Contradiction of Separation through Virtualization and Inter Virtual Machine Communication in Automotive Scenarios

Tobias Holstein
School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden
tobias.holstein@mdh.se

Joachim Wietzke
Department of Mechanical Engineering and Mechatronics
University of Applied Sciences
Karlsruhe, Germany
joachim.wietzke@hs-karlsruhe.de

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 intent of separation is disregarded.
In this paper we investigate current architectures for in-vehicle infotainment systems (IVIS), i.e. mixed critical systems for automotive purposes, and show that regulations and/or requirements break the previous intents of the architecture.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*real-time systems and embedded systems, distributed systems*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

Keywords

Ubiquitous Interoperability, Heterogeneous Platforms, User Interface, Composition, Hypervisor, Virtualization

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 [4].

Cluster instruments and 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 Car Play* and *Google Auto* 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, which at the same time saves weight, energy and space. It also 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 and engineering costs and time.

Another trend in the automotive industry is to introduce more adaptive driver assistance systems (ADAS), which leads to fully autonomous driving vehicles. While a safety critical system is given full control of the car, the driver can use another part of the system to enjoy the IVIS. Merging those two parts into one homogeneous system enhances the requirements for a clear separation.

ECSAW '15, September 07 - 11, 2015, Dubrovnik/Cavtat, Croatia

ISBN 978-1-4503-3393-1/15/09...$15.00

DOI: http://dx.doi.org/10.1145/2797433.2797437

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 multicore 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, 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. Nevertheless, as soon as a communication between separated VMs exists, the previous separation is weakened, if not broken.

In this paper we investigate multi-OS architectures and current approaches in the automotive industry. We show the contradiction by comparing approaches and present a minimalistic approach 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.

2. RELATED WORK

Using multi-OS environments to separate domains have been covered in various publications, which often describe communication between domains in various ways.

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 (e.g. "Xen", "KVM" and "Hyper-V") run directly on the host's hardware to assign the hardware components and to manage guest OSs. Type two (or hosted) hypervisors (e.g. "VMWare", "Virtual Box" and "Virtual PC") run as application and request resources of the host OS.

While [15] explains the history of virtual machine manager (VMM) and shows example use case scenarios, [9] discusses the role of virtualization in embedded systems and states, that the strongest motivation for virtualization is security. The latter also mentions the contradiction: "By their very nature, embedded systems are highly integrated, all their subsystems need to cooperate in order to contribute to the overall function of the system. Isolating them from each other interferes with the functional requirements of the system".

In [8] a display system called "Blink" is presented, that allows to safely multiplex complex graphical content from multiple untrusted virtual machines onto a single GPU. 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. Therefore all applications are specific to the Blink architecture, which makes the use of Apps from popular App Stores difficult, or even impossible if the application is not compatible with the wrapper.

A virtualization concept for mixed critically graphic ECUs and an implementation using a type one hypervisor (i.e. VMM) is presented in [6]. A bidirectional communication between VMs is based on a "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.

In [7] an access control mechanism is presented, that allows to manage applications access to certain screen areas. The mechanism also handles authorization, prioritization and hierarchies for applications. Advanced protocols are used to handle communication between applications and VMM.

In [14] compositing of GUIs from a partitioned IVIS is shown. 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 controlled 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. The later relies on a socket connection to exchange events and request between client and compositor.

2.1 Use Cases for Virtualization

There are several use cases which make the use of virtualization interesting for automotive embedded systems. The following list shows the most popular reasons, based on our literature research:

- 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 [5, 15, 9, 14].
- The composition of OSs form an overall system. Using different OSs in order to take advantage of applications, which are specific to a particular OS [9, 6].
- 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 [9].
- Dedicate hardware resources (e.g. GPU, CAN-Controller) to a particular VM in order to isolate them from other VMs [15, 8].
- Decoupling of development cycles in order to be able to update applications or the OS itself without having to verify the rest of system [14, 1].

A communication between the different VMs in particular is not required to implement those use 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 they are used to

a) enable access to dedicated hardware resources in other domains/VMs ([6], [8], [11]),

b) provide identification mechanism between distinct systems ([12, 17.1.5.1], [6]),

c) provide information about the current state of the system, which also includes access to domain specific data sources ([11, 5.2.2.2.4], [14]),

d) handle input/output operations [14, 6] or

e) provide flexibility to or to simplifiy software development [6].

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

In a simple naive approach two or more OSs share a single hardware platform through a type-one hypervisor, as depicted in fig. 1.

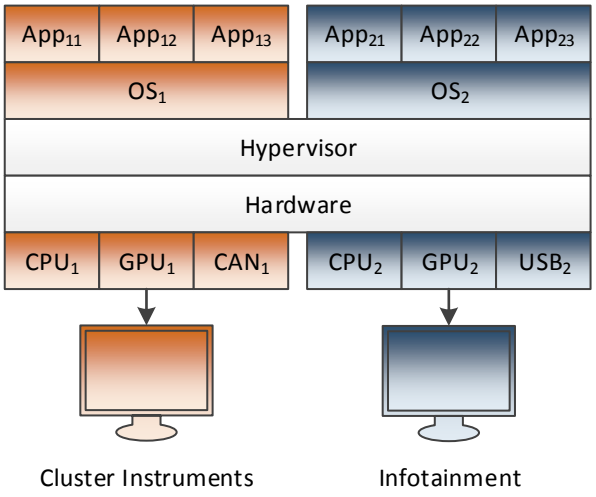

Figure 1: Clear Separation: Two or more OSs (VMs) run concurrently (without interconnections)

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.

Depending on the hardware architecture and the used hypervisor the overall system and encapsulation between VMs can be assumed to be secure. Though, design flaws in hardware architecture may make attacks based on exploited hardware components possible [16].

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 touchscreen for the IVIS and a standard screen for FPK.

However, this approach applies for automotive HMI concepts, where FPK and IVIS are physically separated parts in a car's dashboard. Newer scenarios composite FPK and IVIS functions into one screen [2]. A modern unified UI may provide access to and information from multiple hundreds of device functions [3]. Therefore it is impossible to provide context dependent soft keys for all functions in a cars dashboard. Decisions have to be made to decide which functions should be directly accessibly [13]. In the end a few controls provide access to all functions and, in case of an unified UI in a multi-OS, also to functions in different VMs.

3.2 Layers of Interconnections

The kind of interconnection between two layers varies as well as the layer in which compositions take place [10]. Fig. 2 shows an abstract representation of interconnections for two VMs.

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 might 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 [6, 7, 11]. Whenever applications share one screen, a synchronisation, i.e. state exchange, between applications might be required for visual transitions/composition. For example 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 know each other (fig. 2 (a)). A protocol must exist, which handles the communication. Usually both applications have access to inter-VM-communication channels, such as e.g. a shared memory or a virtual socket connection.

Input events may be received through bus systems (e.g. CAN) or 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)). The difference can be application specific events, which cannot be handled by a generic input device of an OS.

Applications may 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, the output can be used by the compositor. Therefore applications do not have to be adapted 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.

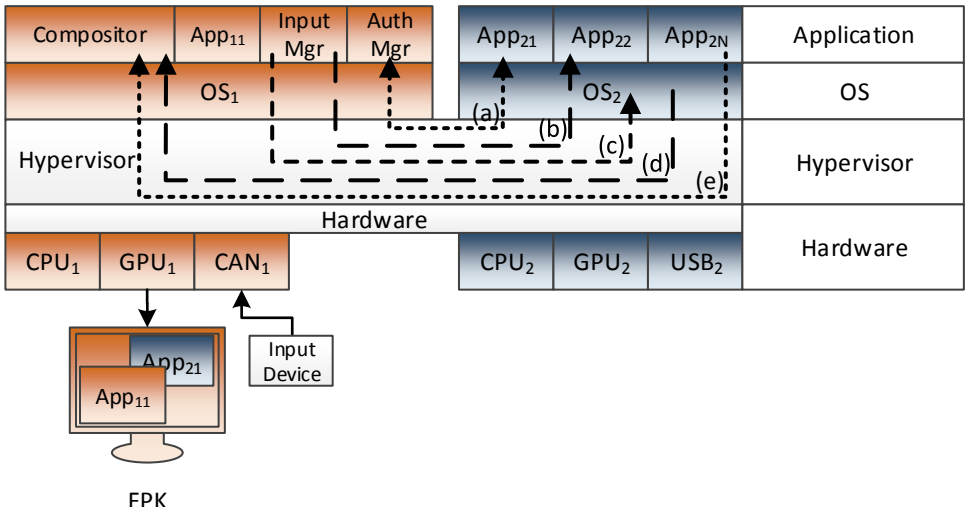

Figure 2: Layers of Interconnections
App: Application; OS: Operating System; Input Mgr: Input Manager; Auth Mgr: Authentication Manager; CAN: Controller Area Network;

3.3 Minimalistic Approach

An approach where communication between VM is kept to a minimum is depicted in fig. 3 and described further on. In this scenario the system has 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.), has internet access as well as 3rd party applications. The user can install/change applications on the GPOS.

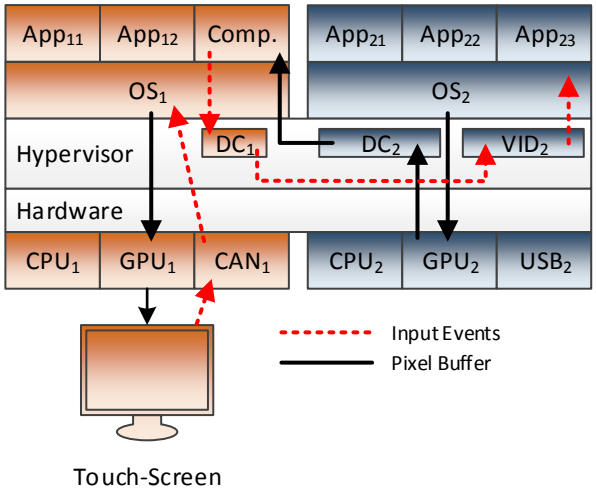

Figure 3: Minimalistic Approach
App: Application; Comp: Compositor; OS: Operating System; DC: Data Container; VID: Virtual Input Device; CAN: Controller Area Network;

Each VM has a shared memory partition containing a data container, which provides information/data to other VMs. Only the owner of a shared memory partition is allowed to write data to the data container. Additionally only a trusted OS (the RTOS) may access those data containers. Therefore it is protected against manipulation from other VMs. The communication is one way, based on "fire and forget" event systems.

While the RTOS has exclusive access to the screen, all other VMs provide pixel buffers only via data container. A pixel buffer can be created on application or OS level, and be used by the compositor to composite the overall GUI. VMs may or may not use hardware rendering through a dedicated GPU. Though, the final rendered output is copied into the data container. The important aspect here is, that only non-interpretable data is provided.

A GPOS does not know about other existing VMs. It receives its input events through virtual devices provided by the hypervisor. Touch events for example have to be prepared by the compositor to fit the local coordinate system of the GPOS before it is transfered via hypervisor to the virtual device of the GPOS.

This approach allows other VMs to be used inside a compositor, without using complex protocols. Although our approach requires only read-only and one-way communication, there might be requirements that cannot be fulfilled.

Possible Extensions.

In order to provide additional information to each VM, the data container can be extended to also contain state information, such as e.g. current applications running, requests to show notifications, etc. Also signing the data inside the data container using certificates through "trusted applications" is a possibility. However, as we assume that every GPOS can already be compromised, the data has to be checked through consistency checks, which ensures, that the RTOS is not compromised.

4. DISCUSSION

In the previous section three different approaches were introduced. The clear separation approach is fulfilling requirements for a hypervisor. The second approach shows interconnections between two VMs, which exist because of

requirements and regulations (as mentioned in 2.2). 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 use functionality from all VMs, while at the same time keep the safety critical systems isolated. However, the amount and level of interconnections depend on the actual requirements of the overall system. Especially the UI layer, where different heterogeneous UIs from different VMs are composited, has usually many requirements regarding the usability and user interaction.

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, such as the Wayland protocol between different VMs. Nevertheless, this approach has to be tested to ensure that actual real world 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.

6. FUTURE WORK

In order to solidify our research further investigations into the use of multi-OS environments have to be conducted.

7. REFERENCES

[1] A. Agizim. IVI system sandboxing: The next frontier for in-vehicle upgrades. online, Apr. 2014.

[2] Audi. Dialoge - Das Audi-Technologiemagazin. online, Jan. 2014.

[3] BMW. BMW Technology Guide: iDrive. online, 2014.

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

[5] N. Feske and C. Helmuth. *Overlay Window Management: User Interaction with Multiple Security Domains*. Technische Berichte. TU, Fak. Informatik, 2004.

[6] S. Gansel, S. Schnitzer, F. Dürr, K. Rothermel, and C. Maihöfer. Towards Virtualization Concepts for Novel Automotive HMI Systems. In G. Schirner, M. Götz, A. Rettberg, M. Zanella, and F. Rammig, editors, *Embedded Systems: Design, Analysis and Verification*, volume 403 of *IFIP Advances in Information and Communication Technology*, pages 193–204. Springer Berlin Heidelberg, 2013.

[7] S. Gansel, S. Schnitzer, A. Gilbeau-Hammoud, V. Friesen, F. Dürr, K. Rothermel, and C. Maihöfer. An Access Control Concept for Novel Automotive HMI Systems. In *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies*, SACMAT '14, pages 17–28, New York, NY, USA, 2014. ACM.

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

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

[10] T. Holstein, M. Wallmyr, R. Land, and J. Wietzke. Current Challenges in Compositing Heterogeneous User Interfaces for Automotive Purposes. In M. Kurosu, editor, *Human-Computer Interaction: Interaction Technologies*, chapter 49. Springer International Publishing, 1 edition, 2015.

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

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

[13] D. Kern and A. Schmidt. Design space for driver-based automotive user interfaces. In *Proceedings of the 1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pages 3–10. ACM, 2009.

[14] A. Knirsch, A. Theis, J. Wietzke, and R. Moore. 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*, pages 63–70, 2013.

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

[16] P. Schnarz, J. Wietzke, and I. Stengel. 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, pages 25–30, New York, NY, USA, 2014. ACM.