

Silver Members

- Aisin AW Co. Ltd.
- Denso
- Fujitsu Ten
- GlobalLogic
- Nissan
- Pioneer

Bronze Members

- Advanced Driver Information Technology
- Advanced Telematic Systems
- AllGo Embedded
- BearingPoint
- China Mobile
- Cinemo
- Componentality
- Electronics and Telecommunications Research Institute
- Eureka Inc.
- Feuerlabs

- Harman
- Hitachi
- Host Concepts
- Hyundai
- Igalia
- JVC Kenwood
- LG Electronics
- Linaro
- Mcloudware
- Microchip
- Micware
- Miracle
- Mitsubishi Electric
- Moscow Design Bureau Compas
- NEC
- NTT Data MSE Corporation
- Nvidia
- OS Systems
- Obigo
- Open Synergy
- Reaktor
- ROSA
- Samsung
- Suntec
- Symbio
- Systena
- Texas Instruments
- Tieto
- Wind River

Source: Automotive Grade Linux [13]

E

Automotive Grade Android (Vehicle ICT Arena) Members List

Core Partners/Members

- Volvo Cars
- Volvo Group

Premium Partners/Members

- ArcCore
- Altran
- Autoliv
- Chalmers tekniska högskola
- Combitech (Main code contributor to the AGA project)
- Cybercom
- Delphi
- HiQ
- Högskolan i Halmstad
- Interaktionsbyrån
- Netgroup Engineering AB
- Pelagicore
- Prevas
- Semcon
- SMSC Sweden
- SP
- Vector Informatik
- Viktoria Swedish ICT
- VTI
- ÅF

Associate Partners/Members

- Actia
- Alpine
- Denso
- Fengco
- Fraunhofer-Chalmers
- Högskolan i Skövde
- Mitsubishi Electric
- Modelon
- NIRA Dynamics
- Qamcon
- QRtech
- Sentient
- Smart Eye
- Swedspot (Main code contributor to the AGA project)
- Time Critical Networks
- Yazaki

Source: Automotive Grade Android [57]

F

Brief Fact Sheet of the Freescale i.MX53 QSB

freescale™
i.MX

i.MX Applications Processors for Multimedia

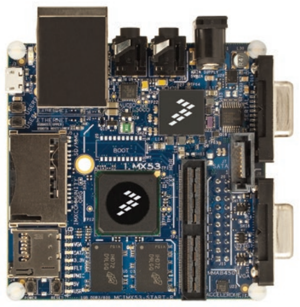
i.MX53 Quick Start Development Board

Cost-effective, multipurpose platform

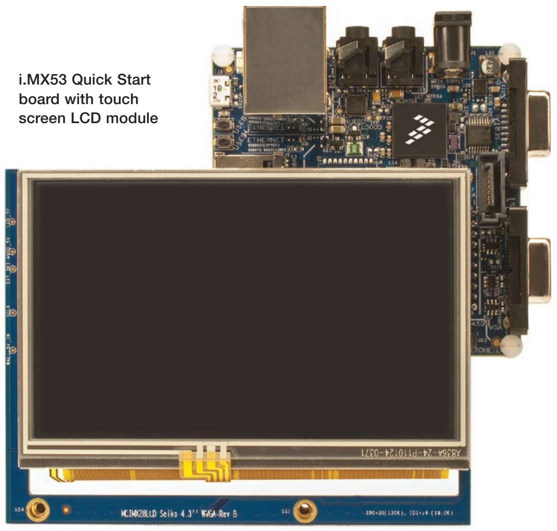
Overview

Freescale delivers a cost-effective, easy-to-use platform designed to simplify product evaluation and speed time to market with Quick Start development boards based on the i.MX family of multimedia applications processors.

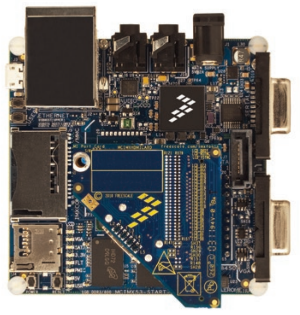
The first in the series, the i.MX53 Quick Start board is a $149 open-source, multipurpose embedded development platform. The i.MX53 comes with a power-efficient ARM® Cortex®-A8 core-based 1 GHz processor with peripherals and hardware accelerated graphics to support applications like human-machine interface (HMI) and support for HD multimedia functions. Also integrated in this platform is the MC34708 power management integrated circuit (PMIC) solution. Complete with highly optimized drivers and software, the i.MX53 enables broad-based applications for the embedded consumer, industrial and medical markets. Supported by a rich ecosystem and a community of developers at imxcommunity.org, the Quick Start board simplifies your out-of-box experience so you can get started quickly.



i.MX53 Quick Start board with touchscreen LCD module



i.MX53 Quick Start board with touch screen LCD module



i.MX53 Quick Start board with HDMI module

Energy Efficient Solutions by Freescale™

Benefits

First-Step Evaluation Platform

At $149, the i.MX53 Quick Start board is priced to attract a broad base of users, including professional developers and hobbyists. The Quick Start board is designed as an entry-level platform allowing you to begin writing code and experimenting before committing to additional development efforts.

Comprehensive, Yet Easy to Use

With an array of peripherals and a breadth of optimized software, the i.MX53 Quick Start board eases system design and allows for a full demonstration of features, such as the fully integrated LCD controller, Ethernet controller and multimedia functionality.

Rich Ecosystem, Vibrant Community

Build on the expertise of Freescale's ecosystem partners to do everything from customizing your application's user interface to using low-cost debuggers and development tools optimized to work with the Quick Start board. Join your fellow i.MX developers online at **imxcommunity.org**, an active community of open source developers.

Software and Tools

The i.MX53 Quick Start board comes pre-installed with the Linux® OS-, Android™ and Windows® Embedded Compact 7 board support packages (BSPs) are also available through third parties. In addition to optimized BSPs, Freescale also provides a large portfolio of optimized video, speech and audio codecs.

A variety of cost-effective debugging tools and complete development suites from partners like SEGGER Microcontroller, Macgraigor and IAR Systems are optimized to work with the Quick Start board. Also included is a VMware® player image running ready-to-go Linux, allowing those with Windows PCs to bypass the typical setup of a standard Linux-based development system.

freescale™

Hardware Features

Processor	• Freescale i.MX53 1 GHz Cortex-A8 Processor • Freescale MC34708 power management integrated circuit (PMIC) • 1 GB DDR3 memory
Display	• LVDS connector • VGA connector • Parallel LCD add-on card (via expansion connector) • HDMI add-on card (via expansion connector)
Audio	• SPDIF output via HDMI add-on card • Freescale SGTL5000 audio codec • Microphone jack • Headphone jack
Expansion Connector	• Enables parallel LCD or HDMI output • Camera CSI port signals • I^{2}C, SSI, SPI signals
Connectivity	• Full-size SD/MMC card slot • microSD card slot • 7-pin SATA data connector • 10/100 Base-T Ethernet port • Two High-Speed USB host ports • Micro USB device por
Debug	• JTAG connector • DB-9 UART port
Miscellaneous	• 3" x 3" 8-layer PCB • Freescale MMA8450QT 3-axis accelerometer • 2-amp, 5 V power supply

Ordering Information

Part Number	Description	MSRP (USD)
MCIMX53-START-R	i.MX53 Quick Start development board	$149
MCIMXHDMICARD	24-bit HDMI output port	$49
MCIMX28LCD	4.3" 800 x 480 WVGA with 4-wire touchscreen	$199
MCIMX-LVDS1	10.1" 1024 x 768 XGA display with capacitive multi-touch	$499

i.MX53 Quick Start Development Board Kit Contents

- i.MX53 1 GHz Cortex-A8 processor
- MC34708 PMIC
- 4 GB microSD card with Linux image
- 5 V power supply with worldwide adapters
- Micro USB cable
- Quick Start Guide
- DVD with VMware player, getting started video, demos and other documents

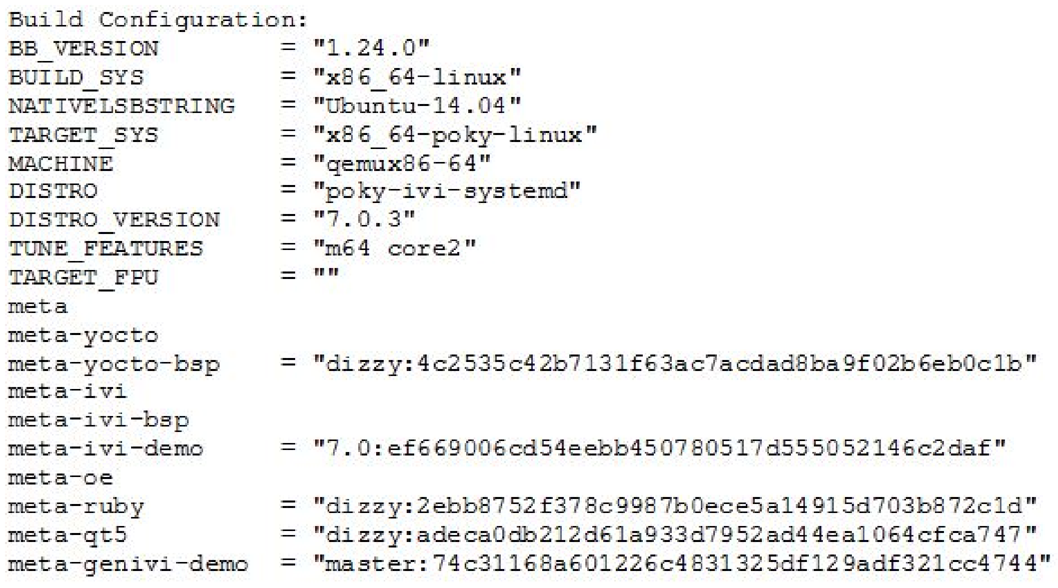
For more information, including a list of Quick Start ecosystem partners, visit freescale.com/iMXQuickStart

Document Number: IMX53RQKSTRTFS REV 3

Source: Freescale [64]

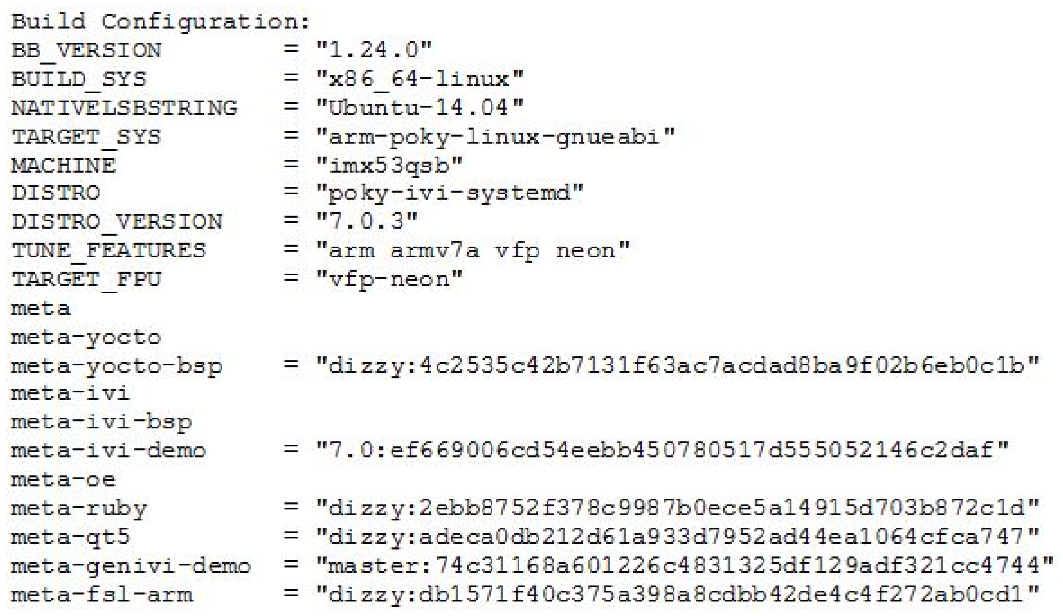
G

Build Configurations for the GENIVI Demo Platform



```
Build Configuration:
BB_VERSION        = "1.24.0"
BUILD_SYS         = "x86_64-linux"
NATIVELSBSTRING   = "Ubuntu-14.04"
TARGET_SYS        = "x86_64-poky-linux"
MACHINE           = "qemux86-64"
DISTRO            = "poky-ivi-systemd"
DISTRO_VERSION    = "7.0.3"
TUNE_FEATURES     = "m64 core2"
TARGET_FPU        = ""
meta
meta-yocto
meta-yocto-bsp    = "dizzy:4c2535c42b7131f63ac7acdad8ba9f02b6eb0c1b"
meta-ivi
meta-ivi-bsp
meta-ivi-demo     = "7.0:ef669006cd54eebb450780517d555052146c2daf"
meta-oe
meta-ruby         = "dizzy:2ebb8752f378c9987b0ece5a14915d703b872c1d"
meta-qt5          = "dizzy:adeca0db212d61a933d7952ad44ea1064cfca747"
meta-genivi-demo  = "master:74c31168a601226c4831325df129adf321cc4744"
```

Figure G.1: GDP for QEMU



```
Build Configuration:
BB_VERSION        = "1.24.0"
BUILD_SYS         = "x86_64-linux"
NATIVELSBSTRING   = "Ubuntu-14.04"
TARGET_SYS        = "arm-poky-linux-gnueabi"
MACHINE           = "imx53qsb"
DISTRO            = "poky-ivi-systemd"
DISTRO_VERSION    = "7.0.3"
TUNE_FEATURES     = "arm armv7a vfp neon"
TARGET_FPU        = "vfp-neon"
meta
meta-yocto
meta-yocto-bsp    = "dizzy:4c2535c42b7131f63ac7acdad8ba9f02b6eb0c1b"
meta-ivi
meta-ivi-bsp
meta-ivi-demo     = "7.0:ef669006cd54eebb450780517d555052146c2daf"
meta-oe
meta-ruby         = "dizzy:2ebb8752f378c9987b0ece5a14915d703b872c1d"
meta-qt5          = "dizzy:adeca0db212d61a933d7952ad44ea1064cfca747"
meta-genivi-demo  = "master:74c31168a601226c4831325df129adf321cc4744"
meta-fsl-arm      = "dizzy:db1571f40c375a398a8cdbb42de4c4f272ab0cd1"
```

Figure G.2: GDP for Freescale i.MX53 QSB