A. Weisbecker, M. Burmester & A. Schmidt (Hrsg.): Mensch und Computer 2015 Workshopband, Stuttgart: Oldenbourg Wissenschaftsverlag, 2015, S. 483-490.

Contradiction of Separation through Virtualization and Intercommunication

Tobias Holstein[1], Joachim Wietzke[2]

Department of Computer Science, University of Applied Sciences Darmstadt[1]
Faculty of Mechatronics, University of Applied Sciences Karlsruhe[2]

Abstract

A trend in automotive infotainment software is to create a separation of components based on different domains (e.g. Navigation, Radio, etc.). This intends to limit susceptibility to errors, simplify maintainability and to organize development based on domains. Multi-OS environments create another layer of separation through hardware/software virtualization. Using a hypervisor for virtualization allows the development of mixed critical systems. However, we see a contradiction in current architectures, which on one side aim to separate everything into virtual machines (VMs), while on the other side allow inter-VM-connectivity. In the end all applications are composited into one homogeneous UI and the previous intend of separation is unsatisfied.

In this paper we investigate current architectures for IVIS, i.e. mixed critical systems for automotive purposes, and show that regulations/requirements break previous intents of the architecture.

1 Introduction

Modern vehicles provide many features to the driver and passengers of a car. Multimedia, navigation, advanced driving assistance and car safety features are standard in every new car. Introducing more and more features raises the complexity of hardware and software in cars. A common car built in 2015 contains up to 110 interconnected electronic control units (ECUs), which operate through multiple hundred thousand lines of code (Ebert & Jones 2009).

Cluster instruments and in-vehicle infotainment systems (IVIS) have been separated onto different ECUs, because of safety critical and non-safety critical features/requirements. However, current trends like fully digital cluster instruments (FPKs) create new design possibilities and combine safety critical applications (e.g. digital speedometer) and non-safety critical applications (e.g. navigation) onto a single screen. Early approaches used hardware layering to composite information on hardware layer (e.g. by using LVDS) in order

to keep both ECUs separated. Though, there are also approaches in building mixed critical systems, where one ECU handles both types of software using virtualization for separation. The term *multi-OS* is further on used to describe a hypervisor based mixed critical system that includes a full IVIS.

Typical scenarios from the industry have been shown in recent automotive presentations. *Apple CarPlay*[1] and *Google's Android Auto*[2] provide examples on how Apps from mobile OSs are seamlessly integrated into IVIS. There, the mobile OS is remotely connected to the IVIS, which only serves as a presentation device, whereas all logic and functionality remains on the mobile phone. In the *multi-OS* approach the user does not have to provide a mobile phone, therefore it is possible to rely on the mobile OS, so that an application for Navigation can be used, whether or not a mobile phone is connected.

This approach is interesting for car manufacturers and original equipment manufacturers (OEMs), because it reduces the amount of ECUs and bus cabling, which at the same time saves weight, energy and space. This also enables high speed data transfer between two systems using e.g. a shared memory. In general it reduces the costs for expensive hardware components, such as the housing or power supply. Further on it enables the use of applications from other OSs, which therefore can reduce development costs and time.

Multi-OS environments use a type-one/baremetall hypervisor to run different OS types (e.g. real-time OSs (RTOS) or general purpose OSs (GPOS)) concurrently on a multi-core hardware. The hypervisor can provide communication channels between virtual machines (VMs), which i.a. allows applications from different VMs or even VMs itself (with mixed criticality) to communicate with each other. In the end applications or VMs are unified into one homogeneous UI onto one or more screens (e.g. through a compositor), while at the same time a clear separation through virtualization is supposed to be ensured.

However, the separation through virtualization seems to be contradictory. A homogeneous UI for the user, which contains access to functions from different VMs, will consequently require inter-VM-communication as soon as shared resources (i.e. input/output modalities) exist. Additionally safety regulations, ISO standards or legal requirements may enforce communication between all parts of the system. However, as soon as a communication between separated VMs exists, the previous separation is weakened, if not broken.

In this paper we investigate the *Multi-OS* architecture and current approaches in the automotive industry. We show the contradiction in some approaches and present a minimalistic example for a *Multi-OS*. The remainder of the paper is structured as follows: In section 2 current research and related work is described. In section 3 different *Multi-OS* environments are shown and discussed. At last a conclusion and future work is presented.

1 Apple CarPlay: https://www.apple.com/de/ios/carplay/

2 Android Auto: https://www.android.com/auto/

2 Related Work

Software, which uses virtualization mechanisms to run multiple operating systems on the same hardware platform, is often referred to as *hypervisor*. A hypervisor can be classified in two different types: Type one (or native, bare metal) hypervisors run directly on the host's hardware to assign the hardware components and to manage guest OSs. Type two (or hosted) hypervisors run as application and request resources of the host OS.

While (Rosenblum & Garfinkel 2005) explains the history of virtual machine manager (VMM) and shows example use case scenarios, (Heiser 2008) discusses the role of virtualization in embedded systems and states, that the strongest motivation for virtualization is security.

The display system *Blink* allows to safely multiplex complex graphical content from multiple untrusted virtual machines onto a single GPU (Hansen 2007). In Blink one VM is given full control of the GPU, while all other VMs use virtual GPUs. An API is required to be implemented in each application in order to access the virtual GPU.

A virtualization concept for mixed critically graphic ECUs and an implementation using a type one hypervisor (i.e. VMM) have been introduced by (Gansel et al. 2013). A bidirectional communication between VMs is based on an *Isolated Communication Channel*. This secure communication requires authentication through an *authentication manager* located in a dedicated VM called *Virtualization Manager*, which also manages shared resources. Further on, access control mechanisms (Gansel et al. 2014) to manage applications access to certain screen areas and to handle authorization, prioritization and hierarchies for applications, have been introduced.

Compositing GUIs from a partitioned IVIS is done by (Knirsch et al. 2013). Partitions, i.e. multiple OSs, run concurrently on a type one hypervisor. Applications running on those OSs are presented in a homogeneous GUI, which is provided by a component called *compositor*. Applications have to provide a GUI that fits into the overall UI concept. The implementation of the inter-VM-communication uses shared memory and an enhanced Wayland protocol.

2.1 Use Cases for Virtualization

There are several use cases which make the use of virtualization interesting for automotive embedded systems. In order to protect a safety relevant part of the overall system, all services and applications that have a high risk of being compromised are separated on VMs (Feske & Helmuth 2004; Rosenblum & Garfinkel 2005; Heiser 2008; Knirsch et al. 2013). The composition of OSs is used to create a new system. Different OSs are used to take advantage of applications, which are specific to a particular OS (Heiser 2008; Gansel et al. 2013). Limiting hardware resources for a VM in order to ensure that a particular VM can never use more resources (CPU, RAM, etc.) as previously defined (Heiser 2008). Dedicate hardware resources (e.g. GPU, CAN-Controller) to a particular VM in order to isolate them from other VMs (Rosenblum & Garfinkel 2005; Hansen 2007). Decoupling of development cycles in order to be able to update applications or the OS itself without having to verify the

rest of system (Knirsch et al. 2013; Agizim 2014). Interconnections between the different VMs in particular are not required in those cases.

2.2 Requirements/Regulations

When developing automotive software, there are regulations, guidelines and demands (e.g. car manufacturer, customer), which form requirements for the software. Interconnections between VMs are also caused by those requirements. Applied software architectures show, that interconnections are used to enable access to dedicated hardware resources in other domains/VMs (Gansel et al. 2013; Hansen 2007; ISO 2002). Identification/Authorization mechanisms between distinct systems need interconnections (ISO 2008; Gansel et al. 2013). Exchanging information about the current state of a system or accessing domain specific data sources (ISO 2002; Knirsch et al. 2013), input/output operations (Knirsch et al. 2013; Gansel et al. 2013) and the demand to provide flexibility for software development (Gansel et al. 2013) requires also interconnections.

3 Architectural Approaches

In this section different architectural approaches will be explained in order to show their differences and corresponding consequences, especially in regard of interconnectivity.

3.1 Clear Separation Approach

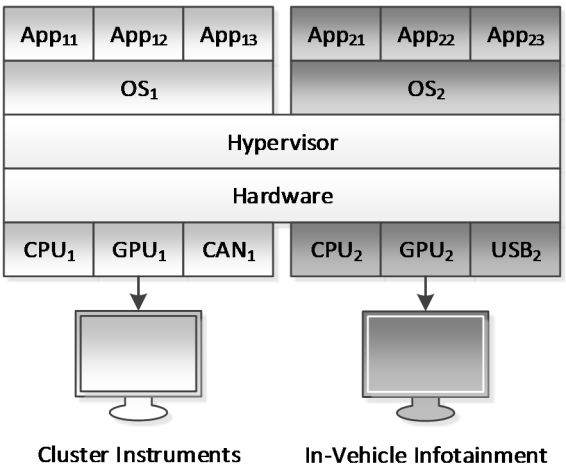

Figure 1: Clear Separation - Two or more OSs (VMs) run concurrently (without interconnections)

In a simple naive approach two or more OSs share a single hardware platform through a type-one hypervisor, as depicted in figure 1. Here, devices are dedicated to a certain OS. This approach assumes that resources do not have to be shared, because each OS has its own

input/output components. Therefore a communication between the different VMs is neither required nor necessary. This architecture can be used to save space for hardware, reduce hardware components and therefore costs. It also makes use of a multi-core CPU and RAM on a single hardware platform for multiple OSs. Despite that, it is similar to two separated devices (i.e. ECUs). Standard scenarios, where FPK and IVIS are clearly separated, will not require interconnections. An example for this architecture with two VMs: one VM provides all functions for the FPK and another VM provides all functions of the IVIS. FPK and IVIS have dedicated input/ouput devices, such as e.g. a touch-screen for the IVIS and a standard screen for FPK.

3.2 Layer of Interconnections

The layers, in which interconnections take place, vary based on the layer of composition (Holstein et al. 2015). The following figure shows an abstract representation of interconnections based on the literature research.

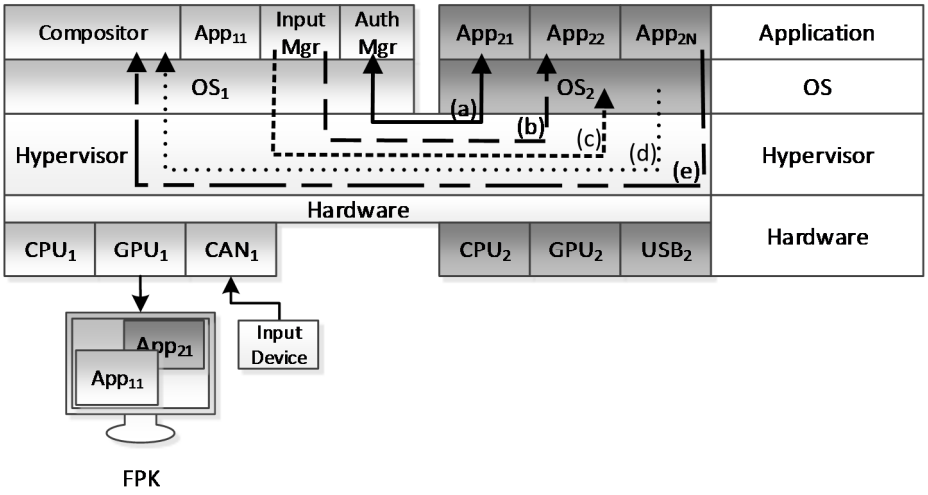

Figure 2: Layers of interconnection; App: Application; OS: Operating System; Input Mgr: Input Manager; Auth Mgr: Authentication Manager; CAN: Controller Area Network;

When resources are shared, interconnections are required. An example could be a touch-screen, which is used as a shared resource for FPK and IVIS. Both VMs provide graphical output, which have to be composited onto one screen. The touch-screen will provide input events to one of the VMs (dedicated device), which have to be dispatched to another VM in case both VMs expect those input events. Applications may be required to authenticate itself to get access to a certain part of a screen (Gansel et al. 2013; Gansel et al. 2014; ISO 2002). Whenever applications share one screen, a synchronisation, i.e. state exchange, between applications is required for visual transitions/composition. This is necessary e.g. to hide visual parts of an application to allow an application on another visual layer below to be

visible. In those cases interconnections are inevitable. This indicates, that connections between VMs can exist on different layers, i.e. a layer in one VM can be connected to another layer in another VM (fig. 2 (c,d)). Additionally, connections can be uni- or bidirectional (fig. 2 (a,b)). A connection on application layers requires both applications to be compatible (fig. 2 (a)), e.g. through a common defined protocol. Both applications have access lower layers in order to use inter-VM-communication channels. Input events are received through dedicated input devices. Those events need to be redirected to a certain application (fig. 2 (b)) or to a virtual device in OS layer (fig. 2 (c)). Special cases are application specific events, which cannot be handled by a generic input device of an OS.

Applications need a connection to a compositor or window manager in order to create, position or display windows (fig. 2 (e)). If the OS itself has an own window manager, which provides an output to a higher layer compositor, applications do not have to be adapted or changed to be used with the compositor (fig. 2 (d)).

Allowing an application in one VM to access functions of the hypervisor, e.g. to restart certain VMs, is another example for a connection on different layers.

A separation of two entities has to be broken, when one of the entities requires global information, i.e. information from/access to one or more other entities. This inevitably requires a connection between those entities.

3.3 Minimalistic Interconnections Approach

An approach, where communication between VMs is kept to a minimum, e.g. (Knirsch et al. 2013), is a mixture of the two previous introduced approaches. A scenario with two VMs: one with RTOS and another one with a GPOS. The RTOS contains a compositor component, critical applications (e.g. speedometer) and has exclusive access to the screen. The GPOS runs applications (e.g. Navigation, Browser, etc.), provides internet access and may provide possibilities to install 3rd party applications. This causes the GPOS to be vulnerable and open for attacks. In this approach we assume, that the GPOS is insecure, and may be infected by a virus, malware or other malicious software.

Therefore, all communication is limited to only provide necessary information, instead of using complex protocols. Interconnections are made between application layer and OS layer of the connected system, which abstracts the interconnection as a local device.

Data, which has to be exchanged with other VMs, is stored in specific data containers inside a shared memory partition owned by the VM, which provides the data. All other VMs have read-only access to it, which protects it against manipulation from other VMs. Therefore, all interconnections are only one way, based on “fire and forget” event systems.

The exchanged data is also kept to a minimum. VMs provide ready-to-use graphical output in form of pixel buffers inside the data container. This non-interpretable data format can be used inside a compositor to composite a homogeneous GUI. This assumes that all applications are implemented to fit into the overall GUI.

4 Discussion

In the previous section two different approaches were introduced. The clear separation approach is fulfilling the previous intent of separation. It also focuses safety and security concerns to the hypervisor and the critical system, which can be seen in current research, where e.g. design flaws in hardware architecture make attacks based on exploited hardware components possible (Schnarz et al. 2014). The second approach shows different interconnections between two VMs, which exist because of requirements and regulations, as mentioned in section 2.2. Those clearly break the previous intent of separation by allowing connections between applications and services from different safety critical and non-safety critical parts of the system. These two approaches represent two extreme cases: a completely separated multi-OS and a completely interconnected multi-OS.

Real world automotive scenarios will use a balanced mixture of those two approaches. The idea is to combine functionality from all VMs into one homogeneous UI, while at the same time keep the safety critical systems safe, i.e. isolated. However, the amount and level of interconnections depend on the actual requirements of the overall system. Especially the UI layer has usually many requirements regarding the usability and user interaction, which lead to interconnections.

The third approach suggests a way of minimizing interconnections between two or more VMs by using only certain interconnections, such as one-way/read-only data container and virtual input/output devices. This is supposed to reduce dependencies and to eliminate the use of complex protocols between different VMs. Nevertheless, this approach is still subject of research, which requires tests with real world scenarios to ensure that all defined requirements can be fulfilled.

5 Conclusion

In this paper an investigation into *multi-OS* architectures and current approaches in the automotive industry has been conducted. Approaches have been compared and it has been shown, that in order to fulfil certain requirements interconnections are inevitable. Further we showed, that interconnections can be categorized based on layers, which can be used to confine access to those layers. A suggestion for a minimalistic approach was shown, in which one way interconnections are used to provide data container to a compositor as well as to dispatch events to other VMs. This article presents work in progress that we plan to continue along the indicated lines.

Literaturverzeichnis

Agizim, A., 2014. IVI system sandboxing: The next frontier for in-vehicle upgrades.

Ebert, C. & Jones, C., 2009. Embedded Software Facts, Figures, and Future. *Computer*, 42(4), pp.42–52.

Feske, N. & Helmuth, C., 2004. *Overlay Window Management: User Interaction with Multiple Security Domains*, TU, Fak. Informatik.

Gansel, S. et al., 2014. An Access Control Concept for Novel Automotive HMI Systems. In *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies*. SACMAT '14. New York, NY, USA: ACM, pp. 17–28.

Gansel, S. et al., 2013. Towards Virtualization Concepts for Novel Automotive HMI Systems. In G. Schirner et al., eds. *Embedded Systems: Design, Analysis and Verification*. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, pp. 193–204.

Hansen, J.G., 2007. Blink: Advanced display multiplexing for virtualized applications. *Proceedings of NOSSDAV*.

Heiser, G., 2008. The Role of Virtualization in Embedded Systems. In *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems*. IIES '08. New York, NY, USA: ACM, pp. 11–16.

Holstein, T. et al., 2015. Current Challenges in Compositing Heterogeneous User Interfaces for Automotive Purposes. In M. Kurosu, ed. *Human-Computer Interaction: Interaction Technologies*. Springer International Publishing.

ISO, 2002. ISO 15005: Road vehicles - Ergonomic aspects of transport information and control systems - Dialogue management principles and compliance procedures.

ISO, 2008. ISO 15408-2: Information Technology - Security Techniques - Evaluation criteria for IT security - Part 2 - Security functional components.

Knirsch, A. et al., 2013. Compositing User Interfaces in Partitioned In-Vehicle Infotainment. In *Mensch {&} Computer 2013: Interaktive Vielfalt, Interdisziplin{ä}re Fachtagung, 8.-11. September 2013, Bremen, Germany - Workshopband*. pp. 63–70.

Rosenblum, M. & Garfinkel, T., 2005. Virtual Machine Monitors: Current Technology and Future Trends. *Computer*, 38(5), pp.39–47.

Schnarz, P., Wietzke, J. & Stengel, I., 2014. Towards Attacks on Restricted Memory Areas Through Co-processors in Embedded multi-OS Environments via Malicious Firmware Injection. In *Proceedings of the First Workshop on Cryptography and Security in Computing Systems*. CS2 '14. New York, NY, USA: ACM, pp. 25–30.