The Automotive Grade Linux

Software Defined Connected Car Architecture

20th June 2018

Final

Table of contents

1. Introduction

With the increasing momentum of electric, connected and self driving cars, the automotive industry is today experiencing a total revolution; consequently it is looking for new solutions to maintain the rapid pace of innovation that the market is demanding while keeping engineering costs under control. The implementation of the software defined vehicle, an autonomous connected automobile whose functions can be customized at run-time, demands an innovative software architecture that can easily scale and drastically reduce software time to market. Open source is certainly a way to create a fast-innovating ecosystem and to shorten software time to market. Automotive Grade Linux (AGL), a collaborative project of The Linux Foundation, aspires to do this by building a de-facto industry standard Linux-based open software platform for automotive applications. However, the complexity of a software defined vehicle and extreme level of configurability requires a system architecture which is flexible, scalable and configurable at run-time. Virtualization is the technology capable of offering this in a secure and efficient way, thanks to its ability to host the isolated execution of different environments concurrently in a single hardware system. For this reason, virtualization is seen as the main software defined vehicle enabler providing the key differentiating factor for Tier-1 and OEM software products.

With this document the AGL Virtualization Expert Group (EG-VIRT), a team of virtualization professionals active in the AGL community, presents the AGL virtualized software defined vehicle architecture. The objectives of this white paper are:

- **Disseminate automotive virtualization** inside and outside AGL
- **Identify virtualization use cases, requirements and solutions** for AGL
- Define the **AGL virtualized software defined vehicle architecture**

EG-VIRT desires to build, connect and combine together open source virtualization solutions around AGL to provide a modular virtualization infrastructure which boosts the creation of innovative advanced driver-assistance systems (ADAS), in-vehicle Infotainment (IVI) and telematics products. In the following pages, the AGL virtualization infrastructure vision is described along with its most important building blocks.

This white paper is organized as follows:

- Section 2 outlines virtualization benefits and challenges in automotive
- Section 3 presents its use cases and requirements
- Section 4 details the AGL approach towards virtualization
- Section 5 shows the AGL Virtualization architecture along with with business models considerations
- Section 6 lists the most interesting virtualization solutions for the AGL community
- Section 7 concludes the document.

What is virtualization?

Virtualization is a technique used to create multiple virtual execution environments by means of resource abstraction or partitioning. These virtual execution environments are then used to consolidate multiple applications on the same hardware in *server*, *desktop* and *embedded environments*.

In *server and desktop environments* the most common virtualization techniques are hypervisors and containers. Hypervisors rely on CPU hardware capabilities (e.g., AMD SVM, Arm Virtualization Extensions or Intel VMX) to create the abstraction of a virtual machine (VM) for the execution of kernel operating systems and applications. This results in minimal overhead and excellent isolation performance. In a server environment hypervisors are used in multi-tenant solutions to isolate workloads belonging to different customers. Containers are a software only mechanism to create execution environments. When running on a Linux host, all containers run on a single Linux kernel instance but in different namespaces for their storage, network, etc.. Containers enable high workload density and provide good deployment time performance, but being a software-only technology they do not provide the same level of isolation provided by hypervisors.

Embedded environments are, on the other hand, very different from servers and desktops due to requirements for power consumption and memory constraints. Therefore, embedded systems use size optimized hypervisors and system partitioners for most solutions. The former are small footprint hypervisors which minimize the number of their components to be more certification-friendly and to provide a smaller attack surface (higher security). System partitioners, on the other hand, limit their functionality to the partition of the system resources only, aiming at an even smaller footprint and at very low overhead (virtual environments are hardware partitions of the system, and therefore run directly on the hardware). More information about Hypervisors, system partitioners and containers can be found in Section 6 of this document. In addition, automotive, aviation, and other embedded systems have functional safety certification requirements that must be met before being deployed. Virtualization can help device manufacturers meet these function safety requirements by isolating the certified components into their own execution environment.

2. Virtualization in automotive

With hundreds of sensors, actuators, and Electronic Control Units (ECUs) that need to communicate together while ensuring the highest performance, safety and security, today’s automobiles are approaching the limits of their complexity. In the future the automotive industry will provide always connected vehicles running advanced self-driving functions and an increasing number of cutting edge applications. In this environment the in-vehicle hardware and software components grow exponentially with the number of applications, causing an explosion of the vehicle’s architectural complexity. Moreover, increased connectivity and applications lead to a larger attack surface that needs to be protected and defended.

To address the market requests and provide self-driving always connected vehicles, the automotive industry needs a hardware and software architecture that guarantees isolation, simplified systems management, high performance, open standards, interoperability, and flexibility. This brings a set of challenges that the automotive industry can achieve by using virtualization techniques as shown in the table below.

	Challenge	Benefit of virtualization
1	**Software Defined Autonomous Car.** New automotive systems functions and services need to adapt much quicker to new requirements users, manufacturers and legal authorities. Time to market is required to be much shorter, and functions and services life cycle needs to be similar to the one of smart mobile applications.	**Abstraction.** Virtualization abstracts the software from the underlying hardware, thus enabling the concept of Software Defined Car. Moreover, it reduces costs and time to market through portability and support for legacy solutions. **Flexibility and Interoperability.** Updates can be automated and performed remotely. Solutions based on different licenses, security levels and operating systems can be combined together.
2	**Costs.** Each time a new function is added, additional sensors, cables and ECUs are added to the vehicle. This increases space, weight, and power consumption as well as having an impact on vehicle cost (hardware, wire harness, maintenance, deployment, etc).	**Consolidation.** The number of ECUs and wiring complexity can be reduced by replacing them with virtualized instances in a single ECU. **Flexible architecture.** Deployment and maintenance can be automated and performed remotely resulting in simplified maintenance. However, software integration complexity depends on the type/number of shared resources between virtual machines. Care should be taken when deciding which resources are shared between virtual machines.
3	**Security.** Third party applications, advanced self driving and infotainment features as well as multiple (also remote) connectivity endpoints increase the security risk and the attack surface. In addition, more complex architectures result in a larger attack surface.	**Isolation.** Virtualized systems separate execution environments (CPU, memory, IO) to implement a multilevel security concept. Isolation guarantees that an application security flaw does not affect other applications running in the system. Finally, vulnerability lifespan can be reduced with remote updates.

4	**Mixed criticality.** Vehicles are embedding different functions with heterogeneous levels of safety. Some of them require Automotive Safety Integrity Level (ASIL) certification.	**Certification.** Virtualization techniques can have a very limited code footprint which eases certification. Concurrent execution of systems with different certification levels is possible; however, the chosen virtualization solution (as well as the underlying hardware platform) must comply with the most stringent certification level requested in the system.

Table 1: Virtualization challenge benefits table

Table 1 shows how virtualization addresses automotive challenges with a number of benefits that can be easily found in today's virtualization solutions. For this reason **virtualization** can be seen as **the software defined connected car enabler**.

3. Automotive virtualization requirements and functions

In the introduction of this document we defined virtualization, a technology which enables the hosting of different execution environments concurrently in a single hardware system. In this section, the workloads - hereinafter referred to as **automotive functions** - that can populate these execution environments and the requirements to create them will be discussed in more detail.

Virtualization provides the best performance in terms of security, isolation and overhead when supported directly by the hardware platform. Table 2 shows the hardware system requirements needed to enable virtualization.

	Electronic Control Unit (ECU) requirements to enable virtualization
	● **Hardware virtualization support** for CPU, Cache, Memory and interrupts to create execution environments (Arm Virtualization Extensions, Intel VT-x and AMD SVM, IOMMU, etc.) ● **Multicore processor** with possibility to allocate one or multiple cores to each execution environment ● **Trusted Computing Module** to isolate safety-security critical applications and assets (Arm TrustZone, Intel Trusted Execution Technology, etc.) ● Optionally, **IO virtualization** support for GPU and connectivity sharing

Table 2: System hardware requirements to enable virtualization

In fact, an ECU equipped with the hardware components listed in Table 2 can run multiple execution environments hosting different automotive functions concurrently, securely and with a good level of performance. Examples of such functions are:

- **Instrument Cluster** displays critical information (speed, signalling, etc.) and needs to be isolated from the rest of the system due to reuse of legacy software or regulations.
- **IVI systems** drive the central console with multimedia/radio functions, Heating, Ventilation and Air Conditioning (HVAC), navigation, rear-view camera, and in many cases, third-party applications. Isolation is used as a security measure against possible vulnerabilities brought by applications installed after market. IVI systems may share display, input, and audio interfaces with the Instrument Cluster.
- **Telematics** performs collection of telemetry data from vehicle, may also serve as OEM cloud connectivity gateway, and can support installation of edge services.
- **Safety critical functions** include ADAS functions such as parking/lane assistant, autonomous driving, digital mirror, etc. These are typically implemented on top of AUTOSAR compliant operating systems.

Thanks to the abstraction and isolation created by virtualization, it is possible to concurrently run heterogeneous types of virtualized automotive functions (e.g., different licenses, operating systems, legacy solutions, etc.). Moreover, virtualized automotive functions define a set of characteristics that the virtualization solution must address. Table 3 shows these requirements in detail.

	Automotive functions requirements for virtualized ECUs	
	Computing	● Static resource partitioning and flexible on-demand resource allocation (CPU, RAM, GPU and IO). ● Memory/IO bus bandwidth allocation and rebalancing.
	Peripherals sharing	● GPU and displays shall be shared between execution environments supporting both fixed (each one talks to its own display or to a specified area on a single display) and flexible configurations (shape, z-order, position and assignment of surfaces from different execution environments may change at run time). ● Inputs shall be routed to one or multiple execution environments depending on current mode, display configuration (for touchscreens), active application (for jog dials & buttons), etc. ● Audio shall be shared between execution environments. Sound complex mixing policies for multiple audio streams and routing of dynamic source/sink devices (BT profiles, USB speakers/ microphones, etc.) shall be supported. ● Network shall be shared between execution environments. Virtual networks with different security characteristics shall be supported

		(e.g., traffic filtering and security mechanisms). ● Storage shall support static or shared allocation, together with routing of dynamic storage devices (USB mass storage).
	Security	● Root of Trust and Secure boot shall be supported for all execution environments. ● Trusted Computing (discrete TPM, Arm TrustZone or similar) shall be available and configurable for all execution environments. ● Hardware isolation shall be supported (cache, interrupts, IOMMUs, firewalls, etc.).
	Performance and Power consumption	● Virtualization performance overhead shall be minimal: 1-2% on CPU/memory benchmarks, up to 5% on GPU benchmarks. ● Predictability shall be guaranteed. Minimal performance requirements shall be met in any condition (unexpected events, system overload, etc.). ● Execution environments fast boot: Less than 2 seconds for safety critical applications, less than 5 seconds for Instrument Cluster, and 10 seconds for IVI. Hibernate and Suspend to RAM shall be supported. ● Execution environments startup order shall be predictable. ● Advanced power management shall be implemented with flexible policies for each execution environment.
	Safety	● System monitoring shall be supported to attest and verify that the system is correctly running. ● Restart shall be possible for each execution environment in case of failure. ● Redundancy shall be supported for the highest level of fault tolerance with fallback solutions available to react in case of failure. ● Real time support shall be guaranteed together with predictive reaction time.

Table 3: Virtualization solution requirements to execute multiple automotive functions in a single multicore platform

4. AGL Virtualization Approach

AGL is building a Linux-based, open software platform to serve as the de-facto industry standard for automotive applications. The AGL virtualization approach follows the same philosophy and aims to provide a virtualization platform that can be used as it is or extended to consolidate different automotive functions in a single hardware platform.

We recognize virtualization as the next differentiating factor for automotive software, and we see such software as a set of interchangeable modules that need to be deployed together, communicate, interact. Any automotive player can combine such modules to provide a unique and customized solutions. AGL does not develop new hypervisors, but leverages on existing open source solutions considering them as modules of its architecture.

For this reason the key pillars of the AGL virtualization approach are:

- **Modularity**: hypervisors, virtual machines, AGL Profiles and automotive functions are seen by the AGL architecture as interchangeable modules that can be plugged in at compilation time (and where possible at runtime). The combination of the modules makes the difference. We can instantiate different modules. Modules can communicate with each other. To achieve modularity, interoperability will be required, especially between open and proprietary components.
- **Openness**: There is no restriction in the way the AGL virtualization platform can be used, deployed and extended. The AGL virtualization architecture supports multiple hypervisors, CPU architectures, software licenses and can be executed as a host and guest.
- **Mixed Criticality**: Applications with different level of criticality are targeted to coexist and run in a virtualized manner. As a consequence, AGL virtualization approach targets to consolidate applications different certification requirements.

The virtualization approach presented in this document and implemented as AGL virtualization platform is fully compliant with the current AGL plans and implementations, as well as it is orthogonal to the AGL application framework.

In fact, today the AGL application framework already supports applications isolation based on namespaces, cgroups and SMACK which relies on files/processes security attributes that are checked by the linux kernel each time an action processes and that work well combined with secure boot techniques. However, when multiple applications with different security and safety requirements (infotainment, instrument cluster, telematics, etc.) need to be executed in the system, the management of these security attributes becomes complex and there is a need of an additional level of isolation to properly isolate these applications from each other.

This is where the AGL virtualization platform comes into the picture, helping to enhance system security and to isolate different applications coming from the AGL community but also from third party developers.

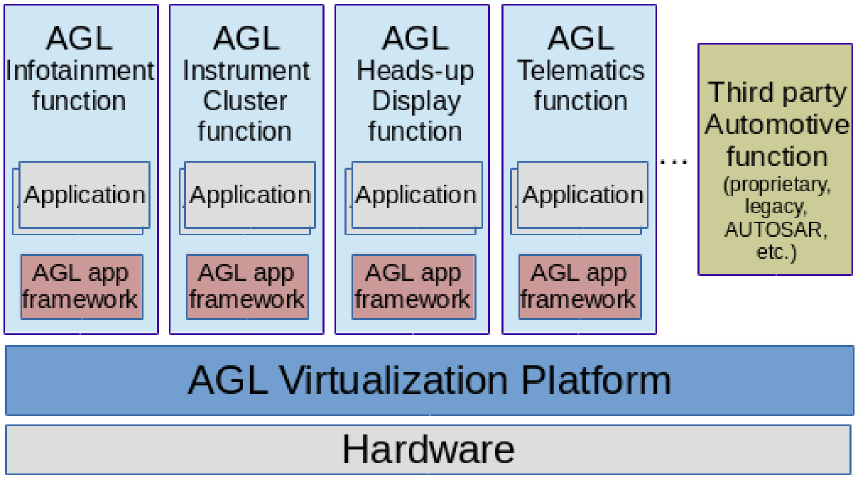
AGL Infotainment function
Application
AGL app framework
AGL Instrument Cluster function
Application
AGL app framework
AGL Heads-up Display function
Application
AGL app framework
AGL Telematics function
Application
AGL app framework
...
Third party Automotive function (proprietary, legacy, AUTOSAR, etc.)
AGL Virtualization Platform
Hardware

Illustration 1: AGL virtualization approach integrated in the AGL architecture

4.1 Business model considerations

Regulations on automotive products require compliance with safety standards such as ISO 26262 (“Road Vehicles - Functional Safety”). ISO 26262 defines four Automotive Safety Integrity Levels (ASIL): ASIL A (the least restrictive), ASIL B, ASIL C and ASIL D (the most restrictive). The entire system, its hardware and its software from the virtualization solution up to the applications, needs to be certified to be integrated in production vehicles.

Among the automotive functions presented in Section 3, IVI typically requires ASIL A or no certification, Instrument Cluster and Telematics typically require ASIL B and more advanced functions such as ADAS and digital mirrors require ASIL C or D.

ISO 26262 suggestions a classic systems engineering approach to software development and provides regulations and recommendations throughout the product development process from conceptual development through decommissioning. Automotive device makers must document and follow their ISO 26262 certified development process. This process provides checkpoints to ensure correctness of design through comprehensive verification and validation. During the development process artifacts are created at every stage to document product design and verification, track change history of work items, show full traceability of testing based on product requirements, provide proof of process control and report process compliance as needed to certification authorities.

However, open source development projects including Automotive Grade Linux prioritize the value of working software (i.e., “code first approach”) over that of comprehensive documentation. This raises the question: ”Can a safety-certifiable software element be developed using open source development practices?” There are many ways to answer this question but the short answer is no. This is not possible without making changes to the current AGL software development practice which is arguably what allows AGL to innovate quickly.

An argument can be made that taking an open source software element through safety certification after development is possible. A relatively small (<10K SLOC) software element could be snapshot (forked) and taken through an effort to reverse engineer requirements and validate the testing effort for certification. The result of such an effort would be a certified open source software element that would have to be managed as a new (forked) product. From this point, all changes going forward would have to be managed with the same rigor required for a safety certifiable software element. In process, this is equivalent to managing a commercial software element. The following questions would then arise:

- Who covers the costs of the required artifacts during design, development and testing?
- Who is the gatekeeper for changes after safety certification?
- Who covers the cost of long term support?
- Will the product be monetized? If so, what are the licensing terms?
- Who bears for the liability of the solution?

The answers to these questions could be addressed by an open source community/project with lots of coordination. Existing effort in this direction are today ongoing in the OSADL community.

Instead of focusing on a new business model and changing the rigor of the AGL open source project we will focus on detailing a comprehensive list of properties for the virtualization solution. It is expected that both open source and proprietary (closed source) software solutions will meet the requirements outlined in this white paper.

OSADL

Open Source Automation Development Lab (OSADL) is an open source community which hosts the Realtime and the Safety Critical Linux projects. The former is an initiative which aims to develop a realtime version of the linux kernel, while the latter targets to create procedures and documents that will lead to a facilitated safety certification Linux-based products.
More information can be found at https://www.osadl.org.

4.2 Interactions with proprietary solutions

In the short term, it is likely that several proprietary virtualization solutions will meet the requirements to support AGL specified use-cases. These proprietary solutions will typically have certification data packages that have been evaluated to assess compliance with the ISO 26262 standard. In the safety case, the virtualization solution is considered a safety element out of context (SEooC). A SEooC is a safety-related element which is not developed for a specific item. This means it is not developed in the context of a particular ECU or system. A SEooC certification data package will provide assumptions of use which must be considered when composing the system/ECU. In the case where AGL is used as a software element that is integrated with a previously certified virtualization solution, the integrator must ensure that the assumptions of use are followed during the system composition process.

5. AGL Virtualization architecture

The AGL virtualization architecture is composed by multiple modules that enable the execution of concurrent applications with different level of criticality (safety and security wise) on a single multicore hardware.

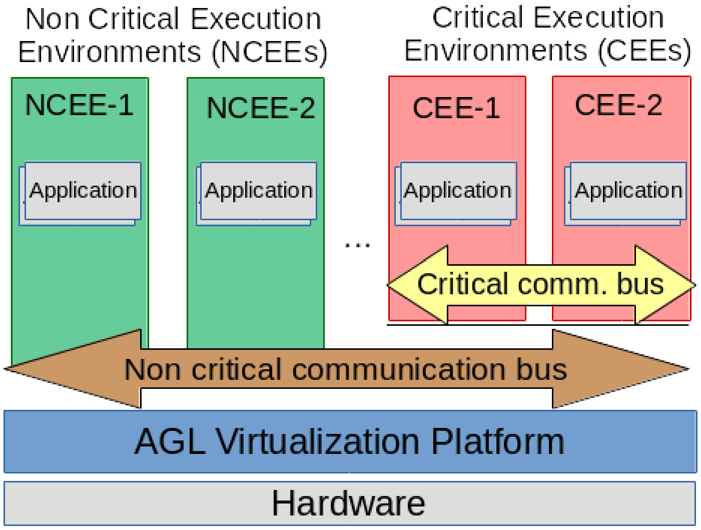
Non Critical Execution Environments (NCEEs)
Critical Execution Environments (CEEs)
NCEE-1
NCEE-2
CEE-1
CEE-2
Application
Application
Application
Application
...
Critical comm. bus
Non critical communication bus
AGL Virtualization Platform
Hardware

Illustration 2: AGL virtualized software defined connected vehicle architecture

The components of such architecture are:

- **Execution Environments (EEs)**: these are silos that run concurrently on the system and enable the execution of different applications. Different types of EE are supported: certified, non certified, trusted, non trusted, open source, proprietary. Some of them are classified as Critical Execution Environments (CEEs), because their applications have an impact of the safety and security of the system. The others are considered Non Critical Execution Environments (NCEEs). They could be implemented as binary applications, combination of a set of libraries with the related application, or full featured operating systems, etc.
- **Communication buses**: Consolidated EEs (and their applications) need to interact and communicate with each other. Two types of communication bus are supported by the architecture:
  - Critical communication bus: it is restricted to Critical Execution Environments (CEEs) only. The communication here has to address important requirements of safety and security.
  - Non critical communication bus: it is open to all the EEs available in the system. It has to address high performance and security requirements.

- **Virtualization Platform**: This module leverages on system software and hardware functions to create the silos for the execution of different EEs. Depending on the type and the combination of the functions used to build this module, the virtualization platform can be implemented using technologies like hypervisors, system partitioners, containers, etc. EEs can communicate with the Virtualization Platform (EE configuration, power management, etc) through specific channels.

Multiple implementations of the above components already exist both open source and commercial solutions. The role of AGL EG-VIRT is to act as technology integrator among these different components, enabling the coexistence of multiple virtualized environments together through different virtualization platforms. For this reason component communication (both horizontal among EEs, and vertical between the EEs and the Virtualization Platform) is one of the next work items for AGL and EG-VIRT. An open source implementation of the communication buses is of pivotal importance for the portability, interoperability, performance, security and safety of the system. This represents in fact one of the most interesting original contributions that EG-VIRT is planning in its near future activities.

6. Automotive virtualization solutions

The virtualization platform functionality can be implemented using different technologies which come with different trade-offs. Some of them are feature-rich and hard to certify, others have a more limited set of functions and for this reason can be more easily certified.

The different virtualization technologies available with both open source and proprietary solutions are hypervisors, system partitioners and containers. Illustration 3 highlights the architecture of each of them, considering as host kernel an operating systems which runs directly on top of the hardware and as guest kernel an operating systems which runs on top of an abstraction layer.

Hypervisors, system partitioners and containers can be combined together to address different requirements and to extend the set of features the system provides to its EEs.

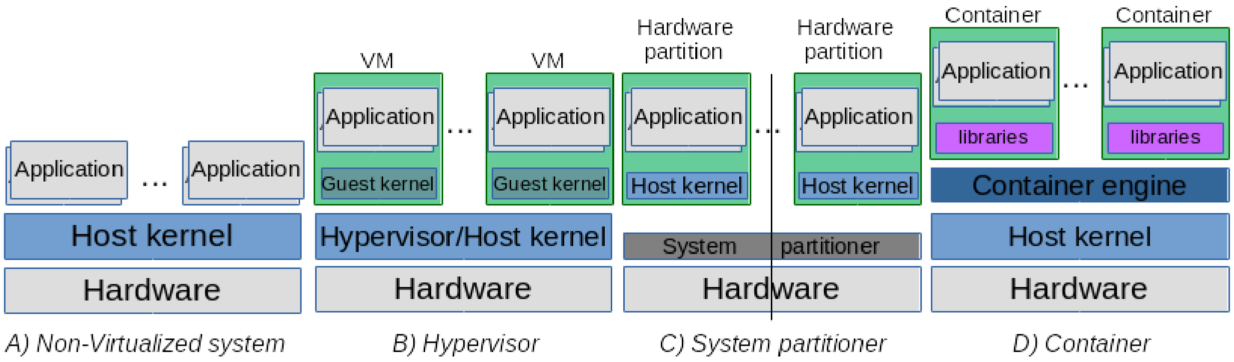

Illustration 3: Virtualization solutions architectures

In the next sections, different open source and proprietary virtualization solutions are detailed.

6.1 Hypervisors

Hypervisors (Illustration 3B) create a virtual hardware abstraction for automotive functions, operating systems and applications to use, leveraging on specific hardware features to create and isolate different execution environments. When run directly on top of the hardware, a hypervisor is defined as Type-1 or bare-metal, while it is considered a Type-2 solution when it is executed on top of an additional software layer. Both Arm and Intel processors are equipped with virtualization extensions and implement directly in hardware memory, timer, and interrupt virtualization. Different operating systems can run unmodified on top of the same hypervisor.

Hypervisors can provide advanced features such as guest to guest communication mechanisms, emulation of devices not present on the system, ability of over commit system resources (memory, CPUs), device sharing and a full set of functions and performance optimization which leverage on the system awareness of being virtualized (para-virtualization). Hypervisors can also provide the capability of statically allocating system resources to virtual machines.

Xen Project - GPL license

Xen is a versatile, general-purpose Type-1 hypervisor with mature community governance. It is developed as a Linux Foundation project and continues improving in response to use in public clouds, enterprise servers, middleboxes, desktops, vehicles and embedded devices. Xen can partition or pool plural resources while optionally virtualizing —securely sharing— singular hardware resources like coprocessors. Xen provides stable interfaces as applications and platforms evolve. For environments with real-time constraints, it can be configured as a partitioning hypervisor, eliminating scheduler overhead, reducing interrupt latency and delegating I/O and memory isolation to hardware with an IOMMU. Xen also provides static CPU assignment and multiple real-time schedulers (including an ARINC 653 scheduler) to further isolate resources and provide real-time guarantees. Xen is available in OpenEmbedded meta-virtualization, and its integration in AGL is planned in future EG-VIRT activities. The Xen community is currently working on implementing a hypervisor configuration compliant with ISO 26262 ASIL-B requirements. For more information, see https://wiki.xenproject.org/wiki/Category:OpenEmbedded.

Kernel-based Virtual Machine (KVM) - GPL license

KVM is a Type-2 hypervisor included in the Linux kernel and implemented as a kernel module. It exploits CPU Virtualization Extensions to execute guest's instructions directly on the host processor(s) and to provide virtual machines (VMs) with isolated execution environments. KVM borrows from the Linux kernel functions such as memory management and CPU scheduling and relies on external user space components to execute virtual machines. In fact, KVM doesn't offer itself machine or device models abstractions (bios, devices, etc.), but uses Quick Emulator (QEMU) for emulating guest hardware devices and instantiating guests. In the KVM paradigm guests are seen by the host as normal POSIX (Portable Operating System Interface for Unix) processes, with QEMU residing in the host userspace and utilizing KVM to take advantage of the hardware virtualization extensions. QEMU and KVM are able to run unmodified guests and support direct device assignment and static CPU allocation. KVM is today already supported in AGL for specific platforms and can be used by enabling the agl-egvirt feature at building time. More information can be found at https://www.linux-kvm.org/page/Main_Page.

L4Re Micro-Hypervisor – GPL license

L4Re is a light-weight, capability-based, real-time, open source operating system with support for virtualization (Type-1 hypervisor). Based on a microkernel architecture, the system is built from user-level components. The kernel is minimized by only providing essential functionality that foremost ensures spatial and temporal isolation among the components it executes, such as address spaces and inter-process communication. The user-level infrastructure provides a POSIX-like environment for running small and trustworthy applications on the L4Re kernel itself, so-called micro-apps, facilitating the construction of systems with small and application-specific trusted computing bases. Both hardware-assisted virtualization as well as paravirtualization employ functionality by the kernel to provide virtual machines on the system. So-called virtual machine monitors provide the necessary virtual platform for VMs to run. Those also offer a rich set of VirtIO functions to provide connectivity for VMs. L4Re supports the Arm, x86 and MIPS multi-core architectures in both 32-bit and 64-bit modes. Besides the open source version a commercial variant is also available. For more information, see https://l4re.org/.

ACRN - BSD license

The open source project ACRN defines a Type-1 hypervisor stack for running multiple software subsystems or domains. As a reference embedded hypervisor implementation, it is flexible and lightweight, and featured with real-time and safety-criticality capability. The ACRN hypervisor is composed of two primary components: the hypervisor and its device model. Like Xen, ACRN has a privileged service OS that has I/O device model, and the VM manager managing and controlling guest VM. It's geared towards IoT and embedded devices, especially for Automotive. For more information, see https://github.com/projectacrn/acrn-hypervisor.

6.2 System Partitioners

Such solutions do not create virtual hardware abstraction, but simply partition the system in different execution environments that can be used to run different automotive functions. System partitioners, shown in Illustration 3C, do not provide advanced virtualization features (e.g., over commitment, device emulation, etc.) and extensively leverage on hardware technologies to limit their footprint in terms of memory and lines of code. As a result, device sharing capabilities are very limited if not directly supported by the hardware. System partitioners can be used to host bare metal applications, operating systems or security workloads directly on the hardware without the need of any abstraction layer (as it is the case of hypervisors).

Jailhouse - GPL license

Jailhouse is a static partitioning hypervisor based on Linux. It runs as bare-metal on the system, i.e. it takes full control over the hardware and needs no external support. Its management interface is based on Linux infrastructure, so one should boot Linux first, then enable Jailhouse, configure it and finally split off parts of the system's resources and assign them to additional cells (Guest OS). The host system is called root cell, which initially controls the complete system resources; when a new cell is created, based on the configuration root cell relinquishes control over some of its CPU, devices and memory to the new cell. As the cells run in isolation, hardware support is required to restrict device accesses to the owner of the partition managing the device. This also makes the Jailhouse a secured option for automotive applications. Jailhouse focuses on two main things: being small and simple, and allowing cells to execute with nearly-zero latency. It supports real-time code run, including bare-metal applications and RTOSes. To know more, visit https://github.com/siemens/jailhouse.

Arm Trusted Firmware (ATF) – BSD license

Arm Trusted Firmware (ATF) is a secure software reference implementation based on Arm TrustZone, a set of security extensions which partitions the system in two execution environments, one only allocated to security and safety workloads (secure world), and the other for the rest (non secure world). It provides support for secure boot, trusted computing functions via the open source project Open Portable Trusted Execution Environment (OPTEE) and power management. More information can be found at :
https://github.com/ARM-software/arm-trusted-firmware.

6.3 Containers

Containers (depicted in Illustration 3D) create abstraction starting from the layers above the Linux kernel. Containers can not run full fledged operating systems but can be used to host Linux applications. No hardware isolation/security enforcement is guaranteed for Containers based EEs, and for this reason their use in AGL is not considered for safety and real time workloads. Using Containers within non-safety critical EEs is considered a good solution for application isolation.

6.4 Commercial solutions

Here below are listed commercial solutions developed by AGL members and part of the AGL ecosystem.

COQOS Hypervisor SDK

COQOS Hypervisor SDK is a next generation hypervisor, specially tailored to the needs of automotive applications. High efficiency and absolute functional reliability are achieved by a lean kernel and support for hardware virtualization. The system is flexible, economical, and functionally reliable due to the minimal trusted code-base. On top of the hypervisor, the SDK provides modular features corresponding to the needs of the customer.
(https://www.opensynergy.com/en/products/coqos-hypervisor).

Crucible

Crucible is a Xen-based separation & security hypervisor that provides hardware partitioning, workload isolation, real-time processing, data confidentiality, secure boot, and runtime integrity. Crucible enables security and isolation throughout the software stack, and supports multiple concurrent guest environments (Linux / Android, VxWorks, baremetal, etc). In addition to secure configuration & positive control of Intel and ARM-based systems, Crucible provides anti-reverse engineering protections and system accreditation / certification artifacts (starlab.io/crucible).

Green Hills Integrity® Multivisor™

INTEGRITY Multivisor is the virtualization service for the safety and security certified INTEGRITY RTOS. Since 2003, INTEGRITY Multivisor has been the industry's only safe and secure certified architecture for simultaneously running one or more guest operating systems alongside life and mission-critical functions on a wide range of multicore SoC's. INTEGRITY Multivisor, with its advanced ASIL D qualified MULTI IDE, enable rapid, optimized and cost effective complex system design without compromising safety, security or performance. (www.ghs.com).

Nautilus

Based on Open Source Xen hypervisor, Nautilus provides a solution accelerator that helps realize a converged digital cockpit for the modern day connected vehicle. It is highly configurable, supports multiple guest operating systems including AGL & Android and can be ported to automotive platforms from Intel, Renesas, Texas Instruments, Qualcomm, NXP, Mediatek & other commercially available Automotive SOC's. GlobalLogic also provides custom development services in the automotive domain around Nautilus (see more at Nautilus).

SYSGO PikeOS

PikeOS is an RTOS including a hypervisor based separation microkernel designed for the highest levels of safety and security. PikeOS technology has been certified on a wide range of projects using various certification standards including DO-178B/C, IEC 61508, EN 50128, IEC 62304 and ISO 26262. It combines a modular, flexible and future proof architecture with a large variety of certification standards. With this full European solution customers benefit in terms of reduction of cost, risk and full system certification lead times. (www.sysgo.com).

VOSYSmonitor

VOSYSmonitor is an automotive system partitioner based on the Arm TrustZone security extensions. It configures two physical partitions (hardware-isolated) in the system and runs a safety critical certified critical operating system (e.g., an RTOS) together with a non critical operating system (AGL, Linux, Android, etc.). VOSYSmonitor is certified ISO 26262 ASIL-C and provides a high level of customization, as it can run any type of operating system in both system partitions. (http://www.virtualopensystems.com/en/products/vosysmonitor).

7. Conclusion

This white paper is the result of the Automotive Grade Linux Virtualization Expert Group (EG-VIRT) activity, which aims to pave the way for open source virtualization in production cars. AGL considers virtualization as part of its architecture that is particularly important to enable multiple AGL profiles (IVI, Telematics, ADAS, etc.) and to implement the concept of software defined vehicle.

In this document, automotive virtualization has been presented with its benefits, challenges, requirements and use cases to foster the use of this technology on all next generation automotive vehicle architectures. Moreover, business model considerations have been presented, highlighting existing gaps between open source projects and functional safety certification. The Linux Foundation is working to fill these gaps and EG-VIRT will rely on the results of this work to build a certified and open source virtualization infrastructure. In addition, a non-exhaustive list of virtualization solution has been presented, detailing the trade-offs of each solution and suggesting the integration of multiple solutions to cover a wider range of requirements.

Finally, the most important contribution of this white paper is the definition of the AGL virtualized software defined vehicle architecture, which has been presented together with its components. This new architecture will serve as input for the future activities of AGL (e.g., Reference Hardware System Architecture, AGL Application and Security Frameworks, etc.) and of EG-VIRT. More in particular, in this document the role of EG-VIRT has been defined as virtualization technology integrator, identifying as key next contribution the development of a communication bus reference implementation for the interaction between Execution Environments and the Virtualization Platform. An open source implementation of this component is seen by EG-VIRT as an enabler of automotive virtualization portability, interoperability, performance, security and safety.

Future EG-VIRT activities will focus on this communication, on extending the AGL support for virtualization (both as a guest and as a host), as well as on IO devices virtualization (e.g., GPU).

8. Contributors

Michele Paolino - Virtual Open Systems
Walt Miner - The Linux Foundation
Daniel Bernal - Arm
Artem Mygaiev - EPAM
Tiejun Chen - VMware
Rich Persaud - OpenXT
Tero Antero Salminen - OpenSynergy
Adam Lackorzynski - Kernkonzept
Ciwan Gouma - SYSGO
Praveen Kumar - Sasaken
Alexander Damisch - Kevix

Jonathan Kline - Star Lab
Denys Balatsko - GlobalLogic
Dan Mender - Green Hills Software
Toni Hoang - Daimler