Towards Virtualization Concepts for Novel Automotive HMI Systems

Simon Gansel[1], Stephan Schnitzer[2], Frank Dürr[2], Kurt Rothermel[2], and Christian Maihöfer[1]

[1] System Architecture and Platforms Department
Daimler AG, Böblingen, Germany
firstname.lastname <at> daimler.com
[2] Institute of Parallel and Distributed Systems
University of Stuttgart, Germany
lastname <at> ipvs.uni-stuttgart.de

Abstract. Many innovations in the automotive industry are based on electronics and software, which has led to a steady increase of electronic control units (ECU) in cars. This brought up serious scalability and complexity issues in terms of cost, installation space, and energy consumption. In order to tackle these problems, there is a strong interest to consolidate ECUs using virtualization technologies. However, current efforts largely neglect legal constraints and certification issues and the resulting technical requirements.
In this paper, we focus on the consolidation of graphics hardware through virtualization, which received a lot of interest in the car industry due to the growing relevance of HMI systems such as head unit and instrument cluster in modern cars. First, we investigate relevant ISO standards and legal requirements and derive seven technical requirements for a virtualized automotive HMI system. Based on these requirements, we present the concept for a *Virtualized Automotive Graphics System* (*VAGS*) that allows for the consolidation of mixed-criticality graphics ECUs.

1 Introduction

Over the years, the automotive industry, which was mainly driven by hardware and mechanics in the past, has changed to an industry where about 90 % of all innovations are driven by electronics and software [3]. In current high-end cars, the IT hardware and software architecture is represented by more than 70 physically separated *electronic control units* (ECU), which are partitioned into different domains and connected via a network of different communication bus systems. To deploy new functionalities, the OEMs often add further ECUs to the vehicle. This trend of "new function, new ECU" has lead to serious scalability issues in terms of cost, installation space, and energy consumption. In order to deal with these problems and to stop this trend of adding further ECUs, there is a strong interest in the car industry to consolidate ECUs using virtualization technologies to share the same hardware between different components.

In this paper, we focus on the consolidation of graphics hardware since it is of high relevance in modern cars. An increasing number of automotive functionalities and applications require highly sophisticated graphical representations in 2D or 3D based on hardware acceleration. For instance, the Head Unit (HU) uses displays integrated into the backside of the front seats and center console to display multimedia content; and displays connected to the Instrument Cluster (IC) show car specific information like current vehicle speed or warnings. Therefore, HU and IC are good candidates for hardware consolidation. Each virtualized ECU runs in a dedicated virtual machine (VM), and a virtual machine monitor (VMM) acts as middleware between VMs and hardware. Besides the already mentioned general benefits, the virtualization of IC and HU provides advantages such as the flexible placement of graphical output on previously separated displays, which is a matter of software implementation only. Moreover, virtualization enables OEMs to deploy custom applications inside a dedicated VM that is isolated from HU and IC.

This paper makes the following two contributions to enable the consolidation of graphics hardware in vehicular systems. First, we thoroughly *analyze relevant ISO standards and legal requirements* and derive *seven technical requirements for a virtualized automotive HMI system.* Such requirements have been largely neglected by current virtualization efforts, which did not target automotive systems with their specific requirements, in particular, with respect to safety. For OEMs, the certifiability of automotive system functionalities is highly relevant. According to [16, ISO 26262], for each functionality safety-criticality shall be identified and mapped to criticality-classes[3]. To fulfill the criticality-level, the severity and likelihood of failures must be determined using, for instance, *failure mode and effects analysis* (FMEA) [25]. Moreover, certifiability also applies to custom third-party applications. For instance, [12, ISO 15005] prohibits displaying movies to the driver while the vehicle is in motion. These specific regulations impose challenging technical requirements to virtualization.

As second contribution, we present a concept for a *Virtualized Automotive Graphics System* (*VAGS*). We elaborate on the challenges that are due to the identified requirements to consolidate mixed-criticality graphics ECUs as used, in particular, by the HU and IC. Although virtualization is a mature technology for general resources like CPU or main memory, existing concepts do neither provide sufficient isolation for accessing shared graphics hardware (GPU) and input devices (e.g., steering wheel buttons), nor do they provide sufficient isolation for implementing the flexible presentation of application windows. Our proposed architecture can be used as starting point for the future implementation of the specified components.

The rest of this paper is structured as follows. In the next section, we analyze automotive standards and guidelines and derive technical requirements. In Section 3, we present the concept for a *VAGS*. In Section 4, we briefly describe a

[3] [16, ISO 26262] specifies five safety requirement levels: Four ASIL (Automotive Safety Integrity Level) ranging from ASIL-A (low criticality) to ASIL-D (high criticality), and one no-criticality level QM (Quality Management)

first proof-of-concept implementation of our concepts. Finally, we discuss related work in Section 5 and conclude this paper with a summary and outlook onto future work in Section 6.

2 Requirements

In this section, we discuss requirements that are relevant for automotive HMI systems. Automotive application development is constrained by ISO standards, automotive design guidelines, legal requirements, and OEM specific demands. The design guidelines (e.g., [1, AAM 2006], [5, ESoP 2008], [18, JAMA 2004]) in the automotive domain are almost completely derived from the following ISO standards.

- [11, ISO 11428] Ergonomic requirements for the perception of visual danger signals.
- [12, ISO 15005] Requirements to prevent impairment of the safe and effective operation of the moving vehicle.
- [14, ISO 16951] Priority-based presentation of messages.
- [15, ISO 2575] Symbols for controls and indicators.
- [13, ISO 15408-2] Security in IT systems.
- [16, ISO 26262] Risk-based assessment of potentially hazardous operational situations and of safety measures.

In the following, we propose seven technical requirements for automotive HMI system. For each of them we added references to relevant sections of the mentioned ISO standards.

R1 – Input Event Handling

R1.1 – Restricted Access Control: For user input events *access control* is required and it shall not violate any of the following constraints [12, ISO 15005]. Applications using dialogues shall not require to use input devices in a way that demands removal of both hands from the steering wheel while driving (5.2.2.2.2). Additionally, exiting a dialog or an application shall always be possible (5.3.3.2.1) unless legally required or traffic-situation-relevant (5.3.3.2.3).

R1.2 – Restricted Processing Time: A *maximum processing time* for input event handling shall be met. For instance, response to tactile user inputs shall not exceed 250 ms (5.2.4.2.3).

R2 – Restricted Window Creation and Positioning

R2.1 – Restricted Visibility of Windows: Usually, graphical applications use API functions to change the *visibility of windows*, e.g., to create, hide, or position them. This functionality must be restricted, and functions not intended to be used by the driver must be inaccessible for him [12, ISO 15005] (5.2.2.2.4).

R2.2 – Priority-based Displaying of Windows: If multiple windows shall be displayed, the importance of each of them must be defined. Importance is represented by priorities, which can depend on safety requirements and software ergonomic aspects (5.2.4.2.4) that must be met by the system (5.2.4.3.3). Moreover, they can depend on urgency and criticality which have to be defined [14, ISO 16951] (3.5). Additionally, appropriate reactions (e.g., behavior in case of conflicts) shall be enforced [14, ISO 16951] (Annex B). Furthermore, country-specific legal requirements constrain the definition of the priorities, e.g., German law requires the constant visibility of the speedometer while the vehicle is in motion (StVZO §57 [19]). Additionally, visual information must be presented in a consistent way [12, ISO 15005] (5.3.2.2.1).

R2.3 – Timing Constraints: An automotive HMI system shall enable applications to provide important information to the driver within given *time constraints*. This means that windows showing information shall be visible within given time constraints [12, ISO 15005] (5.2.4.3.4). If applications require user interaction, e.g., if a user selects a radio channel, the flow of information must not adversely affect driving (5.2.4.2.1). Concretely, according to [1, AAM 2006] Section 2.1, each glance shall not exceed 2 seconds. Hence, any kind of animation shall not run longer than 2 seconds.

R3 – Trusted Channel

R3.1 – Integrity and Confidentiality: In environments where applications run inside VMs, communication is inevitable. This holds for communication that previously used dedicated communication hardware and is now replaced by software-based inter-VM communication. According to [13, ISO 15408-2], communication between applications and hardware must provide integrity and confidentiality, for both user data (14.5.8.2) and software components providing relevant functionality (17.1.5.3). All applications that need trusted communication shall be able to use it (17.1.5.2).

R3.2 – Authentication and Non-Repudiation: Identification shall be assured even between distinct systems (17.1.5.1), which also applies to inter-VM communication. A trusted channel also requires non-repudiation of origin (8.1.1 and 8.1.6.1-3) and receipt (8.2.1 and 8.2.6.1-3). This requires authentication and may also involve cryptographic key management (9.1.1) and key access (9.1.7.1).

R4 – Virtualized Graphics Rendering In our system, multiple VMs have shared access to a single GPU, and therefore the VMM has to provide isolation. That is, unintended interference between applications must not occur.

R4.1 – Priority Handling: Application windows must be assigned a priority which determines how GPU commands are processed [12, ISO 15005] (5.2.4.2.4 and 5.2.4.3.3), [13, ISO 15408-2] (15.2.5.1-2 and 15.2.6.1-2).

R4.2 – Rendering Time Constraints: Not only comparative requirements (like priorities) but also absolute timing requirements have to be fulfilled. A response to a drivers tactile input shall not exceed 250 ms [12, ISO 15005] (5.2.4.2.3). Similarly, emergency signals may require constant redraw rates to represent flashing lights [11, ISO 11428] (4.2.2). This requires appropriate CPU and GPU resources and imposes a minimum frame rate since the delay between two consecutive frames is constraint by an upper bound. The upper bound must be known to determine the effectiveness of safety-critical messages [14, ISO 16951] (Annex F) and also to allow for the definition of delays after which messages are displayed (Annex B).

R4.3 – GPU Resource Isolation: The GPU is a controlled resource according to [13, ISO 15408-2]. To prevent unintended interference, it must be possible to provide guarantees to certain applications that they are provided sufficient GPU resources such as processing time. Therefore, it must be possible to control which GPU resources individual windows, graphical applications, or VMs are allowed to use (15.3.6.1 and 15.3.7.1-2).

R5 – Reconfiguration of Policies A set of permissions that apply to user input events, application windows, and the related scheduling and isolation is called a *policy*. At each point in time, exactly one policy is active, though policies are dynamically switched during runtime depending on the system state.

R5.1 – Dynamic State Changes: In accordance to [12, ISO 15005], a *state change* happens either on user request or automatically by system-defined rules. A state can depend on a current vehicle condition like "vehicle is in motion" which could require the deactivation of applications that are not intended to be used by the driver while the vehicle is in motion (5.2.2.2.4). Otherwise, an automotive HMI system shall provide sufficient information and warnings to provide the driver with the intended purpose in a current state. For every state change, specified *deadlines* apply to determine a consistent and accurate transition between different states. The definition of states and system behavior is explained in more detail in [14, ISO 16951] (3.3 and Annex E).

R5.2 – Dynamic Policy Changes: Authorized software components shall be able to apply changes to policies during runtime. This includes granting and revoking permissions on both, currently active and currently inactive policies. As for R5.1, deadlines apply to dynamic policy changes. Where applicable and allowed, the driver shall be able to change the active policy to manipulate the flow of information (5.3.3.2.3).

R5.3 – Presentation Enforcement: The system-defined rules shall enforce the presentation of legally required messages and traffic-situation-relevant messages. Presentation requires that those messages are visible and perceivable, in particular, if state changes require driver attention [12, ISO 15005] (5.3.2.2.2). Furthermore, state-related information shall be displayed either continuously or upon request by the driver.

R6 – Certifiability For an OEM, certifiability is an essential part of the software development process, e.g., by using methods like FMEA [25]. The development process for certified software, in particular, for high criticality levels, is quite complex and expensive. A key indicator for complexity is the number of function points that correlates with the approximated number of software defects [3]. Hence, a system shall be developed with respect to an easy certification according to [16, ISO 26262].

R7 – System Monitoring System Monitoring puts the focus on logging, detecting, and reacting to events that possibly are relevant to provide safety.

R7.1 – Secure Boot: Derived from [13, ISO 15408-2], the system shall provide *secure boot* to ensure the integrity of the system. Compromising the system (14.6.9.1) or system devices or elements (14.6.9.2) by physical tampering shall be unambiguously detected.

R7.2 – Auditing: The *auditing* of all safety-critical related events shall be guaranteed to ensure traceability of system activities in an automotive HMI system that potentially violate safety or security. Therefore, direct hardware access must not be permitted to ensure that auditing cannot be bypassed. For a potential violation analysis, a fixed set of rules shall be defined for a basic threshold detection, [13, ISO 15408-2] (7.3.2). To indicate any potential violation of the system-defined rules, the monitoring of audited events shall also be based on a set of rules (7.3.8.1) that must be enforced by the system either as an accumulation or a combination of a subset of defined auditable events which are known to threat the system security (7.3.8.2). Similarly, all changes to policies initiated by applications shall be monitored and verified.

R7.3 – Supervision of Timing Requirements: It is a requirement to regulate the flow of information to ensure short and concise groups such that the driver can easily perceive the information with minimal distraction [12, ISO 15005] (5.2.4.2.1). Therefore, specified *time restrictions* need to be verified. This also includes the auditing of driver tactile input and system response time which shall not exceed 250 ms (5.2.4.2.3).

R7.4 – Detection of DoS Attacks: The occurrence of any event representing a significant threat such as a *DoS attack* shall be detectable by the system in real-time or during a post-collection batch-mode analysis [13, ISO 15408-2] (7.3.2).

R7.5 – Perception of Visual Signals: For the perception of visual danger signals, visibility properties like fractions of luminances [11, ISO 11428] (4.2.1.2) and colors of signal lights (4.3.2) have to be monitored. Monitoring is also required for certain safety-critical symbols defined in [15, ISO 2575].

R7.6 – Software Fault Tolerance: [13, ISO 15408-2] requires the detection of defined failures or service discontinuities and a recovery to return to a consistent and secure state (14.7.8.1) by using automated procedures (14.7.9.2). A list of potential failures and service discontinuities have to be supervised by a *watchdog* to detect entering of failure states. Furthermore, for a de-

fined subset of functions that are required to complete successfully, failure scenarios shall be specified that ensure recovery (14.7.11.1).

R7.7 – System Integrity: In case of unrecoverable failures, the system shall be able to switch to *degraded operation mode* to preserve system integrity. A list of failure types shall be defined for which no disturbance of the operation of the system can take place [13, ISO 15408-2] (15.1.7.1). Moreover, the system shall ensure the operation of a set of capabilities for predefined failure types (15.1.6.1). This includes the handling of DoS attacks and detection of illegitimate policy changes. Some events have to be maintained in an internal representation to indicate if any violations take or took place. This includes the behavior of system activities for the identification of potential violations (7.3.10.2-3) like state changes (7.3.10.1).

3 Architecture

In this section, we briefly describe the architecture of a *Virtualized Automotive Graphics System* (*VAGS*) (cf. Fig. 1) that addresses the identified requirements.

Certifiability (R6) applies to the complete development process, all other requirements are represented by the functionalities of the components of our architecture. With respect to certifiability, we follow the approach of a microkernel-based VMM where drivers run in user space rather than kernel space. Therefore, the kernel code size is very small and easier to certify [3]. If driver code crashes, this does not affect the VMM. The *Virtualization Manager* runs in a dedicated VM and exclusively manages shared resources. It contains relevant drivers, e.g., for GPU and input devices. This ensures that access to all shared resources is controlled by a single trustworthy VM. Indirect hardware access by VMs facilitates Virtualized Graphics Rendering (R4) and System Monitoring (R7). Additionally, the Virtualization Manager contains multiple software components ensuring that every hardware access by VMs is in compliance with our requirements. Note that our architecture only shows four exemplarily VMs. However, we do not restrict the number of VMs. Therefore, it is possible to deploy additional VMs if needed. In order to access hardware, the HU and IC VMs communicate with the Virtualization Manager VM. For this bidirectional communication, a Trusted Channel (R3) is required to support secure communication between the different virtual machines. A trusted channel is provided by the cooperation of the *Isolated Communication Channel* and the *Authentication Manager*. The Isolated Communication Channel provides integrity and isolation for communication (R3.1) between applications and the Virtualization Manager. To initiate a connection, applications first have to provide valid credentials to the Authentication Manager, to guarantee non-repudiation of origin and receipt (R3.2). In particular, this is required for the communication between the graphical applications located on HU or IC and the virtualization manager, which needs to be trustworthy to ensure that the active policy is never violated.

Permission and Policy Management (R5) ensures that applications are getting their defined permissions to use functionalities or resources provided by

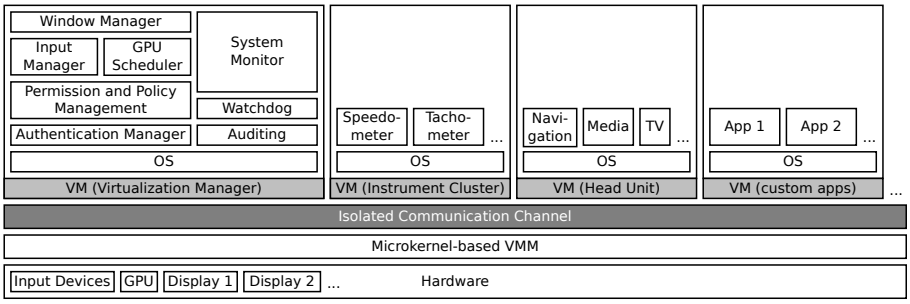

Fig. 1. Architecture

the Input Manager, Window Manager, or GPU Scheduler. Permissions are represented by the active policy, which depends on the current state (R5.1), e.g., "vehicle is parking" or "vehicle is in motion". The policy management is configured by rules that define transitions between policies performed whenever state changes (R5.2) in defined time constraints (R5.3).

The *Input Manager* performs Input Event Handling (R1) and is responsible for dispatching user input events to the intended applications (R1.1). Since the processing of user input is subject to time restrictions, a minimal delivery time for input events to the applications must be ensured (R1.2).

The *GPU Scheduler* is responsible for Virtualized Graphics Rendering (R4) according to drawing requirements and permissions of graphical applications. To this end, applications are assigned priorities that define the amount of dedicated GPU resources (R4.1). Besides priorities, according to (R4.2), deadlines apply to the graphical rendering of certain applications like the tachometer. The GPU scheduler, therefore, has to sequence graphics commands, schedule application requests and provide isolation between different contexts (R4.3).

The *Window Manager* provides the functionality for creating, positioning, and displaying windows of graphical applications. This represents a paradigm shift from fully user-defined window management to restricted window creation and positioning (R2). Applications with sufficient permissions interact with the Window Manager to create windows and to modify properties like size and position (R2.1). Moreover the Window Manager is responsible for correct window stacking (R2.2) and meeting rendering time requirements (R2.3).

In order to guarantee *Secure Boot* (R7.1), the integrity of code that is loaded must be verified, using, for instance, approaches described in [8]. The *Auditing* component (R7.2) traces all relevant system activities and interactions. The gathered traces can be used by the Watchdog and System Monitor components to detect inconsistencies (for R7.3 to R7.7). The *Watchdog* supervises relevant system functionalities and emits signals in case of system malfunctioning as required for R7.3 to R7.6. The *System Monitor* receives signals of detected system malfunctions from the Watchdog. Rules are used to configure its reaction on these signals.

4 Implementation

We have created a proof-of-concept implementation for the main parts of our proposed *VAGS* architecture. The implementation consists of a Window Manager using an hierarchical access control management for display areas and input events. It supports permission negotiation between different virtual machines and applications. The applications create, destroy and move their windows using a dedicated Window Manager API. Based on their permissions, applications are allowed to display their windows in dedicated display areas. Furthermore, each display area is mapped to a depth level representing the priority of the application. This prevents that application windows are overlapped by windows of applications with lower priority. The Window Manager has a dedicated compositing backend which is currently (as an intermediate step) based on X11 compositing.

Linux was used as operating system for the Virtualization Manager and the graphical applications. As a first step we used an x86 standard PC platform and created a set of automotive applications like speedometer and navigation software to demonstrate the feasibility of the concept.

For communication between applications located on different VMs, a transport layer has been implemented. The channels use a custom ring buffer implementation and shared memory to establish data transfer channels. These channels are used for forwarding graphics data like EGL, OpenGL ES 2.0, and API commands of the Window Manager, from the graphics applications to the Virtualization Manager. The management of shared memory is performed by a dedicated component in the Virtualization Manager. To allow applications on different VMs for initiating new connections, in each VM a management process performs the mapping of shared memory segments to applications. The Virtualization Manager performs simple scheduling using synchronization mechanisms of OpenGL ES 2.0.

5 Related Work

The concept of microkernel-based VMMs in virtualization is well known for many years. The focus on safety increased during the last few years, e.g., the NOVA microkernel [26]. Moreover, certifiability became more important, at least in case of the VMM [22].

A large number of work related to virtualization and graphics applications has been described in the literature. Due to space constraints, we only focus on windowing systems, GPU scheduling, and graphics forwarding in the following. According to [17], the X11 Windowing System does not provide security. Trusted X [4] has been proposed to provide security for the X Windowing System targeting the requirements in TCSEC B3 (superseded by [13, ISO 15408-2]) but has not been certified. To provide isolation, an untrusted X server and a window manager is deployed for each security level which impacts scalability. Therefore, mutual isolation of applications is practically impossible due to scalability issues. Nitpicker [7] is a GUI server with security mechanisms and protocols to

provide secure and isolated user interaction using different operating systems. To achieve isolation between these OSes, Nitpicker uses the VMM L4/Fiasco [10]. The EROS Window System (EWS) [24] targets the protection of sensitive information and the enforcement of security policies by providing access control mechanisms and enforcing the user volition. A common denominator of Trusted X, Nitpicker, and EWS is that they only focus on security and thus do not comply with Input Event Handling (R1), Restricted Window Creation and Positioning (R2), and System Monitoring (R5). DOpE [6] is a window server that assures redrawing rates windows of real-time applications and provides a best-effort service for non-real-time applications. DOpE is based on L4/Fiasco [10] for isolation and IPC. However, policies are not enforced. Common to all these windowing systems is the fact that they do not support graphics hardware acceleration and do not provide any timing guarantees for rendering and displaying.

GERM [2] provides GPU resource management targeting fairness without addressing isolation or prioritization. Timegraph [21] enhances these concepts and provides priority-based scheduling of GPU command groups for DRI2. However, for the execution time of an GPU command group no upper bound can be guaranteed and the performance is heavily degraded. Additionally, due to latency induced by synchronous GPU operations, applications using the X Server and double buffering encounter additional problems addressed in [20]. However, the X Server itself does not provide sufficient isolation mechanisms and therefore cannot be used for an automotive HMI system.

VMGL [23] is an approach to transfer OpenGL commands from an OpenGL client to an OpenGL server using a TCP/IP connection. However, using TCP/IP causes significant latency and overhead. Blink [9] is a display system which focuses on the safe multiplexing of OpenGL programs in different VMs. Blink uses an OpenGL Client/Server to transmit the OpenGL commands and data via shared memory to a "Driver VM". The "Driver VM" is responsible for the execution of the OpenGL commands on the GPU. Blink proposes "BlinkGL" which increases performance, but requires applications to be modified.

6 Summary and Future Work

In this paper, we presented requirements for novel automotive HMI systems. From relevant ISO standards, we derived seven technical requirements for the physical consolidation of mixed-criticality graphical ECUs such as head unit and instrument cluster. Additionally we presented VAGS (Virtualized Automotive Graphics System), a novel automotive HMI concept which provides isolation between custom graphics applications running in dedicated VMs. Although these applications are not certified, a VAGS can guarantee that no unintended interference with certified OEM software can take place. We presented a suitable architecture and created a proof-of-concept implementation.
In future work we are going to improve graphics scheduling by using execution time prediction of graphics commands and by using a more suitable scheduling

algorithm. Furthermore, we implement system monitoring, auditing, a watchdog, and integrate authentication concepts. Since the current implementation is based on X11, which has a couple of drawbacks, we plan to switch to a native implementation tailored to embedded hardware. Finally, we evaluate and optimize the performance of our implementation depending on different application scenarios.

Acknowledgement

This paper has been supported in part by the ARAMiS (Automotive, Railway and Avionics Multicore Systems) project of the German Federal Ministry for Education and Research (BMBF) with funding ID 01IS11035.

References

[1] AAM: Statement of Principles, Criteria and Verification Procedures on Driver Interactions with Advanced In-Vehicle Information and Communication Systems. Alliance of Automotive Manufacturers (July 2006)
[2] Bautin, M., Dwarakinath, A., Chiueh, T.: Graphic engine resource management (2008)
[3] Ebert, C., Jones, C.: Embedded software: Facts, figures, and future. Computer 42(4), 42 –52 (April 2009)
[4] Epstein, J., McHugh, J., Pascale, R., Orman, H., Benson, G., Martin, C., Marmor-Squires, A., Danner, B., Branstad, M.: A prototype b3 trusted x window system. In: Proceedings of the 7th Annual Computer Security Applications Conference. pp. 44–55 (Dec 1991)
[5] ESOP: On safe and efficient in-vehicle information and communication systems: update of the European Statement of Principles on human-machine interface. Commission of the European Communities (2008)
[6] Feske, N., Hartig, H.: Dope – a window server for real-time and embedded systems. In: Proceedings of the 24th IEEE Real-Time Systems Symposium. pp. 74–77 (Dec 2003)
[7] Feske, N., Helmuth, C.: A nitpicker's guide to a minimal-complexity secure gui. In: Proceedings of the 21st Computer Security Applications Conference. pp. 85–94 (Dec 2005)
[8] Gallery, E., Mitchell, C.J.: Trusted computing: Security and applications (May 2008)
[9] Hansen, J.G.: Blink: Advanced Display Multiplexing for Virtualized Applications. In: Proceedings of the 17th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV). pp. 15–20 (2007)
[10] Hohmuth, M.: The Fiasco kernel: System Architecure. Technical report: TUD-FI02-06-Juli-2002 (2002)
[11] ISO 11428: Ergonomics – Visual danger signals – General requirements, design and testing. ISO, Geneva, Switzerland (Dec 1996)

[12] ISO 15005: Road vehicles – Ergonomic aspects of transport information and control systems – Dialogue management principles and compliance procecdures. ISO, Geneva, Switzerland (July 2002)
[13] ISO 15408-2: Information technology – Security techniques – Evaluation criteria for IT security – Part 2: Security functional components. ISO, Geneva, Switzerland (Aug 2008)
[14] ISO 16951: Road vehicles – Ergonomic aspects of transport information and control systems (TICS) – Procedures for determining priority of on-board messages presented to drivers. ISO, Geneva, Switzerland (2004)
[15] ISO 2575: Road vehicles – Symbols for controls, indicators and tell-tales. ISO, Geneva, Switzerland (July 2010)
[16] ISO 26262: Road vehicles – Functional Safety. ISO, Geneva, Switzerland (Nov 2011)
[17] J Epstein, J.P.: Trusting x: Issues in building trusted x window systems – or – what's not trusted about x. In: Proceedings of the 14th National Computer Security Conference. vol. 1. National Institute of Standards and Technology, National Computer Security Center (Oct 1991)
[18] JAMA: Guideline for In-vehicle Display Systems – Version 3.0. Japan Automobile Manufacturers Association (Aug 2004)
[19] Janker, H.: Straßenverkehrsrecht: StVG, StVO, StVZO, Fahrzeug-ZulassungsVO, Fahrerlaubnis-VO, Verkehrszeichen, Bußgeldkatalog. C.H. Beck (2011)
[20] Kato, S., Lakshmanan, K., Ishikawa, Y., Rajkumar, R.: Resource sharing in gpu-accelerated windowing systems. In: Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE. pp. 191–200 (April 2011)
[21] Kato, S., Lakshmanan, K., Rajkumar, R., Ishikawa, Y.: Timegraph: Gpu scheduling for real-time multi-tasking environments. In: Proceedings of USENIX Annual Technical Conference. USENIX Association, Berkeley, CA, USA (2011)
[22] Klein, G., Andronick, J., Elphinstone, K., Heiser, G., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: Formal verification of an OS kernel. Communications of the ACM 53(6), 107–115 (June 2010)
[23] Lagar-Cavilla, H.A., Tolia, N., Satyanarayanan, M., de Lara, E.: VMM-independent graphics acceleration. In: Proceedings of the 3rd international conference on Virtual execution environments. pp. 33–43. ACM, New York, NY, USA (2007)
[24] Shapiro, J.S., Vanderburgh, J., Northup, E., Chizmadia, D.: Design of the eros trusted window system. In: Proceedings of the 13th conference on USENIX Security Symposium – Volume 13. USENIX Association, Berkeley, CA, USA (2004)
[25] Stamatis, D.: Failure Mode and Effect Analysis: FMEA from Theory to Execution. ASQ Quality Press (2003)
[26] Steinberg, U., Kauer, B.: Nova: a microhypervisor-based secure virtualization architecture. In: Proceedings of the 5th European conference on Computer systems. pp. 209–222. EuroSys '10, ACM, New York, NY, USA (2010)