S. Boll, S. Maaß & R. Malaka (Hrsg.): Workshopband Mensch & Computer 2013
München: Oldenbourg Verlag, 2013, S. 63–70

Compositing User Interfaces in Partitioned In-Vehicle Infotainment

Andreas Knirsch[1,2], Andreas Theis[2], Joachim Wietzke[2], Ronald Moore[2]

Centre for Security, Communications and Network Research, Plymouth University, UK[1]
Faculty of Computer Science, Hochschule Darmstadt, DE[2]

Abstract

Automotive information and entertainment systems have become an integral part of a car's human-machine interface and already affect a prospective customer's purchase decision. In-Vehicle Infotainment systems combine an increasing number of software-based functionalities of varying importance and purpose on a shared hardware platform. This led to integrated modular architectures to achieve temporal isolation of different classes of applications, developed independently by multiple suppliers. Despite this partitioning on the software level, the user interface has to provide all functionality in a uniform way, blended into the manufacturer's superordinate usage concept. Furthermore, allocation and presentation of graphical content has to respect the car's operating state along with the user's preferences and system interaction. In the following, an approach is presented that enables the integration of segregated and independently rendered graphics into a uniform graphical user interface, while considering a multi-display environment and flexible allocation of different views. Relevant requirements, a prospective architecture, and a prototypical implementation are presented to foster the provisioning of the required computational and graphical power to enable future In-Vehicle Infotainment systems.

1 Introduction

Nowadays, 90 percent of the innovations within the automotive domain are attributed to electronics. Hence, software (SW) based functions became a notable success factor for automobile manufacturers. With a continuously increasing share of a car's SW attributed to In-Vehicle Infotainment (IVI) systems, their engineering and even more their qualities affect the economic success of the manufacturer. From a user perspective, the user interface (UI) is the actual point of contact with those systems. The design of such systems has to cover demands for an appealing front-end to foster a positive user experience (UX). Furthermore, the UI has to reflect the system's functional purpose with regard to the safety-critical environment. Nevertheless, the question how IVI systems can efficiently exploit their functionality and the capabilities of current hardware (HW) platforms is of primary importance.

In the past, IVI systems were rather isolated. Their main task was to provide information and entertain the car's occupants. Within the last decade, they have become an integral part of the in-vehicle system's network and enable the driver to configure and control automotive func-

tions. They are even about to merge with other systems, like the instrument cluster. This results invariably in new requirements as regards safety and security. Current developments make security particularly important: IVI systems feature the communication node between components attached to automotive fieldbus and infrastructure-based wireless communication networks. The interconnection with other systems within the vehicle as well as the environment enables new services and functionality, including future Advanced Driver Assistant Systems (ADAS) (Bolle 2011). This evolution expands the scope of application and leads to the adoption of the wider and more appropriate term In-Car Multimedia (ICM) systems, which is used in the following.

Moreover, the interconnectivity allows future ICM systems to update both data and functionality dynamically during operation. This includes, for example, geographical maps, traffic information and applications (or “apps” as provided for consumer electronics (CE) using “app-stores/-markets"). The evolution from static functional extends to ICM devices that can be updated and enriched regularly or on-demand is a major change in the automotive domain. With respect to the usual system lifecycle of several years, the UX can be efficiently maintained through the entire vehicle lifetime. Hence, this evolution has significant impact on the user interaction. In current systems the user is able basically to configure functionality delivered with the vehicle. Future ICM systems will provide capabilities to adapt the functional extent to personal needs or favors. This creates a new dimension of customization.

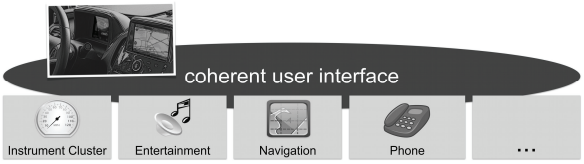

Figure 1: Partitioning Infrastructure

However, a dynamic update of functionality contains risks relevant for safety. Hence, the dynamic functionality may only include non-critical applications. Nevertheless, non-reliable functionalities within such “after-market apps” potentially (a) distract the driver or (b) interfere with critical applications due to the use of shared HW resources. The latter is caused by, for example, various independent SW suppliers causing non-functional incompatibilities, or a non-uniform and unpredictable set of applications due to the users' decisions about what to install and run. Whereas (a) can be covered by a thorough application-specific quality assurance, the risks related to (b) have to be mitigated by an appropriate infrastructure of the ICM system. Such an infrastructure has to segregate functionalities of different criticality, or those that are trusted and those that are not trusted by the vendor of the ICM system. Approaches for segregating functionalities by time and space partitioning via virtualization, the use of different operating systems (OS), and exploiting the capabilities of real-time scheduling to define execution domains (ED), while maintaining efficient intercommunication facilities within the context of ICM systems are presented in (Vergata et al. 2011). Figure 1 depicts exemplary an architectural overview of such a partitioned system.

The segregated computation mitigates risks regarding negative interferences between different applications and error propagation due to an infrastructure based encapsulation. Still, the SW system shares a common HW platform, including shared resources. The allocation and arbitration of such shared resources potentially cause temporal interference as well. For re-

sources that allow only an exclusive usage at any given time, a priority-based arbiter may lead to more predictable system behavior, as discussed in (Knirsch et al. 2012). Time slicing is not appropriate for shared resources that facilitate multiple accessing applications at the same time. This applies especially for data sinks that allow the blending of data streams, such as video and audio. However, these types of data are significant for building an appealing UI. Hence, these have to be considered for establishing a comprehensive infrastructure relying on the segregation of functionality while improving the UX.

Based on the independent development of the SW components, the UIs are built independently from the core functionalities. Nevertheless, they have to comply with design specifications predefined by the car manufacturer or original equipment manufacturer (OEM) to implement a homogeneous "look and feel" and user interaction. With a rising number of applications and after-market "apps", a comprehensive UI component covering all visual presentation and user event handling is no longer feasible. Each application has to provide its own UI to be integrated with – or blended into – the existing ones. This creates a demand for a graphics compositing instance as a segregated component that can cope with multiple graphics sources and the related user interaction (the "back channel") for user presentation and event handling, respectively. Such an instance may act as manager and define what to visualize, where, and in which presentation mode, whereas the graphics sources are segregated SW components. A conceptual architecture for such an instance is presented in the following. The goal is to pave an integration path for independently developed components while enforcing individual run-time polices.

2 Related Work

The partitioning of SW within vehicles is not a new concept. The Automotive Open System Architecture (AUTOSAR) (Bunzel 2011) is a standardized architecture, development approach and application programming interface (API). It fosters an independent development using well-defined interfaces to enable integration onto shared HW platforms. Therefore, abstraction layers help to decouple SW from HW specifics that makes AUTOSAR appear as underlying platform to SW components. Although the target is very similar to the earlier described segregation, it does not detail compositing of graphics to a shared rendering device. Nevertheless, the concept discussed in the following might be transferred to an AUTOSAR conform ICM system using the provided API of AUTOSAR.

Open vehicular SW platforms appeared more recently. Similar to AUTOSAR, they are intended to create abstraction layers that provide access to HW resources and offer domain specific SW services. They aim for reduction of application complexity while fostering parallel execution and reuse of SW components. However, "open" means the platform specification is freely available, which enables everyone to develop platform compatible SW components. Prominent open automotive platforms for ICM are AutoLinQ™, GENIVI and Ford SYNC® (Holle et al. 2011). A side effect of the open platform trend is the introduction of Linux based OSs into the vehicle, which is also applied for evaluation of the herein proposed compositing architecture. Despite the fact no specific platform is addressed by the latter, it may constitute a beneficial enhancement to them to enable independent UIs. GENIVI's "IVI

Layer Management" project addresses compositing and separation of HMI and layer management, but does not yet cover efficient inter-ED UI provisioning (GENIVI 2013).

QNX Software Systems propose their QNX CAR HTML5-based HMI framework to ease integration of applications from CE space (Gryc & Lapierre 2012). A compositing of different UI components might be realized by use of different in-vehicle provisioned web-services, each offering a particular functionality. A "browser" acts as a central compositor. This positively affects the development process through the use of web techniques and may ease the transfer of a predefined design to a working UI. Different service providers could be segregated into dedicated partitions, with the freedom to utilize different OSs. However, the major part of the UI's content has to be rendered within the partition of the "browser". Hence, there is more computational power required for the compositor. Furthermore, a certain service provider may interfere with a more critical one due to the need for interpretation and computation of the content to visualize, which undermines the concept of partitioning. Therefore, it does not provide an adequate solution, although within layered system architectures HTML5 might be applicable as long as the rendering is performed within a segregated partition.

There already exist various graphics compositing window managers for different OSs and providing different features. The latter address, for example, improved accessibility, simplified use and so called "eye candy" to enhance UX. One of the more recent developments is Wayland (Høgsberg 2012), which focuses on a lightweight and efficient internal communication and, therefore, is also applicable to resource-constrained embedded systems. Wayland is also incorporated into the "IVI Layer Management" of GENIVI. Unfortunately, Wayland does not natively facilitate an inter-partition communication. Nevertheless, it is used for the evaluation of the herein proposed concept. Therefore, fundamental communication components were substituted or enhanced.

3 Architectural Drivers for ICM Compositing

The following constraints lay the foundation for the architecture of an ICM Compositor.

The system's functionality is partitioned into segregated execution domains (EDs) to ensure local run-time policies. These include predefined temporal behavior derived from given priority policies and priority levels. The intention is to prevent effectively negative interference between different functionalities – or applications – deployed to different EDs. The partitions may be implemented using virtualization techniques or other, lighter weight encapsulation techniques. Both options do benefit from and are enabled by multicore HW architectures. This implies that the different partitions do not share a common OS (or kernel space), meaning that the options for inter process communication (IPC) are limited. However, efficient communication is necessary to utilize and benefit from the common HW infrastructure.

With respect to the varying safety relevance of different applications, the intercommunication between EDs has to meet certain security related requirements. It has to be ensured that a dynamically installed or updated application cannot cause an error within a safety relevant ED (e.g. instrument cluster) or the compositing ED.

Nowadays, appealing UIs often rely on 3D graphic effects. Therefore, a graphical processing unit (GPU) is usually used as an accelerator to relieve the general purpose CPU. By partitioning the system into several EDs that independently render graphics, a single GPU has to

be shared between multiple OSs, introducing a significant bottleneck. Alternatively, only one ED benefits from the GPU, while the other ED render their graphics without acceleration. Both options are unsatisfactory. For efficient compositing the architecture may employ several GPUs as accelerators for different EDs.

In summary, the integrated modular architecture applied to highly interactive ICM systems requires partitioning. This is caused through the functionalities' different safety relevance and hence, the need to prevent negative interference. Demands for uniform and compelling UIs create requirements for efficient graphic processing. Using dedicated CPU cores for segregated computation is no solution as long as more than one ED relies on graphic acceleration. Thus, we propose the utilization of multiple GPUs to maintain consistently the segregated architecture also for graphics processing. This means decreasing computational load for the CPUs related to graphics processing and hence more effective utilization of HW capabilities. Consequently, negative inter-ED interference is mitigated and additional graphics acceleration for future highly interactive ICM UIs is made available.

4 ICM Compositing Infrastructure

The design is derived from the architectural drivers defined above. Basically, it consists of three conceptual components as detailed in the following and depicted in Figure 2:

UI application (UI-APP): an independent functionality providing an UI artifact. Such an artifact (or surface) may implement comprehensive and extensive menu structures providing access to a set of applications, or only a section of a certain UI screen that has to be blended with other UI-APP's artifacts. This means each application renders its own subset of the UI. UI-APPs are distributed to different EDs, whereas each ED may benefit from a dedicated GPU. The combination of all UI-APPs forms the UI of the ICM system, which means that they represent the source for graphics and the sink for related user events. However, all UI-APPs must comply with the design concept of the overall system.

Compositor: a super-ordinated instance that blends the artifacts provided by different UI-APPs. Therefore, it may, for example, resize (delegated to UI-APP), transpose, and project the graphics with perspective. These artifacts can be regarded as active video streams. This is comparable with applying a texture to a 3D model, whereas here the texture is not an image but an animated and active UI artifact. Active means the UI artifact is still receiving user events (e.g. touch-events). The manipulation of the provided artifact issues demands for a dedicated accelerating GPU for the compositor. Furthermore, the graphical artifacts are received as plain pixel buffers. This obviates the need to interpret information and hence mitigate security issues such as code injection. However, it also means additional efforts, for instance, to changing perspective of displayed artifacts. Nevertheless, using plain pixel buffers has great advantages in terms of loose coupling, maintaining a high degree of freedom for the UI-APPs, but still ensuring compatibility. Additionally, the compositor delegates incoming user events to the related UI-APP, comparable with an input-event mapper. An ICM system employs a single compositing instance, communicating with all UI-APPs. Hence, the compositor is the single instance that is aware of what artifacts are actually displayed to the user. Therefore, it may also map generic input events (e.g. buttons on a multifunction steer-

ing wheel) that are related to the current system context to the corresponding UI-APP or respective ED. This also applies for input preprocessed by speech- or gesture-recognition.

Intercommunication: the facilitator of the compositing infrastructure is an efficient communication. Ideally, by use of a shared memory region that is accessible by both UI-APPs and the compositor. Basically, this is needed to transfer pixel buffer information from UI-APPs to the compositor with adequate throughput to achieve predefined frame rates. The intercommunication also transfers user events from the compositor to UI-APPs, which requires low latency to provide appropriate responsiveness.

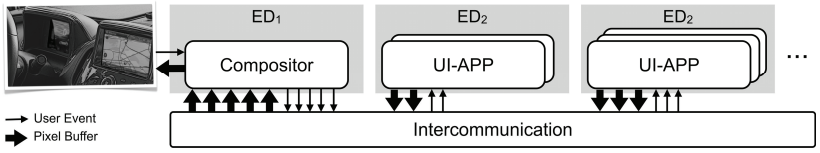

Figure 2: ICM UI Infrastructure

5 Evaluation

A prototype implementation has been built to facilitate evaluation of the proposed design and demonstrate its feasibility. It is not extensive and does not yet provide all functional capabilities of a real-world ICM system. However, the previously defined architectural drivers are covered to address essential features from an architectural viewpoint.

The prototype implementation has to feature at least n EDs, with $n \geq 2$. ED_1 contain a compositor that is blending the independently rendered graphics for visualization on a display. ED_1 is connected to a dedicated GPU to support the modification of artifacts. $ED_{2..n}$ contain UI-APPs rendering 3D graphics, also using dedicated GPUs for graphics acceleration. All EDs run different instances of an OS and have access to a shared memory. This constitutes the minimum criteria to verify the applicability of the above described approach. In the following the actual implementation and its constraints of the prototype are outlined.

The partitioning in the prototype relies on virtualization, where each ED is encapsulated within a dedicated virtual machine (VM). All virtual machines are connected to a shared memory region to prepare the prerequisite for the intercommunication component. This is realized using KVM as virtual machine monitor in conjunction with a virtual inter-VM shared memory PCI device (Kivity et al. 2007, Macdonell 2011). The platform of the host system provides several GPUs passed through to respective VMs for dedicated acceleration. Currently, a GNU/Linux based OS is utilized for the compositor's ED and UI-APPs' EDs. The prototype also supports Android OS based EDs acting as UI-APP to demonstrate the blending of graphical artifacts which are rendered by different OSs. The intercommunication is realized by using Wayland with enhancements to utilize inter-VM shared memory. The implementation of both the compositor and UI-APPs is based on Weston. Within Android the system service for rendering the UI is modified to clone and route surfaces to the compositor using the intercommunication component of the proposed design (Theis 2013). The surfaces are routed without changing the Android applications.

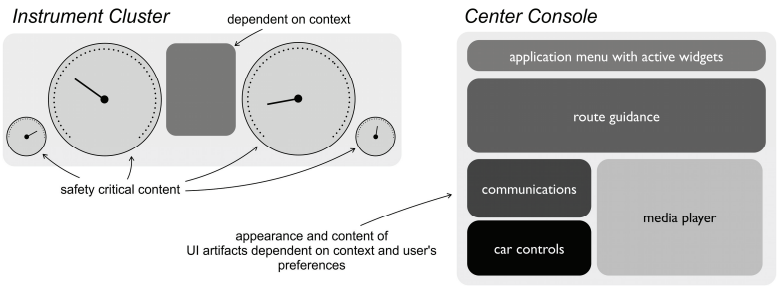

Figure 3: Exemplary ICM UI with different UI-APPs

Figure 3 depicts a prototype UI layout that relies on various UI artifacts rendered by different EDs. The selection and appearance of the content within the center console is adaptable, whereas the instrument cluster must comply with regulations and laws. All UI artifacts are fully active and may be transposed in size and perspective by the compositor instance independent of the UI-APP. Certain content is additionally displayed on the instrument cluster, dependent on the vehicle's or applications' context or user interaction.

The prototype practically demonstrates how a compositor along with graphic acceleration could enable modular UIs without breaching partitioning concepts.

6 Conclusion and Outlook

Appealing UIs are important features of future ICM systems. In parallel, the increasing extent of functionality integrated into such systems creates new challenges. Functionality varies in criticality in terms of safety. This leads to time/space separated SW architectures to enable strong enforcement of run-time polices. Such a partitioned architecture counteracts the realization of a comprehensive, coherent, and compelling UI, which has to appear as an ensemble of one piece. This is amplified as long as only one graphic accelerator is available that has to be shared by applications executed in parallel on multiple CPU cores. The architectural design approach presented addresses this issue and provides an integration path for individually developed SW components of different criticality. Relevant architectural drivers are discussed and the essential design components are illustrated. A prototype supports the evaluation of the design by use of a functional proof-of-concept.

Furthermore, research is planned to incorporate partitions that employ a real-time OS and a more thorough quantitative evaluation of different opportunities to accelerate 3D graphics processing. Additionally, the proposed concept does not yet cover the blending of audio sources, which is necessary for a comprehensive UI compositor.

References

Bolle, M. (2011). Connected Vehicle: i2Car or Car2i? In *carIT-Kongress – Mobilität 3.0*. automotiveIT, Media-Manufaktur.

Bunzel, S. (2011). AUTOSAR – the Standardized Software Architecture. *Informatik-Spektrum*. vol. 34. Springer. pp. 79–83.

GENIVI (2013). GENIVI Open Source Projects: IVI Layer Management. Accessed 01 July 2013 <http://projects.genivi.org/ivi-layer-management>

Gryc, A. & Lapierre, M. (2012). Warum HTML5 die HMI-Technologie der Zukunft ist. Whitepaper. Ottawa: QNX Software Systems.

Høgsberg, K. (2012). Wayland – A new graphics architecture. In *Free and Open source Software Developers' European Meeting (FOSDEM)*. Brussels.

Holle, J., Groll, A., Ruland, C., Cankaya, H. & Wolf, M. (2011). Open Platforms on the Way to Automotive Practice. In *8th ITS European Congress*, Lyon.

Kivity, A., Kamay, Y., Laor, D., Lublin, U. & Liguori, A. (2007). kvm: the Linux Virtual Machine Monitor. In *Proceedings of the Linux Symposium*, vol. 1. pp. 225–230.

Knirsch, A., Schnarz, P. & Wietzke J. (2012). Prioritized Access Arbitration to Shared Resources on Integrated Software Systems in Multicore Environments. In *3rd IEEE International Conference on Networked Embedded Systems for Every Application (NESEA)*. IEEE Computer Society. pp. 1-8.

Macdonell, A. C. (2011). *Shared-Memory Optimizations for Virtual Machines*. Ph.D. dissertation, Department of Computing Science, University of Alberta.

Theis, A. (2013). *User-Interfaces einzelner Android-Apps auf einem entfernten Wayland-Compositor*. MSc Thesis. FB Informatik, Hochschule Darmstadt.

Vergata, S., Knirsch, A. & Wietzke J. (2011). Integration zukünftiger In-Car-Multimediasysteme unter Verwendung von Virtualisierung und Multi-Core-Plattformen. In *Echtzeit*, Springer.

Contact

Andreas Knirsch, E-Mail: andreas.knirsch@plymouth.ac.uk