Konzepte und Mechanismen für die Darstellung von sicherheitskritischen Informationen im Fahrzeug

Von der Fakultät Informatik, Elektrotechnik und
Informationstechnik der Universität Stuttgart
zur Erlangung der Würde eines Doktors der Naturwissenschaften
(Dr. rer. nat.) genehmigte Abhandlung

vorgelegt von

Simon Gansel

aus Tübingen

Hauptberichter:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Mitberichter:	Prof. Dr.-Ing. habil. Roman Obermaisser
Tag der mündlichen Prüfung:	*19.04.2017*

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart
2017

Inhaltsverzeichnis

Danksagungen

Ich möchte mich an dieser Stelle bei allen bedanken, die während der Zeit zur Promotion an meiner Seite standen und mich durch Rat und Tat unterstützt haben.

Meinem Doktorvater Prof. Dr. Kurt Rothermel möchte ich für das Ermöglichen dieser Arbeit danken. Ohne sein entgegengebrachtes Vertrauen und die wertvollen Ratschläge wäre diese Arbeit nicht möglich gewesen. Bei Prof. Dr. Roman Obermaisser möchte ich mich für die Diskussionen und Vorschläge zur Verbesserung dieser Arbeit bedanken.

Durch die gemeinsame Arbeit an dem Projekt ARAMiS hat vor allem Stephan Schnitzer mit Rat und Tat das Entstehen dieser Arbeit ermöglicht. Insbesondere danke ich ihm für die vielen Diskussionen und den Austausch von Wissen. Dies hat die Zusammenarbeit im Rahmen des Projekts und diese Arbeit sehr bereichert. Danken möchte ich auch Dr. Frank Dürr für seine Erfahrungen, die er durch seine Rückmeldungen in diese Arbeit hat einfließen lassen.

Des Weiteren möchte ich mich bei den Kollegen der Daimler AG bedanken, die durch die vielen anregenden Gespräche wertvolle Erfahrungen in diese Arbeit haben einfließen lassen. Genannt sei unter den vielen Dr. Viktor Friesen, der in unseren gemeinsamen Gesprächen sein mathematischen Wissens an mich weitergab. Insbesondere möchte ich auch Dr. Christian Maihöfer und Dr. Matthias Stümpfle danken, die mir die Möglichkeit gaben, an dem Projekt ARAMiS arbeiten zu dürfen und mich in dieser Zeit unterstützt haben.

Nicht zuletzt sei den vielen Studenten und externen Mitarbeitern gedankt, welche das Projekt ARAMiS tatkräftig durch Praktika und/oder Abschlussarbeiten, sowie Implementierungen bereichert haben.

Besonderer Dank gilt auch meiner Familie, die mir in dieser Zeit viel Rückhalt gab. Ohne euer Verständnis und eure Ermutigung in dieser Zeit wäre diese Arbeit nicht möglich gewesen.

Abbildungsverzeichnis

Zusammenfassung

Die zunehmende Verwendung von Anwendungen im Fahrzeug wie Navigation, Videowiedergabe oder Geschwindigkeitsanzeige, welche eine grafische Repräsentation anstatt der physischen Zeigerinstrumente nutzen, geht einher mit einer Zunahme der verbauten digitalen Anzeigen im Fahrzeug. Neben den Anzeigen der Headunit und der Kombiinstrumente gibt es Anzeigen in den Kopfstützen und die Headup-Anzeige. Da meist jede Anzeige ihr eigenes Steuergerät besitzt, führt dieser Trend auch zu einer Zunahme an Steuergeräten. Dies bringt jedoch Skalierungsprobleme und eine zunehmende Komplexität mit sich, sowie erhöhten Bauraumbedarf, zunehmende Kosten und einen höheren Stromverbrauch. Um diesen Problemen begegnen zu können, wird eine Konsolidierung von Steuergeräten angestrebt.

Die Anwendungen im Automobilbereich sind jedoch teils sehr unterschiedlich in ihrer Sicherheitskritikalität, da sie unterschiedlichen Einfluss auf die funktionale Sicherheit des Fahrzeugs haben. So ist die Darstellung mancher Warnlampen sicherheitskritisch, da sie für die Sicherheit der Insassen relevant sind, während das Abspielen einer DVD nur den Qualitätsansprüchen genügen muss. Die unterschiedlichen Anwendungen dürfen sich gegenseitig nicht ungewollt beeinflussen, was eine Isolation erforderlich macht, die bisher durch physisch separierte Hardware-Plattformen realisiert wurde. Dies muss aufgrund der oben genannten Gründe durch Software implementiert werden. Hierzu eignet sich vor allem die Technologie Virtualisierung, welche verschiedene Anwendungen in virtuellen Maschinen kapselt. Die Virtualisierung gewährleistet Isolation derzeit in der Nutzung von Ressourcen wie CPU und Speicher und vermeidet unbeabsichtigte oder böswillige Beeinflussung. Jedoch erstreckt sich die Isolation nicht auf die Nutzung der grafischen Ressourcen wie Anzeigen und GPU und kann insbesondere nicht die Anforderungen im Automobilbereich erfüllen. Der konfliktfreie Zugriff auf Anzeigebereiche unter Berücksichtigung der Sicherheitskritikalität der Anwendungen ist essentiell für die Sicherheit während der Fahrt. Im Rahmen des öffentlich geförderten Projektes ARAMiS wurde dieser Sachverhalt untersucht und geeignete Konzepte entwickelt.

Zusammenfassung

In dieser Arbeit werden unterschiedliche Anforderungen aus Rahmenrichtlinien wie ISO-Standards oder gesetzlichen Bestimmungen analysiert und auf sieben Kategorien von Anforderungen reduziert, welche für das grafische System im Fahrzeug erfüllt werden müssen. Auf Grundlage dieser Anforderungen wird dann eine Architektur für einen Domänen-Server vorgeschlagen, welche mittels Virtualisierung und verschiedener Komponenten Isolation zwischen grafischen Anwendungen mit unterschiedlicher Sicherheitskritikalität bietet.

Insbesondere die gemeinsame Nutzung der Anzeigen durch die Anwendungen mit unterschiedlicher Kritikalität stellt eine besondere Herausforderung dar. Die Konsolidierung von Steuergeräten wie der Headunit und den Kombiinstrumenten ermöglicht die flexible und dynamische Nutzung der viele Anzeigen, die den Anwendungen nun zur Verfügung stehen. Die dynamische Zuweisung der Anzeigebereiche muss die verschiedenen Anforderungen erfüllen und zu jeder Zeit die Ablenkung des Fahrers vermeiden. Zu diesem Zweck ist eine Zugriffskontrolle für die Anzeigebereiche notwendig. Hierzu werden Kontexte verwendet, um dynamisch festzustellen, welche Anwendung auf welchen Anzeigebereich zugreifen darf. Ein Kontext kann aus Sensorinformationen des Fahrzeugs (z. B. die Geschwindigkeit) oder aus Zuständen der Anwendungen (z. B. welcher Eintrag in der Auswahlliste ausgewählt ist) abgeleitet werden. In dieser Arbeit wird ein Zugriffskontrollmodell vorgeschlagen, welches den Zugriff auf die Anzeigebereiche abhängig vom Kontext des Fahrzeugs und der Anwendungen regelt. Für eine möglichst flexible Erweiterbarkeit werden die Berechtigungen für die Anzeigebereiche zwischen Anwendungen, welche beispielsweise von verschiedenen Drittanbietern stammen, delegiert. Das Zugriffskontrollmodell ist vollständig formal definiert und es werden anhand von definierten Zuständen im Modell bestimmte Eigenschaften wie die Konfliktfreiheit bei Zugriff auf Anzeigebereiche bewiesen. Die Evaluation des Zugriffskontrollmodells wird anhand einer Implementierung der Konzepte durchgeführt und zeigt auf, dass die Latenz, die durch die Zugriffskontrolle entsteht, gering genug für Szenarien im Fahrzeug ist.

Zudem wird ein Konzept für das Compositing von Fenstern vorgeschlagen, welche den grafischen Inhalt von Anwendungen enthalten und entsprechend ihrer Größe und Position auf einer Anzeige dargestellt werden. Hierzu wird zwischen rechteckigen Fenstern und Fenstern, die eine beliebige Form annehmen können, unterschieden. Rechteckige Fenster werden meist in den existierenden Fenstersystemen verwendet, für welche zwei populäre Ansätze für das Compositing mehrerer sich teils überdeckender Fenster existieren. In dieser Arbeit wird ein Hybridansatz für das Compositing vorgeschlagen, welcher die Vorteile der beiden Ansätze nutzt,

um ein effizienteres Compositing durchzuführen, was anhand von verschiedenen Szenarien aufgezeigt werden kann. Die Verwendung von Fenstern in beliebiger Form erfordert andere Ansätze für das Compositing. Um durchgängig die flexiblen Möglichkeiten des Zugriffskontrollmodells zu ermöglichen, wird daher ein weiterer Ansatz für ein Compositing vorgeschlagen, welcher als Grundlage für die Definition der Fenster Bitmasken verwendet, die ebenfalls in den Berechtigungen für die Anzeigebereiche verwendet werden. Das Compositing gewährleistet dann, dass nur die Pixel auf der Anzeige geschrieben werden, welche in der Berechtigung für den Zugriff mittels Bitmaske definiert wurde. Anhand geeigneter Evaluationen wird aufgezeigt, dass diese Eigenschaft für das Compositing einen Mehraufwand darstellt, jedoch in Szenarien im Fahrzeug anwendbar ist.

Zur Evaluation der Konzepte für ein Zugriffskontrollmodell und ein Compositing für Anzeigebereiche wird die Systemarchitektur basierend auf Virtualisierung in einem Demonstrator implementiert. Anhand des Demonstrators in Form eines Cockpits, welcher im Rahmen des Projektes ARAMiS entstanden ist, werden verschiedene Szenarien aus dem Fahrzeug demonstriert. Dadurch wird gezeigt, dass eine Konsolidierung der separaten Hardware-Plattformen für die Kombiinstrumente und die Headunit unter Berücksichtigung der verschiedenen Anforderungen für sicherheitskritische Anwendungen im Fahrzeug durch den Einsatz der vorgeschlagenen Konzepte möglich ist.

Abstract

Introduction

The increasing number of applications in vehicles such as navigation, DVD playback, or tachometer (using a graphical representation instead of a physical pointer device) leads to more and more digital displays in vehicles. In addition to the head unit display and the instrument cluster display, there are displays mounted in the head-rests and headup displays used to provide additional infotainment to the driver and passengers of the vehicle. Since each display is connected to its own electronic control unit (ECU), this trend also leads to an increasing number of ECUs [EJ09, Mul11]. This increases the complexity and brings up scalability issues with respect to installation space, high costs and energy consumption. To face these problems a consolidation of ECUs is required.

Due to the consolidation of ECUs such as the head unit and instrument cluster [MKS12] the applications can directly access more displays to present their content. This allows for flexible usage, by dynamically mapping applications to display areas on any connected display. However, the usage of the displays is restricted by different requirements, which prevents driver distraction and provides safety. Since the safety-criticality of the applications (previously located on different ECUs) differs, an isolation mechanism is required to prevent intended or unintended interferences. To this end, the technology virtualization can be used to encapsulate the different applications in virtual machines and isolate them from each other. Virtualization provides isolation of resources such as central processing unit (CPU) and memory that have to be shared between the applications. However, existing isolation concepts do not cover graphical resources such as displays and graphics processing unit (GPU) and therefore do not satisfy the requirements in the automotive domain [GSD+13]. In particular, conflict-free access to display areas, while taking the safety-criticality of the applications into account, is essential for safety while the vehicle is in motion.

In the Project ARAMiS, supported by the German Federal Ministry for Education and Research, academic and industrial partners developed concepts to

overcome these shortcomings of technology virtualization.

Requirements and System Architecture

In this work, various requirements from ISO standards and legal restrictions were analyzed and clustered into seven categories that must be satisfied by the graphical system in a vehicle. The seven categories are "Input Event Handling (R1)", "Restricted Window Creation and Positioning (R2)", "Trusted Channel (R3)", "Virtualized Graphics Rendering (R4)", "Reconfiguration of Policies (R5)", "Certifiability (R6)", and "System Monitoring (R7)". In particular, the requirement categories "Restricted Window Creation and Positioning (R2)" and "Reconfiguration of Policies (R5)" are the main focus of this work. Based on these requirements, the architecture for an infotainment domain server is proposed that provides isolation between applications with different safety-criticalities by using virtualization.

Related Work

The concept of virtualization has been well known for many years. While virtualization provides good isolation of CPU and memory, the concepts for safe sharing of graphic resources such as displays and GPUs are insufficient to fulfill the requirements in the automotive domain. In particular, the dynamic mapping of display areas to applications must satisfy all relevant automotive requirements.

There exists some research (e.g., [FH04, FH05, Han07]) targeting the compositing of windows, but they assume that the window placement is controlled by the user. This is incompatible with automotive requirements and does not provide fine-grained access control. The X-server, which is often used in the operating system Linux, has security issues according to Epstein et al. [EP91]. Epstein et al. propose an extension to provide security to the X-server, but the user controls the compositing without restrictions.

Using context to provide clients with information about located-objects and how those objects change over time, was first introduced by Schilit and Theimer [ST94]. They use context as location, identities of nearby people and objects, and changes to those objects, but access control is not provided. In [BBG04, HYMLWD12, MB03, SN04], role-based access control (RBAC) is combined with contexts to decide which roles need to be assigned and which permissions are valid. Since they either do not provide the ability to delegate permissions between

applications, or they only consider an administrator to be responsible for defining the set of permissions for each context the flexibility of managing permissions is restricted. Context-aware access control models exist using context information similar to roles in the access decision (e.g., [CMT04a, CMT04b]). Nonetheless, they do not provide hierarchically dependent permissions and thus do not support prioritized usage of resources.

Besides the access decisions about which application should access which display area, the efficiency and reliability of the compositing of the graphical content into the screen buffer is crucial for the safety of the graphical system in a vehicle. There exists several window managers providing compositing of application windows. Meyrowitz and Moser [MM81] developed the window manager BRUWIN, which focuses on the adaptability of the graphical system to a variety of devices and operating systems. A display manager is responsible for the compositing of z-ordered windows that can overlap. Basically, they use an expensive Full Compositing approach that skips all windows, which are fully overlapped. The window manager Sapphire, developed by Myers [Mye84a, Mye84b], supports rectangular windows that can overlap. They propose in [Mye86] to store only the covered parts of overlapping windows to minimize the required memory. However, Sapphire does not store the window content in off-screen buffers, which potentially leads to artifacts occurring on the screen and requires costly redraws of the screen content. In [Pik83], a data structure for storing the covered parts of a window to reduce the window update latency is proposed. They extend the domain of the bitmap operator bitblt [GS82] to include fully or partly covered windows. This approach is similar to Tile Compositing, where visible tiles are only kept in the screen buffer. Besides saving a small amount of memory, it is costly to change the tiling since the data needs to be copied between the screen buffer and off-screen data structures.

The X windowing system (X11) [SG86] is commonly used in Linux. The X Server manages windows and provides support to compositors (i.e., by using Xdamage or the Xcomposite extension). The xcompmgr [xco15] compositor uses the Xdamage extension [GP04] to get X event notifications about modified window regions. However, affected applications need to update their content on request, which is sent in z-order since no backbuffers are used. Therefore, X11 uses the Full Compositing strategy for compositing. Wayland [Way15] is a successor of X11 and reduces the functionalities to the main tasks, which is managing and compositing of windows. Weston is an efficient implementation of Wayland and uses the Pixman library [Pix15] for tiling. By using Pixman, the compositor in

Weston is using the Tile Compositing strategy.

Android uses a surface manager to manage the surfaces (widgets and views) of applications. Surfaces represent windows, that are composited by SurfaceFlinger [Sur15]. Thus, Android uses an optimized Full Compositing, which is sufficient for typical Android use cases where windows do not overlap, but not for complex automotive scenarios with overlapping windows.

Context-aware Access Control Model

In current high-end cars there is a static mapping of functionalities to display areas. Convenience requirements and display sizes establish limits for the number of functionalities. The rapid increase of graphic functionalities, the number of displays, and display sizes leads to the desire to use a dynamic mapping instead. Moreover, the mapping often depends on the context of the car or functionalities. For instance, while waiting at traffic lights, another application could use the display area of the speedometer. Or depending on the age of the driver, instruments could be magnified. However, the shared use of the displays by applications with different safety-criticality levels is challenging since the visibility of highly safety-critical applications has to be guaranteed in any situation. Hence, dynamic mapping of display areas to applications must satisfy all relevant automotive requirements, which also includes the prevention of driver distraction caused by the applications. Therefore, access control to display areas is required. Applications shall only get access to a display when the current context allows their presentation on the display. A context can be determined by using the car's sensors (e.g., speed) or the conditions and states of the applications (e.g., the selected entry in a pull-down menu). In this work, an access control model is proposed that grants access to display areas depending on the contexts of the car and the applications. A constrained-permission consists of a display area and a set of contexts in which the access to the defined display area is allowed. An application that wants to access a certain display area requires a constrained-permission for this display area and the restricting contexts of the permission need to be active. To provide new applications access to display areas in a flexible manner, constrained-permissions are delegated to other applications, e.g., developed by third-party developers. The access control model is formally defined and has defined properties. By using states and transitions between states these properties are formally proved.

Efficient Compositing

When the applications have permission to access a display area, they register a window in which their graphical content will be visible. Combining all application windows on a screen, with respect to their size and location, is called compositing. The applications render their output in what are referred to as off-screen buffers and the compositing bitblits it to the screen buffer at the respective window position. The shape of a window is typically rectangular in different sizes, using only the information of the window's position, width and height. Rectangular windows are mostly used in window systems since they are easy to define.

To guarantee the visibility of potentially overlapping windows, compositing has to consider the z-order of the windows. Two compositing strategies, Tile Compositing and Full Compositing, exist that handle overlapping windows. Although both have performance issues depending on how windows overlap. Since automotive embedded platforms are restricted in power consumption, installation space, and hardware cost, their performance is limited. This effectuates the need for highly efficient bitblitting concepts. In order to increase the performance in compositing windows, Hybrid Compositing is proposed, which predicts the required bitblitting times, and chooses the most efficient strategy for each pair of overlapping windows and tiles.

Rectangular windows do not always match the presentation requirements of automotive applications. For instance, the speedometer is often depicted as a tube that requires a circular window. Therefore, a concept is presented that allows for windows in arbitrary shapes. In this work the usage of bitmasks is proposed to define arbitrary shapes that can also be used to describe the display areas in the constrained-permissions, and therefore guarantee conflict-free access in the compositing step. An application receives a permission to a display area, which is described by using a bitmask that indicates the accessible pixels. To register a window an application also uses a bitmask, which can be the bitmask of the permission or only a subset of the accessible pixels. After the application finished writing its content to its off-screen buffer, the compositing bitblits only the accessible pixels of the application's offscreen buffer to the screen.

Implementation

To demonstrate the feasibility of the proposed concepts, a proof-of-concept implementation of the access control model and the compositing for an automotive

HMI system was implemented. To this end, an infotainment domain server was created, which uses virtualization to isolate the safety-critical and non-safety-critical applications from each other.

Various software components were implemented that provide the proposed isolation mechanisms. To isolate the safety-critical applications (e.g., the brake failure warning) from the non-safety-critical applications (e.g., a game), traditionally running on physically isolated IC and HU platforms, different virtual machines were used. A dedicated virtual machine, called Virtualization Manager, provides access control and has exclusive access to the hardware components GPU, displays, and input devices. Isolated communication channels provide session-based FIFO communication between the applications in the VMs and the system components of the Virtualization Manager, which is required to get access to the hardware components. A communication manager creates dedicated communication channels to the system components for each authenticated application. Communication between applications and system components is restricted based on their identification. The context-based access control layer performs access decisions using contexts. The Context Manager provides context handling for applications, which can set the status of contexts. The Constrained-Permission Manager is responsible for granting and revoking constrained-permissions and provides access decisions for the window manager. The Window Manager handles the creation, destroying, and positioning of windows. Applications can grant or revoke constrained-permissions. If an application received a constrained-permission, it is allowed to create a window within the bounds of the received constrained-permission. The Compositing Layer provides operations for resizing and mapping of windows.

Applications directly render into the off-screen buffers of their windows. After the application created content for its window, the Compositor copies the content to the screen-buffer, which makes it visible on the displays. The Compositor uses the Hybrid Compositing strategy, which allows for efficient bitblitting of the window content to the screen-buffers. Both the concepts for rectangular and pixel-exact windows were implemented. Pixel-exact constrained-permissions are implemented using rectangular areas in combination with bitmasks that restrict operations to the allowed pixels. Due to the exclusive access property of the proposed access control model, the compositing does not need to consider the layering of windows.

A cockpit demonstrator is used to demonstrate various automotive scenarios. It contains two automotive 12"displays each with a resolution of 1440×540 pixels, and automotive input devices such as steering wheel buttons and the

central control knob to control the applications. The implementation is running on a Freescale i.MX6 SABRE for Automotive Infotainment quad core embedded board with three GPUs, namely, the GC2000 3D GPU providing OpenGL ES 2.0 support, the GC355 for vector graphics with OpenVG 1.1, and the GC320, which provides compositing of framebuffers by using a 2D API. The OpenGL ES 2.0 API is used by the applications for the rendering of the graphical content into backbuffers. The 2D API of the Image Processing Unit (IPU) is used by the Compositor to bitblit into framebuffers.

Evaluation

The concepts were evaluated by using the implementation that is running on the Freescale i.MX6 SABRE for Automotive Infotainment quad core embedded board.

Typical automotive scenarios for a given set of constrained-permissions and active contexts, using applications such as speedometer, tachometer, navigation, indicators, trip, car status, and menu were setup. These scenarios include constrained-permissions that allow the display of half of the speedometer and the tachometer in the left and the right corner, respectively, in favor of a bigger display area for the presentation of the rear view camera in the middle of the screen. All constrained-permissions are conflict-free and hierarchically granted in order of the criticality of the according applications.

Besides the proof-of-concept of the access control model by using different constrained-permissions in automotive scenarios, the latency introduced by the granting and revoking of permissions was measured and evaluated. The results show that the time required to provide access to a display area is 58 ms for one application. This is below the time constraint of 250 ms for tactile user feedback. Since the changing of 15 permissions is still below 2 s, the time constraint for rear view cameras is also satisfied. The results represent the worst case, which are mainly caused by the communication between the applications and the components of the Virtualization Manager running on separate virtual machines. This demonstrates that the access control for display areas is applicable in automotive scenarios.

To evaluate the compositing concepts, various scenarios for rectangular windows and non-rectangular windows were used. Using the proposed Cache-Hybrid Compositing, the time required to bitblit 17 rectangular windows in an automotive scenario is 51 % faster than compared to Full Compositing. In addition, the

Abstract

Cache-Hybrid Compositing reduces the CPU execution time of the Hybrid Compositing approach by up to 66 %. Additionally, non-rectangular windows using bitmasks were evaluated for different scenarios. To evaluate the scalability, a scenario with 15 application windows was executed. The performance results show that the bitblitting driver implementation of the Freescale i.MX6 SABRE platform is only able to bitblit up to 10 bitmasks in less than 16 ms. Thus, bitblitting is the bottleneck in scenarios with many windows. To compare the performance of the compositing using non-rectangular windows to the compositing using rectangular windows, scenarios with rectangular windows and rectangular bitmasks were used. The comparisons show the frame rate of the applications using compositing with rectangular windows is about twice as fast as using compositing with rectangular bitmasks in automotive scenarios with six windows. Still, the compositing using bitmasks provides sufficient performance for scenarios with only a small number of windows or required update rate of windows less than 60 FPS.

Conclusion

The consolidation of the instrument cluster and the head unit in a single ECU, leads to the desire to use the connected displays versatilely flexibly by dynamically mapping applications to display areas. Due to the restriction in the usage of the displays by different requirements, that should prevent driver distraction and provide safety, suitable concepts for shared displays need to be applied. In this work, the different automotive requirements have been analyzed and seven requirement categories are identified that need to be satisfied by the graphical system in a vehicle. Based on these requirements, a system architecture and components suitable to meet these requirements are proposed. To guarantee the visibility of highly safety-critical applications in any situation, an access control model is proposed. It allows for safe sharing of the displays between non-safety-critical and safety-critical applications. To provide flexibility without compromising safety, the model provides the ability to hierarchically grant access to display areas depending on the contexts of the car and applications. Since the compositing is also crucial for the safety and the performance of a consolidated graphical system in a vehicle, compositing concepts were developed and evaluated. Using automotive scenarios, the feasibility of the proposed concepts were evaluated in a cockpit demonstrator. It was shown that the proposed concepts are suitable to guarantee safe display sharing in a virtualized graphical system and provide enhanced flexibility for the instrument cluster and head unit applications.

1. Einleitung

Die Fahrzeugentwicklung in der Automobilindustrie wird zunehmend getrieben durch Weiterentwicklungen der elektronischen Komponenten, welche oftmals unverzichtbar für die Sicherheit, den Komfort und das Entertainment von Fahrzeugen sind. Bereits 80 % der Innovationen im Automobilbereich finden heutzutage im Bereich der Elektronik und Software statt [LH02, MGR+14]. Meist hat die Entwicklung neuer Funktionalitäten zur Folge, dass diese in separate Steuergeräte implementiert werden, um ungewollte Interferenzen mit bestehenden Steuergeräten zu verhindern. Durch die Aufteilung der Funktionalitäten auf verschiedene Steuergeräte, was eine physikalische Trennung darstellt, soll eine Isolation zwischen Funktionalitäten unterschiedlicher Kritikalität gewährleistet werden.

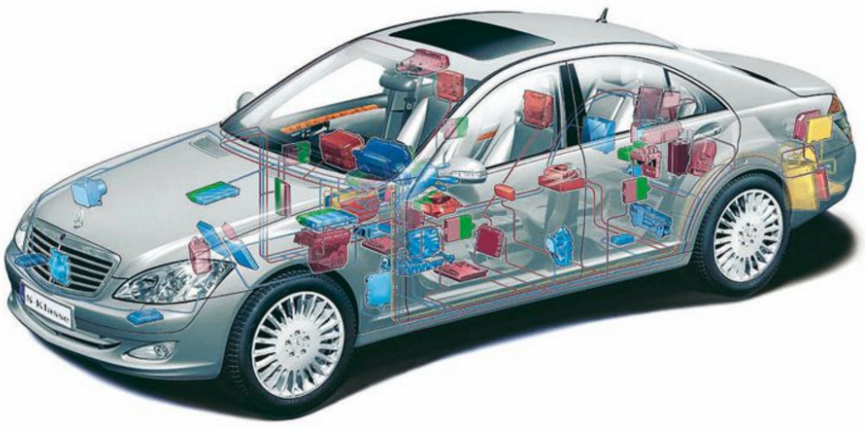

Abbildung 1.1.: Mercedes Benz S-Klasse 221 mit verbauten Steuergeräten (Quelle: Auto Motor und Sport[1])

Dies wiederum führt in der heutigen Fahrzeugentwicklung meist zu einer Zunahme an Steuergeräten, so dass in Premiumfahrzeugen (siehe Abbildung 1.1) zwischen 70 und 100 Steuergeräte verbaut sind [EJ09, Mul11], welche verschiedenen Domänen wie Antrieb oder Infotainment zugeordnet sind. Zudem sind die Steuergeräte über das gesamte Fahrzeug verteilt und mittels eines Netzwerks aus verschiedenen Kommunikationsbussen, wie das Local Interconnect Network (LIN)

[1] http://www.auto-motor-und-sport.de/testbericht/lexikon-bus-systeme-694533.html

und das Controller Area Network (CAN), miteinander verbunden. Dies bedeutet nicht nur einen erheblichen Mehrbedarf an Bauraum für die Steuergeräte im Fahrzeug, sondern ebenfalls höhere Kosten für die Hardware und nicht zuletzt eine Zunahme des Stromverbrauchs, womit dieser Trend an Grenzen stoßen wird. Um

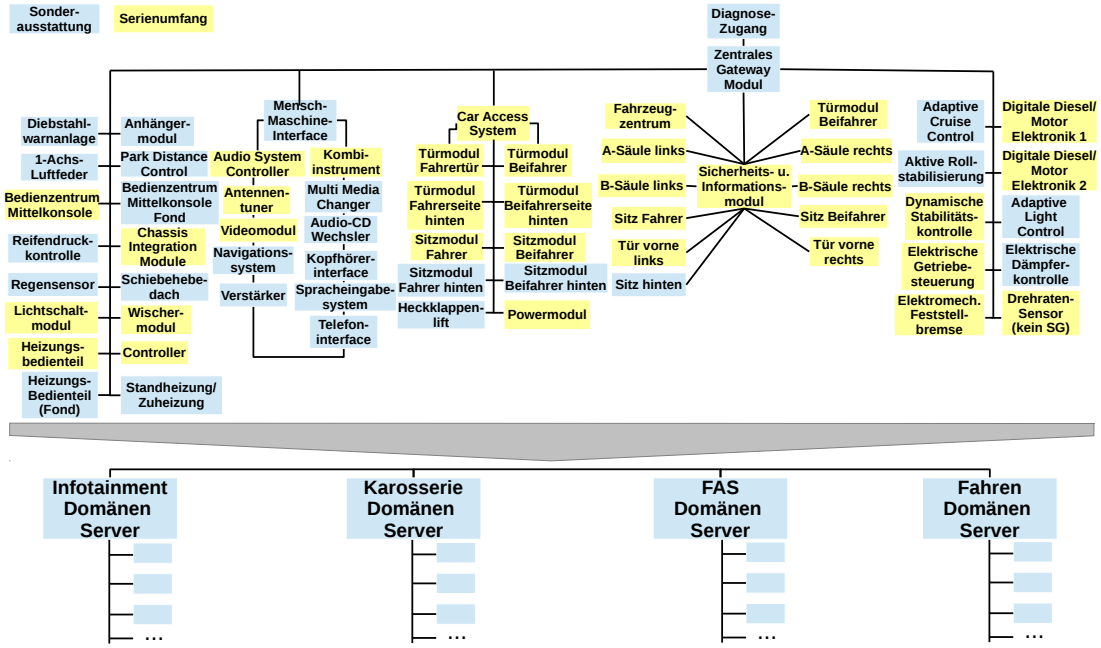

Abbildung 1.2.: Domänenserver-Architektur im Fahrzeug nach [MKS12]

diesem Trend entgegenzuwirken, gibt es die Bestrebung eine Hardwarekonsolidierung von Steuergeräten durchzuführen.

Ein Ansatz [MKS12] hierfür ist die Konsolidierung der Steuergeräte entsprechend ihrer Zuordnung zu Domänen, wie in Abbildung 1.2 dargestellt, in sogenannte *Domänenserver*. Diese übernehmen die Funktionalitäten der Domäne, was eine Zentralisierung der Domänen darstellt. Bei der Konsolidierung unterschiedlicher Funktionalitäten auf einen *Domänenserver* muss jedoch die *Isolation* zwischen den sicherheitskritischen und den nicht-sicherheitskritischen Funktionalitäten, die vormals durch die physikalische Trennung der Steuergeräte gegeben war, weiterhin gewährleistet werden. Es darf keine unbeabsichtigte oder böswillige Beeinflussung einer sicherheitskritischen Funktionalität durch eine nicht-sicherheitskritische stattfinden. Der *Infotainment-Domänenserver* steht hierbei im besonderen Fokus, da er die grafische Darstellung der sicherheitskritischen und nicht-sicherheitskritischen Funktionalitäten gewährleisten muss. Beispielsweise muss garantiert werden, dass das Darstellen wichtiger Warnmeldungen nicht durch das Abspielen einer DVD verhindert oder verzögert werden kann.

Für die Gewährleistung der Darstellung von sicherheitskritischen Inhalten ist es

folglich erforderlich, dass der Zugriff auf die Anzeigebereiche kontrolliert wird und ein beabsichtigtes oder unbeabsichtigtes Überschreiben der sicherheitskritischen Inhalte nicht möglich ist. Dies erfordert die Vergabe von Berechtigungen für den Zugriff von Anwendungen auf bestimmte Anzeigebereiche, welche entsprechend der Kritikalität der Anwendungen vergeben werden müssen.

Da sich hierbei sicherheitskritische und nicht-sicherheitskritische Funktionalitäten dieselbe Hardware teilen, muss die Isolation für einen sicheren Zugriff auf diese, entsprechend der Kritikalität der Funktionalität, gewährleistet werden.

Eine Schlüsseltechnologie, die Isolation zwischen den unterschiedlichen Funktionalitäten bietet, ist *Virtualisierung*. Diese Technologie bietet mittels einer Softwareschicht, teils durch Hardware unterstützt, eine Abschottung einzelner Partitionen, stellt Kommunikationsmechanismen zwischen diesen zur Verfügung und kontrolliert den Zugriff auf die Hardwareressourcen. Die Partitionen selbst beinhalten die Anwendungen mit jeweils eigenem Betriebssystem und werden auch als virtuelle Maschinen bezeichnet. Der *Hypervisor* regelt für die virtuelle Maschinen den Zugriff auf die Ressourcen. Die Technologie Virtualisierung hat sich vor allem im Bereich der Serverzentren erfolgreich durchgesetzt und ermöglicht durch Konsolidierung die Rechneranzahl zu reduzieren, sowie eine gleichmäßige Lastverteilung der Server durchzuführen. Virtualisierung zur Konsolidierung der Steuergeräte im Fahrzeug zu verwenden, ist insbesondere interessant, um dieses Potential in einer Domänenserver-Architektur auszuschöpfen. Da Steuergeräte immer für die Maximallast ausgelegt sind, aber diese in der Regel nicht gleichzeitig bei allen Steuergeräten auftritt, kann durch eine Konsolidierung die Hardware besser ausgelastet werden. Hierbei gilt es jedoch die Anforderungen der Automobilindustrie zu prüfen, inwieweit die Technologie Virtualisierung dies ermöglicht.

Da die Funktionalitäten der heutigen Infotainmentsysteme viele grafische Funktionen und Anwendungen beinhalten, steht insbesondere die Hardwarebeschleunigung für Grafik und Video bei einem *Infotainment-Domänenserver* im Fokus. Die Headunit (HU) ist das wesentliche Steuergerät des Infotainmentsystems im Fahrzeug und stellt verschiedene Anwendungen wie Navigation, Radio und DVD-Wiedergabe auf einer zentralen Anzeige in der Mittelkonsole dar. Diese Anwendungen sind in der Regel nicht sicherheitskritisch, erfordern jedoch eine Schnittstelle in das Kommunikationsnetz des Fahrzeugs um Informationen zu erhalten und Konfigurationen vorzunehmen. Werden in heutiger Zeit die Anwendungen der HU vorwiegend vom Original Equipment Manufacturer (OEM) selbst bzw. in dessen Auftrag entwickelt, was einen gewissen Grad an Vertrauenswürdigkeit und Kontrollierbarkeit der Anwendungen bietet, so zeichnet sich ein Angleichen der

Automobilindustrie an die CE-Industrie (Unterhaltungsindustrie, engl. Consumer Electronics) ab. Dies bedeutet, dass in Zukunft nicht nur der OEM Anwendungen zur Verfügung stellt, sondern auch Anwendungen von anderen Entwicklern über Portale (z. B. der App Store von Apple oder der Play Store von Google) den Weg in das Fahrzeug finden werden, wie es beispielsweise Ford [For13] anstrebt. Diese Anwendungen können unter Umständen unbewusst oder bewusst ein schadhaftes Verhalten haben, welches Einfluss auf sicherheitskritische Anwendungen haben kann, wenn keine geeigneten Maßnahmen dies verhindern.

Abbildung 1.3.: Vier verbaute Anzeigen in einem Mercedes-Benz S 500

Die wachsende Popularität grafischer Funktionen und Anwendungen schlägt sich auch im Trend der zunehmenden Anzahl an verbauten digitalen Anzeigen im Fahrzeug nieder, wie in Abbildung 1.3 zu sehen. So können an das Infotainmentsystem weitere Anzeigen angeschlossen sein, wie die Anzeigen in den Kopfstützen der Vordersitze, welche es den Beifahrern auf den Rücksitzen ermöglichen, ebenfalls auf das Infotainmentsystem zuzugreifen und z. B. DVDs wiederzugeben. Die klassischen Kombiinstrumente sind diesem Trend auch unterworfen und werden bereits in Premiumfahrzeugen durch Anzeigen ersetzt, welche die Informationen wie Geschwindigkeit und Drehzahl in grafischer Form darstellen und auf analoge Zeigerinstrumente verzichten. Diese stellen in der Regel sicherheitskritische Funktionalitäten dar, welche besonderen Anforderungen unterliegen. Dies beinhaltet eine Teilzertifizierung nach ISO 26262 [ISO11].

Durch eine Konsolidierung der HU und der Kombiinstrumente müssen die Anforderungen auch unter dem Gesichtspunkt einer gemeinsamen Nutzung der Hardwareressourcen erfüllt sein. Ein grundlegendes Systemmodell für einen Infotainment-Domänenserver unter Verwendung von Virtualisierung würde wie in

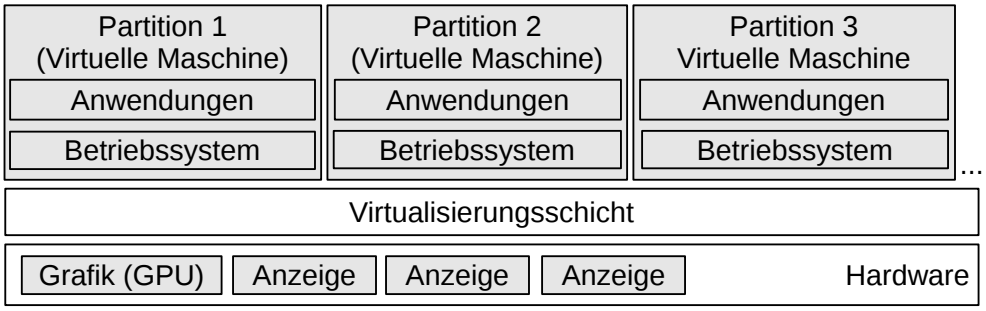

Abbildung 1.4.: Systemmodell eines virtualisierten Infotainment-Domänenservers

Abbildung 1.4 dargestellt aussehen. Die Funktionalität der HU und die der Kombiinstrumente sind voneinander isoliert in jeweils verschiedenen Partitionen gekapselt. Des Weiteren können Anwendungen, welche nicht direkt vom OEM zu Verfügung gestellt werden, in einer separaten Partition betrieben werden.

Im Folgenden werden die Ziele und Fragestellungen dieser Arbeit beschrieben.

1.1. Ziele und Fragestellungen

Das Ziel dieser Arbeit ist die Erstellung von Konzepten und einer Architektur für einen *Infotainment-Domänenserver*, der mittels der Technologie Virtualisierung eine Konsolidierung der HU und der Kombiinstrumente ermöglicht und einen sicheren Zugriff auf die grafischen Hardwareressourcen bietet. Dies soll zum einen die Vorteile einer Konsolidierung – den Bauraumbedarf, den Stromverbrauch und die Hardwarekosten zu reduzieren – bringen, zum anderen soll dadurch eine flexiblere Nutzung der Ressourcen, insbesondere der Anzeigen ermöglicht werden. Hierbei stehen insbesondere die Mechanismen für einen sicheren und isolierten Zugriff der sicherheitskritischen und nicht-sicherheitskritischen Anwendungen, welche sich in unterschiedlichen Partitionen befinden, auf die Graphics Processing Unit (GPU) und die Anzeigen im Fokus dieser Arbeit.

Existierende Virtualisierungslösungen bieten bereits Konzepte [Kai07] für die gemeinsame Nutzung der Hardwareressourcen durch Funktionalitäten mit unterschiedlicher Kritikalität. Die Virtualisierungslösung PikeOS [Pik15] beispielsweise ist für den Bereich der Flugzeugentwicklung zertifiziert, die höhere Sicherheitsanforderungen hat als der Automobilbereich. Dennoch gibt es Unterschiede in den Anforderungen der Automobilentwicklung zur Flugzeugentwicklung.

Es zeigt sich im Bereich der grafischen Funktionen und Anwendungen, dass die bestehenden Virtualisierungslösungen die gemeinsame Nutzung der Grafikressourcen meist unzureichend abdecken, was zu gravierenden Problemen bei der

Darstellung sicherheitskritischer Funktionalitäten führen kann. Dies liegt zum einen daran, dass die heutigen GPUs vor allem auf Leistung optimiert sind und nur unzureichend den gemeinsamen Zugriff auf ihre Ressourcen kontrollieren.

Zum anderen bieten die bestehenden grafischen Systeme, wie sie auf den heutigen Desktop-Systemen eingesetzt werden (z. B. X11 [GP04]), keine ausreichenden Konzepte für eine sichere und flexible Darstellung der sicherheitskritischen bzw. nicht-sicherheitskritischen Anwendungen auf gemeinsam genutzte Anzeigen. Hierfür bedarf es Konzepte, welche es ermöglichen, sowohl die GPU-Ressourcen als auch die digitalen Anzeigebereiche entsprechend den Anforderungen zwischen den sicherheitskritischen und nicht-sicherheitskritischen Anwendungen zu teilen. Die daraus entstehenden Fragestellungen werden im Folgenden dargelegt.

Die Kritikalität der Anwendungen kann sicherheitskritisch und nicht-sicherheitskritisch sein, was folglich bei einer gemeinsamen Nutzung der Ressourcen klare Regeln des Zugriffs und der Nutzung erfordert. Die Fragestellung lautet, wie die Architektur eines *Infotainment-Domänenservers* sein muss, damit alle Anforderungen erfüllt werden, welche an die HU oder an die Kombiinstrumente gestellt werden. Es gibt unterschiedliche Anforderungen, die bei der Entwicklung beachtet werden müssen. Neben den staatlichen Vorgaben in Form der StVZO [Jan11], welche als Rechtsverordnung die Zulassung von Fahrzeugen für den Verkehr regelt, gibt es ISO Richtlinien und Richtlinien der Automobilindustrie, welche Vorgaben für den Entwicklungsprozess machen. Des Weiteren legen die OEMs selbst Anforderungen fest, um sowohl die Qualität, als auch die Innovationen ihrer Systeme voranzutreiben. Ein Infotainment-Domänenserver muss folglich also diese unterschiedlichen Anforderungen berücksichtigen, was insbesondere mit Fokus auf die sicherheitskritischen grafischen Anwendungen zu analysieren ist.

Neben den bereits genannten Vorteilen einer Konsolidierung ergibt sich ein weiteres Teilziel, das insbesondere auf die Nutzung der Anzeigen abzielt. Die Zunahme der Anzeigen im Fahrzeug lässt nicht zwangsläufig eine flexible Nutzung der Anzeigen seitens des Kunden zu, da eine strikte Separierung bzw. Zuordnung der Funktionalitäten nach Sicherheitskritikalität zu den Anzeigen erforderlich ist. Die sichere Darstellung von sicherheitskritischen Inhalten muss auf den gemeinsam genutzten Anzeigen im Fahrzeug gewährleistet sein. Dennoch soll eine flexible Nutzung seitens des Fahrers bzw. der Passagiere möglich sein, sodass für bestimmte Funktionalitäten sich die Wahl der Darstellung auf den Anzeigen steuern lässt.

Somit ergibt sich die Fragestellung, wie die Beeinflussung von Funktionalitäten mit hoher Kritikalität durch nicht-vertrauenswürdige oder bösartige Anwendungen vermieden werden kann. Hierfür bedarf es geeigneter Konzepte, welche die

Nutzung der Anzeigebereiche regelt und den Zugriff auf die Pixel kontrolliert. Liegen die grafischen Inhalte der Anwendungen vor und sind die Anzeigebereiche zugeordnet, dann müssen diese Inhalte auf den Anzeigen dargestellt werden.

Da einige Anwendungen für die Gewährleistung der funktionalen Sicherheit relevant sind, müssen alle Anzeigebereiche entsprechend der festgelegten Zugriffe dargestellt werden. Dies bedeutet, dass z. B. die Darstellung des Warnsymbols für die Störung der Bremsanlage an der festgelegten Position vollständig sichtbar sein muss und nicht verfälscht sein darf. Dies kann beispielsweise mit Bitmasken-Vergleichen überprüft werden, wie es die i.MX6 Plattform für einzelne Anzeigebereiche mittels eines Integritätsprüfers [Fre] für die Sicherheitsanforderungsstufe ASIL B [ISO11] ermöglicht. Die in den Anzeigebereichen darzustellenden grafischen Inhalte der Anwendungen müssen dann zu einem Gesamtbild zusammengesetzt (genannt Compositing) werden. Da die Steuergeräte, welche im Bereich Infotainment eingesetzt werden, wesentlich weniger Leistung bei der Darstellung von grafischen Inhalten als Desktopsysteme bieten, ergibt sich die Notwendigkeit eines effizienten Compositing-Verfahrens, das die Ressourcen des Systems schont.

Anwendungen in grafischen Fenstersystemen zeichnen nach dem Registrieren ihrer Fenster ihre Inhalte in der Regel in Puffer, welche dann für die Darstellung auf den Anzeigen in den Speicher der Anzeigen übertragen werden müssen. Für das Compositing wird zwischen zwei verschiedenen Ansätzen unterschieden.

Der erste Ansatz und gleichzeitig das gebräuchliche Vorgehen der gängigen Fenstersysteme ist die Verwendung von rechteckigen Fenstern. Dies bedeutet, das Fenster, in welchem eine Anwendung ihren grafischen Inhalt darstellen kann, besitzt eine rechteckige Form, die sich mittels Positionsangabe, Breite und Höhe einfach beschreiben lässt. Entsprechend kopiert das Compositing auf effizienter Art und Weise mittels Bitblitting – hardware-unterstütztes Kopieren des Puffers in den Anzeigespeicher – den rechteckigen grafischen Inhalt der Anwendungen aus dem Puffer in den Anzeigespeicher. Können Fenster sich überdecken, d. h. neben der zwei-dimensionalen Position auf der Anzeige kommt eine weitere Dimension in z-Richtung hinzu, dann muss das Compositing dies berücksichtigen.

Es gibt zwei Verfahren, welche im Allgemeinen in den Fenstersystemen eingesetzt werden. Das *fenster-basierte* Compositing kopiert die Fenster entsprechend der z-Richtung, um die Überdeckung der Fenster zu berücksichtigen. Das *kachel-basierte* Compositing kopiert nur den sichtbaren Teil (Kacheln) der Fenster, wobei die Reihenfolge dann keine Rolle spielt. Da beide Ansätze nicht optimal sind, ist die Fragestellung, wie das Compositing optimiert werden kann.

Im Folgenden werden die wissenschaftlichen Beiträge beschrieben.

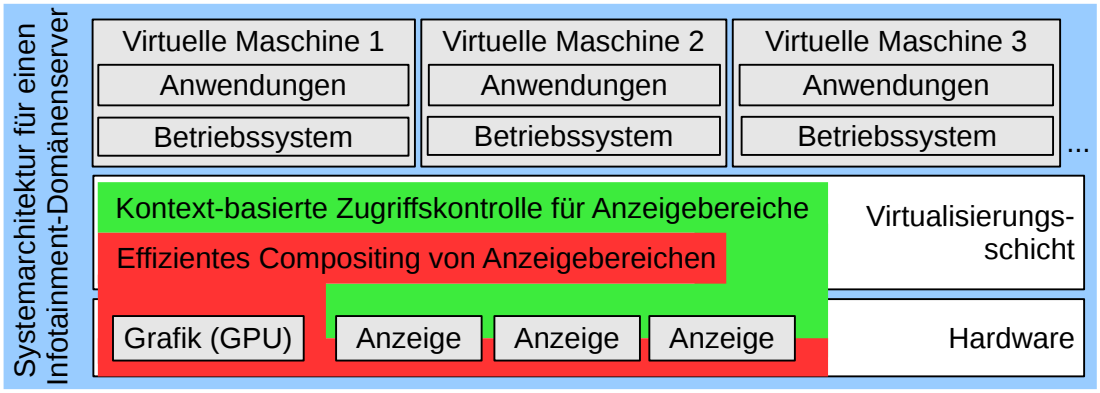

Abbildung 1.5.: Einordnung der Themen in das virtualisierte System

1.2. Wissenschaftlicher Beitrag

Zu den genannten Zielen aus Abschnitt 1.1 werden in diesem Abschnitt die wissenschaftlichen Beiträge dieser Arbeit beschrieben. Legt man den Beiträgen die Abbildung 1.5 des Systemmodells für einen virtualisierten Infotainment-Domänenserver zugrunde, dann liegen die Schwerpunkte der Beiträge im Systemmodell, wie in Abbildung 1.5 dargestellt. Während die Systemarchitektur des Infotainment-Domänenservers die Gesamtarchitektur basierend auf Virtualisierung darstellt, stellen die Konzepte für die kontext-basierte Zugriffskontrolle für Anzeigebereiche und das effiziente Compositing von Anzeigebereichen Schnittstellen für den Zugriff auf die Grafikhardware und Anzeigen dar.

Im Folgenden werden die einzelnen Themen ausführlicher beschrieben.

Anforderungen und Systemarchitektur eines Infotainment-Domänenserver

Für den Entwicklungsprozess eines Fahrzeugs sind unterschiedliche Anforderungen relevant, die sich aus ISO-Standards, rechtlichen Anforderungen, Automobilrichtlinien und OEM-spezifische Anforderungen ergeben. Im Rahmen dieser Arbeit wurde eine Analyse der unterschiedlichen Anforderungen durchgeführt, die die Entwicklung von HMI-Systemen im Fahrzeug beeinflussen. Daraus wurden sieben Anforderungen abgeleitet, die sich in mehrere Teilanforderungen aufteilen lassen. Auf Grundlage dieser sieben Anforderungen wurde eine Systemarchitektur entwickelt, die als Infotainment-Domänenserver bezeichnet wird. Sie konsolidiert unter anderem die Funktionalitäten der HU und der Kombiinstrumente und gewährleistet eine Isolation von sicherheitskritischen und nicht-sicherheitskritischen Anwendungen auf Grundlage der Technologie Virtualisierung und verschiedener

Komponenten für den sicheren Zugriff auf die Grafikressourcen.

Kontext-basierte Zugriffskontrolle für Anzeigebereiche

Es wurde ein formales Modell definiert, das die Darstellung von sicherheitskritischen und nicht-sicherheitskritischen Inhalten an den Kontext des Fahrzeugs und der Anwendungen knüpft, indem Berechtigungen für Anzeigebereiche abhängig von Kontexten vergeben werden. Der Zugriff erfolgt auf der Granularität einzelner Pixel. Um die Flexibilität der Verwendung der Anzeigebereiche durch das Zugriffskontrollmodell zu erhöhen, können Berechtigungen an andere Anwendungen weitergegeben werden, wenn die vergebenen Berechtigungen nicht mehr zulassen, als die vergebende Anwendung selbst darf. Dadurch können ungenutzte Anzeigebereiche durch andere Anwendungen verwendet werden. Durch die Hierarchie der Vergabe kann jederzeit die vergebene Berechtigung entzogen werden, wobei die Vergabe selbst eine Delegationsbeziehung zwischen den Anwendungen erfordert. Mittels formal definierten Eigenschaften, wie der konfliktfreie Zugriff auf Anzeigebereiche, kann mathematisch bewiesen werden, dass die Eigenschaften durch die Verwendung eines Protokolls nicht verletzt werden können, wenn die initial vergebenen Berechtigungen die Eigenschaften nicht verletzen. Mittels geeigneter Evaluationen kann zudem gezeigt werden, dass die zusätzliche Latenz durch die Verwendung der Berechtigungen im erfüllbaren Rahmen der Anforderungen liegt.

Effizientes Compositing von Anzeigebereichen

Für das Compositing von Anzeigebereichen wurde eine hybride Strategie (Hybrid-Compositing) entwickelt, welche die Nachteile der beiden vorwiegend eingesetzten Ansätze vermeidet. Hierzu wird für jedes Fenster, das überlagert wird, berechnet, ob das Kopieren des gesamten Fensters effizienter ist oder nur dessen sichtbare Teile. Dazu wurde ein Vorhersagemodell entworfen, das die benötigte Ausführungszeit für das Bitblitting abschätzt. Zur Optimierung der Strategie (Cache-Hybrid-Compositing) wird zusätzlich ein Cache verwendet, welche die bereits getroffenen Entscheidungen speichert und für die spätere Verwendung bereithält.

Eine weitere Variante erlaubt es, beliebige Fensterformen zu definieren und auf Granularität einzelner Pixel anzuwenden. Für die Definition der Fenster werden Bitmasken verwendet, welche auf Basis einzelner Pixel angeben, welche Pixel für das Fenster geschrieben werden. Das Compositing verwendet entsprechend die Bitmasken dann beim Bitblitting. Durch die Verwendung der pixel-definierten

Fenster wird in Kombination mit dem Zugriffskontrollmodells für Anzeigebereiche ein Überdecken der Fenster verhindert, da die Definition der Fenster disjunkt erfolgt. Somit ergibt sich eine Durchgängigkeit der Zugriffskontrolle für Anzeigebereiche von der Fensterregistrierung bis zur Darstellung auf der Anzeige.

Demonstrator Virtualized-Car-Telematics (VCT)

Im Rahmen des Projektes ARAMiS wurde ein Demonstrator [RAL+15] erstellt, welcher die Implementierung der wesentlichen Konzepte der zuvor beschriebenen Beiträge demonstriert. Zu diesem Zweck wurde die Systemarchitektur des Infotainment-Domänenservers basierend auf der Virtualisierungslösung PikeOS von Sysgo [Pik15] auf einer Freescale i.MX6 SABRE [Fre15] Plattform implementiert. Durch Anbinden zweier Anzeigen an die Plattform können die Anwendungen der HU und der Kombiinstrumente, sowie Anwendungen von Drittanbietern dargestellt werden, welche mittels Berechtigungen der Zugriffskontrolle geregelt Zugriff auf die Anzeigen erhalten. Das Compositing gewährleistet entsprechend den Berechtigungen eine konfliktfreie Darstellung auf den Anzeigen. Des Weiteren wurde eine Anbindung der Eingabegeräte des Lenkrads und des Dreh-Drückstellers durchgeführt, sodass die Anwendungen mittels dieser Eingabegeräte gesteuert werden können. Hierbei wurde ebenfalls eine Zugriffskontrolle der Eingabeereignisse durchgeführt. Mittels Anwendungen der HU und der Kombiinstrumente, sowie Anwendungen von Drittanbietern wurden Szenarien im Automobil erstellt, die die Flexibilität und Leistungsfähigkeit der entwickelten Konzepte aufzeigen.

Weitere verwandte wissenschaftliche Beiträge

Zusätzlich zu den vorgestellten wissenschaftlichen Beiträgen, wurden weitere verwandte Beiträge zur Gesamtarchitektur geleistet. Diese Beiträge beschäftigen sich mit der Ressourcenverwaltung der GPU, welche eine gemeinsam genutzte Ressource darstellt und sowohl von sicherheitskritischen als auch nicht-sicherheitskritischen Anwendungen genutzt werden kann. Da weder die Ausführungsdauer eines Grafikbefehls auf der GPU bekannt ist und beliebig lange sein kann, noch die GPU während der Verarbeitung eines Grafikbefehls unterbrochen werden kann, ist für eine Verteilung der Rechenzeit ein nicht-präemptives Scheduling erforderlich. Hierzu ist eine Abschätzung der Ausführungszeit von Grafikbefehlen auf der GPU notwendig, welche dann für ein Scheduling der Befehle genutzt wird. Eine weitere Veröffentlichung ist in Planung, die sich insbesondere mit einer verbesserten Abschätzung der Laufzeit der Grafikbefehle mittels Machine-Lerning befasst.

Wesentliche Teile dieser Arbeit wurden bereit in [GSD+13,GSGH+14,GSC+15, GSGH+15] veröffentlicht. Des Weiteren wurden als weitere verwandte Beiträge [SGDR14,SGDR16,RAL+15] veröffentlicht. Die Veröffentlichungen wurden alle in Zusammenarbeit mit anderen Autoren erstellt. Herr Professor Dr. Kurt Rothermel und Dr. Frank Dürr der Universität Stuttgart, sowie Dr. Christian Maihöfer sorgten durch ihre Anmerkungen und Rückmeldungen zu den Themen und insbesondere der inhaltlichen Struktur für die Qualität der Veröffentlichungen. Die weiteren Anteile an diesen Veröffentlichungen sind wie folgt.

Im Hinblick auf das Thema „Anforderungen und Systemarchitektur eines Infotainment-Domänenservers“ wurde die wissenschaftliche Publikation [GSD+13] veröffentlicht. In Zusammenarbeit mit Stephan Schnitzer wurde eine Analyse der Anforderungen durchgeführt und wesentliche Anforderungen abgeleitet, die zu einer Systemarchitektur für HMI-Systeme im Automobil führten. Der eigene geleistete Anteil an dieser Arbeit betrug ca. 45 %.

Zu dem Thema „Zugriffskontrolle von Anzeigebereichen“ wurden zwei wissenschaftliche Publikationen [GSGH+14] und [GSGH+15] veröffentlicht. Teile der Publikation [GSGH+14] wurden im Rahmen einer Diplomarbeit [GH13] von Ahmad Gilbeau-Hammoud bearbeitet, welcher unter anderen auch Autor der Publikation [GSGH+14] ist. Stephan Schnitzer trug insbesondere durch seine Ratschläge zur Entwicklung der Konzepte bei und sorgte durch seine Rückmeldungen für eine Optimierung der sprachlichen Qualität der Veröffentlichungen. Ebenso half Viktor Friesen die Lesbarkeit der Veröffentlichungen zu verbessern, welches insbesondere die formalen Definitionen betraf. Ulrich Krämer half mit Rückmeldungen zur technischen Umsetzung für eine Optimierung der Konzepte in Publikation [GSGH+15]. Der eigene geleistete Anteil an den beiden Publikationen [GSGH+14] und [GSGH+15] betrug jeweils ca. 75 %.

Für die Hybrid-Compositing-Strategie wurde die wissenschaftliche Publikation [GSC+15] veröffentlicht. Als Grundlage der Publikation [GSC+15] diente die Masterarbeit [Cec14] von Riccardo Cecolin, welcher ebenfalls als Autor an der Publikation beteiligt war. Stephan Schnitzer verbesserte durch seine Rückmeldungen die sprachliche Qualität der Veröffentlichung. Der eigene geleistete Anteil an dieser Arbeit betrug ca. 75 %.

Der Demonstrator (VCT) im Projekt ARAMiS wurde in [RAL+15] veröffentlicht. Der eigene geleistete Anteil an dieser Arbeit betrug ca. 5 %.

Zum Thema „Abschätzung der Ausführungszeit von Grafikbefehlen“ wurde die wissenschaftliche Publikation [SGDR14] veröffentlicht. Des Weiteren wurde zum Thema „Echtzeit-Scheduling für 3D-GPU-Rendering“ die wissenschaftliche Publi-

kation [SGDR16] veröffentlicht. Der eigene geleistete Anteil an diesen Arbeiten betrug jeweils ca. 10 %.

1.3. Projekt ARAMiS

Da die wesentlichen Konzepte dieser Arbeit im Rahmen des öffentlich geförderten Projekts ARAMiS [Ara15] durchgeführt wurden, wird in diesem Abschnitt ein kurzer Überblick über das Projekt und dessen Ziele gegeben. Die wesentlichen Inhalte stammen aus der Vorhabensbeschreibung des Projektes ARAMiS. Im Rahmen des Projektes ist in Zusammenarbeit mit den Projektpartnern in ARAMiS, insbesondere der Firmen Freescale, Sysgo, OpenSynergy und der Universität Stuttgart, ein Demonstrator entstanden, der die Konzepte dieser Arbeit demonstriert. Das Projekt ARAMiS ist ein gefördertes Projekt des Bundesministeriums für Bildung und Forschung (BMBF) und steht für Automotive, Railway and Avionics Multicore Systems. Ziel des Projektes ist es, durch die Verwendung der Multicore-Technologie eine technologische Basis zur Erhöhung der Sicherheit, der Verkehrseffizienz und des Komforts in den drei Mobilitätsdomänen Automobil, Avionik und Bahn zu schaffen. Die gewonnenen Erkenntnisse, die sich aus dem Projekt ergaben, sollen die Grundlage für die Vernetzung von eingebetteten Systemen zu Cyber Physical Systems (CPS) bilden. Durch dieses Projekt wurde ein Beitrag für den Erhalt bzw. zur Stärkung der deutschen Unternehmen im weltweiten Wettbewerb in den Domänen Automobil, Avionik und Bahn geleistet.

Das Projekt hatte eine Gesamtlaufzeit von drei Jahren, zuzüglich einer Verlängerung um vier Monate. Offiziell startete das Projekt im Dezember 2011 und lief bis März 2015. Mit einem geplanten Budget von ca. 36 Millionen Euro war ARAMiS ein vergleichsweise großes öffentlich gefördertes Projekt.

Paradigmenwechsel durch den Einsatz der Multicore-Technologie

Seit einiger Zeit werden bereits Multicore-Prozessoren anstatt von Singlecore-Prozessoren in PCs und Servern eingesetzt. Dies liegt in erster Linie an der Leistungsgrenze der Singlecore-Prozessoren, die sich nur noch in geringem Maße weiter anheben lässt. Diese Entwicklung ist ebenfalls bei den eingebetteten Systemen zu beobachten, die einen Anteil von über 90 % der weltweit eingesetzten Prozessoren ausmachen. Der relative Anteil der in den Domänen Automobil, Avionik und Bahn eingesetzten Prozessoren betrug 2009 jedoch noch unter 10 % [Cor09]. Dadurch konzentriert sich die Entwicklung der Multicore-Prozessoren eher an

absatzstarken Märkten wie der Kommunikations- oder Entertainmentbereich.

Für den Einsatz von Multicore-Systemen in den Domänen Automobil, Avionik und Bahn müssen einige funktionale und nicht-funktionale Anforderungen erfüllt werden, die sich aus verschiedenen ISO-Standards, rechtlichen Einschränkungen und OEM-spezifischen Vorgaben ergeben. Dies sind insbesondere: Echtzeitfähigkeit, Leistungsfähigkeit, Zuverlässigkeit und Verfügbarkeit, zertifizierbare Funktionssicherheit (Safety), Sicherheit gegen Angriffe (Security), Kompatibilität zu bestehenden Konzepten und Energieeffizienz.

Die Übertragung bereits existierender Lösungen für Multicore-Systeme aus anderen Bereichen, wie Entertainment, ist hierbei nur bedingt möglich. Ebenfalls können die bereits entworfenen Systeme und Konzepte für Singlecore-Systeme nur sehr eingeschränkt für neue Multicore-Systeme eingesetzt werden, da sich aufgrund der Parallelisierbarkeit erheblich komplexere Systemzustände ergeben können, welche die Sicherstellung der genannten Anforderungen erschweren.

Hieraus ergibt sich die Notwendigkeit spezialisierte Architekturen und Methoden zu entwerfen, welche die Funktionalitäten mit ihren unterschiedlichen Kritikalitäten anforderungsgemäß auch auf Multicore-Systemen ausführbar machen.

Virtualisierung in Multicore-Systemen

Die Technologie Virtualisierung bietet Schutzmaßnahmen, um die Funktionalitäten entsprechend ihrer Kritikalitäten zu kapseln und eine gegenseitige Beeinträchtigung zu verhindern. Dieser Ansatz hat bereits weite Verbreitung im Bereich der Rechenzentren gefunden, lässt aber aufgrund der teils sehr unterschiedlichen Ausrichtung keine direkte Übertragung auf die eingebetteten Systeme der Domänen Automobil, Avionik und Bahn zu. Es gilt hierbei die geforderten Anforderungen dieser Domänen in geeigneter Weise mittels Virtualisierung umzusetzen und dabei die ungelösten bzw. neuen Herausforderungen zu lösen. Insbesondere stellt sich die Herausforderung, den Anwendungen mit unterschiedlicher Kritikalität, trotz der Isolation, eine gemeinsame Nutzung von Hardware-Ressourcen zu ermöglichen. Diesen Konflikt zu lösen, liegt im Fokus des Projektes.

Ziel in der Domäne Automobil

Für die Domäne Automobil ergibt sich aus Sicht des Projekts die Zielsetzung der optimalen Nutzung von Multicore-Systemen unter Berücksichtigung der automobil-spezifischen Rahmenbedingungen der verschiedenen Fahrzeugdomänen (Infotainment, Fahrdynamik, Chassis, Antriebsstrang, ...). Hierbei muss die ISO

26262 [ISO11] bei der Entwicklung von Sicherheitsarchitekturen berücksichtigt werden. Des Weiteren werden Konzepte zur effizienten Parallelisierbarkeit von Anwendungen mit unterschiedlicher Kritikalität, sowie Konzepte zur Sicherstellung von Aspekten der Sicherheit und die Unterstützung der Softwareentwicklung durch einen geeigneten Entwicklungsprozess (z. B. konform zur Entwicklung von AUTOSAR [Aut]) betrachtet. Entscheidend ist, dass die Isolation der Funktionen gewährleistet bleibt, was ein Ziel der Technologie Virtualisierung ist. Damit soll es ermöglicht werden, eine neue energieeffiziente Architektur für das Bordnetz, die auf zentralisierten Einheiten (Domänen-Servern) aufbaut, zu erschaffen und neue hochvernetzte Funktionen in den Fahrzeugdomänen aufzubauen.

Struktur des Projektes

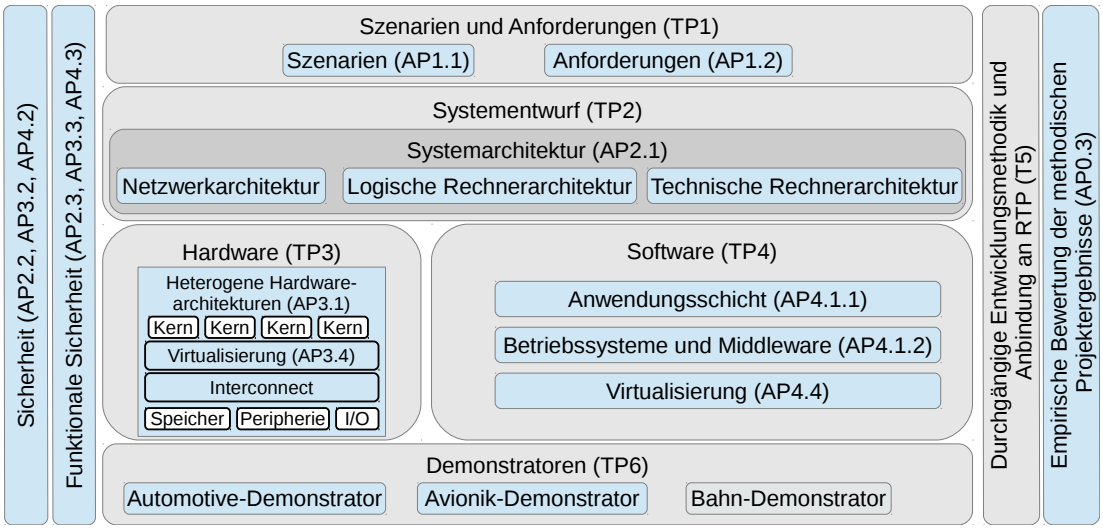

Abbildung 1.6.: Projektstruktur in ARAMiS

Die Projektstruktur setzt sich aus einzelnen Teilprojekten (TPs) und Arbeitspaketen (APs) zusammen. Der Ablauf des Projekts ist in der Mitte der Abbildung 1.6 dargestellt und beginnt mit den Szenarien und Anforderungen, über den Systementwurf und der Hardware- und Softwareentwicklung bis hin zu den Demonstratoren. Aspekte der funktionalen Sicherheit (engl. Safety) und Sicherheit (engl. Security) sowie Zertifizierbarkeit sind nicht den einzelnen Modulen zuordenbar, sondern müssen durchgängig über alle Ebenen betrachtet werden (vgl. Abbildung 1.6 links). Dasselbe gilt für die Methoden und Werkzeuge, welche ebenfalls übergreifend betrachtet werden müssen (vgl. Abbildung 1.6 rechts).

Im Folgenden werden der Fokus und die Beiträge der einzelnen TPs beschrieben.

In TP1 – Szenarien und Anforderungen – wurden die relevanten Szenarien für Multicore-Systeme in den Domänen festgelegt. Aus diesen Szenarien wurden dann funktionale und nicht-funktionale Anforderungen abgeleitet, die sich für Multicore-Systeme und dem Einsatz von Virtualisierung ergeben. In TP2 – Systementwurf – wurde auf Grundlage der zuvor ermittelten Szenarien und Anforderungen ein System erstellt. Hierbei wurden die Anforderungen der funktionalen Sicherheit unter Anwendung der Technologie Virtualisierung betrachtet. Der Entwurf wurde als Grundlage für die folgenden TPs 3, 4 und 5 verwendet. In TP3 – Hardware – wurde auf Grundlage der Anforderungen aus TP2 Hardware entworfen. In TP4 – Software – wurden die Softwarearchitekturen unter Einsatz der Technologie Multicore und Virtualisierung erstellt, welche für die Erfüllung der Anforderungen notwendig sind. In TP5 – Durchgängige Entwicklungsmethodik und Werkzeuge – wurden Werkzeuge entworfen, welche die Erstellung von Multicore-System durchgängig unterstützen. In TP6 – Demonstratoren – wurden in den einzelnen Domänen Demonstratoren erstellt (zwei je Domäne Automobil und Avionik, sowie ein Demonstrator in der Domäne Bahn), die der Validierung der Konzepte und der allgemeinen Machbarkeit der Umsetzung von Multicore-Systemen mit Virtualisierung dienten. Es wurden insbesondere die Szenarien und Anforderungen aus TP1 demonstriert bzw. validiert, was die Durchgängigkeit des Projekts darstellen soll. Die Gesamtkoordination des Projekts erfolgte durch TP0. Umfassend wurde ebenfalls die finale Aufbereitung und Verteilung der Projektergebnisse, sowie eine empirische Evaluierung des Projekterfolgs von TP0 durchgeführt.

Die maßgeblichen Beiträge dieser Arbeit wurden im Rahmen von TP1, TP2, TP4 und TP6 erstellt. In TP1 wurden die relevanten Anforderungen an ein grafisches Fenstersystem analysiert und definiert und in TP2 die Systemarchitektur erstellt. In TP4 wurden die wesentlichen Konzepte für den gemeinsamen Zugriff auf die Ressourcen aus Sicht der grafischen Anwendungen mit unterschiedlicher Kritikalität entworfen und das effiziente Compositing entwickelt. In TP6 entstand ein Demonstrator, der die Machbarkeit der Konzepte aufzeigt.

Partner im Projekt ARAMiS

In ARAMiS kooperierten Hersteller aus den Mobilitätsdomänen, insbesondere der Domäne Automobil und Avionik, deren Zulieferer, sowie Halbleiterhersteller, Softwarehersteller und Forschungseinrichtungen. Dies ermöglichte über die gesamte Entwicklungsprozesskette eine Entwicklung neuer Konzepte, die durch die Kompetenz der unterschiedlichen Partner getragen wurde.

Ergebnisse des Projekts ARAMiS

Die Ergebnisse des Projekts ARAMiS sind vielfältiger Natur. Rein nach Fakten beurteilt, wurden innerhalb dieser drei Jahre über 120 Dokumente erstellt, die die Ergebnisse der Teilprojekte und die entwickelten Konzepte enthalten. Dazu kommen über 80 wissenschaftliche Veröffentlichungen und Konferenzbeiträge. Nicht zuletzt zeigen die fünf Demonstratoren in den Domänen die Machbarkeit der entwickelten Konzepte und die neuen Möglichkeiten der Nutzung auf.

Mit Fokus auf dieser Arbeit zeigt das Projekt ARAMiS, dass eine sichere und zuverlässige Nutzung der Multicore-Technologie unter Anwendung der Virtualisierung möglich ist. Die Nutzung gemeinsamer Ressourcen durch Anwendungen mit unterschiedlicher Kritikalität bedarf Konzepte für den konfliktfreien Zugriff, die im Rahmen des Projekts unter Berücksichtigung der Anforderungen, entwickelt wurden. Insbesondere wurde die Nutzung der grafischen Ressourcen betrachtet. In dieser Arbeit werden im Folgenden diese Konzepte und Mechanismen für die Darstellung von sicherheitskritischen Informationen im Fahrzeug betrachtet.

1.4. Struktur der Arbeit

Die weitere Struktur der Arbeit ist wie folgt aufgebaut. In Kapitel 2 werden die Anforderungen eines Infotainment-Domänenservers auf Grundlage des Systemmodells aus den verschiedenen International Organization for Standardization (ISO)-Standards, Richtlinien und rechtlichen Einschränkungen abgeleitet. Des Weiteren wird eine Systemarchitektur eines Infotainment-Domänenservers definiert, die die Anforderungen berücksichtigt bzw. zu deren Erfüllung beiträgt. Darauf aufbauend wird dann in Kapitel 3 eine kontext-basierte Zugriffskontrolle für Anzeigebereiche definiert, die zwischen den Anwendungen der verschiedenen virtuellen Maschinen und den Anzeigen steht und festlegt, welche Anwendung auf welche Anzeigebereiche zugreifen darf. In Kapitel 4 wird dann das effiziente Compositing von Anzeigebereichen, welches dann die erstellten Fensterinhalte auf die Anzeigen bringt, vorgestellt. Der Demonstrator Virtualized-Car-Telematics (VCT), welcher in Kapitel 5 beschrieben wird, setzt einen virtualisierten Infotainment-Domänenserver um, der die Konzepte der drei Schwerpunkte der Beiträge demonstriert. Das Kapitel 6 fasst die Arbeit zusammen und gibt einen Ausblick auf weitere Themen, welche wissenschaftlich betrachtet werden können.

2. Anforderungen und Systemarchitektur eines Infotainment-Domänenservers

Für die Verwendung eines Infotainment-Domänenservers ergeben sich Besonderheiten, wie gemeinsam genutzte GPU und Anzeigen durch HU und Kombiinstrumente. In diesem Kapitel wird eine Analyse der Anforderungen durchgeführt, die sich aus den verschiedenen Rahmenrichtlinien im Automobilbereich wie ISO-Standards oder gesetzlichen Vorgaben wie die StVZO [Jan11] ergeben. Es werden hierbei sieben Anforderungen abgeleitet, welche sich teils weiter aufgliedern lassen. Unter Anwendung der ermittelten Anforderungen wird dann eine Systemarchitektur für einen Infotainment-Domänenserver abgeleitet, welche aus verschiedenen Komponenten besteht, die die Einhaltung der Anforderungen gewährleisten.

Im folgenden Abschnitt werden zuerst die Anforderungen für die HMI-Systeme im Fahrzeug gefolgt von der Systemarchitektur beschrieben.

2.1. Anforderungen an HMI-Systeme im Fahrzeug

Die Fahrzeugentwicklung wird durch verschiedene Anforderungen bestimmt, welche die OEMs in ihren Entwicklungsprozessen beachten müssen. Die Anforderungen setzen sich aus Rahmenrichtlinien wie ISO-Standards, rechtlichen Anforderungen, Automobilrichtlinien und durch den OEM gegebene Anforderungen zusammen. In Abbildung 2.1 sind die unterschiedlichen Rahmenrichtlinien dargestellt und Beispiele dafür angegeben, welche die Fahrzeugentwicklung beeinflussen.

Eine Besonderheit ist, dass die Automobilrichtlinien (z. B. AAM 2006 [AAM06], ESoP 2008 [ESO08], JAMA 2004 [JAM04]), welche die Human-Machine-Interface (HMI)-Systeme im Fahrzeug betreffen, beinahe vollständig von ISO-Standards abgeleitet werden können, so dass im Folgenden nur auf die entsprechenden ISO-

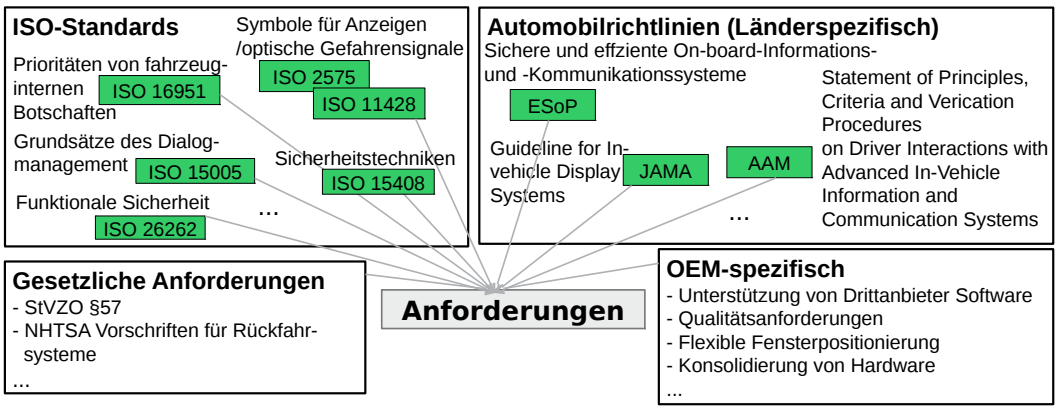

Abbildung 2.1.: Darstellung von Anforderungen für die Fahrzeugentwicklung

Standards verwiesen wird. An ISO-Standards sind die Folgenden relevant:

- ISO 11428 [ISO96] Ergonomie - Optische Gefahrensignale - Allgemeine Anforderungen, Gestaltung und Prüfung.
- ISO 15005 [ISO02] Straßenfahrzeuge - Ergonomische Aspekte von Fahrer-informations- und -assistenzsystemen - Grundsätze und Prüfverfahren des Dialogmanagements.
- ISO 16951 [ISO04] Straßenfahrzeuge - Ergonomische Aspekte von Verkehrs-informations - Prozeduren zur Prioritätsbestimmung für fahrzeuginterne Botschaften an den Fahrer.
- ISO 2575 [ISO10] Straßenfahrzeuge - Symbole für Bedienteile, Anzeigen und Warnleuchten.
- ISO 15408-2 [ISO08] Informationstechnik - IT-Sicherheitsverfahren - Evaluationskriterien für IT-Sicherheit - Teil 2: Funktionale Sicherheitsanforderungen.
- ISO 26262 [ISO11] Straßenfahrzeuge - Funktionale Sicherheit (Teil 1-10).

Der folgende Abschnitt beschreibt sieben Anforderungen für HMI-Systeme im Fahrzeug. Für jede Anforderung wird entsprechend ein Verweis auf die relevanten Abschnitte in den ISO-Standards oder anderen Richtlinien verwiesen.

2.1.1. R1 – Eingabeverarbeitung

R1.1 – Zugriffskontrolle: Bezugnehmend auf ISO 15005 [ISO02] ist eine *Zugriffskontrolle* für die Eingaben des Benutzers erforderlich. Insbesondere Anwendungen (z. B. von Drittanbietern), die nicht unter besonderer Berücksichtigung der ISO 15005 [ISO02] entwickelt wurden, dürfen die folgenden Beschränkungen nicht verletzen. Anwendungen, die Dialoge verwenden, sollen keine Eingaben des

Benutzers erfordern, die das Entfernen beider Hände vom Lenkrad während der Fahrt zur Folge haben (5.2.2.2.2). Zusätzlich soll es möglich sein, dass ein Dialog oder eine Anwendung beendet oder verlassen werden kann (5.3.3.2.1), wenn dies nicht rechtlich oder aufgrund der Verkehrssituation vorgeschrieben ist (5.3.3.2.3).
R1.2 – Beschränkte Verarbeitungszeit: Für die Verarbeitung von Eingabeereignissen soll eine maximale Verarbeitungszeit existieren, welche nicht überschritten werden darf. Dies kann beispielsweise eine Rückmeldung auf Benutzereingaben sein, welche 250 ms nicht überschreitet (5.2.4.2.3).

2.1.2. R2 – Einschränkung der Erstellung und Positionierung von Fenstern

R2.1 – Einschränkung der Sichtbarkeit von Fenstern: Normalerweise verwenden grafische Anwendungen Funktionen, welche in einer API beschrieben werden, um die *Sichtbarkeit von Fenstern* zu ändern, z. B. Fenster zu erzeugen, verstecken oder die Position zu ändern. Diese Funktionalitäten müssen beschränkt werden und Funktionen, welche nicht dazu gedacht sind, dass der Benutzer sie verwendet, unzugänglich gemacht werden (ISO 15005 [ISO02] (5.2.2.2.4)).
R2.2 – Prioritätsbasiertes Anzeigen von Fenstern: Werden mehrere Fenster dargestellt, muss die Wichtigkeit jedes einzelnen Fensters definiert sein. Die Wichtigkeit wird durch Prioritäten repräsentiert, die von Sicherheitsanforderungen und Aspekten der Softwareergonomie abhängen können (5.2.4.2.4) und das System einhalten muss (5.2.4.3.3). Des Weiteren kann sie von der Dringlichkeit und Kritikalität abhängen, die festgelegt werden müssen (ISO 16951 [ISO04] (3.5)). Zusätzlich müssen angemessene Reaktionen (z. B. Verhalten im Falle eines Konfliktes) durchgesetzt werden (ISO 16951 [ISO04] (Annex B)). Darüber hinaus beschränken länderspezifische Anforderungen die Definition der Prioritäten. Es ist z. B. nach deutschem Recht erforderlich, dass die Geschwindigkeitsanzeige sichtbar ist, wenn das Fahrzeug in Bewegung ist (StVZO §57 [Jan11]). Außerdem müssen Informationen auf konsistente Art und Weise dargestellt werden (ISO 15005 [ISO02] (5.3.2.2.1)). Dies heißt, dass z. B. Systemdialoge oder Statusmeldungen wenn möglich immer an derselben Stelle der Anzeige dargestellt werden.
R2.3 – Zeitliche Beschränkungen: Ein HMI-System im Fahrzeug muss es Anwendungen ermöglichen, dem Fahrer innerhalb bestimmter *zeitlicher Beschränkungen* wichtige Informationen zu liefern. Das heißt, dass Fenster, die Informationen darstellen, innerhalb einer zeitlichen Beschränkung sichtbar sein müssen (ISO 15005 [ISO02] (5.2.4.3.4)). Falls die Anwendungen Interaktionen seitens des

Benutzers erfordern, z. B. ein Benutzer wählt einen Radiokanal, dann darf der Informationsfluss das Lenken des Fahrzeugs nicht nachteilig beeinflussen (5.2.4.2.1). Konkret bedeutet dies nach AAM 2006 [AAM06] Abschnitt 2.1, dass der Blick nicht mehr als 2 s von der Straße abgewandt sein darf. Folglich darf jegliche Art von Animation, die eine Ablenkung darstellt, nicht länger als 2 s dauern.

2.1.3. R3 – Vertrauenswürdiger Kanal

R3.1 – Integrität und Vertraulichkeit: Kommunikation ist in einer Umgebung, in welcher die Anwendungen innerhalb einer virtuellen Maschine laufen, unumgänglich. Dies gilt insbesondere für Kommunikation, welche vormals dezidierte Kommunikationshardware verwendete und nun durch eine softwarebasierte Inter-VM-Kommunikation ersetzt wurde.

Laut ISO 15408-2 [ISO08] muss die Kommunikation zwischen Anwendungen und Hardware unter Berücksichtigung von Integrität und Vertraulichkeit stattfinden. Dies gilt insbesondere für die Benutzerdaten (14.5.8.2), aber auch für die Softwarekomponenten, welche die relevante Funktionalität zur Verfügung stellen (17.1.5.3). Alle Anwendungen, welche eine vertrauenswürdige Kommunikation benötigen, müssen in der Lage sein diese nutzen zu können (17.1.5.2).

R3.2 – Authentifikation und Nachweisbarkeit: Selbst zwischen eindeutigen Systemen (17.1.5.1), die an der Inter-VM-Kommunikation teilnehmen, muss die Identifikation gewährleistet sein. Ein vertrauenswürdiger Kanal muss die Unleugbarkeit der Herkunft (8.1.1 und 8.1.6.1-3) und des Empfangs (8.2.1 und 8.2.6.1-3) gewährleisten. Dies erfordert eine Authentifikation und unter Umständen ein kryptografisches Schlüsselmanagement (9.1.1) mit Schlüsselzugriff (9.1.7.1).

2.1.4. R4 – Virtualisierte Grafikberechnung

Da in einem virtualisierten System mehrere unterschiedliche virtuelle Maschinen auf dieselbe GPU geteilten Zugriff haben, muss es eine Isolation geben, um eine unbeabsichtigte, gegenseitige Beeinflussung von Anwendungen zu verhindern.

R4.1 – Prioritäten: Die Fenster der Anwendungen benötigen eine zugewiesene Priorität, die festlegt, wie die Befehle der GPU verarbeitet werden (ISO 15005 [ISO02] (5.2.4.2.4 und 5.2.4.3.3), ISO 15408-2 [ISO08] (15.2.5.1-2 und 15.2.6.1-2)).

R4.2 – Zeitliche Beschränkungen beim Erstellen der grafischen Inhalte: Neben komparativen Anforderungen (wie Prioritäten) müssen ebenfalls absolute zeitliche Anforderungen erfüllt werden. Eine Rückmeldung auf die taktile Eingabe des Fahrers darf 250 ms nicht überschreiten (ISO 15005 [ISO02] (5.2.4.2.3)).

Ähnlich dazu erfordern Gefahrensignale eine konstante Aktualisierungsrate (zwischen 2 Hz und 3 Hz), welche z. B. Blinklichter repräsentieren (ISO 11428 [ISO96] (4.2.2)). Dies erfordert genügend verfügbare Ressourcen der Central Processing Unit (CPU) und der GPU und führt zu einer minimalen Aktualisierungsrate, da die Verzögerung zwischen zwei aufeinanderfolgenden Bildern durch eine obere zeitliche Schranke beschränkt ist. Diese obere zeitliche Schranke muss bekannt sein, um die Wirksamkeit von sicherheitskritischen Meldungen zu bestimmen (ISO 16951 [ISO04] (Annex F)) und sie erlaubt auch die Festlegung der Verzögerungen, nach welcher die Meldungen dargestellt werden sollen (Annex B).

R4.3 – Isolation der GPU-Ressource: Die GPU ist laut ISO 15408-2 [ISO08] eine kontrollierte Ressource, da sie von mehreren Anwendungen gleichzeitig genutzt werden kann. Um unbeabsichtigte, gegenseitige Beeinflussung zu verhindern, muss es möglich sein, bestimmten Anwendungen Garantien bezüglich erforderlichen Ressourcen der GPU wie Verarbeitungsdauer zu geben. Deswegen muss eine Kontrollmöglichkeit bestehen, welche es erlaubt, die Ressourcen der GPU zur Nutzung auf einzelne Fenster, grafische Anwendungen oder virtuelle Maschinen zu verteilen (15.3.6.1 und 15.3.7.1-2).

2.1.5. R5 – Rekonfiguration der Richtlinien

Eine Menge an Berechtigungen, welche die Ereignisse für Benutzereingaben, Anwendungsfenster, das zugehörige Scheduling und die Isolation reguliert, werden *Richtlinien* genannt. Zu jedem Zeitpunkt darf nur genau eine Richtlinie aktiv sein. Richtlinien können jedoch dynamisch zur Laufzeit, aber immer nur abhängig des Zustands des Systems gewechselt werden.

R5.1 – Dynamische Zustandsänderungen: Von ISO 15005 [ISO02] abgeleitet, kann eine *Zustandsänderung* entweder auf Anfrage des Benutzers oder automatisch aufgrund von definierten Regeln seitens des Systems stattfinden. Ein Zustand kann von den aktuell gegebenen Bedingungen des Fahrzeugs wie „Fahrzeug fährt“ abhängen, welche das Deaktivieren von Anwendungen, die nicht dazu bestimmt sind, während der Fahrt genutzt zu werden, nötig machen (5.2.2.2.4).

Andererseits muss ein HMI-System im Fahrzeug genügend Informationen und Warnungen bereitstellen, um dem Benutzer die angedachte Absicht in einem bestimmten Zustand zu vermitteln. Für jede Zustandsänderung müssen spezifische *Fristen* gelten, um die konsistente und akkurate Transition zwischen verschiedenen Zustände bestimmen zu können. Die Definition der Zustände und des Systemverhaltens wird im Detail in ISO 16951 [ISO04] (3.3 und Annex E) erklärt.

R5.2 – Dynamische Richtlinienänderungen: Autorisierte Softwarekomponenten sollen in der Lage sein, Änderungen der Richtlinien zur Laufzeit durchzuführen. Dies beinhaltet die Vergabe und den Entzug von Berechtigungen für derzeit aktive und inaktive Richtlinien. Laut R5.1 gelten für die dynamischen Wechsel der Richtlinien bestimmte Fristen. Der Fahrer soll in der Lage sein, den Wechsel der aktiven Richtlinie durchzuführen, um den Informationsfluss zu verändern (5.3.3.2.3), wenn dies in der Situation anwendbar und erlaubt ist.
R5.3 – Gewährleistung der Darstellung: Die seitens des Systems festgelegten Regeln sollen die Präsentation der rechtlich erforderlichen Meldungen durchsetzen, die aufgrund der Verkehrssituation notwendig sind. Die Präsentation der Meldungen erfordert, dass diese sichtbar und wahrnehmbar sind, insbesondere, wenn die Zustandsänderungen die Aufmerksamkeit des Fahrers erfordern (ISO 15005 [ISO02] (5.3.2.2.2)). Des Weiteren sollen zustandsrelevante Informationen entweder fortlaufend oder auf Anfrage des Fahrers dargestellt werden.

2.1.6. R6 – Zertifizierbarkeit

Die Zertifizierbarkeit ist für den OEM ein essentieller Teil des Softwareentwicklungsprozesses, z. B. durch die Verwendung der Methode FMEA [Sta03]. Der Entwicklungsprozess für zertifizierte Software, im Speziellen mit einem hohen Kritikalitätslevel, ist in der Regel sehr komplex und teuer. Eine Messgröße für die Komplexität ist die Anzahl der Funktionspunkte, die mit der geschätzten Anzahl an Softwarefehlern korreliert [EJ09]. Demzufolge soll ein System entwickelt werden, welches eine einfache Zertifizierung nach ISO 26262 [ISO11] ermöglicht.

2.1.7. R7 – Systemüberwachung

Die Überwachung des Systems fokussiert auf das Aufzeichnen, Erkennen und Reagieren auf Ereignisse, die möglicherweise relevant für die Sicherheit des Systems sind. Hierfür sind unterschiedliche Anforderungen relevant, welche im Folgenden in sieben Anforderungen untergliedert und jeweils im Detail beschrieben werden.
R7.1 – Sicheres Hochfahren des Systems: Abgeleitet von dem ISO-Standard ISO 15408-2 [ISO08] soll es möglich sein, das System *sicher hochzufahren*, um die Integrität des Systems zu gewährleisten. Das Kompromittieren des Systems (14.6.9.1), der Systemgeräte oder der Systemelemente (14.6.9.2) durch jegliche Art physikalischer Manipulation sollen immer eindeutig erkannt werden können.
R7.2 – Auditing: Das *Auditing* aller sicherheitsrelevanten Ereignisse soll garantiert werden, um die Nachvollziehbarkeit der Systemaktivitäten in einem HMI-Sys-

tem im Fahrzeug, welches potenziell die Sicherheit verletzen kann, sicherzustellen. Deshalb darf es keinen direkten Zugriff auf die Hardware geben, um zu gewährleisten, dass das Auditing nicht umgangen werden kann. Zudem soll eine Menge an Regeln für eine Basiserkennung definiert werden, um eine Analyse potentieller Sicherheitsübertretungen zu ermöglichen (ISO 15408-2 [ISO08] (7.3.2)). Um jede potentielle Sicherheitsübertretung der seitens des Systems definierten Regeln zu erkennen, soll eine Überwachung der protokollierten Ereignisse ebenfalls auf Grundlage einer Menge an Regeln (7.3.8.1) stattfinden. Diese Überwachung muss durch das System entweder als eine kumulierte oder eine kombinierte Untermenge an definierten und protokollierten Ereignissen, welche die Sicherheit des Systems beeinträchtigen können, durchgeführt werden (7.3.8.2).

R7.3 – Überwachung der zeitlichen Anforderungen: Es ist erforderlich den Informationsfluss zu regulieren, um eine möglichst knappe und klare Gruppierung der Informationen zu gewährleisten, welche es dem Fahrer ermöglicht, die Informationen mit minimaler Ablenkung wahrzunehmen (ISO 15005 [ISO02] (5.2.4.2.1)). Hierzu müssen die bestimmten *zeitlichen Beschränkungen* verifiziert werden. Dies beinhaltet ebenfalls das Auditing von taktilen Benutzereingaben und die Reaktionszeit des Systems, welche 250 ms nicht überschreiten soll (5.2.4.2.3).

R7.4 – Erkennung von DoS-Attacken: Das Auftreten jedes Ereignisses, das eine signifikante Bedrohung wie eine *Denial of Service (DoS)-Attacke* darstellt, muss seitens des Systems in Echtzeit oder während der Analyse einer gebündelten Menge an gesammelten Ereignissen erkannt werden (ISO 15408-2 [ISO08] (7.3.2)).

R7.5 – Wahrnehmung von visuellen Signalen: Für die Wahrnehmung von visuellen Warnsignalen müssen Eigenschaften der Sichtbarkeit wie Helligkeitsanteile (ISO 11428 [ISO96] (4.2.1.2)) und die Farben der Warnsignale (4.3.2) überwacht werden. Die Überwachung ist ebenfalls für bestimmte sicherheitskritische Symbole, definiert in ISO 2575 [ISO10], erforderlich.

R7.6 – Toleranz von Softwarefehlern: Laut ISO 15408-2 [ISO08] ist es erforderlich, dass bestimmte Fehler und Diskontinuitäten der Systemdienste erkannt werden. Dies beinhaltet ebenfalls die Wiederherstellung bzw. die Überführung des Systems in einen konsistenten und sicheren Zustand (14.7.8.1) unter Verwendung von automatisierten Prozeduren (14.7.9.2). Eine Liste an potentiellen Fehlern und möglichen Diskontinuitäten der Systemdienste muss durch einen sogenannten *Watchdog* überwacht werden, um den Eintritt in einen Fehlerzustand zu erkennen. Des Weiteren müssen für eine definierte Untermenge an Funktionen, die erfolgreich ausgeführt werden müssen, Ausfallszenarien definiert werden, welche eine Wiederherstellung ermöglichen (14.7.11.1).

R7.7 – Systemintegrität: Im Falle eines Fehlers, welcher das Wiederherstellen des Systems nicht mehr zulässt, soll in einen sogenannten *degradierten Betriebsmodus* gewechselt werden, welcher die Systemintegrität erhält. Eine Liste an verschiedenen Fehlertypen soll definiert werden, für welche keine Störung des Systembetriebs stattfinden kann (ISO 15408-2 [ISO08] (15.1.7.1)). Darüber hinaus soll das System den Betrieb einer bestimmten Funktionsmenge für vordefinierte Fehlertypen gewährleisten (15.1.6.1). Dies beinhaltet den Umgang mit DoS-Attacken und die Erkennung von unberechtigten Richtlinienänderungen.

Manche Ereignisse müssen in Form einer internen Repräsentation unterhalten werden, um festzustellen, ob eine Verletzung stattfindet oder stattfand. Dies beinhaltet das Verhalten der Systemaktivitäten zur Identifikation potentieller Übertretungen (7.3.10.2-3) sowie Zustandsänderungen (7.3.10.1).

2.2. Systemarchitektur für moderne HMI-Systeme im Fahrzeug

In diesem Abschnitt wird die Systemarchitektur eines Infotainment-Domänenservers beschrieben, der die in Abschnitt 2.1 beschrieben Anforderungen adressiert und auf Virtualisierung basiert (siehe Abbildung 1.5). In Abbildung 2.2 ist die Systemarchitektur bestehend aus mehreren Komponenten dargestellt, die maßgeblich dazu dienen die Anforderungen zu erfüllen.

Die Anforderung **Zertifizierbarkeit (R6)** stellt eine Ausnahme dar, da diese für den gesamten Entwicklungsprozess gilt, während alle anderen Anforderungen durch die Funktionalitäten der Komponenten erfüllt werden. In Bezug auf die Zertifizierbarkeit wird als Ansatz ein mikrokern-basierter Hypervisor verwendet, der sich dadurch auszeichnet, dass die Treiber für die Hardware nicht im Kernel-Modus laufen, sondern im Benutzer-Modus. Dies führt zu einer sehr kleinen Softwarecodebasis des Hypervisors, die einfacher zu zertifizieren ist [EJ09], da die Treiber nicht im kritischen Teil des Hypervisors laufen und ein Absturz der Treiber nicht den Hypervisor zum Absturz bringt. Dies erhöht die Stabilität des Systems und erlaubt den selektiven Neustart der Treiber bzw. der virtuellen Maschinen, in welchen sie laufen, ohne das gesamte System neu zu starten.

Wie in Abbildung 2.2 dargestellt, läuft der *Virtualisierungsmanager* in einer separaten virtuellen Maschine und regelt die Zugriffe auf die gemeinsam genutzten Ressourcen. Er beinhaltet die Treiber für die GPU und die Eingabegeräte. Dies stellt sicher, dass der Zugriff auf die gemeinsam genutzten Ressourcen durch

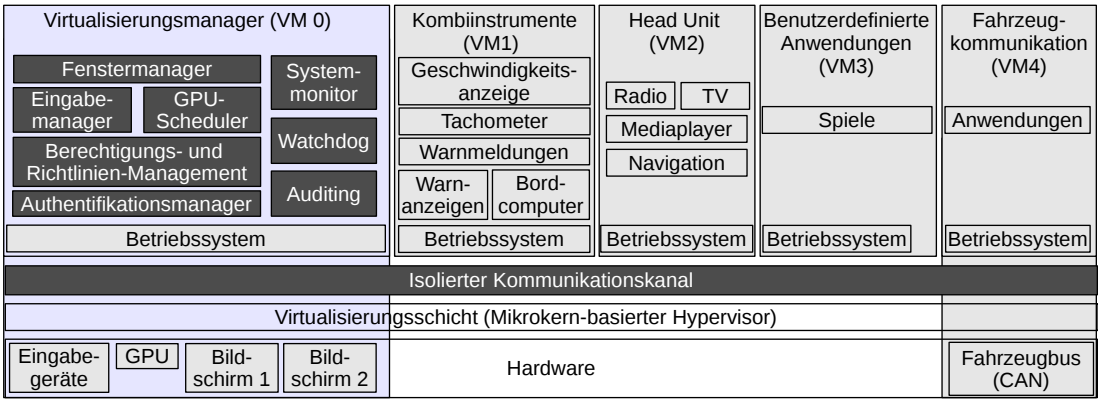

Abbildung 2.2.: Systemarchitektur für einen Infotainment-Domänenserver

eine einzige vertrauenswürdige virtuelle Maschine kontrolliert wird. Der indirekte Hardwarezugriff erleichtert die Anforderungen **Virtualisierte Grafikberechnung (R4)** und **Systemüberwachung (R7)**. Zusätzlich beinhaltet der Virtualisierungsmanager mehrere Komponenten, die sicherstellen, dass der Zugriff von den virtuellen Maschinen auf die Hardware in Übereinstimmung mit den Anforderungen stattfindet. Die Abbildung 2.2 stellt fünf virtuelle Maschinen dar. Dies ist exemplarisch und kann, falls erforderlich, um weitere virtuelle Maschinen ergänzt werden. Es ist neben dem Virtualisierungsmanager jeweils eine virtuelle Maschine für die Kombiinstrumente und die HU vorgesehen, die Anwendungen mit unterschiedlicher Kritikalität haben. Eine weitere virtuelle Maschine kann für Anwendungen des Benutzers verwendet werden, die nicht vertrauenswürdig sind.

Um eine sichere und zuverlässige Kommunikation mit dem Netzwerk (z. B. CAN) im Fahrzeug, insbesondere den Steuergeräten, zu ermöglichen, wird eine dezidierte virtuelle Maschine verwendet. Alle Daten, die von anderen Steuergeräten benötigt werden (z. B. Geschwindigkeit oder Drehzahl des Motors), können von Anwendungen in dieser virtuellen Maschine mittels Hardwareschnittstelle (z. B. CAN-Schnittstelle) aus dem Fahrzeugnetzwerk empfangen und an die Anwendungen in anderen virtuellen Maschinen gesendet werden. Diese virtuelle Maschine muss sicher und zuverlässig sein, damit keine unbeabsichtigte und böswillige Manipulation über diese Schnittstelle im Fahrzeugnetzwerk möglich ist.

Damit die virtuellen Maschinen, die keinen direkten Zugriff auf die Hardware haben (z. B. HU), die Hardware nutzen können, müssen sie mit dem Virtualisierungsmanager kommunizieren. Für die bidirektionale Kommunikation ist ein **vertrauenswürdiger Kanal (R3)** erforderlich, der die sichere und vertrauliche Kommunikation zwischen den virtuellen Maschinen gewährleistet. Ein vertrauens-

würdiger Kanal wird durch die Kooperation des *isolierten Kommunikationskanals* und des *Authentifikationsmanagers* ermöglicht. Der isolierte Kommunikationskanal stellt die Integrität und die Isolation für die Kommunikation zwischen dem Virtualisierungsmanager und den Anwendungen bereit, wie es von der Anforderung (R3.1) gefordert wird. Für die initiale Verbindung muss eine Anwendung Authentifikationsinformationen an den Authentifikationsmanager senden, damit die Unleugbarkeit zwischen dem Empfänger und dem Sender von Daten gewährleistet werden kann (R3.2). Insbesondere ist dies für die Kommunikation zwischen den grafischen Anwendungen der Kombiinstrumente bzw. der HU mit dem Virtualisierungsmanager notwendig, da diese Kommunikation vertrauenswürdig sein muss, um zu gewährleisten, dass die geltenden Richtlinien nicht verletzt werden.

Das *Berechtigungs- und Richtlinien-Management* stellt sicher, dass die Anwendungen Berechtigungen für die Verwendung der Funktionalitäten oder Ressourcen erhalten, die für die Komponenten *Eingabemanager*, *Fenstermanager* und *GPU-Scheduler* benötigt werden. Berechtigungen werden durch aktive Richtlinien repräsentiert, die vom aktuellen Zustand (R5.1) abhängen, z. B. „Fahrzeug parkt“ oder „Fahrzeug fährt“. Das Richtlinien-Management ist mittels Regeln konfigurierbar und führt definierte Transitionen zwischen Richtlinien durch, wenn Zustandsänderungen (R5.2) in definierten zeitlichen Beschränkungen (R5.3) vorliegen.

Der *Eingabemanager* ist für die Verteilung der Eingabeereignisse an die entsprechenden Anwendungen zuständig, welche die erforderlichen Berechtigungen besitzen. Um dies gewährleisten zu können, muss der Eingabemanager alle Ereignisse der Eingabegeräte erhalten. Abhängig von den Richtlinien muss sichergestellt werden, dass die Eingabeereignisse an die dafür bestimmten Anwendungen (R1.1) gehen. Welche Berechtigungen gültig sind bzw. welche Richtlinien gelten, bestimmt das Berechtigungs- und Richtlinien-Management. Dadurch wird sichergestellt, dass alle relevanten Sicherheitsanforderungen eingehalten werden. Da die Verarbeitung der Eingabeereignisse innerhalb bestimmter zeitlicher Beschränkungen erfolgen muss, ist eine minimale Verteilungszeit der Eingabeereignisse an die Anwendungen vom Eingabemanager einzuhalten (R1.2). Um das Abfangen und Verändern von Eingabeereignissen zu verhindern, muss jede Anwendung über einen sicheren Kanal mit dem Eingabemanager kommunizieren.

Der *GPU-Scheduler* ist dafür zuständig, dass die Anforderungen an die **virtualisierte Grafikberechnung (R4)** erfüllt werden. Der GPU-Scheduler muss den Zugriff auf die GPU verwalten, sodass die entsprechenden Anforderungen und Berechtigungen der grafischen Anwendungen eingehalten werden. In Situationen, in welchen mehrere Anwendungen GPU-Ressourcen benötigen, muss der

GPU-Scheduler die Ausführung der verschiedenen Befehle koordinieren. Sicherheitskritische Anwendungen haben immer Priorität vor nicht-sicherheitskritischen Anwendungen. Das heißt, dass Anwendungen eine Priorität zugewiesen bekommen, welche festlegt, wie viel eine Anwendung an Ressourcen der GPU nutzen darf (R4.1). Da davon ausgegangen werden kann, dass die Anzahl an Anwendungen, welche eine GPU nutzen, größer ist, als die Anzahl verfügbarer GPUs, muss der GPU-Scheduler die grafischen Befehle sequentialisieren und deren Ausführung koordinieren. Des Weiteren gilt für manche Anwendungen (z. B. die Geschwindigkeitsanzeige) eine maximale Zeit pro Bild für das Berechnen der grafischen Inhalte (R4.2), welche eingehalten werden soll. Eine GPU verwendet gemeinsam genutzte Ressourcen, wie Speicher und den Zugriff auf den Systembus. Dies kann sicherheitskritische Anwendungen beeinflussen, wenn Manipulationen am Speicher (z. B. das Überschreiben von grafischen Daten durch andere Anwendungen) durchgeführt werden. Zusätzlich kann die Sicherheit verletzt werden, wenn die grafischen Daten von anderen Anwendungen gelesen werden und somit sensible Daten entwendet werden können. Dies erfordert, dass der GPU-Scheduler die Adressen des Speicherbereichs der verschiedenen Anwendungen isoliert (R4.3).

Der *Fenstermanager* stellt die Funktionalitäten für die Erstellung, Positionierung und das Darstellen der Fenster der grafischen Anwendungen zur Verfügung. Dies stellt ein Paradigmenwechsel vom benutzerdefinierten Fenstermanagement hin zum **eingeschränkten Erzeugen und Positionieren von Fenstern (R2)** dar. Anwendungen mit entsprechenden Rechten können mit dem Fenstermanager interagieren, um Fenster zu erzeugen oder deren Eigenschaften wie Größe und Position zu verändern (R2.1). Des Weiteren ist der Fenstermanager dafür zuständig, dass es entweder nicht zu einer Überlagerung von Fenstern kommt oder die Fenster in der richtigen Reihenfolge dargestellt werden (R2.2). Ändern sich die gültigen Richtlinien z. B. aufgrund von Zustandsänderungen, dann ist es erforderlich, dass die betroffenen Fenster entsprechend angepasst werden oder entfernt werden. Dabei müssen die zeitlichen Anforderungen für die Darstellung der grafischen Inhalte ebenfalls vom Fenstermanager eingehalten werden (R2.3).

Um das sichere Hochfahren des Systems (R7.1) zu gewährleisten, muss die Integrität des Softwarecodes, welcher geladen wird, verifiziert werden. Hierzu können Ansätze wie in [GM09] beschrieben, angewandt werden. Die Komponente *Auditing* ist für das Aufzeichnen aller relevanten Systemaktivitäten und Interaktionen zuständig, wie in der Anforderung (R7.2) gefordert. Diese Aufzeichnungen können dann von dem *Watchdog* und dem *Systemmonitor* verwendet werden, um Inkonsistenzen zu erkennen (R7.3 bis R7.7). Der Watchdog überwacht alle rele-

vanten Systemfunktionalitäten und signalisiert, wenn das System Fehlfunktionen aufzeigt, wie in R7.3 bis R7.6 gefordert. Der Systemmonitor empfängt die Signale über das erkannte Systemfehlverhalten vom Watchdog. Entsprechend definierte Regeln werden dann als Reaktion auf die Signale angewandt.

2.3. Verwandte Arbeiten

Es gibt eine große Vielfalt an verwandten Arbeiten, welche sich mit dem Thema HMI-Systeme beschäftigt. Insbesondere die Virtualisierung von Systemen mit Fokus auf Grafik birgt unterschiedliche Herausforderungen. Deshalb sind die verwandten Arbeiten in vier Teile aufgegliedert. Im ersten Teil werden verwandte Arbeiten zum Thema Virtualisierung behandelt, welche auf die funktionale Sicherheit fokussieren. Der zweite Teil umfasst die Fenstersysteme, für welche viele der aufgestellten Anforderungen aus Abschnitt 2.1 relevant sind. Der dritte Teil behandelt die verwandten Arbeiten zum Thema GPU-Scheduling. Im vierten Teil werden die verwandten Arbeiten zum Thema Weiterleitung von grafischen Daten behandelt, das zur Übertragung der Daten an die GPU benötigt wird, da der Virtualisierungsmanager mit Zugriff auf die GPU und die einzelnen Anwendungen in separaten virtuellen Maschinen ausgeführt werden.

2.3.1. Virtualisierung

Die Technologie Virtualisierung wurde in vielen wissenschaftlichen Arbeiten in unterschiedlichen Aspekten beleuchtet. Auch wenn das Konzept der mikrokernbasierten Hypervisoren in Bereich der Virtualisierung bereits seit vielen Jahren bekannt ist, legte sich der Fokus auf funktionale Sicherheit erst innerhalb der letzte Jahre. Insbesondere für eingebettete Plattformen gibt es gute Gründe mikrokern-basierte Hypervisoren zu verwenden (vgl. [Hei08]). Aufgrund der sehr geringen Softwarecodebasis, die eine Verifikation der Funktionalität des Hypervisors praktikabel macht, kann die Zuverlässigkeit deutlich erhöht werden und die Möglichkeiten eines Angriffs auf die Virtualisierungsschicht verringert werden.

Darüber hinaus wird die Zertifizierbarkeit von Hypervisoren immer wichtiger. Beispielsweise haben Klein et al. [KAE+10] für den mikrokern-basierten Hypervisor seL4 eine formale Verifikation durchgeführt, die die funktionale Korrektheit des Systems belegen soll. Der Mikrokernel NOVA [SK10] ist ein Hypervisor mit einer minimalen Softwarecodebasis, welche unmodifizierte Betriebssysteme unter Verwendung von Hardwareunterstützung der CPU für Virtualisierung

ausführen kann. Die Verwendung von Hardwareunterstützung reduziert zudem den Leistungseinbruch deutlich. Da eingebettete Plattformen jedoch oft keine Hardwareunterstützung der CPU für Virtualisierung besitzen, können nur paravirtualisierte Betriebssysteme eingesetzt werden. Der mikrokern-basierte Hypervisor OKL4 [OKL15] ist speziell für eingebettete Systeme ausgelegt und basiert auf dem Mikrokern L4, welcher auf Grundlage der wissenschaftlichen Arbeiten von Liedkte (vgl. [Lie93, Lie95, LHH97, LDE+01]) entwickelt wurde. Hierbei steht insbesondere die Optimierung der Geschwindigkeit der Ausführung der Virtualisierung im Vordergrund. So steht beispielsweise die Optimierung der schnellen Kontextwechsel auf ARM basierten Plattformen im Fokus [vSH07]. Eine weitere Variante von L4 ist der mikrokern-basierte Hypervisor PikeOS [Pik15] von Sysgo. PikeOS wird vor allem in der Flugzeugentwicklung zur Isolation verschiedener Funktionalitäten auf einem Steuergerät eingesetzt. Bieten diese mikrokern-basierten Hypervisoren bereits eine gute Isolation für Anwendungen in verschiedenen virtuellen Maschinen, so ist der Zugriff auf Ressourcen wie GPU und Anzeigen meist nur exklusiv für eine Partition möglich.

2.3.2. Fenstersysteme

Da das Fenstersystem in der vorgestellten Architektur eine zentrale Rolle spielt und insbesondere wichtig für die Erfüllung der Anforderungen ist, werden im Folgenden verwandte Arbeiten aus dem Bereich der Fenstersysteme vorgestellt.

Laut Epstein et al. [EP91] bietet das Fenstersystem X11 keine bzw. nur unzureichend Sicherheit. Als Bewertungsgrundlage verwendet Epstein et al. die Anforderungen aus TCSEC B3 [Nat85] (diese sind in ISO 15408-2 [ISO08] enthalten) und analysiert das Fenstersystem X [SG86]. Mit Trusted X [EMP+91] wird eine Lösung vorgeschlagen, welche Sicherheit bietet und dabei auf die Erfüllung der Anforderungen in TCSEC B3 abzielt. Der entstandene Prototyp soll sicherstellen, dass das X-Protokoll mit minimaler Auswirkung auf die Anwendungen keine versteckten Kommunikationskanäle zulässt, jedoch wurde aufgrund des hohen Aufwandes keine Zertifizierung durchgeführt. Es werden verschiedene Sicherheitslevels verwendet, die eine Zuordnung der Anwendungen entsprechend ihrer benötigten Sicherheitsklasse ermöglichen und entsprechend die Anwendungen der unterschiedlichen Sicherheitslevel isolieren. Um Isolation zu gewährleisten, werden ein X-Server und ein Fenstermanager pro Sicherheitslevel eingesetzt. Dies führt jedoch zu Skalierungsproblemen, da insbesondere bei vielen Sicherheitslevels die Leistung deutlich abnimmt. Das heißt, dass gegenseitige Isolation von Anwendun-

gen aufgrund der Skalierungsprobleme praktisch nicht möglich ist. Hier skaliert ein monolithischer Ansatz für das Fenstersystem, wie in der Architektur für den Infotainment-Domänenserver in Abbildung 2.2 dargestellt, wesentlich besser. Des Weiteren hat der Anwender volle Kontrolle über die Fensterdarstellung, sodass die Anforderungen des Automobilbereichs nicht sichergestellt werden können.

Nitpicker [FH05] ist ein grafisches System, das Sicherheitsmechanismen und Protokolle bietet, die die sichere und isolierte Interaktion des Benutzers unter Verwendung von verschiedenen Betriebssystemen unterstützt. Das heißt, dass Eingabeereignisse nicht von allen Anwendungen abgefangen werden können, sondern nur an die dafür vorgesehenen Anwendungen gehen. Außerdem soll vermieden werden, dass die Anwendungen auf die Fensterinhalte anderer Anwendungen zugreifen können. Um dies zu gewährleisten, erhält der Grafikserver von Nitpicker exklusive Zugriffsrechte auf die GPU und auf den Anzeigespeicher. Die Anwendungen stellen einen Speicherbereich zur Verfügung, in welchen sie ihre grafischen Inhalte für ihre Fenster ablegen. Nitpicker nimmt diese Inhalte und stellt sie auf der Anzeige dar (siehe [FH04]). Die Eingabeereignisse werden dann entsprechend an die Anwendung gesendet, deren Fenster gerade durch den Anwender z. B. mittels Mauseingabe fokussiert ist. Um Anwendungen mit unterschiedlichen Sicherheitsanforderungen zu isolieren, werden virtuelle Maschinen mit eigenem Betriebssystem verwendet. Nitpicker läuft in einer dedizierten virtuellen Maschine, die mit den Anwendungen mittels Inter-Partition Communication (IPC) kommuniziert. Um Isolation zwischen den verschiedenen Betriebssystemen zu gewährleisten, nutzt Nitpicker den Virtual Machine Monitor (VMM) L4/Fiasco [Hoh02].

Das Fenstersystem DOpE [FH03] stellt die Aktualisierungsrate der Fenster von Echtzeitanwendungen sicher und bietet für die Anwendungen, welche nicht echtzeitfähig sind, bestmöglich die verbleibenden Ressourcen. Die bedeutet, dass Überlastsituationen verhindert werden, welche durch nicht-echtzeitfähige Anwendungen oder Benutzeraktivitäten, die die Aktualisierung von Fenstern nach sich ziehen, ausgelöst werden. DOpE basiert, wie Nitpicker, auf L4/Fiasco [Hoh02] und nutzt die Isolationseigenschaft und die Kommunikation mittels IPC. Jedoch können keine Richtlinien definiert werden. Alle diese Fenstersysteme haben gemeinsam, dass die Anwendungen keinen Zugriff auf die GPU besitzen und ihnen im Gegensatz zu der Systemarchitektur für einen Infotainment-Domänenserver in Abschnitt 2.2 keine Hardwarebeschleunigung für die Berechnung ihrer grafischen Inhalte zur Verfügung steht. Des Weiteren werden keine zeitlichen Garantien für die Berechnung und Darstellung der grafischen Inhalte gegeben.

Das EROS Fenstersystem „EWS“ (engl. EROS Window System) [SVNC04] zielt

auf den Schutz von sensitiven Informationen und die Verwendung von Sicherheitsrichtlinien in einem Zugriffskontrollsystem, das durch den Anwender gesteuert wird, ab. EWS basiert auf dem EROS Betriebssystem und bietet ein Anzeigesystem mit Doppelpufferung der grafischen Inhalte der Anwendungen, das nur minimale Ressourcen für die Verwaltung benötigt. In EWS setzt jede Veränderung der grafischen Darstellung eine Benutzereingabe voraus.

Trusted X, Nitpicker, DOpE und EWS haben gemeinsam, dass sie nur auf Sicherheit und nicht auf funktionale Sicherheit fokussieren, womit die Anforderungen Eingabeverarbeitung (R1), Einschränkung der Erstellung und Positionierung von Fenstern (R2) und Systemüberwachung (R7) nicht erfüllt werden.

2.3.3. GPU-Scheduling

Für das Verwalten der GPU-Ressourcen gibt es im Wesentlichen zwei relevante Arbeiten. Dies ist zum einen GERM [BDC08], das die Ressourcen der GPU nach Fairness verteilt, jedoch dabei keinerlei Isolation oder Priorisierung von Anwendungen kennt. Zum anderen ist dies Timegraph [KLRI11], das auf den Konzepten von GERM aufbaut und mittels Prioritäten ein Scheduling der GPU-Befehlsgruppen (Basis ist der Treiber Nouveau [Nou16]) durchführt. Jedoch kann für die Ausführungszeit einer GPU-Befehlsgruppen keine obere Schranke garantiert werden und die Leistung kann durch die Verwendung von Timegraph erheblich sinken. Darauf aufbauend wurde von Kato et al. [KLIR11] das Konzept erweitert, um die Latenz, die durch synchrone GPU-Befehle bei der Verwendung eines X-Servers [SG86] und der Doppelpufferung der grafischen Inhalte entsteht, zu reduzieren. Hierzu wird versucht, eine mögliche Prioritäteninversion, welcher dadurch entstehen kann, dass der X-Server durch Anwendungen mit unterschiedlichen Prioritäten angefragt wird, zu verhindern. Dies bedeutet, dass Befehle, welche vom X-Server an die GPU gesendet werden, dieselbe Priorität haben müssen, wie die Anwendungen, zu welcher dieser Befehl gehört. Da der X-Server jedoch keine ausreichenden Isolationsmechanismen bietet, eignet sich der Einsatz des X-Servers nicht als grafisches Fenstersystem im Automobil.

2.3.4. Effiziente Übertragung von Grafikbefehlen

Für das effiziente Übertragen von Grafikdaten und -befehlen wurde der Ansatz VMGL [LCTSdL07] entwickelt. Hierbei werden OpenGL-Befehle von einem OpenGL-Klient an einen OpenGL-Server mittels einer TCP/IP-Verbindung übertragen. Der Server führt dann die empfangenen OpenGL-Befehle auf der Grafik-

karte aus und stellt die berechneten Bilder auf seiner Anzeige dar. Durch die Verwendung von TCP/IP kommt es jedoch zu einer nicht unerheblichen Latenz, welche insbesondere durch synchrone Befehle verursacht wird. Blink [Han07] ist ein Fenstersystem, welches auf die sichere Darstellung von mehreren OpenGL-Anwendungen in verschiedenen virtuellen Maschinen fokussiert. Blink verwendet ebenfalls eine Klient-Server-Architektur für die Übertragung der OpenGL-Befehle, jedoch werden hierbei die Daten über gemeinsam genutzten Speicher an eine dezidierte virtuelle Maschine (Treiber-VM) übertragen, welche die Treiber enthält. Diese Treiber-VM ist verantwortlich für die Ausführung der OpenGL-Befehle auf der GPU. Um die Leistung zu erhöhen, wird zusätzlich ein Konzept (genannt BlinkGL) vorgestellt, welche zusätzliche Befehle einführt, die die Übertragung beschleunigen sollen. Die erfordert jedoch die Anpassung der Anwendungen und ist nicht mehr konform zu dem Standard OpenGL.

2.4. Zusammenfassung

In diesem Kapitel wurden die Anforderungen an ein neuartiges grafisches Fenstersystem beschrieben, welche in einer virtualisierten Umgebung im Fahrzeug erfüllt werden müssen und die Funktionalitäten unterschiedlicher Steuergeräte bedienen. Von den relevanten Richtlinien, ISO-Standards und gesetzlichen Vorschriften wurden sieben technische Anforderungen abgeleitet, welche für eine physikalische Konsolidierung von grafischen Steuergeräten mit unterschiedlicher Kritikalität wie die Headunit und die Kombiinstrumente relevant sind und in einem Infotainment-Domänenserver konsolidiert werden können.

Es wurde eine Systemarchitektur für einen Infotainment-Domänenserver vorgestellt, welche Isolation zwischen grafischen Anwendungen bietet, die in verschiedenen virtuellen Maschinen laufen. Obwohl nicht alle Anwendungen zertifiziert sind, kann die Systemarchitektur garantieren, dass es zu keiner ungewollten Beeinflussung mit zertifizierter Software des OEMs kommen kann und die sieben technischen Anforderungen erfüllt werden können. Um dies zu gewährleisten, sind Konzepte notwendig, welche in den folgenden Kapiteln beschrieben werden. Der Schwerpunkt dieser Arbeit liegt auf den Anforderungen R2 (Einschränkung der Erstellung und Positionierung von Fenstern) und R5 (Rekonfiguration der Richtlinien). Zu den weiteren Anforderungen wurden in diesem Kapitel mögliche Lösungsansätze und verwandte Arbeiten in Kürze beschrieben.

Das folgende Kapitel beschreibt die Zugriffskontrolle für Anzeigebereiche.

3. Kontext-basierte Zugriffskontrolle für Anzeigebereiche

In diesem Kapitel wird ein Zugriffskontrollmodell für Anzeigebereiche abhängig von Kontexten beschrieben. Die Zugriffskontrolle gewährleistet, dass für den Fahrer des Fahrzeugs die Anwendungen sichtbar sind, welche in bestimmten Kontexten des Fahrzeugs oder der Anwendungen erforderlich bzw. zugelassen sind.

Die Zugriffskontrolle für Anzeigebereiche stellt eine Schicht zwischen den einzelnen Anwendungen in den virtuellen Partitionen und den Anzeigen dar, wie in Abbildung 3.1 dargestellt. Bevor eine Anwendung grafische Inhalte darstellen kann, muss der Zugriff auf die Anzeigen entsprechend durch die Zugriffskontrollschicht gewährt werden.

Im Folgenden wird zunächst das Systemmodell für eine Zugriffskontrolle von Anzeigebereichen erläutert. Nach der Definition der Entitäten für das Modell werden auf Grundlage der Anforderungen in Kapitel 2.1 Anforderungen für ein Zugriffskontrollmodell abgeleitet, für welches Korrektheitseigenschaften definiert werden. Nach der Definition eines Protokolls zur Modifikation der Zustände im Modell wird mit formalen Beweisen nachgewiesen, dass die Korrektheitseigenschaften in jedem Zustand erfüllt sind, wenn sie im Startzustand gelten.

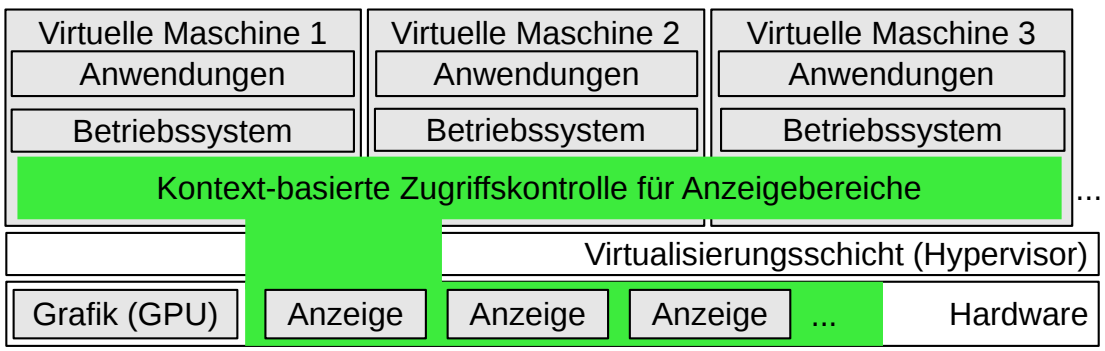

Abbildung 3.1.: Zugriffskontrollschicht für Anzeigebereiche

3.1. Grundlagen

In diesem Abschnitt werden die wichtigsten Grundbegriffe und Arten der Zugriffskontrolle erklärt, die für ein Verständnis des formal beschriebenen Zugriffskontrollmodells hilfreich sind. Zusätzlich werden im Anhang A die wichtigsten mathematische Grundlagen beschrieben, welche für die Definition des Zugriffskontrollmodells und die Beweise der Korrektheitseigenschaften des Modells notwendig sind.

3.1.1. Zugriffskontrolle

Computersysteme bestehen und verwalten viele *Objekte*, die vor unerlaubtem Zugriff geschützt werden müssen. In diesem Abschnitt sollen die wesentlichen Modelle und Mechanismen für Zugriffskontrolle beschrieben werden, womit ein grundlegendes Verständnis für Zugriffskontrollmodelle vermittelt werden soll. Die Grundlagen sind aus [Tan09] entnommen.

Zugriffskontrolle erfordert Schutzmechanismen, die regeln sollen, wer (*Subjekt*) auf was (*Objekt*) zugreifen darf. Geschützt vor Zugriff werden Objekte wie Hardware (z. B. Prozessor, Speicherbereiche, Drucker) oder Software (z. B. Prozesse, Dateien, Webseiten). Objekte sind eindeutig identifizierbar (z. B. über einen Namen) und es gibt festgelegte Operationen, die ein Subjekt (z. B. ein Prozess) auf einem Objekt ausführen kann. Zum Beispiel kann man in der Regel auf Dateien per Lese- und Schreiboperation zugreifen. Subjekten kann man entweder den Zugriff vollständig verbieten, das heißt, ein Subjekt darf keine Operation auf ein Objekt ausführen, oder der Zugriff wird nur für bestimmte Operationen gewährt. Der Zugriff eines Prozesses, z. B. auf eine Datei A, kann auf die Operation Lesen eingeschränkt werden. Eine Berechtigung gibt dann an, dass ein bestimmtes Subjekt auf ein bestimmtes Objekt mit einer Menge an Operationen zugreifen darf.

Zugriffsmatrix und Zugriffslisten In der Regel wird das sogenannte Konzept der *Domänen* verwendet. Eine *Domäne* ist eine Menge an Paaren (Objekt, Rechte), die festlegt, mit welchen Operationen auf das Objekt zugegriffen werden darf. Die Rechte legen hierbei fest, welche Operationen ausgeführt werden dürfen. Weist man nun Subjekten Domänen zu, dann ist festgelegt, wer auf was mit welchen Rechten zugreifen darf. Eine wichtige Fragestellung ist, wie das System überprüfen kann, welches Objekt zu welcher Domäne gehört. Eine Möglichkeit stellt die Verwendung einer *Zugriffsmatrix* dar, in welcher für jedes Objekt (eine

Spalte pro Objekt) und für jede Domäne (eine Zeile pro Domäne) die Rechte in den Feldern eingetragen sind. Der Nachteil einer solchen Matrix ist, dass diese zentral abgelegt und verwaltet werden muss. Eine Alternative dazu stellen *Zugriffslisten* dar, die im Grunde die Spalten der Zugriffsmatrix pro Objekt speichern. Das heißt, je Objekt werden die Rechte der entsprechenden Domänen gespeichert. Der Vorteil ist, dass die Rechte eines Objektes nicht zentral gespeichert und relativ einfach geändert werden können. Dazu muss nur die entsprechende Zugriffsliste angepasst werden. Aufwändig wird es jedoch, wenn die Domänen geändert werden sollen. Hierbei müssen unter Umständen alle Zugriffslisten angepasst werden.

Capability Die Zugriffsmatrix kann auch nach Zeilen aufgeteilt werden. Jedes Subjekt besitzt dann eine Liste an Objekten, auf die es mit den darin aufgelisteten Rechten zugreifen darf. Dies stellt seine Domäne dar. Ein Element der Liste wird *Capability* (zu Deutsch: „Befähigung“) genannt. Jede Capability stellt die Rechte dar, die das Subjekt für ein bestimmtes Objekt hat. Eine Capability besteht aus einem eindeutigen Identifikator (ID) für das Objekt und einer Maske für die Rechte, die über ein Bit gesetzt werden.

Die Capabilities in den Listen müssen vor Manipulation geschützt werden, da sonst jedes Subjekt sich selbst Rechte geben könnte. Eine Möglichkeit, dies zu tun, ist, dass die Capabilities nicht direkt die Informationen über die Rechte enthalten, sondern nur einen Verweis auf eine Stelle, die das Betriebssystem verwaltet und welche entsprechend geschützt ist. Im Unterschied zu Zugriffslisten, die es relativ einfach ermöglichen Rechte zu entziehen, kann es bei Capabilities aufwändig sein, wenn selektiv bestimmte Rechte entzogen werden sollen. Dies erfordert es unter Umständen, dass viele Listen der Capabilities anzupassen sind. Da weder Zugriffsmatrizen noch Capabilities statisch sind, spielt der Aufwand für das Anlegen und Entfernen von Objekten, sowie das Ändern von Zugriffsrechten ebenfalls eine große Rolle. Die zentrale Frage ist, ob ein System mit verfügbaren Mechanismen bestimmte Sicherheitsrichtlinien einhalten kann oder nicht. Dies zu beweisen ist meist nicht trivial bzw. kann für allgemeingültige Systeme oft auch dazu führen, dass sie theoretisch nicht sicher sind oder es nicht entscheidbar ist.

Im Folgenden werden zwei Referenzmodelle bzw. Richtlinien für Zugriffskontrolle vorgestellt, welche unter anderem auf diese Fragestellung abzielen. Des Weiteren wird ein Konzept für rollenbasierte Zugriffskontrolle vorgestellt.

3.1.2. Zugriffskontrollmodelle

Es gibt zwei häufig verwendete Referenzmodelle, die für Zugriffskontrollmodelle verwendet werden. Nach Nusser [Nus98] sind das die *wahlfreie* Zugriffskontrolle (engl.: „**D**iscretionary **A**ccess **C**ontrol (DAC)“) und die *regelbasierte* Zugriffskontrolle (engl.: „**M**andatory **A**ccess **C**ontrol (MAC)“), auch Multilevel Security genannt. Diese werden vom Verteidigungsministerium der USA in den „**T**rusted **C**omputer **S**ystem **E**valuation **C**riteria (TCSEC) [Nat85]“ definiert. Zusätzlich gibt es ein Konzept für *rollenbasierte* Zugriffskontrollmodelle (engl.: „**R**ole **B**ased **A**ccess **C**ontrol (RBAC)“), das im Anschluss zu den beiden Referenzmodellen beschrieben wird.

Wahlfreie Zugriffskontrolle Die wahlfreie Zugriffskontrolle erlaubt es Subjekten (z. B. den Benutzern oder Prozessen), die Rechte für ihre Daten selbst zu setzen. Die Annahme ist, dass es für jedes Objekt einen Besitzer gibt, der in der Regel ein Subjekt ist. Außerdem gilt, dass jeder Besitzer eines Objektes nach eigenem Ermessen bzw. nach eigener Diskretion (engl.: „Discretionary“) die Rechte der anderen Subjekte für dieses Objekt ändern kann. Neben dem Besitzermodell gibt es das hierarchische Modell, das z. B. einen privilegierten Administrator hat, der über den Subjekten steht und ebenfalls die Rechte auf die Objekte ändern kann. Dies wird in gängigen Betriebssystemen wie Linux angewandt.

Regelbasierte Zugriffskontrolle Im Unterschied zur wahlfreien Zugriffskontrolle sind bei der regelbasierten Zugriffskontrolle nicht die einzelnen Subjekte für die Zuweisung der Rechte zuständig, sondern das System setzt die Sicherheitsrichtlinien. Dies ist vor allem in Umgebungen sinnvoll, die strenge Sicherheitsrichtlinien haben, wie das Militär oder Krankenhäuser. Es muss gewährleistet werden, dass die gegebenen Regeln der Organisation, die über den Zugriff bzw. die Lese- und Schreibrechte bestimmen, nicht durch individuelle Personen beliebig geändert werden können. Die regelbasierten Zugriffskontrollen verwenden sogenannte Markierungen, die als Zugriffskontrollinformation dienen. Hierzu können Kategorien verwendet werden, in welche die Objekte eingeordnet werden. Ein Beispiel dafür ist das Modell von Bell und LaPadula [BL73]. Die Zugriffskontrolle gewährleistet, dass ein Subjekt nur dann Lesezugriff auf ein Objekt bekommt, wenn es höher oder gleich eingestuft ist als die Klassifizierung des Objektes. Zudem muss die Berechtigung des Subjekts die assoziierten Kategorien des Objektes beinhalten. Im Gegensatz dazu ist es für den Schreibzugriff notwendig, dass die Klassifizierung

des Objektes höher oder gleich eingestuft ist als das Subjekt. Dies verhindert, dass ein Subjekt mittels Kopie des Objektes die Klassifizierung verringern kann.

Rollenbasierte Zugriffskontrolle Die Idee hinter der rollenbasierten Zugriffskontrolle (engl.: „**R**ole **B**ased **Ac**cess **C**ontrol (RBAC)“, vgl. [SCFY96]) ist, dass die Aufgaben, die ein Subjekt wahrnimmt, einer Rolle zugeordnet wird und sich daran bestimmte Zugriffsrechte koppeln lassen. Einem Subjekt kann zeitlich beschränkte Rollen zugeordnet werden, die sich aus einer zeitlich beschränkten Tätigkeit ergeben. Daran gebunden ist, dass ein Subjekt bei mehreren möglichen Rollen sich für eine oder mehrere aktive Rollen entscheiden muss, die für eine *Sitzung* gelten. Die Rollen können *hierarchisch geordnet* werden. Das heißt, Rollen können vererbt werden. Allgemeine Rollen vererben ihre Rechte an spezialisierte Rollen, die zusätzliche Rechte besitzen. Mittels der rollenbasierten Zugriffskontrolle können durch Abstraktion die Zugriffskontrollinformationen in Form von Rollen an die Subjekte zugewiesen werden.

3.2. Systemmodell und Anforderungen

In diesem Abschnitt wird das Systemmodell für eine Zugriffskontrolle, die mittels Berechtigungen Anwendungen das Schreiben von Pixeln erlaubt, beschrieben. Anforderungen, die sich aus den Richtlinien oder ISO-Standards des Automobilbereichs ableiten, werden im Anschluss zum Systemmodell beschrieben.

3.2.1. Systemmodell

Das Systemmodell, dargestellt in Abbildung 3.2, besteht aus zwei Teilen. Dies ist zum einen ein Anteil, der sich Offboard befindet (d. h. außerhalb des Fahrzeuges) und zum anderen ein Anteil, der sich Onboard befindet (d. h. im Fahrzeug).

Onboard-Komponenten

Die Komponenten, die sich Onboard befinden, regeln den Zugriff der Anwendungen auf die *gemeinsamen Anzeigen*. Die Anwendungen nutzen den *Bildschirmbereich* um ihre grafischen Inhalte darzustellen, wobei sie einen *Anzeigebereich* des Bildschirmbereichs nur nutzen dürfen, wenn sie eine *Berechtigung* dafür haben. Ein Anzeigebereich ist eine zweidimensionale Menge aus Pixeln und eine Untermenge des gesamten Bildschirmbereichs. Jeder Pixel eines Anzeigebereichs ist eindeutig durch seine Position identifizierbar.

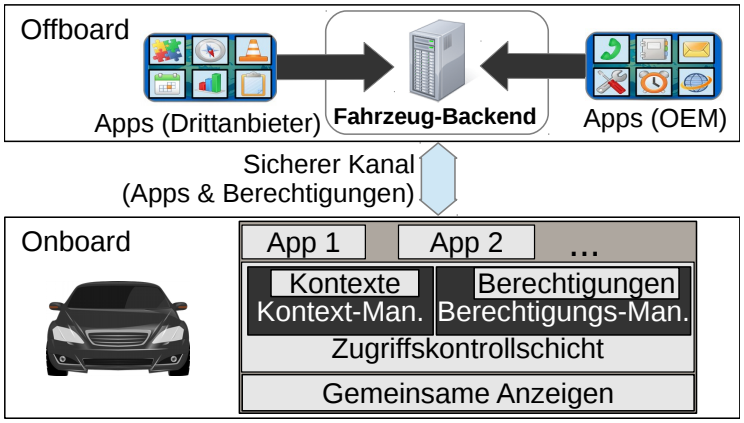

Abbildung 3.2.: Systemmodell einer Zugriffskontrolle für Anzeigebereiche

Um eine Berechtigung zu erhalten, muss eine Anwendung sich bei der *Zugriffskontrollschicht* authentifizieren. Damit Anwendungen identifiziert werden können, besitzt jede Anwendung einen eindeutigen Identifikator (ID). Die Vergabe einer Berechtigung an eine Anwendung wird vom *Berechtigungs-Manager* durchgeführt und hängt von *Kontexten* ab. Ähnlich zu [Dey01] gilt, ein Kontext ist „... jede Information, welche genutzt werden kann, um die Situation einer Entität zu charakterisieren“. Konkret bedeutet dies, ein Kontext stellt eine bestimmte Situation des Fahrzeugs oder einer Anwendung dar, die entweder *aktiv* oder *inaktiv* ist. Da der Zugriff auf einen Anzeigebereich abhängig der Kontexte stattfindet, legt eine Berechtigung fest, welche Anwendung in welchen Kontexten auf einen Anzeigebereich zugreifen darf. Da somit die Kontexte essentiell die funktionale Sicherheit durch die Darstellung von sicherheitskritischen Anwendungen beeinflussen, müssen die Informationen zur Bestimmung der Kontexte (z. B. Sensordaten des Fahrzeugs) fälschungssicher und zuverlässig sein. Die Annahme ist, dass insbesondere die Daten der Sensoren des Fahrzeuges, die über einen Fahrzeugbus (z. B. CAN) gesendet werden, weder beabsichtigt noch unbeabsichtigt verfälscht werden können, sodass die Ermittlung der Kontexte auf zuverlässigen Daten basiert.

Ein Kontext kann von einer Anwendung mittels des *Kontext-Managers* festgelegt werden. Jede Änderung der Kontexte meldet der Kontext-Manager an den Berechtigungs-Manager, welcher entsprechend die Zugriffe der Anwendungen entzieht bzw. gewährt, sollten Berechtigungen sich durch Kontextwechsel ändern.

Offboard-Komponenten

In der heutigen Fahrzeugentwicklung nimmt der Anteil der Entwicklung von Software weiter zu [EJ09]. Hier ist die Annahme, dass ein *dezentraler Softwa-*

reentwicklungsprozess angewandt wird z. B. durch service-basierte Softwareentwicklung [KNP04]. In diesem Prozess sind viele verschiedene Abteilungen des OEMs, Zulieferer, sowie Drittanbieter (z. B. Entwickler, die Anwendungen für das Fahrzeug entwerfen, welche der Kunde sich zusätzlich kaufen kann) beteiligt.

Ein Szenario ist in Abbildung 3.3 zu sehen. Die Entwicklung der Software seitens des OEMs wird von verschiedenen Abteilungen und Zulieferern durchgeführt. Der OEM stellt ein Basissystem für die HMI, Schnittstellendefinitionen und Zertifizierungsrichtlinien zur Verfügung. Firma 1 ist unabhängig vom OEM und stellt eine Navigationssoftware zur Verfügung. Weitere Firmen können für die Navigationssoftware Erweiterungen liefern, die seitens der Firma 1 zertifiziert werden. Die Vergabe der Berechtigungen erfordert eine *Delegationsbeziehung* zwischen den unterschiedlichen Parteien im Vergabeprozess. Um dennoch die Kontrolle über den Verteilungsprozess der Anwendungen zu haben, ist für die Verteilung von Anwendungen und Berechtigungen ein zentralisiertes Fahrzeug-Backend zuständig. Dies bedeutet, Anwendungen, die einerseits von Drittanbietern oder vom OEM kommen, werden über einen sicheren Kanal an die Fahrzeuge verteilt. Hierbei werden ebenfalls die Berechtigungen für den Zugriff auf die Anzeigen mit verteilt.

3.2.2. Anforderungen

Für die Darstellung von sicherheitskritischen Anwendungen sind die Anforderungen in R2 – Einschränkung der Erstellung und Positionierung von Fenstern – aus Kapitel 2 relevant. Laut R2.1 – Einschränkung der Sichtbarkeit von Fenstern – muss es eine Zugriffskontrolle für die Anzeigebereiche geben, auf die sicherheitskritische und nicht-sicherheitskritische Anwendungen zugreifen. Weder Anwender noch Anwendungen dürfen die Möglichkeit haben, Manipulationen an Anzeigebereichen durchzuführen, ohne dass die Zugriffskontrolle dies autorisiert hat. Damit keine Inkonsistenzen auftreten, darf jeder Pixel zu jedem Zeitpunkt nur genau einer Anwendung zugeordnet sein. Zusammen mit R2.2 – Prioritätsbasiertes Anzeigen von Fenstern – ergibt sich, dass eine Vergabe der Anzeigebereiche von den

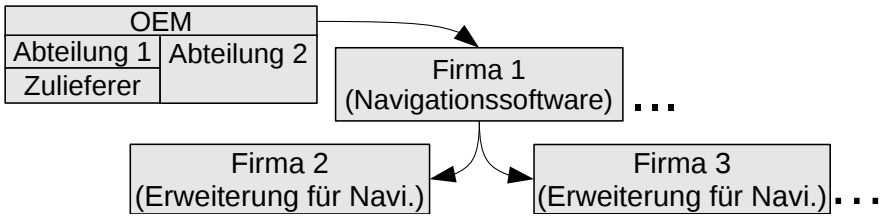

Abbildung 3.3.: Szenario für einen dezentralen Softwareentwicklungsprozess

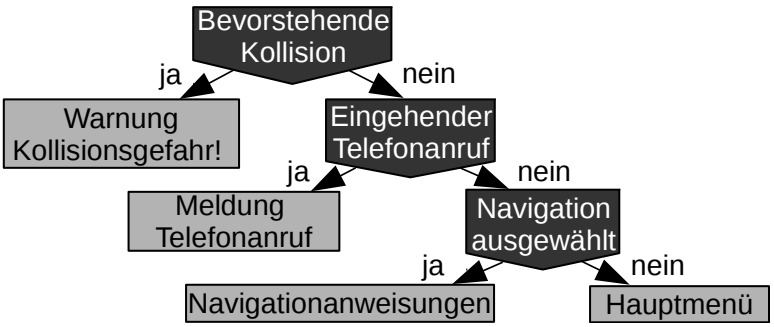

Abbildung 3.4.: Beispiel für eine Zuordnung von Kontexten

Prioritäten der Anwendungen bzgl. ihrer Sichtbarkeit abhängt. Die Berechtigung für den Zugriff auf einen Anzeigebereich muss prioritätsbasiert erfolgen. Durch R2.3 – zeitliche Beschränkungen – sind maximale Ausführungszeiten zu beachten.

Neben den Prioritäten der Anwendungen spielen die Anforderungen R5 – Rekonfiguration der Richtlinien – eine wichtige Rolle für eine Zugriffskontrolle der Anzeigebereiche. Eine Richtlinie besteht aus einer Menge an Berechtigungen, die sich direkt auf die Darstellung von Anwendungen auswirken. Das heißt, die Zugriffskontrolle muss die Richtlinien, die für die Darstellung gelten, gewährleisten. Da die Richtlinien zustandsabhängig sind, muss jede Zustandsänderung, die zu einer Änderung der Richtlinien führt von der Zugriffskontrolle berücksichtigt werden. Dies ist nach R5.2 – Dynamische Richtlinienänderungen – erforderlich. Beispielsweise muss die Geschwindigkeitsanzeige im Zustand „Fahrzeug fährt“ angezeigt werden. Die Zugriffskontrolle muss der Anwendung die Berechtigung für den Anzeigebereich gewähren, sobald die Zustandsänderung erfolgt. Die Zugriffskontrolle muss nach Anforderung R5.3 – Gewährleistung der Darstellung – die Präsentation z. B. rechtlich erforderlicher Meldungen bzw. Anwendungen gewährleisten. Das heißt, dass die Vergabe einer Berechtigung auch von äußeren Faktoren, d. h. den Kontexten, wie die Verkehrssituation, abhängen kann. Beispielsweise erfolgt die Darstellung eines Warnhinweises über eine mögliche Kollisionsgefahr sobald sich das vorausfahrende Auto zu schnell nähert. Um Konfliktfreiheit zwischen zwei Berechtigungen zu gewährleisten, dürfen diese entweder den Zugriff nicht auf dieselbe bzw. einen Teil desselben Anzeigebereichs gewähren oder die zugehörigen Kontexte dürfen nicht zur selben Zeit aktiv sein. Darf der grafische Inhalt einer Anwendung abhängig von bestimmten Kontexten nicht sichtbar sein, dann muss der Zugriff auf den Anzeigebereich entzogen werden, wenn diese Kontexte aktiv sind. Es erfordert z. B. der Kontext „Fahrzeug fährt“, dass die Darstellung eines Videos für den Fahrer nicht sichtbar ist und ihn nicht ablenkt.

Abbildung 3.4 stellt ein Beispiel dar, in welchem ein Anzeigebereich in der Mitte

der Anzeige der Kombiinstrumente von vier verschiedenen Anwendungen genutzt wird. Entsprechend den Anforderungen im Automobilbereich soll in diesem Fall die Entscheidung, welche Anwendung exklusiven Zugriff auf den Anzeigebereich erhält, abhängig der Kontexte stattfinden. Dies sind die Kontexte „Bevorstehende Kollision“, „Eingehender Telefonanruf“ und „Navigation ausgewählt“. Der Zugriff auf den Anzeigebereich soll maximal einer Anwendung abhängig der Kritikalität bzw. der Priorität zur selben Zeit gewährt werden. Sei die Annahme, dass das „Hauptmenü“ die Berechtigung hat, die Fahrzeuginformationen wie Kilometerstand darzustellen. Wenn Kontext „Navigation ausgewählt“ aufgrund der Eingabe des Fahrers aktiv wird, dann müssen die Berechtigungen angepasst werden, sodass die „Navigationsanweisungen“ anstatt des „Hauptmenüs“ sichtbar sind. Sobald der Fahrer einen Telefonanruf erhält, wird der Kontext „Eingehender Telefonanruf“ aktiv und die „Meldung Telefonanruf“ wird sichtbar, die es dem Fahrer erlaubt, den Anruf anzunehmen. Während die Aufmerksamkeit des Fahrers auf den Telefonanruf gelenkt ist, verringert sich die Distanz zum vorausfahrenden Fahrzeug unter ein kritisches Maß, sodass die „Warnung Kollisionsgefahr“ angezeigt wird. Wie an diesem Beispiel zu sehen ist, bedeutet dies, dass die Sichtbarkeit von Anwendungen ebenfalls von den Kontexten anderer Anwendungen abhängt. So werden die „Navigationsanweisungen“ nur dann angezeigt, wenn die Kontexte „Eingehender Telefonanruf“ und „Bevorstehende Kollision“ nicht aktiv sind.

Um den verteilten Entwicklungsprozess zu erleichtern, soll es dem OEM möglich sein, kontext-basierte Berechtigungen für Anzeigebereiche an Softwareunternehmen oder an individuelle Softwareentwickler zu vergeben. Diese sollen dann ebenfalls in der Lage sein Berechtigungen zu vergeben, so dass eine hierarchische Weitergabe von Berechtigungen möglich ist. Dies ermöglicht es dem OEM – ohne selbst eine zentrale Zertifizierungsstelle für alle Anwendungen zu sein – die Anforderungen in Kapitel 2.1 zu erfüllen. Dazu kann er eingeschränkte Berechtigungen weitergeben, die die Verletzung relevanter Anforderungen verhindern (z. B. die Berechtigungen erlauben den Zugriff auf die Anzeigen nur bei stehendem Fahrzeug) oder bestimmte Anforderungen durch Drittanbieter zertifizieren lassen.

3.3. Konzept des Zugriffskontrollmodells

In einer Zugriffskontrolle wird festgelegt, welche *Subjekte* auf welche *Objekte* zugreifen dürfen. In einem Zugriffskontrollmodell für Anzeigen stellen die Subjekte *Anwendungen* und die Objekte *Anzeigebereiche*, auf deren Pixel die Anwendungen zugreifen, dar. Eine Anwendung benötigt eine *Berechtigung* um auf eine Anzeige

zugreifen zu dürfen. Der Zugriff ist abhängig der *Kontexte* des Fahrzeugs und der Anwendungen. Das heißt, eine Anwendung benötigt eine Berechtigung, die auf bestimmte Kontexte festgelegt ist, um auf einen bestimmten Anzeigebereich zugreifen zu dürfen. Diese Art der Berechtigung heißt *kontext-beschränkte Berechtigung*. Nur wenn die Kontexte einer kontext-beschränkten Berechtigung zu den aktuellen Kontexten des Fahrzeugs oder der Anwendungen passen, darf auf den zugehörigen Anzeigebereich zugegriffen werden. Zur flexiblen Nutzung der Anzeigebereiche kann eine Anwendung, die eine kontext-beschränkte Berechtigung *cp* – nach der englischen Übersetzung „**c**ontext constrained **p**ermission“ – besitzt, einer anderen Anwendung eine kontext-beschränkte Berechtigung *vergeben*, wenn sie nicht über die Größe des Anzeigebereichs und der Kontexte von *cp* hinausgeht.

Da die Zugriffskontrolle für Anzeigebereiche garantiert, dass der Zugriff für jeden Pixel zu jeder Zeit nur genau einer Anwendung gewährt wird, kann bei einer Weitergabe einer kontext-beschränkten Berechtigung die vergebende Anwendung nicht mehr auf den entsprechenden Anzeigebereich zugreifen. Dazu ist das *Entziehen* einer kontext-beschränkten Berechtigung nötig, damit die vergebende Anwendung wieder Zugriff auf den entsprechenden Anzeigebereich bekommt.

Die Vergabe und das Entziehen von kontext-beschränkten Berechtigungen ermöglicht es, die Zugriffskontrolle für Anzeigebereiche in dezentralen Entwicklungsprozessen zu verwenden. Der OEM kann die Entwicklung bestimmter Anwendungen oder Anwendungskomponenten an einen Zulieferer delegieren. Dieser Zulieferer kann dann Teile von Anwendungen an weitere Zulieferer delegieren usw. Dazu muss eine *Delegationsbeziehung* zwischen dem Vergebenden und dem Empfänger einer kontext-beschränkten Berechtigungen bestehen, die von beiden bestätigt ist. Im Folgenden werden die Entitäten des Zugriffskontrollmodells definiert.

3.3.1. Objekte und Subjekte

Wie in Abbildung 3.5 dargestellt, sind *Anzeigebereiche* Objekte, auf die *Anwendungen* (Subjekte) Zugriff erhalten, wenn eine Berechtigung vorliegt. Ein Anzeigebereich besteht aus einer Menge aus Pixeln, wobei der kleinste Anzeigebereiche ein einzelner Pixel ist, der als *atomares Objekt* bezeichnet wird. Der gesamte Anzeigebereich wird *Bildschirmbereich* genannt und besteht aus allen Pixeln.

Definition 1 $\boldsymbol{\Lambda} = \{\lambda_1, ..., \lambda_n\}$ *ist eine endliche Menge, bestehend aus Pixeln (Atomare Objekte). Ein Anzeigebereich ist eine Untermenge der Menge der verfügbaren Pixel. Formal ist ein Anzeigebereich o ein Objekt* $o \in \boldsymbol{O} = \mathcal{P}(\Lambda)\backslash\emptyset$*, wobei O die Menge aller Anzeigebereiche darstellt.*

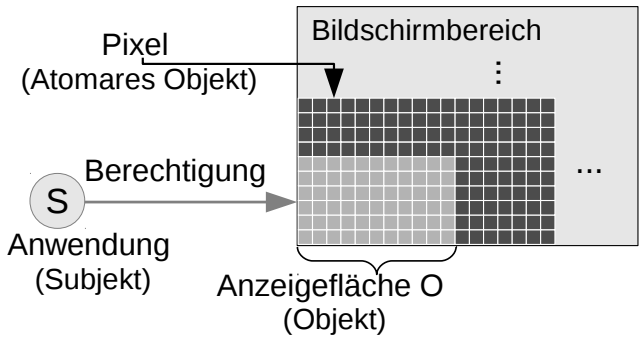

Abbildung 3.5.: Subjekte und Objekte

Damit eine Anwendung (Subjekt) Zugriff auf einen Anzeigebereich bekommt, benötigt sie eine Berechtigung.

Definition 2 $\boldsymbol{S} = \{s_1, ..., s_n\}$ *ist eine Menge aus Anwendungen (Subjekte) mit* $n \geq 1$.

3.3.2. Kontexte

Im Fahrzeug hängt das Anzeigen von Anwendungsinhalten meist vom gegebenen Kontext des Fahrzeug oder der Anwendungen ab. Wie in Abbildung 3.6 dargestellt, werden zwischen drei verschiedenen Quellen für Kontexte unterschieden. Die Datenquelle *Sensordaten des Fahrzeugs* liefert Informationen über den Status des Fahrzeugs (z. B. die Motordrehzahl oder den Status der Bremsen) oder die Bedingungen der Umgebung (z. B. die Distanz zum vorderen Fahrzeug). Die Datenquelle *Kommunikationsereignisse* ist zuständig für Ereignisse, die aufgrund von eingehenden Informationen mittels Kommunikationsgeräten stattfinden z. B. Telefonanrufe, SMS oder Emails. Die dritte Datenquelle *Benutzerschnittstellenereignisse* behandelt Ereignisse, die aufgrund von Benutzereingaben ausgelöst werden z. B. die Auswahl des Radios im Menü. Die Daten, die die Datenquellen bereitstellen, werden von den *Kontextlieferanten* interpretiert. Die Kontextlieferanten bestimmen, um welchen Kontext es sich handelt und ob dieser *aktiv* oder *inaktiv* ist. Die Kontextlieferanten sind in der Regel Anwendungen. Da die Daten der Datenquellen Einfluss auf die Darstellung von sicherheitskritischen Anwendungen haben, müssen die gelieferten Informationen zur Ermittlung der Kontexte vertrauenswürdig und zuverlässig sein. Dies gilt für Informationen, die z. B. über die Sensoren des Fahrzeugs ermittelt werden und Warnungen oder Fehlerzustände darstellen, wie die Kollisionswarnung. Jegliche Manipulation der Daten, die teils über den Fahrzeugbus (z. B. CAN) gesendet und von unterschiedlichen Steuergeräten gelesen werden, durch Fehler im System oder durch externes Einwirken

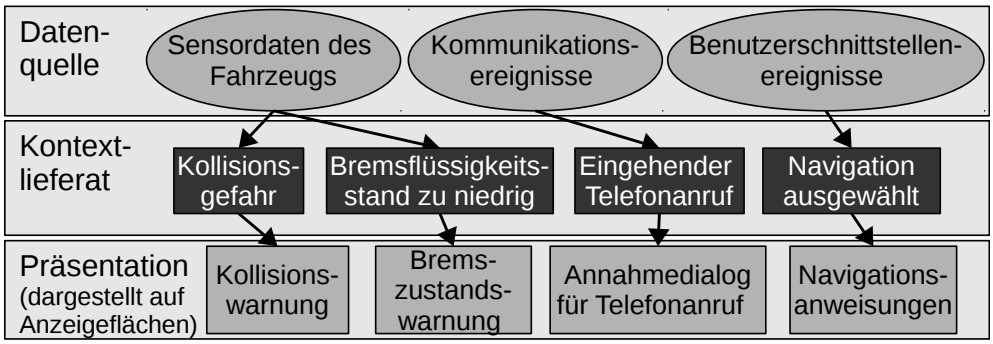

Abbildung 3.6.: Beispiele für Kontexte im Fahrzeug

kann die funktionale Sicherheit des Fahrzeugs gefährden. Da dies nicht im Fokus dieser Arbeit steht, sei auf die in Kapitel 3.8 genannten verwandten Arbeiten verwiesen, die sich mit der sicheren Kommunikation im Fahrzeug befassen.

In diesem Modell stellen die Anwendungen die Kontextlieferanten dar und jeder Kontext ist genau einer Anwendung zugeordnet, die den Zustand des Kontexts bestimmt. Damit die Kontexte eindeutig bestimmt werden können, wird jedem Kontext eine ID zugeordnet, die innerhalb einer Anwendung eindeutig ist. Das bedeutet, ein Kontext ist eindeutig identifizierbar durch seine Zuordnung zu einer Anwendung und seiner ID. Im Folgenden werden Kontexte formal definiert.

Definition 3 $\boldsymbol{C} = (S \times \mathbb{N})$ *ist die Menge aller Kontexte. Jeder Kontext wird durch eine zugehörige Anwendung und einer ID repräsentiert.*
$\boldsymbol{CTX} = \{f : C \to A\}$ *ist eine Menge aus Funktionen, welche für jeden Kontext dessen Status* $\boldsymbol{A} = \{aktiv, inaktiv\}$ *zurückliefern.*

Ein Kontext (z. B. mit ID 1) stellt einen Zustand wie *„Fahrzeug fährt“* dar, der von der Anwendung Geschwindigkeitsanzeige s_{GA} bestimmt wird und eindeutig durch $(s_{GA}, 1) \in C$ identifiziert werden kann. Dies bedeutet, wenn die aktuelle Geschwindigkeit über $0\,km\backslash h$ ist, dann setzt die Geschwindigkeitsanzeige den Status des Kontexts $(s_{GA}, 1)$ auf *aktiv*, ansonsten auf *inaktiv*. Seien beispielsweise $ctx \in CTX$ eine Menge von Kontexten und der Kontext $(s_{GA}, 1) \in C$ mit Status *aktiv* ein Element von ctx, dann gilt $((s_{GA}, 1), aktiv) \in ctx$. Seien $a, a' \in A$ und $a = aktiv$ dann liefert die Funktion $inv(a) = a'$ mit $a' = inaktiv$ zurück.

3.3.3. Kontext-beschränkte Berechtigungen

Eine kontext-beschränkte Berechtigung stellt eine Berechtigung dar, die nur für bestimmte Kontexte gilt. Eine Anwendung darf auf einen Anzeigebereich nur

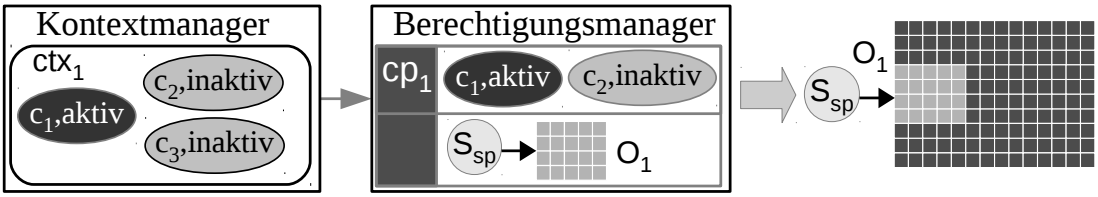

Abbildung 3.7.: Beispiel für Kontexte und kontext-beschränkte Berechtigungen

dann zugreifen, wenn sie eine kontext-beschränkte Berechtigung besitzt, die Zugriff auf diesen Anzeigebereich gewährt und die Kontexte mit den aktuellen übereinstimmen. Eine kontext-beschränkte Berechtigung ist formal wie folgt definiert.

Definition 4 $\boldsymbol{CP}$: *$CTX \times S \times O$ ist eine Menge von kontext-beschränkten Berechtigungen, die festlegen, welche Anwendungen Zugriff auf bestimmte Anzeigebereiche für bestimmte Kontexte erhalten.*

Wie in Abbildung 3.7 dargestellt, soll die Anwendung s_{GA} nur sichtbar sein, wenn das Fahrzeug fährt (d. h. Kontext $c_1 = (s_{GA}, 1) \in C$ ist aktiv). Die Anwendung s_{GA} besitzt eine kontext-beschränkte Berechtigung cp_1, die Zugriff auf der Anzeigebereich o_1 gewährt und auf den Kontext c_1, sowie auf den inaktiven Kontext $c_2 = (s_{GA}, 2)$ beschränkt ist. Die Anwendung s_{GA} ist genau dann sichtbar, bzw. erhält Zugriff auf den Anzeigebereiche o_1, wenn die Menge der aktuellen Kontexte $ctx_1 \in CTX$ die beiden Kontexte $(c_1, aktiv)$ und $(c_2, inaktiv)$ enthält.

Kontext-beschränkte Berechtigungen können in Konflikt stehen, wenn sie gleichzeitig Zugriff auf dieselben oder Teile desselben Anzeigebereichs gewähren. Dies kann geschehen, wenn die Kontexte der Berechtigungen zur selben Zeit erfüllt sind und die vergebenen Anzeigebereiche nicht schnittfrei sind. Die kontext-beschränkten Berechtigungen müssen daher *konfliktfrei* sein, sodass nicht mehr als eine Anwendung zur selben Zeit Zugriff auf dieselben Pixel erhält.

Seien die beiden kontext-beschränkten Berechtigungen cp und cp' mit $cp = (C_1, s_1, o_1) \in CP$ und $cp' = (C_2, s_2, o_2) \in CP$ definiert. Die beiden kontext-beschränkten Berechtigungen cp und cp' sind genau dann *konfliktfrei*, wenn die Schnittmenge der beiden Anzeigebereiche o_1 und o_2 leer ist oder der Kontext c in der Kontextmenge C_1 und C_2 ist, aber der Status sich jeweils unterscheidet, sodass beide Kontexte nicht zur selben Zeit erfüllt werden können. Im Folgenden wird die *Konfliktfreiheit* formal definiert.

Definition 5 *Seien die kontext-beschränkten Berechtigungen $cp = (C_1, s_1, o_1) \in CP$ und $cp' = (C_2, s_2, o_2) \in CP$. cp und cp' sind* ***konfliktfrei*** $\Leftrightarrow cp \sqcap cp' = \emptyset \Leftrightarrow o_1 \cap o_2 = \emptyset \vee \exists cx = (c, a) \in C_1, \exists cx' = (c', a') \in C_2 : c = c' \wedge a \neq a'$.

Sei $Im(ctx) = \{ctx(c)|c \in C\}$. *Dann gilt* $cp \sqsubseteq cp' \Leftrightarrow Im(ctx_{cp'}) \subseteq Im(ctx_{cp}) \wedge o_{cp} \subseteq o_{cp'}$.

Seien $cp_{GA} = (\hat{C}, s_{GA}, o) \in CP$ und $cp_{AT} = (\tilde{C}, s_{AT}, \tilde{o}) \in CP$ die kontext-beschränkten Berechtigungen der Geschwindigkeitsanzeige s_{GA} und des Abstandsregeltempomats s_{AT}. Des Weiteren sei der Anzeigebereich o deckungsgleich mit dem Anzeigebereich $\tilde{o}$. In diesem Fall gilt, dass die kontext-beschränkten Berechtigungen cp und cp' genau dann *konfliktfrei* sind, wenn die Kontextmengen $\hat{C}$ und $\tilde{C}$ nicht für dieselbe Kontextmenge gültig sind und somit gleichzeitig auf den selben Anzeigebereich Zugriff gewähren können.

Um die Flexibilität bei der Vergabe der kontext-beschränkten Berechtigungen zu erhöhen, wird im Folgenden die hierarchische Vergabe und das Entziehen von kontext-beschränkten Berechtigungen beschrieben.

Hierarchische Vergabe von kontext-beschränkten Berechtigungen

Eine Anwendung, die eine kontext-beschränkte Berechtigung cp erhalten hat, kann ihrerseits eine kontext-beschränkte Berechtigung cp' unter bestimmten Bedingungen an andere Anwendungen vergeben. Erstens müssen beide Anwendungen in einer Delegationsbeziehung stehen, die für die Vergabe von kontext-beschränkten Berechtigungen notwendig ist. Zweitens muss der Anzeigebereich von cp' identisch oder eine Teilfläche des Anzeigebereichs in cp sein. Drittens muss cp' im Vergleich zu cp mindestens auf dieselben Kontexte oder durch weitere Kontexte beschränkt sein. Viertens müssen alle weiteren kontext-beschränkten Berechtigungen, welche basierend auf cp durch die Anwendung vergeben wurden, ebenfalls konfliktfrei sein. Wenn diese Bedingungen alle erfüllt sind, dann wurde die kontext-beschränkte Berechtigung cp' erfolgreich vergeben und die vergebende Anwendung kann nicht mehr auf den Anzeigebereich in cp zugreifen, wenn deren Kontexte erfüllt sind. Im Folgenden wird eine Menge definiert, die pro Anwendung die erhaltenen und vergebenen kontext-beschränkten Berechtigungen enthält.

Definition 6 $\boldsymbol{CONS} = \{f : S \rightarrow \mathcal{P}(S \times CP) \times \mathcal{P}(S \times CP)\}$ *enthält für jede Anwendung die beiden Mengen* $\mathcal{P}(S \times CP)$*, welche die kontext-beschränkten Berechtigungen auf Anwendungen abbilden und die* erhaltenen *und* vergebenen *kontext-beschränkten Berechtigungen einer Anwendung darstellen.*

Seien $cons \in CONS, s \in S$. Die Menge $received_{cp}(cons, s) := \{r|(r, g) \in cons(s)\}$ ist die Menge der kontext-beschränkten Berechtigungen, die Anwendung s erhalten (engl. received) hat. Die Menge $granted_{cp}(cons, s) := \{g|(r, g) \in$

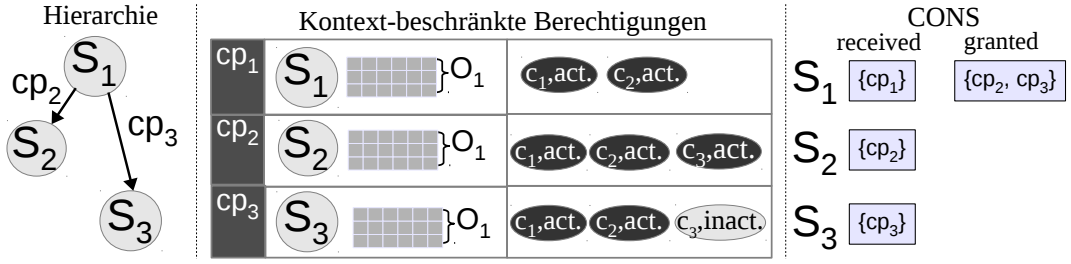

Abbildung 3.8.: Beispiel für eine Vergabe von kontext-beschränkten Berechtigungen der Anwendung S_1 an S_2 und S_3

$cons(s)\}$ steht für die Menge der kontext-beschränkten Berechtigungen, die Anwendung s vergeben (engl. granted) hat. $(s', cp') \in received_{cp}(cons, s)$ gibt an, dass die Anwendung s die kontext-beschränkte Berechtigung cp' von Anwendung s' erhalten hat. $(s', cp') \in granted_{cp}(cons, s)$ hingegen gibt an, dass die Anwendung s' die kontext-beschränkte Berechtigung cp' an Anwendung s' vergeben hat.

In Abbildung 3.8 ist ein Beispiel für die Vergabe von kontext-beschränkten Berechtigungen zwischen der Anwendung S_1 an S_2 und S_3 dargestellt. Sei die Menge $cons \subseteq CONS$, die Menge der momentanen kontext-beschränkten Berechtigungen, und habe die Anwendung S_1, die kontext-beschränkte Berechtigung cp_1 von der Anwendung $\hat{S}$ (nicht in Abbildung 3.8 dargestellt) erhalten. Die kontext-beschränkten Berechtigungen $\{cp_2\}$ und $\{cp_3\}$ wurden von Anwendung S_1 an die Anwendungen S_2 und S_3 vergeben. Im Beispiel sind die Anzeigebereiche der kontext-beschränkten Berechtigungen cp_1, cp_2 und cp_3 identisch. Die kontext-beschränkten Berechtigungen cp_2 und cp_3 sind dennoch konfliktfrei ($cp_2 \sqcap cp_3 = \emptyset$), da der Kontext c_3 in der kontext-beschränkten Berechtigung cp_2 im Status *aktiv* sein muss, in cp_2 jedoch im Status *inaktiv*, was nicht gleichzeitig sein kann. Da jeder Kontext, der die kontext-beschränkte Berechtigung cp_1 erfüllt, auch entweder cp_2 oder cp_3 erfüllt, hat Anwendung S_1 folglich keinen Zugriff mehr auf den Anzeigebereich o_1. Diesen kann sie nur wieder erhalten, wenn sie entweder cp_2 oder cp_3 entzieht. Dies bedeutet, die Anzeigebereiche, die eine Anwendung tatsächlich nutzt, hängen nicht nur von den momentanen Kontexten, sondern auch von den kontext-beschränkten Berechtigungen ab, die sie an andere Anwendungen vergeben hat.

Es wird eine Menge von Funktionen $received_o$ definiert, die zu den Mengen aus kontext-beschränkten Berechtigungen und Kontexten, die Anzeigebereiche einer Anwendung zurückliefern, für die sie Berechtigungen erhalten hat. Die Menge an Funktionen $granted_o$ gibt die vergebenen Anzeigebereiche einer Anwendung abhängig der kontext-beschränkten Berechtigungen und Kontexte zurück.

Definition 7 *$received_o = \{f : CONS \times CTX \times S \rightarrow O\}$ ist eine Menge aus Funktionen, die für die Anwendungen die Anzeigebereiche zurückliefern, auf welche diese abhängig der Berechtigungen und Kontexte Zugriff erhalten haben. Seien $ctx \in CTX, cons \in CONS$ und $s, s' \in S$. Dann ist $received_o(cons, ctx, s) =$*

$$\{o \in O | (s', cp) \in received_{cp}(cons, ctx, s) \wedge cp = (ctx_{cp}, s, o) \wedge ctx_{cp} \subseteq ctx\}.$$

Seien $cons \in CONS, s \in S$ und $ctx \in CTX$. Die Menge aller Anzeigebereiche, welche eine Anwendung in Abhängigkeit zu den Berechtigungen und Kontexten erhalten hat, ist definiert als

$$\Lambda_{received}(cons, ctx, s) := \bigcup received_o(cons, ctx, s).$$

Vergleichbar dazu ist $granted_o = \{f : CONS \times CTX \times S \rightarrow O\}$ eine Menge aus Funktionen, die für die Anwendungen die Anzeigebereiche zurückliefern, welche diese abhängig der Berechtigungen und Kontexte vergeben haben.
Seien $ctx \in CTX, cons \in CONS$ und $s, s' \in S$. Dann ist $granted_o(cons, ctx, s) =$

$$\{o \in O | (s', cp) \in granted_{cp}(cons, ctx, s) \wedge cp = (ctx_{cp}, (s, o)) \wedge ctx_{cp} \subseteq ctx\}.$$

Seien $cons \in CONS, s \in S$ und $ctx \in CTX$. Die Menge aller Anzeigebereiche, welche abhängig der Berechtigungen und Kontexte von einer Anwendung an andere vergeben wurden, ist definiert als

$$\Gamma_{granted}(cons, ctx, s) := \bigcup granted_o(cons, ctx, s).$$

Um die *Abhängigkeit* zwischen den Anwendungen bezüglich den vergebenen und erhaltenen kontext-beschränkten Berechtigungen anzuzeigen, wird der Vergleichsoperator $<_{cp}$ verwendet. Dieser gibt an, ob eine Anwendung direkt oder indirekt über eine Kette aus Vergaben bezüglich einer kontext-beschränkten Berechtigung von einer anderen Anwendung abhängt.

Definition 8 *Seien $s, s' \in S, cp \in CP, cons \in CONS$. Dann gelte $\boldsymbol{s <_{cp} s'} \Leftrightarrow$*

$$\exists s_1, ..., s_n \in S; \exists cp_1, ..., cp_{n-1} \in CP : \tag{8.1}$$
$$s_1 = s' \wedge s_n = s \wedge cp \sqsubseteq cp_{n-1} \wedge \tag{8.2}$$
$$\forall i : 1 \leq i < n : (s_i, cp_i) \in received_{cp}(cons, s_{i+1}) \wedge \tag{8.3}$$
$$\forall i : 1 \leq i < n - 2 : cp_{i+1} \sqsubseteq cp_i \tag{8.4}$$

$s <_{cp} s'$ bedeutet, dass die Anwendung s die kontext-beschränkte Berechtigung cp entweder direkt erhalten hat oder diese mittels mehrerer Vergaben von s' über

andere Anwendungen zu Anwendung s gekommen ist. Die Anwendung s hängt von Anwendung s' bzgl. der kontext-beschränkten Berechtigung cp ab.

Seien $s, s' \in S$. Die Anwendung s ist *unabhängig* von Anwendung s' und umgekehrt bezüglich kontext-beschränkten Berechtigungen. Formal, $s \neq_{cp} s' \Leftrightarrow \nexists cp \in CP : s <_{cp} s' \vee s' <_{cp} s$. Das heißt, die Anwendungen s und s' haben sich gegenseitig keine kontext-beschränkten Berechtigungen vergeben.

Ähnlich zum Vergleichsoperator $<_{cp}$ kann die Abhängigkeit der Anwendungen bzgl. Anzeigebereichen angegeben werden. Es wird ein Vergleichsoperator $<_{o,ctx}$ definiert, der abhängig einer Kontextmenge ctx und eines Anzeigebereichs o angibt, ob eine Anwendung direkt oder indirekt eine Teilmenge von o erhalten hat.

Definition 9 *Seien $s, s' \in S, o \in O$, $cons \in CONS$. Dann gelte $\boldsymbol{s <_{o,ctx} s'} \Leftrightarrow$*

$$\exists s_1, ..., s_n \in S; \exists o_1, ..., o_{n-1} \in O : \tag{9.1}$$

$$s_1 = s' \wedge s_n = s \wedge o \subseteq o_{n-1} \wedge \tag{9.2}$$

$$\forall i : 1 \leq i < n : (s_i, o_i) \in received_o(cons, ctx, s_{i+1}) \wedge \tag{9.3}$$

$$\forall i : 1 \leq i < n - 2 : o_{i+1} \subseteq o_i \tag{9.4}$$

Seien $s, s' \in S$. Die Anwendung s ist *unabhängig* von Anwendung s' und umgekehrt bezüglich des Anzeigebereichs o mit Kontextmenge ctx. Wie bei dem Vergleichsoperator $<_{cp}$ gilt auch hier, $s \neq_{o,ctx} s' \Leftrightarrow \nexists o \in O : s <_{o,ctx} s' \vee s' <_{o,ctx} s$. Das heißt, s und s' haben sich mit Kontextmenge ctx keinen Anzeigebereich bzw. keine Teilmenge von o mittels kontext-beschränkten Berechtigungen vergeben.

Kontext-beschränkte Berechtigungen werden zwischen Anwendungen nur dann vergeben, wenn diese sich in einer *Delegationsbeziehung* befinden. Jede Anwendung kann in einer Delegationsbeziehung mit einer oder mehreren Anwendungen stehen. Ohne Delegationsbeziehung können nicht-sicherheitskritische Anwendungen keine kontext-beschränkten Berechtigungen an sicherheitskritische Anwendungen vergeben, sodass keine Abhängigkeiten bzgl. der Anzeigebereiche zwischen diesen Anwendungen entstehen. Eine Delegationsbeziehung wird nur angelegt, wenn eine gegenseitige Übereinkunft zwischen den Anwendungen besteht.

Definition 10 *Jede Anwendung wird auf eine Menge an Anwendungen abgebildet, mit welchen sie in einer Delegationsbeziehung steht. $D = \mathcal{P}(S \times S)$ repräsentiert die Menge aller möglichen Kombinationen an Delegationsbeziehungen zwischen Anwendungen. Die Anwendungen $s, s' \in S$ mit der Menge $d \in D$ befinden sich jeweils genau dann in einer Delegationsbeziehung zu dem anderen, wenn $(s, s') \in d$ und $(s', s) \in d$ gilt.*

Die Delegationsbeziehungen decken sich mit dem dezentralen Entwicklungsprozess in Abbildung 3.3. Mittels der Delegationsbeziehungen kann im Entwicklungsprozess die Verteilung von kontext-beschränkten Berechtigungen an verschiedene Zulieferer eingeschränkt werden. Um während der Laufzeit das Bereitstellen der Anwendungen auf dem System zu ermöglichen, können Anwendungen dynamisch festlegen, mit welchen Anwendungen sie eine Delegationsbeziehung eingehen.

3.4. Protokoll

In diesem Abschnitt werden *Anfragen* beschrieben, die die Anwendungen abschicken können, um die kontext-beschränkten Berechtigungen, Kontexte oder Delegationsbeziehungen zu ändern. Dazu werden *Regeln* definiert, die festlegen, wie *Transitionen* durchgeführt werden. Eine Transition stellt ein Zustandsübergang dar, der von einem Zustand $u \in U$ zu einem Zustand $u' \in U$ übergeht.

3.4.1. Anfragen

Eine Transition wird durch eine *Anfrage* einer Anwendung ausgelöst. Die Anfrage kann kontext-beschränkte Berechtigungen hinzufügen oder entfernen, den Status eines Kontexts ändern oder eine Delegationsbeziehungen aufbauen oder auflösen.

Definition 11 *Eine Anfrage* $r \in \boldsymbol{RP} = RA \times S \times S \times CP$ *besteht aus dem Aktionsmodus* $\boldsymbol{RA} = \{append, discard\}$*, der vergebenden Anwendung, der erhaltenden Anwendung und der kontext-beschränkten Berechtigung, welche zwischen den beiden Anwendungen ausgetauscht werden soll. Eine Anfrage* $\boldsymbol{RC} = S \times C \times A$ *besteht aus der Anwendung, dem betreffenden Kontext und dem Status, zu welchem der Kontext geändert werden soll. Eine Anfrage* $\boldsymbol{RD} = RA \times S \times S$ *besteht aus dem Aktionsmodus* $\boldsymbol{RA} = \{append, discard\}$*, der anfragenden Anwendung und der Anwendung, mit welcher eine Delegationsbeziehung eingegangen werden soll. Die Menge aller möglichen Anfragen ist* $\boldsymbol{R} = RP \cup RC \cup RD$*.*

3.4.2. Transitionen

Als nächstes werden Transitionen zwischen Zuständen definiert. Um Inkonsistenzen zu verhindern, werden die Transitionen durch Regeln beschränkt. Können die Regeln nicht angewandt werden bzw. wird gegen die Regeln verstoßen, dann wird eine Transition nicht ausgeführt. Eine Transition ist wie folgt definiert.

Definition 12 *$trans : U \times R \rightarrow U$ ist eine Funktion, die eine Transition mittels einer Anfrage von einem Zustand zu einem anderen Zustand in U durchführt.*

Um eine kontext-beschränkte Berechtigung cp an eine Anwendung s' zu geben, muss eine Anwendung s eine Anfrage $r \in RP$ mit $ra = append$ initiieren. Die Anfrage wird akzeptiert, wenn s eine kontext-beschränkte Berechtigung erhalten hat, die eine Obermenge der zu vergebenden kontext-beschränkten Berechtigung cp ist und cp konfliktfrei zu allen bereits vergebenen kontext-beschränkten Berechtigungen von s ist. Anwendung s kann die kontext-beschränkte Berechtigung durch zusätzliche Kontexte weiter einschränken. Der Status eines neuen Kontexts kann nur von Anwendung s gesetzt werden. Dies wird in Regel 1 formal definiert.
Regel 1: Es muss die folgende Bedingung $cond_1$ erfüllt sein.

$$
\begin{aligned}
&cond_1 = (r = (ra, s, s', cp) \in RP \wedge ra = append \wedge cp = (\hat{ctx}, \hat{s}, \hat{o}) \wedge \\
&s \neq s' \wedge s' = \hat{s} \wedge \\
&[\forall cp' \in \{\hat{cp} \in CP | \exists (\hat{s}, \hat{cp}) \in granted_{cp}(cons, s)\} : cp \sqcap cp' = \emptyset] \wedge \\
&[\exists \tilde{cp} \in \{\hat{cp} \in CP | \exists (\hat{s}, \hat{cp}) \in received_{cp}(cons, s)\} : cp \sqsubseteq \tilde{cp} = (\tilde{ctx}, \tilde{s}, \tilde{o}) \wedge \tilde{s} = s] \wedge \\
&\forall ((s'', n), a) \in \hat{ctx} \backslash \tilde{ctx} : s'' = s)
\end{aligned}
$$

Ist Bedingung $cond_1$ erfüllt, dann wird die Transition ausgeführt. Hierzu wird die Funktion $add_{cp} : CONS \times S \times S \times CP \rightarrow CONS$ verwendet, welche eine kontext-beschränkte Berechtigung in die Mengen der erhaltenen und vergebenen Berechtigungen der beteiligten Anwendungen hinzufügt. Seien $s, s' \in S$ und $cp = (C_{cp}, s_{cp}, o_{cp}) \in CP$. Es gilt $cons' = add_{cp}(cons, s, s', cp) \Leftrightarrow$

$$
\begin{aligned}
&cons'(s) = (received_{cp}(cons, s), granted_{cp}(cons, s) \cup \{(s', cp)\}) \wedge && (12.1.1) \\
&cons'(s') = (received_{cp}(cons, s') \cup \{(s, cp)\}, granted_{cp}(cons, s')) \wedge && (12.1.2) \\
&[\forall s'' \in S \backslash \{s', s\} : cons'(s'') = cons(s'')] && (12.1.3)
\end{aligned}
$$

Die Funktion add_{cp} fügt die kontext-beschränkte Berechtigung cp zur Menge der erhaltenen Berechtigungen $received_{cp}$ und vergebenen Berechtigungen $granted_{cp}$ der Anwendung s und der Anwendung s' in $cons$ hinzu (siehe 12.1.1-2). Alle anderen Anwendungen werden von dieser Funktion nicht beeinflusst (12.1.3).

Um eine kontext-beschränkte Berechtigung cp zu entziehen, kann eine Anwendung s eine Anfrage $r \in RP$ mit $ra = discard$ verwenden. Wenn die Anwendung s die kontext-beschränkte Berechtigung cp zuvor vergeben hat, dann ist die Anfrage gültig und es werden alle kontext-beschränkte Berechtigungen, welche von

cp abhängen, entzogen. Dies ist formal durch Regel 2 beschrieben.
Regel 2: Es muss die Bedingung $cond_2$ erfüllt sein.

$$\begin{aligned} cond_2 =&(r = (ra, s, s', cp) \in RP \wedge ra = discard \wedge \\ &(s \neq s' \wedge (s', cp) \in granted_{cp}(cons, s) \wedge (s, cp) \in received_{cp}(cons, s'))) \end{aligned}$$

Wenn die Bedingung $cond_2$ erfüllt ist, dann wird die Transition durchgeführt. Dazu wird die Funktion $del_{cp} : CONS \times S \times S \times CP \rightarrow CONS$ verwendet, welche die betreffende kontext-beschränkte Berechtigung und alle davon Abhängenden entfernt. Seien $s, s' \in S, cp = (C_{cp}, s_{cp}, o_{cp})$ und $cp' = (C_{cp'}, s_{cp'}, o_{cp'}) \in CP$. Dann ist $cons' = del_{cp}(cons, s, s', cp) \Leftrightarrow$

$$[\forall s_1 \in S : s_1 <_{cp} s \Rightarrow \tag{12.2.1}$$
$$received_{cp}(cons', s_1) = received_{cp}(cons, s_1) \setminus \tag{12.2.2}$$
$$\{(s_2, cp') \in S \times CP | s_2 <_{cp'} s_1 \wedge cp' \sqsubseteq cp\} \wedge \tag{12.2.3}$$
$$granted_{cp}(cons', s_1) = granted_{cp}(cons, s_1) \setminus \tag{12.2.4}$$
$$\{(s_2, cp') \in S \times CS | s_1 <_{cp'} s_2 \wedge cp' \sqsubseteq cp\}] \wedge \tag{12.2.5}$$
$$cons'(s) = (received_{cp}(cons, s), granted_{cp}(cons, s) \setminus \{(s', cp)\}) \wedge \tag{12.2.6}$$
$$[\forall s_3 \in S \setminus \{s', s\}; \forall cp' \in CP : cp' \sqsubseteq cp \wedge s \neq_{cp'} s_3 \vee s <_{cp'} s_3 \tag{12.2.7}$$
$$\Rightarrow cons'(s_3) = cons(s_3)] \tag{12.2.8}$$

Die Funktion del_{cp} wird ausgeführt, wenn die Bedingung $cond_2$ erfüllt ist. Die Funktion del_{cp} entfernt die kontext-beschränkte Berechtigung cp und alle davon abhängigen Berechtigungen aus der Menge der erhaltenen Berechtigungen $received_{cp}$ (siehe 12.2.2-3) und aus der Menge $granted_{cp}$ (siehe 12.2.4-6) der kontext-beschränkten Berechtigungen in $cons$, womit diese zu $cons'$ wird. Die anderen Anwendungen, die keine abhängigen kontext-beschränkten Berechtigungen zu cp haben, werden durch die Funktion nicht beeinflusst (siehe 12.2.7-8).
Eine Anwendung s kann mit Anfrage $r \in RC$ den Status eines Kontexts c ändern. Die Anfrage ist gültig, wenn die anfragende Anwendung den Kontext einführte.
Regel 3: Es muss die folgende Bedingung $cond_3$ erfüllt sein.

$$cond_3 = (r = (s, c, a) \in RC \wedge c = ((\tilde{s}, n), a) \in ctx \wedge \tilde{s} = s)$$

Wenn die Bedingung $cond_3$ erfüllt ist, dann wird die Transition ausgeführt. Die Transition verwendet die Funktion $set_{ctx} : CTX \times C \times A \rightarrow CTX$ um einen

Kontext zu der Menge der derzeitigen Kontexte hinzuzufügen. Formal gilt für die Funktion: $set_{ctx}(ctx, c, a) = ctx' \Leftrightarrow ctx'(c) = a \wedge \forall c' \in C \backslash \{c\} : ctx'(c') = ctx(c')$.

Damit eine Delegationsbeziehung zu einer anderen Anwendung eingegangen werden kann, muss eine Anwendung eine Anfrage $r \in RD$ initiieren. Die Anfrage wird akzeptiert, wenn die anfragende Anwendung und die Anwendung, zu welcher eine Delegationsbeziehung eingegangen werden soll, nicht identisch sind.
Regel 4: Es muss die Bedingung $cond_4$ erfüllt sein.

$$cond_4 = r = (ra, s, s') \in R_d \wedge s \neq s' \wedge ra = append$$

Die Anwendung s kann eine Anfrage $r \in RD$ mit $ra = append$ initiieren, um zu der Anwendung s' eine Delegationsbeziehung einzugehen. Die Delegationsbeziehung ist nur dann vollständig, wenn die Anwendung s' ebenfalls eine Anfrage $r \in RD$ mit $ra = append$ zu der Anwendung s initiiert hat. Erst danach können die beiden Anwendungen kontext-beschränkte Berechtigungen austauschen.
Eine Delegationsbeziehung kann mit Anfrage $r \in RD$ initiiert werden.
Regel 5: Um die Regel 5 zu erfüllen, muss die Bedingung $cond_5$ erfüllt sein.

$$cond_5 = (r = (ra, s, s') \in R_d \wedge s \neq s' \wedge ra = discard \wedge s \neq_{cp} s')$$

Damit eine Anwendung s eine Delegationsbeziehung zu der Anwendung s' auflösen kann, muss sie eine Anfrage $r \in RD$ mit $ra = discard$ initiieren. Außerdem müssen die beiden Anwendungen verschieden sein ($s \neq s'$) und dürfen sich keine kontext-beschränkten Berechtigungen gegeben haben ($s \neq_{cp} s'$).

Als nächstes wird die Funktion $trans$ mit den fünf Regeln definiert.

Seien $u = (cons, ctx, d) \in U$ und $r \in R$ mit $r = (ra, s, s', cp) \in RP$, $r = (ra, s, s') \in RD$ oder $r = (s, c, a) \in RC$. Dann gilt

$$trans(u, r) = \begin{cases} (add_{cp}(cons, s, s', cp), ctx, d), & \text{wenn } cond_1 \quad \text{(Regel 1)} \\ (del_{cp}(cons, s, s', cp), ctx, d), & \text{wenn } cond_2 \quad \text{(Regel 2)} \\ (cons, set_{ctx}(ctx, c, a), d), & \text{wenn } cond_3 \quad \text{(Regel 3)} \\ (cons, ctx, d \cup \{s, s'\}), & \text{wenn } cond_4 \quad \text{(Regel 4)} \\ (cons, ctx, d \backslash \{s, s'\}), & \text{wenn } cond_5 \quad \text{(Regel 5)} \\ u, & \text{ansonsten} \end{cases}$$

3.5. Korrektheit

Um die Korrektheit des Zugriffskontrollmodells zu verifizieren, wird ein *System* definiert, das aus Sequenzen von Zuständen und Anfragen besteht. Unter Verwendung von *Korrektheitseigenschaften* werden dann Sätze definiert. Die Sätze können mit dem formalen System bewiesen werden, indem man die Beweismethode *vollständige Induktion* über die Zustände des Systems anwendet. Ein Zustand ist *sicher*, wenn er die Korrektheitseigenschaften erfüllt. Der Beweis zeigt, dass das System sicher ist, wenn der initiale Zustand die Korrektheitseigenschaften erfüllt. Wenn der initiale Zustand z. B. aus nur einer Anwendung mit einer kontext-beschränkten Berechtigung besteht, dann ist dieser Zustand unabhängig der Kontexte ein sicherer Zustand. Der Beweis impliziert, dass mittels des Protokolls in dem Zugriffskontrollmodell nur sichere Zustände erreicht werden können, wenn der initiale Zustand sicher ist. Es folgt die formale Definition der Korrektheitseigenschaften.

3.5.1. Korrektheitseigenschaften für die funktionale Sicherheit

In diesem Abschnitt werden *Zustände* im Zugriffskontrollmodell definiert. Ein Zustand besteht aus Mengen aus kontext-beschränkten Berechtigungen, aus Kontexten und aus Delegationsbeziehungen zwischen den Anwendungen. Auf Grundlage der Zustände können Korrektheitseigenschaften definiert und verifiziert werden.

Definition 13 *$U : CONS \times CTX \times D$ repräsentiert die Menge der vergebenen und erhaltenen kontext-beschränkten Berechtigungen, der Kontexte und der Delegationsbeziehungen. U ist die Menge aller Zustände. Ein Element $u \in U$ wird ein Zustand in U genannt.*

Es wird zwischen Berechtigungen unterschieden, die eine Anwendung für einen Anzeigebereich erhalten hat und den Anzeigebereichen, die sie tatsächlich *nutzt*. Ein Anzeigebereich wird von einer Anwendung genutzt, wenn sie die Pixel einer Anzeigebereich schreiben kann. Wenn das der Fall ist, dann hat sie eine entsprechende Berechtigung erhalten, aber keine entsprechende Berechtigung vergeben.

Definition 14 *$used_o : CONS \times CTX \times S \to \mathcal{P}(O)$ ist eine Funktion, die die Menge der Anzeigebereiche zurückgibt, welche abhängig der vergebenen Berechtigungen und der Kontexte von einer Anwendung genutzt werden. Seien $o \in O$,*

$ctx' \in CTX$ und $cons \in CONS$. Dann gelte $o \in used_o(cons, ctx, s) \Leftrightarrow$

$$\exists \hat{o} \in O : \hat{o} \in received_o(cons, ctx, s) \wedge o \subseteq \hat{o} \wedge \tag{14.1}$$

$$\forall o' \in O : o' \in granted_o(cons, ctx, s) \Rightarrow o \cap o' = \emptyset \tag{14.2}$$

$\Omega_{used} : CONS \times CTX \rightarrow \mathcal{P}(O)$ ist eine Funktion, die die Menge der Anzeigebereiche zurückliefert, welche abhängig der Berechtigungen in $CONS$ und der aktuell gültigen Kontexte in CTX von den Anwendungen genutzt werden. Seien $cons \in CONS$ und $ctx \in CTX$. Dann gelte

$$\Omega_{used}(cons, ctx) := \bigcup \{o \in O | \exists s \in S : o \in used_o(cons, ctx, s)\}$$

$\Phi_{used}(cons, ctx, s) := \bigcup used_o(cons, ctx, s)$ ist die Menge der Mengen aller genutzten Anzeigebereiche der Berechtigungen in cons mit den Kontexten ctx einer Anwendung s.

Im Folgenden werden Korrektheitseigenschaften für einen Zustand u in U definiert. Zustände, die die Korrektheitseigenschaften erfüllen, werden *sichere Zustände* genannt. Eine Sequenz aus sicheren Zuständen wird *sichere Zustandssequenz* genannt.

Definition 15 ***Konfliktfreiheitseigenschaft (KFE):***
*Ein Zustand erfüllt die **Konfliktfreiheitseigenschaft (KFE)**, wenn jede kontext-beschränkte Berechtigung konfliktfrei ist. Sei Zustand $u = (cons, ctx, d) \in U$. Zustand u erfüllt die Konfliktfreiheitseigenschaft $\Leftrightarrow$*

$$\begin{aligned}
&\forall s, s', s_1, s_2 \in S \forall cp, cp' \in CP : s' \neq_{cp} s \wedge s' \neq_{cp'} s \wedge cp \neq cp' \wedge \\
&(s_1, cp) \in received_{cp}(cons, s) \wedge (s_2, cp') \in received_{cp}(cons, s') \\
&\Rightarrow (cp \sqcap cp' = \emptyset)
\end{aligned}$$

*Es existiert maximal eine kontext-beschränkte Berechtigung, die die Berechtigung für einen Anzeigebereich für eine Menge an Kontexten zur selben Zeit gewährt. Diese Eigenschaft impliziert den **exklusiven Zugriff**. Das heißt, auf jeden Anzeigebereich schreibt in jeder Kontextkombination nur maximal eine Anwendung.*

Definition 16 ***Vollständigkeitseigenschaft (VE):***
*Ein Zustand erfüllt die **Vollständigkeitseigenschaft (VE)**, wenn durch die kontext-beschränkten Berechtigungen gewährleistet ist, dass für jeden Pixel zu jeder Zeit mindestens eine Berechtigung vorliegt, sodass eine Anwendung darauf*

zugreifen kann.
Sei $u = (cons, ctx, d) \in U$. Zustand u erfüllt die Vollständigkeitseigenschaft $\Leftrightarrow$

$$\Omega_{used}(cons, ctx) = AO$$

Das heißt, dass jeder Pixel von mindestens einer Anwendung verwaltet wird und somit eine entsprechende Berechtigung vergeben wurde. Eine Anwendung, die eine Berechtigung für einen Pixel besitzt, kann diesen an eine andere Anwendung weitergeben, jedoch nicht löschen. Dadurch wird sichergestellt, dass es keine „toten" Pixel gibt, für die keine Anwendung eine Berechtigung zum Schreiben besitzt.

Definition 17 ***Delegationseigenschaft (DE):***
In einem Zustand, in welchem die ***Delegationseigenschaft (DE)*** *erfüllt ist, sind alle Berechtigungen nur zwischen Anwendungen ausgetauscht worden, welche in einer Delegationsbeziehungen stehen. Sei Zustand $u = (cons, ctx, d) \in U$. Zustand u erfüllt die Delegationseigenschaft* $\Leftrightarrow$

$$\begin{aligned}
&\forall s, s' \in S, \forall cp \in CP : s \neq s' \wedge s <_{cp} s' \Rightarrow \\
&\quad \exists s_0, ..., s_{n+1} \in S : s_0 = s \wedge s_{n+1} = s' \wedge \\
&\quad \forall i \in \{0, ..., n\} \subset \mathbb{N}_0 : s_i <_{cp} s_{i+1} \wedge (s_i, s_{i+1}) \in d \wedge (s_{i+1}, s_i) \in d
\end{aligned}$$

Dies bedeutet, dass die Vergabe der kontext-beschränkten Berechtigungen nur zwischen Anwendungen stattfindet, die sich in einer Delegationsbeziehung befinden. Dies soll verhindern, dass nicht vertrauenswürdige Anwendungen Berechtigungen an sicherheitskritische Anwendungen vergeben und diese dadurch beeinträchtigen.

3.5.2. System

Ein *System* besteht aus den *Sequenzen aus Anfragen* und den *Sequenzen aller Zustände*, die von einem initialen Zustand aus erreicht werden können. Für die Definition der Sequenzen wird zunächst die folgende Schreibweise festgelegt, die einen Index für die Elemente einer Sequenz darstellt.

$$I^n \subset \mathbb{N}_0 \textit{ ist eine endliche Menge mit } I^n = \{0, 1, 2, 3, ..., n\}$$

Es folgen die Definitionen der Sequenzen aus Zuständen und des Systems.

Definition 18 *Die* ***Menge aus Sequenzen aus Elementen*** *ist eine Menge aus n-Tupeln und definiert als $X^{I^n} = \{(x_0, ..., x_i, ..., x_n) | x_i \in X \wedge i \in I^n \wedge x_i =$*

$f(i)$ mit $f : I^n \to X\}$. $(x_0, x_1, ..., x_n) \in X^{I^n}$ ist eine Sequenz mit $x_0 := x_0 \in X$, $x_1 := x'_1 \in X$,..., $x_n := x_n^{(n)} \in X$.
*Der **Operator** $\succ$ gibt an, ob ein Element Teil einer Sequenz aus Elementen ist. Sei $(x_0, x_1, ..., x_n) \in X^{I^n}$. Es gilt $x \succ (x_0, x_1, ..., x_n) \Leftrightarrow \exists i \in I^n : x = x_i$.*

Eine Sequenz ist eine geordnete Menge aus Elementen und der **Operator** $\succ$ gibt an, ob ein Element Teil einer Sequenz mit zugehöriger Sequenznummer ist. Nach der generischen Definition von Sequenzen werden als nächstes die Sequenzen aus Anfragen und die Sequenzen aus Zuständen definiert.

Definition 19 *Für eine **Sequenz aus Anfragen** $(r_0, ..., r_{n-1}) \in R^{I^{n-1}}$ ist die **Sequenz aus Zuständen**, welche durch $(r_0, ..., r_{n-1})$ erzeugt wird, definiert als $(u_0, u_1, ..., u_n) \in U^{I^n}$ mit $\forall i \in I^{n-1} : u_{i+1} = trans(u_i, r_i)$.*

Unter Verwendung der Definitionen der Sequenzen aus Anfragen und Zuständen wird ein *System* definiert, welches aus einem initialen Zustand und allen mittels des Protokolls und den entsprechenden Sequenzen aus Anfragen erreichbaren Zuständen besteht. Zusätzlich wird ein **Operator** $\gg$ definiert, welcher angibt, ob eine Transition (u, r, u') Teil eines *Systems* ist oder nicht.

Definition 20 *Ein **System $\Psi(u_{start})$** $\subset R^{I^n} \times U^{I^n}$ wird durch den initialen Zustand u_{start} festgelegt. Sei $x_r = (r_0, ..., r_{n-1}) \in R^{I^{n-1}}$, $x_u = (u_0, ..., u_n) \in U^{I^n}$. Es gilt $(x_r, x_u) \in \Psi(u_{start}) \Leftrightarrow u_0 = u_{start} \wedge \forall i \in I^n \backslash \{0\} : u_i = trans(u_{i-1}, r_{i-1})$
Seien $(u, r, u') \in U \times R \times U$, $u_0 \in U$. Es gilt $\boldsymbol{(u, r, u') \gg \Psi(u_0)} \Leftrightarrow$*

$$\exists x_r \in R^{I^{n-1}}, \exists x_u \in U^{I^n}, \exists i \in I^n \backslash \{0\} : \tag{20.1}$$

$$(x_r, x_u) \in \Psi(u_0) \wedge u_i \succ x_u \wedge u_{i+1} \succ x_u \wedge r_i \succ x_r \wedge \tag{20.2}$$

$$(u, r, u') = (u_{i-1}, r_{i-1}, u_i). \tag{20.3}$$

Transition (u, r, u') ist Teil eines Systems, wenn eine Sequenz aus Anfragen und Zuständen (siehe 20.1) existiert, die u, r und u' enthält (siehe 20.2) und es eine Transition von Zustand u zu Zustand u' mittels Anfrage r gibt (siehe 20.3).

Ein System, welches nur aus sicheren Zustandssequenzen besteht, wird ein *sicheres System* genannt. Dies bedeutet, ein sicheres System erfüllt die Korrektheitseigenschaften in jedem erreichbaren Zustand innerhalb der Sequenzen. Als nächstes werden die formalen Definitionen für einen *sicheren Zustand* und ein *sicheres System*, welches nur aus *sicheren Zustandssequenzen* besteht, gegeben.

Definition 21 *$u \in U$ ist ein **sicherer Zustand** $\Leftrightarrow$ u erfüllt die Korrektheitseigenschaften. $(u_0, ..., u_n) \in U^{I^n}$ ist eine **sichere Zustandssequenz** $\Leftrightarrow \forall i \in$*

I^n : u_i ist ein sicherer Zustand. Ein System $\Psi(u_0) \subset U^{I^n} \times R^{I^n}$ mit $x_r \in R^{I^{n-1}}$ und $x_u = (u_0, ..., u_n) \in U^{I^n}$ ist ein ***sicheres System*** $\Leftrightarrow \forall(x_r, x_u) \in \Psi(u_0)$: *x_u ist eine sichere Zustandssequenz.*

Es folgt der Beweis, dass die Korrektheitseigenschaften aus Abschnitt 3.5.1 immer erfüllt sind, wenn der initiale Zustand die Korrektheitseigenschaften erfüllt.

3.5.3. Beweis der Korrektheit des Systems

Aufgrund der formalen Definition der Zustände und der Transitionen des Zugriffskontrollmodells, können für den Korrektheitsbeweis Sätze definiert werden, die aus den Korrektheitseigenschaften aus Abschnitt 3.5.1 abgeleitet sind. Dies ermöglicht es, einen formalen Beweis über die funktionale Sicherheit des Systems zu führen.
Seien $u, u', u_0 \in U$, $u' = (cons', ctx', d')$, $u = (cons, ctx, d)$ und $r \in R$.

Satz 1: Alle Sequenzen in $\Psi(u_0)$ erfüllen die Konfliktfreiheitseigenschaft (KFE) für alle initialen Zustände u_0, welche die KFE erfüllen $\Leftrightarrow$
$\forall(u, r, u') \in U \times R \times U : (u, r, u') \gg \Psi(u_0) \Rightarrow u, u' \in U$ erfüllt die KFE.

Satz 2: Alle Sequenzen in $\Psi(u_0)$ erfüllen die Vollständigkeitseigenschaft (VE) für alle initialen Zustände u_0, welche die VE und die KFE erfüllen $\Leftrightarrow$
$\forall(u, r, u') \in U \times R \times U : (u, r, u') \gg \Psi(u_0) \Rightarrow u, u' \in U$ erfüllt die VE.

Satz 3: Alle Sequenzen in $\Psi(u_0)$ erfüllen die Delegationseigenschaft (DE) für alle initialen Zustände u_0, welche die DE erfüllen $\Leftrightarrow$
$\forall(u, r, u') \in U \times R \times U : (u, r, u') \gg \Psi(u_0) \Rightarrow u, u' \in U$ erfüllt die DE.

Satz 1 sagt, dass alle Sequenzen in einem System $\Psi(u_0)$ die Konfliktfreiheitseigenschaft genau dann erfüllen, wenn Zustand u und alle Zustände u', die direkt von Zustand u mit einer Anfrage erzeugt werden können, die Konfliktfreiheitseigenschaft erfüllen. Satz 2 und 3 stellen dieselben Behauptungen auf, aber jeweils für die Vollständigkeitseigenschaft und die Delegationseigenschaft. Mittels dieser Sätze lässt sich zeigen, dass jedes System $\Psi(u_0)$ ein sicheres System ist, wenn der initiale Zustand u_0 die Korrektheitseigenschaften aus Abschnitt 3.5.1 erfüllt.

Um die Korrektheit der Sätze 1, 2 und 3 zu beweisen, wird jeweils ein Lemma definiert. Die Sätze werden dann mittels vollständiger Induktion über die Zustände des Systems und die Lemmata bewiesen. Zuerst werden drei Lemmata definiert, die für die Korrektheitsbeweise der Sätze 1 und 2 verwendet werden.

Lemma 1 sagt aus, dass nach einer Transition mit Regel 1 die Anzeigefläche o, die in der Berechtigung cp definiert ist, von der Menge der genutzten Anzeigeflächen der Anwendung s in die Menge der Anwendung s' übergeht. Dies gilt, wenn die Kontextmenge ctx_{cp} Teilmenge der Kontextmenge ctx des Zustands u ist.

Lemma 1: Seien $u_0 \in U$, $\Psi(u_0)$ ein System und $(u, r, u') \in U \times R \times U$ mit $(u, r, u') \gg \Psi(u_0)$. Sei Zustand $u = (cons, ctx, d)$, der die KFE erfüllt, $u' = (cons', ctx, d)$, $r = (ra, s, s', cp)$ mit $ra = append$ und die Bedingung $cond_1$ erfüllt. Dann gilt $trans(u, r) = (add_{cp}(cons, s, s', cp), ctx, d)$ mit $cp = (ctx_{cp}, s', o)$ und $ctx_{cp} \subseteq ctx$:

$$\Phi_{used}(cons', ctx, s) = \Phi_{used}(cons, ctx, s) \backslash o \wedge \tag{L1.1}$$

$$\Phi_{used}(cons', ctx, s') = \Phi_{used}(cons, ctx, s') \cup o \tag{L1.2}$$

Laut Lemma 1 befindet sich nach der Vergabe einer Berechtigung die betreffende Anzeigenfläche o in der Menge der genutzten Anzeigenflächen $\Phi_{used}(cons', ctx, s')$ der Anwendung s', wenn die Kontextmenge ctx_{cp} eine Teilmenge von ctx in Zustand u ist (L1.2). Die Anzeigenfläche o befindet sich dann nicht mehr in der Menge der genutzten Anzeigenflächen $\Phi_{used}(cons', ctx, s)$ der Anwendung s (L1.1).

Beweis von Lemma 1:

Es wird zuerst (L1.1) gezeigt. Nach $cond_1$ mit $ctx_{cp} \subseteq ctx$ und $cp \sqsubseteq \tilde{cp} = (\tilde{ctx}, \tilde{s}, \tilde{o})$ gilt $\tilde{ctx} \subseteq ctx_{cp} \subseteq ctx$ und $\tilde{o} \in received_o(cons, ctx, s)$ mit $o \subseteq \tilde{o}$. Mit $cp \sqcap cp' = \emptyset$ gilt für alle vergebenen Berechtigungen von s, dass $o \cap granted_o(cons, ctx, s) = \emptyset$ ist. Das heißt, $o \subseteq used_o(cons, ctx, s)$. Sei nun $\hat{o} \in used_o(cons, ctx, s)$ eine Anzeigenfläche mit $o \subseteq \hat{o}$. Zu zeigen sind die beide folgenden Aussagen:

$$o \not\subseteq \Phi_{used}(cons', ctx, s) \tag{I}$$

$$(\hat{o} \backslash o) \subseteq \Phi_{used}(cons', ctx, s) \tag{II}$$

Da für $(\hat{o} \cap o) = o$ gilt, muss für $o \not\subseteq \Phi_{used}(cons', ctx, s)$ nur gezeigt werden, dass $\hat{o} \not\subseteq \Phi_{used}(cons', ctx, s)$ gilt. Es wird Aussage (I) und dann (II) gezeigt:

(I) : Wegen Aussage 12.1.1 gilt, dass sich die Berechtigung cp in der Menge der vergebenen Berechtigungen $granted_{cp}(cons', s)$ der Anwendung s befindet. Daraus folgt, dass in Zustand u' die Anzeigefläche $o \in granted_o(cons', ctx, s)$ ist, da zudem $ctx_{cp} \subseteq ctx$ gilt (vgl. Definition 7). Da $o \subseteq \hat{o}$ gilt, ist offensichtlich, dass daraus $\hat{o} \cap o = o \neq \emptyset$ folgt. Dies bedeutet jedoch, dass die Bedingung 14.2 nicht erfüllt ist und damit die Aussage (I) gilt. Es befin-

den sich keine Teilmengen der Anzeigefläche $\hat{o}$ in der Menge der genutzten Anzeigeflächen der Anwendung s in Zustand u'.

(II) : Es muss gezeigt werden, dass die Anzeigefläche $(\hat{o}\backslash o)$ die Bedingungen der Definition 14 erfüllt. Hierfür sind die folgenden zwei Aussagen zu zeigen:

$$(\hat{o}\backslash o) \subseteq \Lambda_{received}(cons', ctx, s) \qquad \text{((a), Bedingung 14.1)}$$
$$(\hat{o}\backslash o) \cap \Gamma_{granted}(cons', ctx, s) = \emptyset \qquad \text{((b), Bedingung 14.2)}$$

(a): Aufgrund $\hat{o} \subseteq \Phi_{used}(cons, ctx, s)$ gilt, dass $\hat{o} \subseteq \Lambda_{received}(cons, ctx, s)$ ist. Nach der Transition $trans(u, r) = (add_{cp}(cons, s, s', cp), ctx, d)$ ist die Menge der erhaltenen Berechtigung in Zustand u und u' mit $received_{cp}(cons, ctx, s) = received_{cp}(cons', ctx, s)$ identisch. Das heißt, es gilt $\Lambda_{received}(cons, ctx, s) = \Lambda_{received}(cons', ctx, s)$. Dann gilt $(\hat{o}\backslash o) \subseteq \hat{o} \subseteq \Lambda_{received}(cons, ctx, s) = \Lambda_{received}(cons', ctx, s)$ und die Aussage (a).

(b): Da $\hat{o} \in used_o(cons, ctx, s)$ gilt, folgt mit der Definition 14.2, dass auch $\hat{o} \cap \Gamma_{granted}(cons, ctx, s) = \emptyset$ gilt. Nach Ausführung der Transition gilt dann $\Gamma_{granted}(cons', ctx, s) = \Gamma_{granted}(cons, ctx, s) \cup o$ aufgrund von $(s', cp) \in granted_{cp}(cons', ctx, s)$ und $ctx_{cp} \subseteq ctx$. Mit folgender Aussage für $\hat{o}\backslash o$ ist die Aussage (b) (d. h. Definition 14.2) ebenfalls erfüllt:

$$\begin{aligned}
&(\hat{o}\backslash o) \cap \Gamma_{granted}(cons', ctx, s) \\
&= (\hat{o}\backslash o) \cap (\Gamma_{granted}(cons, ctx, s) \cup o) && \text{(nach Def. 12, Regel 1)} \\
&= ((\hat{o}\backslash o) \cap \Gamma_{granted}(cons, ctx, s)) \cup ((\hat{o}\backslash o) \cap o) && \text{(Distributivgesetz)} \\
&= \emptyset \cup ((\hat{o}\backslash o) \cap o) && (\hat{o} \cap \Gamma_{granted}(cons, ctx, s) = \emptyset) \\
&= \emptyset && (\text{da } (B\backslash A) \cap A = \emptyset^{1})
\end{aligned}$$

Dies bedeutet, es gilt (b) $(\hat{o}\backslash o) \cap \Gamma_{granted}(cons', ctx, s) = \emptyset$.

Als nächstes wird (L1.2) gezeigt. Es ist zu zeigen, dass die Anzeigefläche o die Bedingungen der Definition 14 für $cons'$ erfüllt. Der Beweis ist ähnlich zu dem Beweis von L1.1. Dazu werden die folgenden beiden Aussagen gezeigt:

$$o \subseteq \Lambda_{received}(cons', ctx, s') \qquad \text{(I)}$$
$$o \cap \Gamma_{granted}(cons', ctx, s') = \emptyset \qquad \text{(II)}$$

(I): Da nach der Transition $trans(u, r) = (add_{cp}(cons, s, s', cp), ctx, d)$ für die Menge der erhaltenen Berechtigungen $received_{cp}(cons', ctx, s') =$

[1] Beweis für $(B\backslash A) \cap A = \emptyset$ siehe Anhang A.2

$received_{cp}(cons, ctx, s') \cup \{(s', cp)\}$ gilt und mit $ctx_{cp} = ctx$ dann auch $received_o(cons', ctx, s') = received_o(cons, ctx, s') \cup \{o\}$ gilt, kann direkt (I) $o \subseteq \Lambda_{received}(cons', ctx, s')$ gefolgert werden.

(II): In Zustand u hat die Anwendung s keine Berechtigung cp' mit $cp' \sqcap cp \neq \emptyset$ an eine andere Anwendung vergeben (siehe $cond_1$), womit $\Gamma_{granted}(cons, ctx, s) \cap o = \emptyset$ gilt. Ebenso gilt vor der Transition die Aussage $\Gamma_{granted}(cons, ctx, s') \cap o = \emptyset$. Sei $\Gamma_{granted}(cons, ctx, s') \cap o \neq \emptyset$. Dann gilt $o \subseteq \Lambda_{received}(cons, ctx, s')$, womit $\Gamma_{granted}(cons, ctx, s) \cap o \neq \emptyset$ folgt. Dies steht in Widerspruch zu $\Gamma_{granted}(cons, ctx, s) \cap o = \emptyset$. Damit gilt $\Gamma_{granted}(cons, ctx, s') \cap o = \emptyset$. Nach Definition der Funktion add_{cp} (vgl. 11.1.2) in Regel 1 gilt $granted_{cp}(cons', s') = granted_{cp}(cons, s')$, da sich die Menge der vergebenen Berechtigungen nicht geändert hat. Es folgt $\Gamma_{granted}(cons', ctx, s') \cap o = \Gamma_{granted}(cons, ctx, s') \cap o = \emptyset$. Alle Mengen der erhaltenen Berechtigungen der anderen Anwendungen sind unverändert (vgl. 11.1.3). Folglich ist die Aussage L1.2 erfüllt.

Lemma 2 sagt aus, dass nach einer Transition mit Regel 2 die Anzeigeflächen o' (Teilmengen von Anzeigefläche o) sich wieder in die Mengen der genutzten Anzeigeflächen der Anwendungen verschieben, welche den Zugriff vergeben hatten.

Lemma 2: Sei $u_0 \in U$, $\Psi(u_0)$ ein System und $(u, r, r') \in U \times R \times U$ mit $(u, r, u') \gg \Psi(u_0)$. Sei der Zustand $u = (cons, ctx, d)$, der die KFE erfüllt, $u' = (cons', ctx', d')$, $r = (ra, s, s', cp)$ mit $ra = discard$ und die Bedingung $cond_2$ erfüllt. Es gilt: $trans(u, r) = (del_{cp}(cons, s, s', cp), ctx, d)$ mit $cp = (ctx_{cp}, s', o)$, $ctx_{cp} \subseteq ctx$ und

$$[\forall \hat{s} \in S \backslash \{s\} : used_o(cons', ctx, \hat{s}) = used_o(cons, ctx, \hat{s}) \backslash \{o' \in O | o' \subseteq o \wedge \hat{s} <_{o', ctx} s\}] \tag{L2.1}$$

$$\Phi_{used}(cons', ctx, s) = \Phi_{used}(cons, ctx, s) \cup o \tag{L2.2}$$

Beweis von Lemma 2:

Zuerst wird L2.1 bewiesen. Sei $s \neq s'$ und die KFE in Zustand u erfüllt. Mittels der Funktion $del_{cp}(cons, s, s', cp)$ wird die Berechtigung cp und alle davon abhängenden Berechtigungen (d. h., $cp' \sqsubseteq cp$) entzogen. Daraus folgt, dass für cp mit $ctx_{cp} \subseteq ctx$ und jede davon abhängige Berechtigung cp' die Anzeigefläche o bzw. die davon abhängige Anzeigefläche $o_{cp'}$ der Berechtigung cp' aus der Menge der genutzten Anzeigeflächen der Anwendungen entfernt wird. Daraus folgt $\forall \hat{s} \in S \backslash \{s\} : \Lambda_{received}(cons', ctx, \hat{s}) = \Lambda_{received}(cons, ctx, \hat{s}) \backslash \bigcup \{o' \in O | o' \subseteq$

$o \wedge \hat{s} <_{o',ctx} s\}$. Das heißt, dass die Definition 14.2 nicht erfüllt ist und damit alle o' nicht mehr in den Mengen der genutzten Anzeigeflächen der entsprechenden Anwendungen sind. Das bedeutet, es gilt $\forall \hat{s} \in S \backslash \{s\} : used_o(cons', ctx, \hat{s}) = used_o(cons, ctx, \hat{s}) \backslash \{o' \in O | o' \subseteq o \wedge \hat{s} <_{o',ctx} s\}$. Für alle o' mit $\hat{s} <_{o',ctx} s$ gilt, dass $o' \subseteq o$ ist. Aufgrund KFE sind die vergebenen kontext-beschränkten Berechtigungen, die o' enthalten konfliktfrei. Es gilt, dass nach Vergabe (siehe Lemma 1) die vergebenen o' entweder in der Menge *used* der vergebenden oder der erhaltenden Anwendung sind. Das heißt, die Vereinigung der Anzeigenflächen o' führt zu $\bigcup\{o' \in O | o' \subseteq o \wedge \hat{s} <_{o',ctx} s\} = o$. Es gilt die Aussage L2.1.
Als nächstes wird L2.2 bewiesen. Aufgrund der Regel 2 mit Bedingung $cond_2$ gilt $(s', cp) \in granted_{cp}(cons, ctx, s)$. Es gilt dann auch $o \subseteq \Lambda_{received}(cons, ctx, s)$ wegen $ctx_{cp} \subseteq ctx$, womit die Bedingung 14.1 erfüllt ist. Aufgrund der Funktion del_{cp} (vgl. 11.2.6) in Regel 2 gilt, dass für die Mengen $granted_o(cons', ctx, s) = granted_o(cons, ctx, s) \backslash \{o\}$ und $\Gamma_{granted}(cons', ctx, s) = \Gamma_{granted}(cons, ctx, s) \backslash o$ gilt. Als nächstes wird gezeigt, dass die Aussage $o \cap \Gamma_{granted}(cons', ctx, s) = \emptyset$ gilt, womit die Bedingung 14.2 erfüllt ist:

$$
\begin{aligned}
o \cap \Gamma_{granted}(cons', ctx, s) &= o \cap (\Gamma_{granted}(cons, ctx, s) \backslash o) \\
&= \emptyset \qquad \text{(da } A \cap (B \backslash A) = \emptyset\text{)}
\end{aligned}
$$

Somit ist $o \cap \Gamma_{granted}(cons', ctx, s) = \emptyset$ und die Definition 14 erfüllt. Es gilt die Aussage L2.2.

Laut Lemma 1 und 2 werden die Mengen der genutzten Anzeigeflächen durch die Transitionen für das Vergeben bzw. das Entziehen einer kontext-beschränkten Berechtigung nicht verändert. Wird eine Anzeigefläche an eine Anwendung vergeben, dann befindet sich die Anzeigefläche in deren Menge der genutzten Anzeigeflächen. Beim Entziehen einer Berechtigung befindet sie sich wieder in der Menge der genutzten Anzeigeflächen der vergebenden Anwendung.

Das folgende Lemma 3 ist ähnlich zu Lemma 2. Der Unterschied ist, dass im Falle einer Kontextänderung die genutzten Anzeigeflächen von vergebenen Berechtigungen der Anwendung s sich entweder in der Menge $used_0$ der Anwendung s oder der von s' befinden (vgl. Regel 1 und Bedingung $cond_1$). Eine Änderung der Nutzung einer Anzeigefläche hängt davon ab, ob die Kontextänderung zu einer Berechtigungsänderung geführt hat.

Lemma 3: Sei $u_0 \in U$, $\Psi(u_0)$ ein System und $(u, r, r') \in U \times R \times U$ mit $(u, r, u') \gg \Psi(u_0)$. Sei der Zustand $u = (cons, ctx, d)$, der die KFE erfüllt, $u' =$

$(cons', ctx', d')$, $r = (s, c, a)$ und Bedingung $cond_3$ erfüllt. Dann gilt $trans(u, r) = (cons, set_{ctx}(ctx, c, a), d)$, $\hat{o} = \bigcup\{o_{cp} \in O | cp = (ctx_{cp}, s'', o_{cp}) \wedge s'' <_{cp} s \wedge ctx_{cp} \not\subseteq ctx'\}$, $\tilde{o} = \bigcup\{o_{cp} \in O | cp = (ctx_{cp}, s'', o_{cp}) \wedge s'' <_{cp} s \wedge ctx_{cp} \subseteq ctx'\}$ und

$$[\forall \hat{s} \in S \backslash \{s\} : used_o(cond, ctx', \hat{s}) = (used_o(cond, ctx, \hat{s}) \backslash \{o' \in O | o' \subseteq \tilde{o} \wedge \hat{s} <_{o',ctx} s\}) \cup \{o'' \in O | o'' \subseteq \hat{o} \wedge \hat{s} <_{o'',ctx'} s\}] \tag{L3.1}$$

$$\Phi_{used}(cons, ctx', s) = (\Phi_{used}(cons, ctx, s) \cup \tilde{o}) \backslash \hat{o} \tag{L3.2}$$

Beweis von Lemma 3: Die Funktion $ctx' = set_{ctx}(ctx, c, a)$ ändert den Zustand eines Kontexts, was dazu führen kann, dass sich durch kontext-beschränkte Berechtigungen, die den Kontext c in ihrer Kontextmenge haben, die Zugriffe auf die zugehörigen Anzeigeflächen ändern, d. h. die $used_o$ Mengen sich ändern.

Es wird zuerst L3.1 gezeigt. Gilt für die Berechtigung $cp = (ctx_{cp}, s_{cp}, o_{cp})$ in Zustand u für die Kontexte $ctx_{cp} \subseteq ctx$ und in Zustand u' für die Kontexte $ctx_{cp} \not\subseteq ctx$, dann wird der Zugriff auf o_{cp} entzogen. Somit gilt $o_{cp} \in received_o(cons, ctx, s_{cp})$ und $o_{cp} \notin received_o(cons, ctx', s_{cp})$. Da dies alle Anzeigeflächen o_{cp} betrifft, die von der Anwendung s an andere Anwendungen vergeben wurden, entspricht dies der Menge $\tilde{o}$. Da aufgrund der KFE gilt, dass $\tilde{o} \cap \hat{o} = \emptyset$ und folglich $\{o' \in O | o' \subseteq \tilde{o} \wedge \hat{s} <_{o',ctx} s\} \cap \{o'' \in O | o'' \subseteq \hat{o} \wedge \hat{s} <_{o'',ctx'} s\} = \emptyset$ gilt, ist die Definition 14.2 nicht erfüllt und die Anzeigeflächen o' nicht in der Menge der genutzten Anzeigeflächen der Anwendungen. Also gilt $\forall \hat{s} \in S \backslash \{s\} : \Lambda_{received}(cons, ctx', \hat{s}) \cap \bigcup\{o' \in O | o' \subseteq \tilde{o} \wedge \hat{s} <_{o',ctx} s\} = \emptyset$. Da bei $\tilde{o}$ alle Berechtigungen für o_{cp} betrachtet werden, für welche $\hat{s} <_{cp} s$ gilt, sind die Anzeigeflächen o'' der Menge $\{o' \in O | o' \subseteq \tilde{o} \wedge \hat{s} <_{o',ctx} s\}$ identisch zu den Anzeigeflächen o_{cp}. Dies gilt, da für alle o' mit $\hat{s} <_{o',ctx} s$ und $o' \subseteq o$ ist. Aufgrund KFE sind die vergebenen kontext-beschränkten Berechtigungen, die o' enthalten konfliktfrei. Es gilt, dass nach Vergabe (siehe Lemma 1) die vergebenen o' entweder in der Menge *used* der vergebenden oder der erhaltenden Anwendung sind. Es folgt $\tilde{o} = \bigcup\{o' \in O | o' \subseteq \tilde{o} \wedge \hat{s} <_{o',ctx} s\}$. Somit gilt Teil eins von L3.1.

Nun muss noch die Vereinigung mit der Menge $\{o'' \in O | o'' \subseteq \hat{o} \wedge \hat{s} <_{o'',ctx'} s\}$ in L3.1 gezeigt werden. Gilt für die Berechtigung $cp = (ctx_{cp}, s_{cp}, o_{cp})$ in Zustand u für die Kontexte $ctx_{cp} \not\subseteq ctx$ und in Zustand u' für die Kontexte $ctx_{cp} \subseteq ctx$, dann wird der Zugriff auf o_{cp} gewährt. Somit gilt $o_{cp} \notin received_o(cons, ctx, s_{cp})$ und $o_{cp} \in received_o(cons, ctx', s_{cp})$. Dies betrifft alle Anzeigeflächen, die von der Anwendung s eine Berechtigung erhalten haben. Die Anzeigeflächen entsprechen der Menge $\hat{o} := \cup\{o_{cp} \in O | cp = (ctx_{cp}, s'', o_{cp}) \wedge s'' <_{cp} s \wedge ctx_{cp} \subseteq ctx'\}$. Da bei $\hat{o}$ alle Berechtigungen für o_{cp} betrachtet werden, für welche $s'' <_{cp} s$ gilt,

sind die Anzeigeflächen o' der Menge $\{o'' \in O | o'' \subseteq \hat{o} \wedge \hat{s} <_{o'',ctx'} s\}$ identisch zu den Anzeigeflächen o_{cp}. Dies gilt, da für alle o'' mit $\hat{s} <_{o'',ctx'} s$ und $o'' \subseteq o$ gilt. Aufgrund KFE sind die vergebenen kontext-beschränkten Berechtigungen, die o'' enthalten konfliktfrei. Es gilt, dass nach Vergabe (siehe Lemma 1) die vergebenen o'' entweder in der Menge *used* der vergebenden oder der erhaltenden Anwendung sind. Somit folgt $\hat{o} = \bigcup\{o'' \in O | o'' \subseteq \hat{o} \wedge \hat{s} <_{o'',ctx'} s\}$. Mit $ctx_{cp} \subseteq ctx$ gilt $\hat{o} \subseteq \Lambda_{received}(cons, ctx, \hat{s})$ und $\forall \hat{s} \in S \backslash \{s\} : \Lambda_{received}(cons, ctx', \hat{s}) = (\Lambda_{received}(cons, ctx, \hat{s}) \backslash \bigcup\{o' \in O | o' \subseteq \tilde{o} \wedge \hat{s} <_{o',ctx} s\}) \cup \{o' \in O | o' \subseteq \hat{o} \wedge \hat{s} <_{o',ctx'} s\}$, womit die Bedingung 14.1 erfüllt bleibt. Für alle $o'' \subseteq \hat{o}$ mit $\hat{s} <_{o'',ctx'} s$ folgt $\forall \hat{s} \in S \backslash \{s\} : granted_{o''}(cons, ctx', s) = granted_{o''}(cons, ctx, s) \backslash o''$. Womit $\forall \hat{s} \in S \backslash \{s\} : \Gamma_{granted}(cons, ctx', \hat{s}) = \Gamma_{granted}(cons, ctx, \hat{s}) \backslash o''$ folgt. Ähnlich Lemma 2, wird gezeigt, dass $\forall \hat{s} \in S \backslash \{s\} : o'' \cap \Gamma_{granted}(cons, ctx', \hat{s}) = \emptyset$ gilt und Bedingung 14.2 erfüllt ist. $\forall \hat{s} \in S \backslash \{s\} :$

$$
\begin{aligned}
o'' \cap \Gamma_{granted}(cons, ctx', \hat{s}) &= o'' \cap (\Gamma_{granted}(cons, ctx, \hat{s}) \backslash o'') \\
&= \emptyset \qquad (\text{da } A \cap (B \backslash A) = \emptyset)
\end{aligned}
$$

Es gilt $\forall \hat{s} \in S \backslash \{s\} : o'' \cap \Gamma_{granted}(cons, ctx', \hat{s}) = \emptyset$. Damit ist dann L3.1 erfüllt.

Es wird nun L3.2 gezeigt. Die Anwendung s kann kontext-beschränkte Berechtigungen an andere Anwendungen vergeben haben, die von Kontext c abhängen. Das heißt, eine Berechtigung $cp = (ctx_{cp}, s_{cp}, o_{cp})$, für die in Zustand u mit $ctx_{cp} \subseteq ctx$ der Zugriff auf o_{cp} gewährt wird, wird bei $ctx_{cp} \not\subseteq ctx'$ entzogen. Damit erhält s den Zugriff auf o_{cp} zurück. Dies entspricht der Menge $\tilde{o}$. Durch das Schalten des Kontexts c wird jede Berechtigung $cp' = (ctx_{cp'}, s_{cp'}, o_{cp'})$, für die in Zustand u mit $ctx_{cp'} \not\subseteq ctx$ die Anwendung $s_{cp'}$ keinen Zugriff auf $o_{cp'}$ hatte, mit $ctx_{cp'} \subseteq ctx'$ in u' der Zugriff gewährt. Diese Anzeigeflächen sind in der Menge $\hat{o}$. Zu zeigen ist, dass für s die Definition 14 für die Menge $\tilde{o}$ erfüllt ist, aber für $\hat{o}$ nicht. Wegen der KFE gilt $\tilde{o} \cap \hat{o} = \emptyset$ und alle $o_{cp} \in \hat{o}$ mit $o_{cp} \notin recevived_o(cons, ctx', s)$ erfüllen die Definition 14.1 nicht, da $\Lambda_{received}(cons, ctx', s) = ((\Lambda_{received}(cons, ctx, s) \cup \tilde{o}) \backslash \hat{o})$ gilt. Für alle $o_{cp'} \in \tilde{o}$ mit $o_{cp'} \in recevived_o(cons, ctx', s)$ gilt, dass $\tilde{o} \subseteq \Lambda_{received}(cons, ctx, s)$ wegen $ctx_{cp'} \subseteq ctx$ folgt und Definition 14.1 erfüllt ist. Es gilt für alle $o_{cp'} \subseteq \tilde{o}$, dass $granted_o(cons, ctx', s) = granted_o(cons, ctx, s) \backslash o_{cp'}$ gilt. Damit gilt $\Gamma_{granted}(cons, ctx', s) = \Gamma_{granted}(cons, ctx, s) \backslash \tilde{o}$. Ähnlich Lemma 3.1 wird gezeigt, dass $\tilde{o} \cap \Gamma_{granted}(cons, ctx', s) = \emptyset$ gilt:

$$
\begin{aligned}
\tilde{o} \cap \Gamma_{granted}(cons, ctx', \hat{s}) &= \tilde{o} \cap (\Gamma_{granted}(cons, ctx, \hat{s}) \backslash \tilde{o}) \\
&= \emptyset \qquad (\text{da } A \cap (B \backslash A) = \emptyset)
\end{aligned}
$$

Folglich ist Definition 14.2 erfüllt und $\tilde{o}$ befindet sich in der Menge der genutzten Anzeigeflächen der Anwendung s, womit Aussage L3.2 bewiesen ist.

Das folgende Lemma KFE sagt, dass eine Transition von einem Zustand, der die KFE erfüllt, immer in einem Zustand endet, der ebenfalls die KFE erfüllt.

Lemma KFE: Alle Sequenzen in dem System $\Psi(u_0)$ erfüllen die KFE für alle initialen Zustände u_0, welche die KFE erfüllen $\Leftrightarrow \forall(u, r, u') \in U \times R \times U$: $(u, r, u') \gg \Psi(u_0) \Rightarrow u, u' \in U$ erfüllt die KFE.

Beweis von Lemma KFE: In Bezug auf den Abschnitt 3.4 ist eine Anfrage $r \in R$ entweder in RP, in RC oder in RD. Die Fälle $r \in RD$ und $r \in RC$ sind trivial, da die Regeln 4 und 5 (vgl. Abschnitt 3.4) nicht die Menge der kontext-beschränkten Berechtigungen in *cons* ändern und deshalb nicht für das Lemma KFE relevant sind. Im Falle von $r = (ra, s, s', cp) \in RP$ müssen die folgenden drei Unterfälle betrachtet werden:

(zu Regel 1): Sei $r \in RP$, $ra = append$ und die Bedingung $cond_1$ erfüllt. Es folgt, dass die Transition $trans(u, r) = (add_{cp}(cons, s, s', cp), ctx, d)$ zur Menge der erhaltenen kontext-beschränkten Berechtigungen $received_{cp}(cons', s')$ $= received_{cp}(cons, s') \cup \{(s, cp)\}$ führt. Zuerst wird im Folgenden gezeigt, dass die kontext-beschränkte Berechtigung cp konfliktfrei zu allen bereits vergebenen kontext-beschränkten Berechtigungen von Anwendung s ist. Danach wird gezeigt, dass cp konfliktfrei zu allen anderen kontext-beschränkten Berechtigungen ist, die nicht von der Anwendung s abhängen, womit gefolgert werden kann, dass die KFE im Zustand u' erfüllt sind.

Da in Zustand u die KFE gilt und die Bedingung $cond_1$ erfüllt ist, folgt, dass $\forall(\tilde{s}, \tilde{cp}) \in granted_{cp}(cons, s) : cp \sqcap \tilde{cp} = \emptyset$ mit der kontext-beschränkten Berechtigung cp durch Anwendung s vergeben wird. Wegen Bedingung $cond_1$ gilt $(s, cp) \notin received_{cp}(cons, s')$. Somit ist cp in Zustand u' konfliktfrei zu den bereits vergebenen kontext-beschränkten Berechtigungen von s.

Mit Bedingung $cond_1$ folgt, dass $\exists \hat{s} \in S$ mit $(\hat{s}, \hat{cp}) \in received_{cp}(cons, s)$: $cp \sqsubseteq \hat{cp}$ gilt. Da Zustand u die KFE erfüllt, gilt, dass die kontext-beschränkte Berechtigung $\hat{cp}$ konfliktfrei zu allen kontext-beschränkten Berechtigungen der anderen Anwendungen in Zustand u ist. Aus $cp \sqsubseteq \hat{cp}$ und $(s, cp) \in received_{cp}(cons', s')$ folgt, dass cp konfliktfrei zu allen kontext-beschränkten Berechtigungen ist, die nicht von s in Zustand u' abhängen.

Es folgt aus $\forall cp'' \in CP, \forall s, s_1, s_2 \in S''$: $(s_1, cp'') \in received_{cp}(cons', s'')$ und $(s_2, cp) \in received_{cp}(cons', s')$ und $s' \neq_{cp} s''$ und $s' \neq_{cp''} s$, dass $cp \sqcap cp'' = \emptyset$.

(zu Regel 2): Sei $r \in RP$, $ra = discard$ und die Bedingung $cond_2$ erfüllt. Dann gilt die Transition mit $trans(u, r) = (del_{cp}(cons, s, s', cp), ctx, d)$. Da in Zustand u die KFE erfüllt ist, folgt $\forall(\tilde{s}, \tilde{cp}) \in granted_{cp}(cons, s) : cp \sqcap \tilde{cp} = \emptyset$ mit der kontext-beschränkten Berechtigung cp vergeben durch die Anwendung s. Des Weiteren gilt, dass cp in Zustand u von der Anwendung s' (d. h., $(s, cp) \in received_{cp}(cons, s')$) erhalten wurde und konfliktfrei zu allen anderen kontext-beschränkten Berechtigungen ist. Das bedeutet:
$\forall s, s', s_1, s_2 \in S, \forall cp, cp' \in CP : cp \neq cp' \wedge (s_1, cp) \in received_{cp}(cons, s) \wedge$
$(s_2, cp') \in received_{cp}(cons, s') \wedge s' \neq_{cp} s \wedge s' \neq_{cp'} s \Rightarrow cp \sqcap cp' = \emptyset$.
Weiter gilt, dass alle kontext-beschränkten Berechtigungen in der Menge der erhaltenen kontext-beschränkten Berechtigungen im Zustand u konfliktfrei sind und somit cp ebenfalls konfliktfrei ist. Nach der Transition gilt, dass die Funktion $del_{cp}(cons, s, s', cp)$ durchgeführt wurde und folglich $(s, cp) \notin received_{cp}(cons, s')$ gilt. Außerdem sind alle kontext-beschränkten Berechtigungen entfernt worden, die von cp abhängen. Es gilt $\forall s_1 \in S : s_1 <_{cp} s \Rightarrow received_{cp}(cons', s_1) = received_{cp}(cons, s_1) \setminus$
$\{(s_2, cp') \in S \times CP | s_2 <_{cp'} s_1 \wedge cp' \sqsubseteq cp\}$. Daraus folgt in Zustand u', dass cp und alle davon abhängigen kontext-beschränkten Berechtigungen aus der Menge der erhaltenen und vergebenen kontext-beschränkten Berechtigungen aller Anwendungen entfernt wurden, wobei alle anderen kontext-beschränkten Berechtigungen nicht verändert wurden. Das heißt, $\forall s_3 \in S \setminus \{s', s\}; \forall cp' \in CP : cp' \sqsubseteq cp \wedge s \neq_{cp'} s_3 \vee s <_{cp'} s_3 \Rightarrow cons'(s_3) = cons(s_3)$.
Somit gilt, dass alle in Zustand u' verbleibenden kontext-beschränkten Berechtigungen konfliktfrei sind und der Zustand u' die KFE erfüllt.

(sonst): Es gilt $u' = u$. Da der Zustand u die KFE erfüllt, folgt, dass der u' ebenfalls die KFE erfüllt.

Somit ist das Lemma KFE bewiesen.

Das folgende Lemma VE sagt aus, dass eine Transition von einem Zustand, der die VE erfüllt, immer in einem Zustand endet, welcher ebenfalls die VE erfüllt.

Lemma VE: Alle Sequenzen in dem System $\Psi(u_0)$ erfüllen die VE und KFE für alle initialen Zustände u_0, welche die VE erfüllen $\Leftrightarrow$
$\forall(u, r, u') \in U \times R \times U : (u, r, u') \gg \Psi(u_0) \Rightarrow u, u' \in U$ erfüllt die VE.

Beweis von Lemma VE: Sei $\Omega_{used}(cons, ctx) = AO$. Es muss gezeigt werden, dass $\Omega_{used}(cons', ctx') = AO$ gilt. Die Anfrage r ist entweder in RP, RC oder RD (vgl. Abschnitt 3.4). Der Fall $r \in RD$ ist trivial, da $cons' = cons$ und $ctx' = ctx$ gilt (vgl. Abschnitt 3.4 Regel 4 und 5) und somit keine Verän-

derung der Zugriffsberechtigungen erfolgt, welche auf VE eine Auswirkung haben könnte. Dies heißt, der Beweis muss für den Fall $r \in RP$ und $r \in RC$ mit jeweils Regel 1, 2 und 3 geführt werden. Mittels Aussage (A1): $\{\breve{o} \in O \mid \exists s \in S : \breve{o} \in used_o(cons, ctx, s)\} \stackrel{!}{=} \{\breve{o} \in O \mid \exists s \in S : \breve{o} \in used_o(cons', ctx, s)\}$ wird auf das Folgende geschlossen:

$$\begin{aligned} AO &= \Omega_{used}(cons, ctx) \\ &= \bigcup \{\breve{o} \in O \mid \exists s \in S : \breve{o} \in used_o(cons, ctx, s)\} \\ &= \bigcup \{\breve{o} \in O \mid \exists s \in S : \breve{o} \in used_o(cons', ctx, s)\} \\ &= \Omega_{used}(cons', ctx) \end{aligned}$$

Dies bedeutet, dass nur die Aussage (A1) bewiesen werden muss. Hierfür muss für die vier Fälle gezeigt werden, dass die Vereinigung der Mengen der genutzten Anzeigenflächen sich nicht verändert, wenn eine Transition durchgeführt wird.

(Regel 1): Sei $r = (ra, s, s', cp) \in RP$ mit $ra = append$ und sei $cond_1$ erfüllt. Hierfür gilt $trans(u, r) = (cons', ctx, d)$ mit $cons' = add_{cp}(cons, s, s', cp)$ und $u = (cons, ctx, d)$ und $cp = (ctx_{cp}, s_{cp}, o_{cp})$. Demzufolge gilt: $\forall \hat{s} \in S \backslash \{s, s'\} : \Phi_{used}(cons', ctx, \hat{s}) = \Phi_{used}(cons, ctx, \hat{s})$. Die Menge der genutzten Anzeigeflächen ist trivialerweise gleich. Es ist zu zeigen: $\Phi_{used}(cons', ctx, s) \cup \Phi_{used}(cons', ctx, s') = \Phi_{used}(cons, ctx, s) \cup \Phi_{used}(cons, ctx, s')$. Dies kann wie folgt bewiesen werden:

$$\begin{aligned} &\Phi_{used}(cons', ctx, s) \cup \Phi_{used}(cons', ctx, s') \\ &= (\Phi_{used}(cons, ctx, s) \backslash o) \cup \Phi_{used}(cons', ctx, s') && \text{(L1.1)} \\ &= (\Phi_{used}(cons, ctx, s) \backslash o) \cup (\Phi_{used}(cons, ctx, s') \cup o) && \text{(L1.2)} \\ &= (\Phi_{used}(cons, ctx, s) \backslash o) \cup (o \cup \Phi_{used}(cons, ctx, s')) && \text{(Kommutativgesetz)} \\ &= ((\Phi_{used}(cons, ctx, s) \backslash o) \cup o) \cup \Phi_{used}(cons, ctx, s') && \text{(Assoziativgesetz)} \\ &= \Phi_{used}(cons, ctx, s) \cup \Phi_{used}(cons, ctx, s') && (\text{da } o \subseteq \Phi_{used}(cons, ctx, s))^2 \end{aligned}$$

Womit die Vereinigung der Mengen der genutzten Anzeigeflächen in Zustand u und u' gleich ist.

(Regel 2): Sei $ra = discard$ und $cond_2$ erfüllt. Es werden die relevanten Anwendungen betrachtet, das heißt, die Anwendungen in der folgenden Menge: $\hat{S} := \{\hat{s} \in S \mid \exists o' \in O : \hat{s} <_{o,ctx} s \wedge o' \subseteq o\}$. Demzufolge gilt $\forall \tilde{s} \in S \backslash \hat{S} : \Phi_{used}(cons', ctx, \tilde{s}) = \Phi_{used}(cons, ctx, \tilde{s})$.

[2] Mit $A \subseteq B \Rightarrow (B \backslash A) \cup A = B$ (Beweis siehe Anhang A.2)

Folgendes ist dann zu zeigen:

$$\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s})\} \cup \Phi_{used}(cons, ctx, s) =$$
$$\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons', ctx, \hat{s})\} \cup \Phi_{used}(cons', ctx, s)$$

Sei $o'' := \bigcup\{o' \in O | \exists \hat{s} \in \hat{S} : o' \subseteq o \wedge s'' <_{o',ctx} s\}$. Nach Lemma 2 gilt: $o = o''$. Daraus wird geschlossen:

$$\begin{aligned}
&\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons', ctx, \hat{s})\} \cup \Phi_{used}(cons', ctx, s) \\
&= \bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s}) \backslash o''\} \cup \Phi_{used}(cons, ctx, s) && \text{(L2.1)} \\
&= \bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s}) \backslash o''\} \cup (\Phi_{used}(cons, ctx, s) \cup o) && \text{(L2.2)} \\
&= \bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s}) \backslash o''\} \cup (o \cup \Phi_{used}(cons, ctx, s)) && (\text{K}^3) \\
&= (\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s}) \backslash o''\} \cup o) \cup \Phi_{used}(cons, ctx, s) && (\text{A}^4) \\
&= \bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s})\} \cup \Phi_{used}(cons, ctx, s) && (\text{da } o = o'')^5
\end{aligned}$$

(Regel 3): Sei $r = (s, c, a) \in RC$ und $cond_3$ erfüllt. Hierfür gilt $trans(u, r) = (cons, ctx', d)$ mit $ctx' = set_{ctx}(ctx, c, a)$ und $u = (cons, ctx, d)$.
Es gibt genau eine Anwendung, die einen Kontext eingeführt hat. Dies ist die Anwendung, die eine bzw. mehrere Berechtigungen wie folgt vergeben hat: $s \in S : (s', cp') \in granted_{cp}(cons, s) \wedge cp' = (ctx_{cp'}, s', o_{cp'}) \wedge (s, cp) \in received_{cp}(cons, s) \wedge cp = (ctx_{cp}, s, o_{cp}) \wedge cp' \sqsubseteq cp \wedge ((c, inv(a)) \in ctx_{cp} \vee (c, a) \in ctx_{cp})$. Des Weiteren müssen die Anwendungen betrachtet werden, welche eine kontext-beschränkte Berechtigung von der Anwendung s erhalten haben und von Kontext c abhängen. Dies sind die Anwendungen in der folgenden Menge: $\hat{S} := \{\hat{s} \in S | \exists cp \in CP : \hat{s} <_{cp} s \wedge cp = (ctx_{cp}, s, o_{cp}) \wedge ((c, a) \in ctx_{cp} \vee (c, inv(a)) \in ctx_{cp})\}$.
Dann muss ähnlich zu Regel 2 das Folgende gezeigt werden:

$$\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s})\} \cup \Phi_{used}(cons, ctx, s) =$$
$$\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons', ctx, \hat{s})\} \cup \Phi_{used}(cons', ctx, s)$$

Die Zustandsänderung des Kontexts c führt zu den folgenden zwei Fällen:

(Fall 1) Den Anwendungen in $\hat{S}$ wird durch Berechtigung $cp = (ctx_{cp}, s, o_{cp})$ der

[3] Kommutativgesetz
[4] Assoziativgesetz
[5] Mit $o'' \subseteq \bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s})\}$ und $A \subseteq B \Rightarrow (B \backslash A) \cup A = B$)

Zugriff auf die Anzeigefläche entzogen ($ctx_{cp} \subseteq ctx$ und $ctx_{cp} \not\subseteq ctx'$).

(Fall 2) Den Anwendungen in $\hat{S}$ wird durch Berechtigung $cp = (ctx_{cp}, s, o_{cp})$ der Zugriff auf die Anzeigefläche gewährt ($ctx_{cp} \not\subseteq ctx$ und $ctx_{cp} \subseteq ctx'$).

Es gelte $\tilde{o} := \bigcup\{o_{cp} \in O | cp = (ctx_{cp}, s'', o_{cp}) \wedge s'' <_{cp} s \wedge ctx_{cp} \not\subseteq ctx'\}$ und $\hat{o} := \bigcup\{o_{cp} \in O | cp = (ctx_{cp}, s'', o_{cp}) \wedge s'' <_{cp} s \wedge ctx_{cp} \not\subseteq ctx'\}$, sowie $o' := \bigcup\{o \in O | o \subseteq \hat{o} \wedge s'' <_{o,ctx'} s\}$ und $o'' := \bigcup\{o \in O | o \subseteq \tilde{o} \wedge s'' <_{o,ctx'} s\}$. Nach Lemma 3 gilt: $o' = \hat{o}$ und $o'' = \tilde{o}$. Daraus wird geschlossen:

$$
\begin{aligned}
&\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s})\} \cup \Phi_{used}(cons, ctx, s) \\
&= (\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s}) \backslash o''\} \cup o') \cup \Phi_{used}(cons, ctx', s) && \text{(L3.1)} \\
&= (\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s}) \backslash o''\} \cup o') \\
&\quad \cup ((\Phi_{used}(cons, ctx, s) \backslash \hat{o}) \cup \tilde{o}) && \text{(L3.2)} \\
&= (\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s}) \backslash o''\}) \\
&\quad \cup (o' \cup ((\Phi_{used}(cons, ctx, s) \backslash \hat{o}) \cup \tilde{o})) && \text{(A}^5\text{)} \\
&= (\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s}) \backslash o''\}) \\
&\quad \cup ((o' \cup (\Phi_{used}(cons, ctx, s) \backslash \hat{o})) \cup \tilde{o}) && \text{(A}^5\text{)} \\
&= (\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s}) \backslash o''\}) \\
&\quad \cup (\Phi_{used}(cons, ctx, s) \cup \tilde{o}) && (\text{da } o' = \hat{o})^6 \\
&= (\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s}) \backslash o''\}) \cup (\tilde{o} \cup \Phi_{used}(cons, ctx, s)) && \text{(K}^7\text{)} \\
&= ((\bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s}) \backslash o''\}) \cup \tilde{o}) \cup \Phi_{used}(cons, ctx, s) && \text{(A}^5\text{)} \\
&= \bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s})\} \cup \Phi_{used}(cons, ctx, s) && (\text{da } o'' = \tilde{o})^8
\end{aligned}
$$

(sonst): Da Zustand u die VE erfüllt und $u' = u$ gilt, erfüllt auch u' die VE. Somit ist das Lemma VE erfüllt.

Das folgende Lemma DE sagt aus, dass eine Transition von einem Zustand, der die DE erfüllt, immer in einem Zustand endet, welcher ebenfalls die DE erfüllt.

Lemma DE: Alle Sequenzen in dem System $\Psi(u_0)$ erfüllen die DE für alle initialen Zustände u_0, welche die DE erfüllen $\Leftrightarrow$
$\forall (u, r, u') \in U \times R \times U : (u, r, u') \gg \Psi(u_0) \Rightarrow u, u' \in U$ erfüllt die DE.

[5] Assoziativgesetz
[6] Mit $\hat{o} \subseteq \Phi_{used}(cons, ctx, s)$ und $A \subseteq B \Rightarrow (B \backslash A) \cup A = B$)
[7] Kommutativgesetz
[8] Mit $o'' \subseteq \bigcup\{\breve{o} \in O | \exists \hat{s} \in \hat{S} : \breve{o} \in used_o(cons, ctx, \hat{s})\}$ und $A \subseteq B \Rightarrow (B \backslash A) \cup A = B$)

Beweis von Lemma DE: Um das Lemma DE zu beweisen, wird ein Widerspruchsbeweis verwendet. Seien $s, s' \in S, o \in O$, $s \neq s'$. Angenommen, Zustand $u = (cons, ctx, d)$ erfüllt die DE, aber Zustand u' nicht.
Dazu werden die folgenden zwei Fälle betrachtet, welche die DE verletzt könnten.
(Fall 1): Kontext-beschränkte Berechtigungen wurden zwischen Anwendungen ausgetauscht, die sich in keiner Delegationsbeziehung befinden (Regel 1).
(Fall 2): Die Delegationsbeziehung zwischen zwei oder mehr Anwendungen wurde aufgelöst, aber es existieren noch vergebene kontext-beschränkte Berechtigungen zwischen diesen Anwendungen (Regel 5).
Folglich müssen die beiden Regeln (R1) und (R5) betrachtet werden.
Es wird zuerst der Fall 1 gezeigt: Sei o.B.d.A. eine kontext-beschränkte Berechtigung cp von der Anwendung s an die Anwendung s' vergeben worden. Da in Zustand $u' = (cons', ctx, d)$ die DE nicht erfüllt ist, muss Folgendes gelten:

$$\{o \in O | (s', cp) \in received_{cp}(cons, s) \wedge cp = (ctx_{cp}, (s, o)) \wedge$$
$$(s, cp') \in granted_{cp}(cons, s') \wedge cp' = (ctx_{cp'}, (s', o)) \wedge ctx_{cp'} \subseteq ctx_{cp}\} \neq \emptyset$$

und entweder gilt $(s, s') \notin d$ oder $(s', s) \notin d$. Aufgrund Regel 1 gilt, dass Transition $trans(u, r) = u'$ mit $u' = (add_{cp}(cons, s, s', cp), ctx, d)$ nicht durchgeführt wird, da die Bedingung $cond_1$ nicht erfüllt ist. Damit gilt für den Zustand $u = u'$. Das heißt, es gilt Folgendes: $\Gamma_{received}(cons', ctx, s) \cap \Lambda_{granted}(cons', ctx, s') = \Gamma_{received}(cons, ctx, s) \cap \Lambda_{granted}(cons, ctx, s') = \emptyset$. Dies ist jedoch im Widerspruch zur Annahme, da die Berechtigung cp dann nicht vergeben wurde.
Nun wird der Fall 2 gezeigt: Sei $i, j \in I^n$ und $s, s', s_0, s_1, ...s_j, ...s_i \in S$ mit $s <_{cp} s_1 <_{cp} ... <_{cp} s_j <_{cp} ... <_{cp} s_{i+1} <_{cp} s'$. Löse o.B.d.A. die Anwendung s_j die Delegationsbeziehung zu Anwendung s_{j+1} auf. Der Zustand u' erfüllt dann nicht mehr die DE und das Folgende ist gültig: $\Gamma_{received}(cons, ctx, s) \cap \Lambda_{granted}(cons, ctx, s') = o$ und $(s_j, s_{j+1}) \notin d'$. Aufgrund von Regel 5 gilt, dass die Transition $trans(u, r) = u'$ mit $d' = d \backslash \{s, s'\}$ (Regel 5) nicht angewandt werden kann, da die Bedingung $cond_5$ nicht erfüllt ist. Da jedoch der Zustand u die DE wegen $\Gamma_{received}(cons, ctx, s) \cap \Lambda_{granted}(cons, ctx, s') \neq \emptyset$ und $(s_j, s_{j+1}) \notin d' = d$ nicht erfüllt, ist dies im Widerspruch zur Annahme und es gilt Lemma DE.

Schließlich werden die Sätze 1,2 und 3 mittels vollständiger Induktion bewiesen.
Sei $(x_r, x_v) \in \Psi(u_{start})$ mit $x_r = (r_0, ..., r_{n-1}) \in R^{I^{n-1}}$, $x_u = (u_0, ..., u_n) \in U^{I^n}$.
Sei $u_0 = u_{start}$ der initiale Zustand, welcher die Zustände x_u durch die folgenden Transitionen erzeugt: $\forall i \in I^n \backslash \{0\} : u_i = trans(u_{i-1}, r_{i-1})$.

Sei $\forall i \in I^n \backslash \{0\} : (v_{i-1}, r_{i-1}, v_i) \gg \Psi(v_0)$, vgl. Definition 20.3.

Basis: Zustand u_0 erfüllt die Korrektheitseigenschaften. Mit $u_1 = trans(u_0, r_0)$ folgt, dass u_1 die Korrektheitseigenschaften aufgrund der Lemmata KFE, VE, und DE erfüllt.

Induktionshypothese: Der Zustand u_i erfüllt die Korrektheitseigenschaften.

Induktionsschritt: Erfülle u_i die Korrektheitseigenschaften. Durch die Lemmata folgt, dass $u_{i+1} = trans(u_i, r_i)$ die Korrektheitseigenschaften erfüllt. □

Aufgrund des Beweises kann gefolgert werden, dass alle Systeme $\Psi(u_0)$ sichere Systeme sind, wenn der Zustand u_0 die Korrektheitseigenschaften erfüllt. Dies bedeutet, dass die Regeln des Protokolls die Korrektheitseigenschaften nicht verletzen. Deshalb entspricht eine Konfiguration des Zugriffskontrollmodells, in welcher z. B. eine Anwendung die gesamte Anzeigenfläche mittels einer kontext-beschränkten Berechtigung zugewiesen bekommt und diese nicht von einem Kontext abhängt – dieser Zustand erfüllt die Korrektheitseigenschaften – einem sicheren System.

3.6. Implementierung

Um die Anwendbarkeit des Berechtigungskonzeptes in einer fahrzeugnahen Evaluation zu zeigen, wurde eine prototypische Implementierung erstellt.

Dazu wurde für die Implementierung eine virtualisierte Umgebung verwendet, welcher der Zielarchitektur aus Abbildung 2.2 entspricht und in Abschnitt 2.2 beschrieben ist. Zu diesem Zweck wurde die Virtualisierungslösung PikeOS von Sysgo verwendet und drei virtuelle Maschinen für den Virtualisierungsmanager, die Anwendungen des Kombiinstruments und die Anwendungen der Headunit verwendet, wie in Abbildung 3.9 dargestellt. Eine weitere virtuelle Maschine für die Kommunikation mit dem Fahrzeugnetzwerk, in diesem Falle CAN, wurde nicht verwendet, da der Aufwand für die Simulation des CAN-Fahrzeugbusses sehr hoch ist. Diese virtuelle Maschine wäre notwendig, um eine sichere und zuverlässige Kommunikation mit dem Fahrzeugbus zu gewährleisten und die Informationen zur Bestimmung der Kontexte des Fahrzeuges zu erhalten. Das relevante Fahrzeugverhalten wurde stattdessen mittels einer Anwendung simuliert, die mittels TCP/IP-Verbindung zu den einzelnen Anwendungen die entsprechenden Informationen (z. B. die Geschwindigkeit) zur Bestimmung der Kontexte sendet.

Der Virtualisierungsmanager hat den exklusiven Zugriff auf die Hardwarekomponenten GPU, Anzeigen und Eingabegeräte. Die Implementierung besteht aus den Softwarekomponenten, wie in Abbildung 3.9 dargestellt. Diese werden im Folgenden genauer beschrieben.

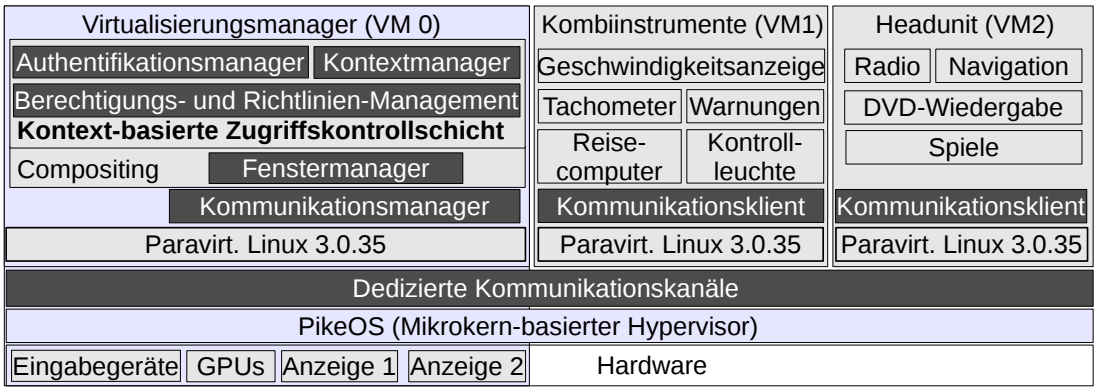

Abbildung 3.9.: Architektur für das Berechtigungssystem in einer virtualisierten Umgebung

Die Anwendungen kommunizieren mittels des dedizierten Kommunikationskanals mit dem Virtualisierungsmanager bzw. mit den einzelnen Komponenten. Bevor eine Anwendung ein Fenster beim Fenstermanager registrieren kann, benötigt sie eine kontext-beschränkte Berechtigung für einen Anzeigebereich vom Berechtigungs- und Richtlinien-Management. Die kontext-beschränkten Berechtigungen für Anzeigebereiche werden in Form von Extensible Markup Language (XML)-Dateien definiert und über die Testanwendungen an das Berechtigungs- und Richtlinien-Management übertragen. Die exakten Pixel der Anzeigebereiche, auf die zugegriffen werden darf, werden in Form einer Bitmaske in der XML-Datei angegeben. Eine Bitmaske stellt ein zweidimensionales Feld dar, in welchem jeder Eintrag einem Pixel entspricht, auf das zugegriffen werden darf (Eintrag ist 1) oder nicht (Eintrag ist 0). Das Berechtigungs- und Richtlinien-Management vergibt die Berechtigungen entsprechend der Kontexte, welcher der Kontextmanager dem Berechtigungs- und Richtlinien-Management übermittelt. Liegt einer Anwendung eine Berechtigung vor, kann sie ein Fenster entsprechend der Bitmaske der Berechtigung beim Fenstermanager registrieren und einen Embedded-System Graphics Library (EGL)-Kontext [EGL16] für die Erstellung von 3D-Inhalten mittels OpenGL ES 2.0 [Ope16a] erzeugen. Liegt ein gültiger EGL-Kontext vor, kann die Anwendung die OpenGL ES 2.0-Befehle zur Berechnung der 3D-Inhalte über einen dezidierten Kommunikationskanal an den Virtualisierungsmanager übertragen, welcher die Befehle an die 3D-GPU weiterleitet. Das Compositing wird mit der Bildwiederholrate der Anzeige synchronisiert, welche 60 Hz beträgt. Mit dieser Rate werden Inhalte der markierten Fenster entsprechend der Bitmaske in den Anzeigespeicher zur Darstellung auf den Anzeigen kopiert.

Die *dedizierten Kommunikationskanäle* bieten sitzungsbasierte first in, first out

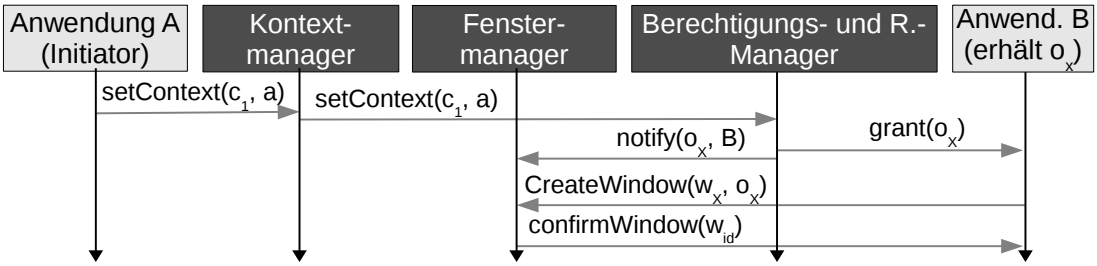

Abbildung 3.10.: Interaktion des API-Aufrufs $setContext(CTX\ c, CA\ a)$ und Fenstervergabe

(FIFO)-Kommunikation basierend auf gemeinsamen Speicher zwischen den Anwendungen in der Partition und dem Virtualisierungsmanager an. Jede Anwendung erhält mittels eines eigenen Kommunikationskanals über den Virtualisierungsmanager Zugriff auf die Hardwarekomponenten. Der *Kommunikationsmanager* speichert die Zertifikate der Anwendungen und leitet sie an den *Authentifizierungsmanager* weiter. Dieser erstellt dedizierte Kommunikationskanäle zwischen den Systemkomponenten und den authentifizierten Anwendungen. Der Authentifizierungsmanager gewährt den Anwendungen die Kommunikation mit den Systemkomponenten abhängig ihrer Identität. Die *kontextbasierte Zugriffskontrollschicht* trifft die Zugriffsentscheidungen abhängig der Kontexte. Der *Kontextmanager* liefert die notwendigen Informationen über die Kontexte und erlaubt es Anwendungen den Status der Kontexte mittels des API-Aufrufs $setContext(CTX\ c, CA\ a)$ mit Kontext c und Status a zu ändern (vgl. Regel 2 in Abschnitt 3.4).

Abbildung 3.10 zeigt die Nachrichten, die beim Schalten eines Kontexts zur Vergabe einer Berechtigung und der Registrierung eines Fensters führen. Abbildung 3.11 zeigt, wie beim Kontextwechsel der Zugriff einer Anwendung auf einen Anzeigebereich entzogen und das Fenster beim Fenstermanager gelöscht wird.

In beiden Abbildungen 3.10 und 3.11 setzt die Anwendung A den Kontext $c_1 = (A, 1)$ auf Status a durch Senden des Befehls $setContext(c_1, a)$ an den Kontextmanager. Der Kontextmanager ändert den Status des Kontexts und informiert das Berechtigungs- und Richtlinien-Management, welches die betroffe-

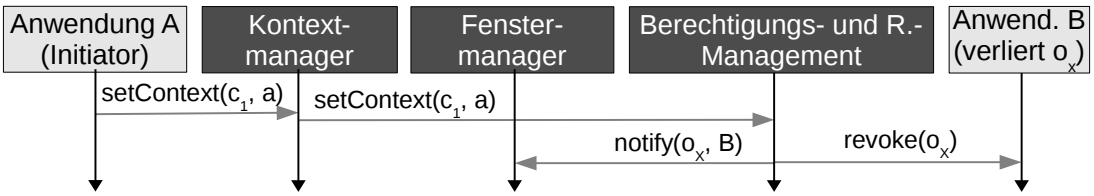

Abbildung 3.11.: Interaktion des API-Aufrufs $setContext(CTX\ c, CA\ a)$ und Entfernen eines Fensters

nen Berechtigungen ermittelt und den Fenstermanager mit $nofify(o_x, B)$ (o_x ist die neue Bitmaske) informiert. In Abbildung 3.10 erhält die Anwendung B mittels $grant(o_x)$ die neue Berechtigung und erzeugt ein Fenster für die neue Berechtigung mit $createWindow(w_x, o_x)$ (o_x ist die Bitmaske und w_x das Fenster). Der Fenstermanager bestätigt der Anwendung B das Erstellen des Fensters mit $confirmWindow(W_{id})$. In Abbildung 3.11 führt die Kontextänderung dazu, dass die Anwendung C die Berechtigung für den Anzeigebereich o_x verliert und mit $revoke(o_x)$ darüber informiert wird. Der Fenstermanager entfernt nach Erhalt von $nofify(o_x, B)$ das entsprechende Fenster, das dann nicht mehr sichtbar ist.

Das *Berechtigungs- und Richtlinien-Management* ist zuständig für die Verwaltung der kontext-beschränkten Berechtigungen und trifft die Zugriffsentscheidungen für den *Fenstermanager*. Der Fenstermanager ist verantwortlich für das Erstellen, Entfernen und Positionieren von Fenstern. Die Anwendungen können kontext-beschränkte Berechtigungen an andere Anwendungen vergeben, indem sie den Aufruf $grantConsPerm(CTX[(c_1, a_1), (c_2, a_2), ...], uuid_A, o)$ (vgl. Regel 1.1 in Abschnitt 3.4) mit der Liste der Kontexte $[(c_1, a_1), (c_2, a_2), ...]$, der Zielanwendung $uuid_A$ und des Anzeigebereichs o verwenden. Für das Entziehen von kontext-beschränkten Berechtigungen, dient Aufruf $revokeConsPerm(ID\ cp_{id})$ (vgl. Regel 1.2 in Abschnitt. 3.4) mit ID cp_{id} der kontext-beschränkten Berechtigung.

Die kontext-beschränkten Berechtigungen sind in XML-Dateien mit den IDs der Kontexte und die zu vergebenden Pixel als Bitmasken beschrieben. Die Anwendungen können innerhalb des erhaltenen Anzeigebereichs Fenster erstellen, verändern und positionieren. Die Bitmaske stellt eine rechteckige Fläche dar, die die vergebenen Pixel markiert, sodass das Compositing nur diese Pixel auf der Anzeige darstellt. Beispielsweise können für die Geschwindigkeitsanzeige die zugriffsberechtigten Pixel als runder Bereich mittels Bitmaske definiert werden.

Das Compositing stellt eine API für das Verändern und Verknüpfen von Fenstern bereit. Die Anwendungen zeichnen direkt in die *Bildspeicher*, welche pro Fenster jede Anwendung besitzt und aus welcher das Compositing die grafischen Inhalte mittels Bitblitting und den Bitmasken in die *Anzeigespeicher* der beiden Anzeigen kopiert. Da jeder Pixel nur genau einer Anwendung zugewiesen ist, muss das Compositing keine Überdeckungen von Anwendungsfenstern berücksichtigen.

Für die prototypische Implementierung wurde der in Kapitel 5 beschriebene Cockpit-Demonstrator verwendet. Der Aufbau verwendet zwei 12" Fahrzeuganzeigen mit je einer Auflösung von 1440×540 Pixel und typische Eingabegeräte wie Lenkradtasten und Dreh-Drück-Steller für das Steuern der Anwendungen.

Für den Aufbau wurde die Hardwareplattform i.MX6 SABRE von Freesca-

le [Fre15] verwendet, welche für den Infotainmentbereich ausgelegt ist. Die Plattform besitzt eine CPU mit vier Kernen und drei GPUs des Herstellers Vivante [Viv16]. Dies sind die 3D-GPU GC2000, welche OpenGL ES 2.0 Unterstützung anbietet, die 2D-GPU GC350 für Vektorgrafiken mit OpenVG 1.1 und die 2D-GPU GC320 mit einer 2D-API für das Compositing.

3.7. Evaluation

Mittels der Implementierung aus Kapitel 3.6 wurde die Latenz, die das Zugriffskontrollmodell bei der Erstellung von Anwendungsfenstern erzeugt, gemessen. Da manche Kontextwechsel sicherheitskritisch sind, verzögert die Zeit, die das Ändern der Zugriffsrechte benötigt, die Sichtbarkeit von sicherheitskritischen Anwendungen. Wie in [GSD+13] beschrieben, muss die Sichtbarkeit wichtiger Informationen innerhalb bestimmter Zeitbeschränkungen gegeben sein. Die Zeitbeschränkung hängt vom Anwendungsfall ab und wird durch den OEM auf Grundlage von Richtlinien, ISO-Standards der Automobilindustrie und rechtlichen Anforderungen bestimmt. Dessen ungeachtet kann als maximale Latenz für die Ausgabe grafischer Inhalte 2 Sekunden (siehe [AAM06]) angenommen werden. Das Bild der Rückfahrkamera muss in weniger als 2 Sekunden nach Einlegen des Rückwärtsgangs für den Fahrer sichtbar sein (wie durch die „US National Highway Traffic Safety Administration“ gefordert). Das heißt, 2 Sekunden kann als generische obere Schranke angenommen werden. Anderseits sollen zeitkritische Benutzereingaben eine typische Latenz von 250 ms nicht überschreiten (siehe [GSD+13, AAM06]). Das bedeutet, nach einer Benutzereingabe muss innerhalb von 250 ms eine sichtbare Reaktion erfolgen, welche die Änderung mindestens einer Zugriffsberechtigung innerhalb dieser Frist erfordert. Folglich ist die Zeitverzögerung, die durch das Ändern der Zugriffe auf die Anzeigebereiche entsteht und es den Anwendungen erlaubt neue Fenster anzulegen, essentiell für die Darstellung von sicherheitskritischen Anwendungen, da deren Sichtbarkeit dadurch verzögert wird. Im Folgenden wird das verwendete Szenario beschrieben.

3.7.1. Szenario

Die Latenz, die durch die Zugriffskontrolle verursacht wird, hängt von der Anzahl der Berechtigungsänderungen ab. Eine Berechtigung kann vergeben bzw. entzogen werden, indem ein entsprechender Kontext aktiv oder inaktiv geschaltet wird. Das betreffende Fenster ist dann nicht mehr sichtbar bzw. wird sichtbar. Die La-

tenz, eine neue Berechtigung zu *erhalten*, wird als Δt_g definiert. Dies ist die Zeit zwischen dem Senden des Befehls $setContext(CTX\ c, A\ a)$ für das Setzen eines Kontexts c und dem Empfangen des Befehls $confirmWindow(WinID\ id)$, das die ID des erzeugten Fensters zurückliefert. Die Latenz, eine bestehende Berechtigung *entzogen* zu bekommen, wird als Δt_c definiert. Dies ist die Zeit zwischen dem Senden des Befehls $setContext(CTX\ c, A\ a)$ für das Setzen eines Kontexts c und dem Empfang des Befehls $revoke(o_x)$, der den Zugriff auf den Anzeigebereich o_x entzieht. Abbildung 3.12 stellt die Nachrichten dar, die zwischen den Anwendungen B und C und den Komponenten ausgetauscht werden müssen.

Es wurden zwei verschiedene Lastsimulationen mit bis zu 16 Testanwendungen durchgeführt. In der ersten Lastsimulation (*direkte Vergabe* von Berechtigungen) stellt eine Anwendung 15 kontext-beschränkte Berechtigungen für 15 getrennte Anzeigebereiche bereit (siehe als Beispiel Abbildung 3.13). Durch das Setzen eines Kontexts erhalten 15 Anwendungen Zugriff auf jeweils einen der Anzeigebereiche. Nachdem jede der Anwendungen ein Fenster erstellt hat, entzieht die vergebende Anwendung durch Deaktivieren des Kontexts alle Berechtigungen. In der zweiten Lastsimulation (*sequenzielle Vergabe* von Berechtigungen) stellt jede der 15 Anwendungen eine kontext-beschränkte Berechtigung für jeweils einen getrennten Unterbereich eines Anzeigebereichs bereit. Die kontext-beschränkten Berechtigungen, die sequenziell voneinander abgeleitet sind, gewähren den Anwendungen Zugriff auf jeweils einem Teil des Anzeigebereichs (siehe als Beispiel Abbildung 3.14). Durch das Setzen eines Kontexts erhält und vergibt jede der 15 Anwendungen Zugriff auf einen Unterbereich. Ausgenommen jedoch die erste und letzte Anwendung in der Reihe, die nur Zugriff gewähren bzw. erhalten. Nachdem alle Anwendungen eine Berechtigung erhalten und ein Fenster angemeldet haben, wird durch Deaktivieren des Kontexts allen Anwendungen die Berechtigungen entzogen.

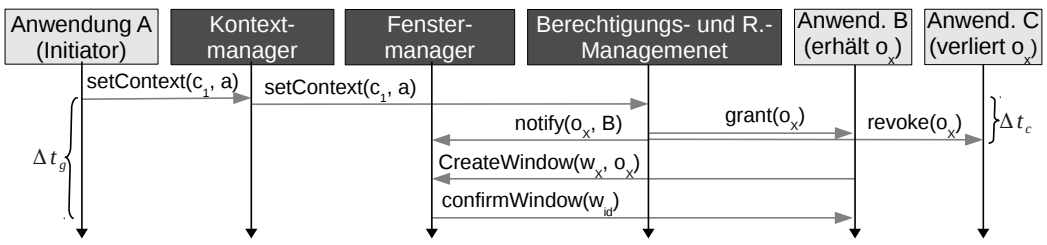

Abbildung 3.12.: Szenario für das Messen der Latenz der Zugriffskontrolle

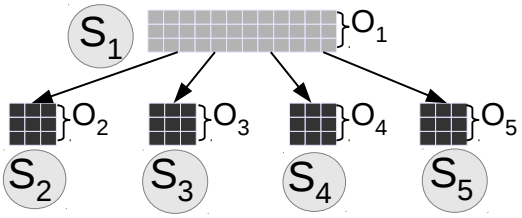

Abbildung 3.13.: Bsp.: Vier direkt vergebene Berechtigungen

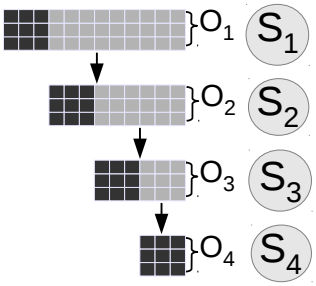

Abbildung 3.14.: Bsp.: Vier sequenziell vergebene Berechtigungen

3.7.2. Evaluationsergebnisse

Für beide Lastsimulationen wurden die Latenzen Δt_g und Δt_c für eine bestimmte Anzahl an kontext-beschränkten Berechtigungen und Anwendungen gemessen. Die durchschnittlichen Latenzen über 4000 Durchläufe pro Anzahl Anwendungen für die Szenarien *direkte Vergabe* und *sequenzielle Vergabe* von Berechtigungen sind in Abbildung 3.15 und 3.16 dargestellt. Des Weiteren sind die maximalen und minimalen Latenzen dargestellt. Die Latenz Δt_g hängt linear von der Anzahl der betroffenen Anwendungen ab, die eine neue Berechtigung erhalten und ein neues Fenster anmelden (siehe Abbildung 3.15 und 3.16). Dies liegt daran, dass die Anfragen vom Berechtigungs-und Richtlinien-Manager sequentiell abgearbeitet werden müssen, um keine Konflikte bei der Vergabe der Berechtigungen zu erzeugen. Die Latenz wird durch die Interaktion zwischen den Komponenten verursacht, die sich auf unterschiedlichen virtuellen Maschinen befinden und über die dedizierten Kommunikationskanäle mittels gemeinsamen Speichers kommunizieren. Bei steigender Anzahl an Anwendungen wird die Latenz zunehmend durch

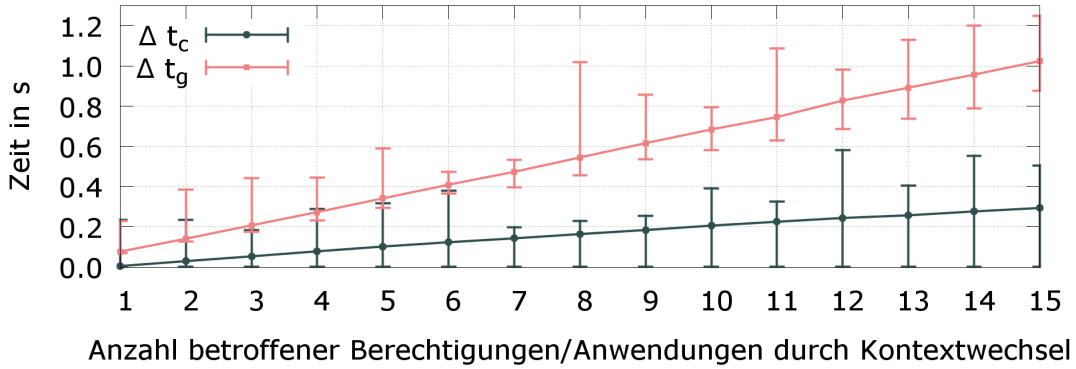

Abbildung 3.15.: Latenzen durch Vergabe bzw. Entzug direkt vergebener Berechtigungen an eine bestimmte Anzahl an Anwendungen

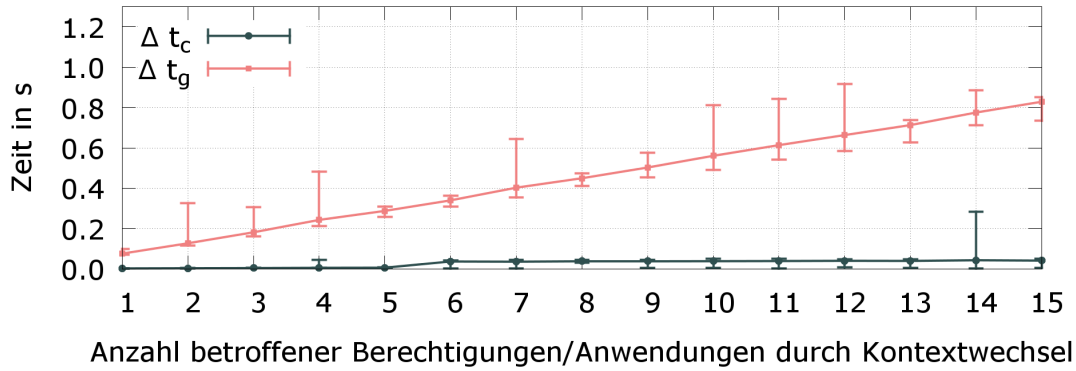

Abbildung 3.16.: Latenzen durch Vergabe bzw. Entzug sequentiell vergebener Berechtigungen an eine bestimmte Anzahl an Anwendungen

das CPU-Scheduling beeinflusst, da die Plattform nur vier CPU-Kerne zur Verfügung stellt, die von den virtuellen Maschinen und den Anwendungen gemeinsam genutzt werden müssen.

Bei der Vergabe von 15 Berechtigungen und dem Anmelden von 15 neuen Anwendungsfenstern bleibt die Latenz auch im Durchschnitt bei beiden Lastsimulationen bei ca. einer Sekunde. Im schlechtesten Fall liegt sie bei der direkten Vergabe bei knapp über 1,2 s und bei der sequentiellen Vergabe bei 0,9 s. Beim Entzug ist der Unterschied zwischen der sequentiellen und direkten Vergabe zu erwarten, da der Entzug der Berechtigung von S_1 automatisch alle anderen Berechtigungen entzieht, während bei der direkten Vergabe alle Berechtigungen einzeln entzogen werden müssen. Da die Zunahme der Latenz durch die Vergabe bzw. den Entzug der Berechtigungen linear mit der Anzahl der Anwendungen ansteigt, ergibt sich eine maximale Anzahl an Anwendungen, deren Berechtigungen gleichzeitig geschaltet werden können.

Situationen, in welchen viele neue Fenster erzeugt werden müssen, kommen nur bei Modusänderungen von großen Anzeigebereichen vor. Beispielsweise wenn die Rückfahrkamera ausgeschaltet werden muss oder die Anzeige des Kombiinstruments von DVD-Wiedergabe im Stand zu Fahrmodus zurückgeschaltet werden muss. Dennoch liegt mit einer Sekunde Latenz die Implementierung des Berechtigungssystems noch weit unter der oberen Schranke von maximal 2 s.

Um auf eine Benutzereingabe mit einer maximalen Latenz von 250 ms zu reagieren, ist lediglich das Schalten einer einzigen Berechtigung notwendig. Da die gemessene Latenz in beiden Lastsimulationen bei unter 100 ms bzw. etwas über 200 ms liegt, ist diese Anforderung erfüllt. Falls nach dieser ersten Reaktion weitere Berechtigungen zu schalten sind, gilt für diese die obere Schranke von 2 s. Wie

viele gleichzeitig schaltbare Berechtigungen innerhalb dieser Schranke geschaltet werden können, ist von der Leistungsfähigkeit der verwendeten Hardwareplattform abhängig, die für die entsprechenden Anforderungen ausgelegt sein muss. Von der im Rahmen dieser Evaluation verwendeten Hardwareplattform wird diese Schranke für bis zu 15 Anwendungen eingehalten, da selbst hierbei die Latenz nur knapp über 1,2 s liegt.

3.8. Verwandte Arbeiten

Feingranulare Zugriffskontrolle für Anzeigen oder grafische Ressourcen, wie sie in Kapitel 3 vorgestellt wurde, ist teilweise in wissenschaftlichen Arbeiten betrachtet worden. Meist steht die sichere Interaktion mit dem Benutzer auf Desktop-Systemen im Fokus und nicht die funktionale Sicherheit in Ausführung und Darstellung von sicherheitskritischen Anwendungen, wie im Automobil benötigt. Im Folgenden werden Fenstersysteme und Zugriffskontrollmodelle betrachtet.

3.8.1. Fenstersysteme

Feske et al. veröffentlichten ein Konzept [FH04] zur Verwaltung von überdeckenden Anwendungsfenstern, die in verschiedenen virtuellen Maschinen ausgeführt werden. Die Anwendungen schreiben ihre grafischen Inhalte in einen Speicherbereich (engl. „buffer“), der in rechteckigen Bereichen (engl. „views“), ähnlich zu Fenstern, auf dem Bildschirm dargestellt wird. Das Konzept wird in Nitpicker [FH05] verwendet, um eine sichere Benutzerschnittstelle aufzubauen, die es mit systemnahen Mechanismen erlaubt, Sicherheitsprobleme, verursacht durch Spionage-Software oder trojanische Pferde, zu verhindern. In Nitpicker ist der Grafikserver dafür verantwortlich, die „buffer“ der Anwendungen in den „views“, die sich auch überdecken können, darzustellen. Zwar gewährleistet Nitpicker den Schutz der Anwendungen vor Ausspähen, aber es werden keine Fenster vor Überdeckung geschützt und die Sichtbarkeit von Anwendungen kann nicht gewährleistet werden.

Das Fenstersystem Blink [Han07] bringt die grafischen Inhalte verschiedener virtueller Maschinen sicher auf eine GPU. Im Unterschied zu Nitpicker können Anwendungen aller Partitionen die GPU für die Berechnung der grafischen Inhalte nutzen. Wie schon bei Nitpicker wird auch bei Blink keine Zugriffskontrolle bzgl. der Fensterplatzierung durchgeführt, die durch den Benutzer bestimmt wird.

Epstein et al. [EP91] analysierten Sicherheitsprobleme in X11, die durch schwa-

che Authentifikation, unbeschränkte gemeinsame Nutzung von X-Ressourcen zwischen verschiedenen Anwendungen oder überdeckenden Fenstern verursacht werden. Als Grundlage dienen die Anforderungen aus TCSEC B3 [Nat85]. Auf Basis der Analyse wurde das System Trusted X [EMP+91] entwickelt, das Mechanismen zur Verhinderung von Sicherheitsproblemen beinhaltet. Die Anwendungen werden Sicherheitslevels zugeordnet, die jeweils eine Instanz eines X-Servers ausführen. Dadurch soll die Isolation der Anwendungen bzgl. ihrer Fenster gewährleistet werden, was jedoch zu Lasten der Geschwindigkeit der Darstellung geht. Um die Geschwindigkeit der Darstellung zu verbessern, wurde Trusted X zur Nutzung von Hardwarebeschleunigung mittels einer GPU erweitert [Eps96]. Da die X-Server selbst keine Zugriffskontrolle für die Anzeigen besitzen und die Anwendungen bzw. die X-Server über die Positionierung der Fenster im Sicherheitslevel nicht informiert werden bzw. diese nicht regulieren können, ist eine Gewährleistung der Sichtbarkeit sicherheitskritischer Anwendungen nicht möglich. Vor allem da der Benutzer die volle Kontrolle über die Fensterplatzierung hat.

3.8.2. Zugriffskontrolle

Der Schutz gemeinsam genutzter Ressourcen durch eine Zugriffskontrolle (vgl. [Lam74]) wurde bereits seit den Anfängen der Betriebssysteme wissenschaftlich untersucht. Nach der klassischen Einteilung ist das Zugriffskontrollmodell, das in Kapitel 3 beschrieben ist, der wahlfreien Zugriffskontrolle zuzuordnen. Der Zugriff auf die Ressource Anzeige wird aufgrund der Identität gewährt. Des Weiteren kann der Zugriff mittels Berechtigungen weitergegeben werden, wobei die Berechtigungen abhängig der Kontexte sind. Es können jederzeit auf Basis der Hierarchie Berechtigungen entzogen werden. Im Folgenden sind die verwandten Arbeiten in drei Themenblöcken geordnet. Zuerst werden die zustandsbasierten Zugriffskontrollmodelle beschrieben. Dann folgen die Modelle basierend auf Hierarchien. Zuletzt werden die kontext-basierten Modelle beschrieben.

Zugriffskontrollmodelle basierend auf Zuständen Die Zugriffskontrollmodelle werden auf Basis von Zuständen definiert, welche bestimmte Eigenschaften erfüllen. Bell und LaPadula [BL76] entwarfen ein Modell für das sichere Teilen von Informationen zwischen verschiedenen Benutzern, welches eine Informationsflusskontrolle ermöglichen soll. Das Modell basiert auf Zuständen, die sich mittels Transitionen ändern können und verwendet eine Zugriffskontrollmatrix für den beschränkten Zugriff auf die Daten, welches die Geheimhaltung der Informationen

gewährleisten soll. Der beschränkte Zugriff von Benutzern und Anwendungen auf Dateien und Ressourcen beachtet das Level der Sensibilität und der Sicherheit. Dies beinhaltet jedoch keine exklusiven Zugriffe auf Ressourcen oder beachtet die gegebenen Kontexte. Zusätzlich wird keine dezentrale Berechtigungsverwaltung unterstützt, was die Verwendung in einem dezentralen Entwicklungsprozess erschwert.

Wang et al. [WZT09] entwarfen eine Erweiterung zu dem Modell von Bell und LaPadula, die für hierarchische Organisationen ausgelegt ist und zwischen dem lesenden und schreibenden Zugriff unterscheidet. Hierbei wird jedoch ebenfalls nur auf den Informationsfluss fokussiert, aber keine Unterstützung für die Berechtigungsvergabe mit exklusiven Rechten geboten.

Hierarchie-basierte Zugriffskontrollmodelle Die Zugriffskontrolle basiert auf hierarchischen Berechtigungen. Wilde et al. [WN05] verwenden hierarchische Beziehungen und führen zwei Arten von Gruppen ein. *Anwender* und *Gruppen* können Mitglieder von *Gruppen* sein, was dazu dient die hierarchische Struktur einer Organisation einfacher abbilden zu können. Die Autorisierung verwendet das Konzept von Rollen, die für Klassen von Ressourcen definiert und den Benutzern bzw. Gruppen zugeordnet sind. Zusätzlich gibt es die Administratorrolle, die für jede Ressource gilt und dazu berechtigt, Benutzer bzw. Rollen hinzuzufügen oder zu entfernen. Rollen können voneinander abgeleitet werden oder auch unabhängig von anderen Rollen Rechte gewähren. Um die Ressourcen dynamisch verwalten zu können, gibt es ressourcenbasierte Gruppen, welche eine Ressource mit Benutzern und Gruppen verknüpft.

Das bedeutet, man kann eine neue Ressource definieren, die von allen Benutzern einer bestehenden Ressource gelesen werden kann. Wird der Zugriff der Benutzer der anderen Ressource geändert, dann wirkt sich das auch auf die neue Ressource aus. Bietet dieses Konzept zwar für Ressourcen, welche von mehreren Benutzern verwendet werden eine einfache Möglichkeit der Zuordnung, so wird dabei weder Priorität noch Konfliktfreiheit berücksichtigt.

Birget et al. [BZNR01] verwenden Hierarchien für Benutzer und Ressourcen. Des Weiteren wird eine weitere Hierarchie eingefügt, die sich durch das Vereinigen der Benutzer- und Ressourcenhierarchie ergibt und zur Festlegung der Zugriffsbeziehungen dient. Diese neue Hierarchie vereinfacht zwar die Zugriffskontrollverwaltung, kann jedoch nur verwendet werden, wenn das System sich nur langsam verändert.

Butt et al. [BAK+02] betrachten die sichere Nutzung von Ressourcen im Grid-

Computing, einer losen Kopplung von Rechnern zur Lösung rechenintensiver Aufgaben, mittels einer feingranularen Zugriffskontrolle. Hierbei soll in erster Linie verhindert werden, dass die Integrität der Ressourcen durch bösartige Programme verletzt werden kann. Butt et al. schlagen einen zweistufigen Ansatz vor, der als erste Stufe die Verwendung eines eingeschränkten Kommandozeileninterpreters (engl. „Shell“ genannt) und eines Laufzeitmonitors zur sicheren Ausführung vorsieht. Als zweite Stufe wird ein aktiver Laufzeitmonitor verwendet, der die unsachgemäße Nutzung eines Programms verhindern soll. Der Kommandozeileninterpreter ist dafür zuständig, die Nutzungsmöglichkeiten des Benutzers für die Ressourcen zu prüfen. Die Richtlinien werden in einer Konfigurationsdatei des Systems gespeichert und legen die Rechte fest wie das Ausführen von Programmen oder das Lesen von Dateien für die Benutzer stattfindet. Die Konfigurationsdatei kann eine Liste enthalten, die den Zugriff erlaubt oder verweigert. Möchte ein Benutzer ein Programm ausführen, dann wird geprüft, ob er das Programm ausführen darf und ob er Zugriff auf die benötigten Dateien hat. Dies wird vor jeder Ausführung eines Befehls durchgeführt. Die Verwaltung der Rechte wird mittels Zugriffslisten oder mittels Capability-Listen durchgeführt. Diese ermöglichen aber keinen Zugriff mittels Prioritäten bzw. passen sich bei Zustandsänderungen nicht dynamisch an.

Kontext-abhängige Zugriffskontrollmodelle Es existiert eine Vielzahl an wissenschaftlichen Arbeiten, die sich mit dem Thema Kontexte in der Zugriffskontrolle beschäftigen. Schilit und Theimer [ST94] führten als erste die *Kontextabhängigkeit* ein und verwendeten als *Kontext* Standorte, Identitäten von naheliegenden Personen und Objekten, sowie Änderungen dieser Objekte. Sie fokussieren auf das Bereitstellen von Informationen über standortabhängige Objekte und wie diese Objekte sich über die Zeit hinweg verändern. Der Zugriff erfolgt ortsbezogen, ohne dass es weitere Kontrollen gibt.

Eine Reihe von Arbeiten [SN04, HYMLWD12, BBG04, MB03] beschäftigt sich mit der rollenbasierten Zugriffskontrolle, die Kontexte zur Festlegung der aktiven Rollen der zugehörigen Berechtigungen verwenden.

Mostefaoui et al. [MB03] stellen ein Modell für eine kontextbasierte Zugriffskontrolle für verteilte Systeme vor. Der Fokus liegt auf der Erstellung einer minimalen generischen Architektur, die aus einzelnen, miteinander verbundenen Komponenten besteht. Anhand der Kontexte werden dann die Sicherheitsrichtlinien dynamisch angepasst und den Nutzern, welchen bestimmte Rollen zugewiesen sind, entsprechend der Zugriff auf die geschützten Ressourcen gewährt oder verwei-

gert. Hierbei werden die Aspekte Repräsentation und Zeit beachtet, die sich in unterschiedlichen Arten auf die Verwendung der Kontexte auswirken und so bei der Vergabe der Berechtigungen beachtet werden müssen.

Strembeck et al. [SN04] schlagen ein rollenbasiertes Zugriffskontrollmodell vor, welches, ähnlich dem Zugriffskontrollmodell in Kapitel 3, die Zugriffe abhängig des Kontexts vergibt. Diese rollenbasierten Zugriffskontrollbeschränkungen werden Kontextbeschränkungen genannt und regeln den Zugriff dynamisch in Abhängigkeit der Kontexte und der gegebenen Rolle. Die Kontextbeschränkungen werden dann im Rahmen der Rollenerstellung abgeleitet und in einem rollenbasierten Zugriffskontrollsystem angewandt.

Bhatti et al. [BBG04] erweitern ein gegebenes rollenbasiertes Zugriffskontrollmodell, um eine vertrauensbasierte und kontextabhängige Komponente. Laut Bhatti et al. hängt die Sicherheit von Diensten im Internet entscheidend von der Vertrauenswürdigkeit der Benutzer ab, welche durch Kontexte beeinflusst wird. Die Zugriffskontrolle muss deshalb abhängig der Kontexte sein. Verringert sich die Vertrauenswürdigkeit eines Benutzers durch eine Änderung der Kontexte, dann werden die Zugriffsrechte entsprechend angepasst, um einen Missbrauch zu verhindern. Für die Verwaltung der Zugriffsrechte bzw. der Rollen ist ein Sicherheitsbeauftragter (engl. System Security Officer) zuständig, welcher insbesondere auch die Grenzwerte der Kontexte für die Vertrauenswürdigkeit festlegen kann.

Hongyue et al. [HYMLWD12] untersuchen den Einfluss von Kontexten im Entscheidungsprozess der Zugriffskontrolle. Des Weiteren werden die Faktoren von Kontexten in der Zugriffskontrolle klassifiziert und anhand von vier Aspekten formalisiert. Die vier Kategorien von Kontexten sind dann die Plattformsicherheit, die Vertrauenswürdigkeit der Benutzer, der Raum und die Zeit. Auf Grundlage dessen werden dann die Entitäten und Beziehungen im rollenbasierten Zugriffskontrollmodell und die Kontextbeschränkungen neu festgelegt. Dies soll eine feingranulare Zugriffskontrolle ermöglichen, welche die unterschiedlich klassifizierten Kontexte bei der Zugriffskontrolle berücksichtigt.

Diese vorgestellten Modelle ermöglichen jedoch keinen konfliktfreien Zugriff und keine Ableitung der Berechtigungen. Die Erstellung der Berechtigungen wird in diesen Modellen von Systemadministratoren bzw. bestimmten festgelegten Rollen übernommen, was eine Einschränkung der Flexibilität ist.

Zugriffskontrollmodelle, welche nicht rollenbasiert aber abhängig des Kontexts sind, verwenden die Kontextinformationen ähnlich den Rollen für die Zugriffsentscheidung. Corradi et al. [CMT04a] sehen in der allgegenwärtigen Bereitstellung von Diensten, welche von mobilen Benutzern genutzt werden, die Herausforderung

für die Zugriffskontrolle von Ressourcen. Da Benutzer fortlaufend ihren Standort ändern, kommt es zu Änderungen der verfügbaren Ressourcen und Diensten. Corradi et al. schlagen ein Zugriffskontrollmodell vor, welches Kontexte verwendet, um den Zugriff auf Ressourcen zu regulieren. Die Kontexte werden dazu mit Zugriffsberechtigungen verknüpft, was dazu führt, dass Anwender ihre Berechtigungen erhalten bzw. verlieren, wenn sie in einen bestimmten Kontext eintreten bzw. einen bestimmten Kontext verlassen. Mittels der Kontexte lassen sich laut Corradi et al. die vollständige Menge an Berechtigungen für die Benutzer bestimmen, sodass eine rollenbasierte bzw. identitätsbasierte Zugriffskontrolle für den Zugriff auf eine Ressource nicht notwendig ist. Dies bedeutet, der Kontext wird als zentraler Aspekt der Zugriffskontrolle gesehen und nicht der Benutzer bzw. seine Rollen. Corradi et al. entwarfen dazu eine Middleware, genannt UbiCOSM (engl. **Ubi**quitous **Co**ntext-based **S**ecurity **M**iddleware), welche die Spezifikation von Sicherheitsrichtlinien mittels Kontexten verwendet und entsprechende Prozesse zur Zugriffsbeschränkung bietet. In einer weiteren Arbeit erweitern Corradi et al. [CMT04b] ihre Middleware UbiCOSM, sodass Anwender selbst ihre Anforderungen für ihre Privatsphäre festlegen können. Dies ermöglicht es, dass die persönlichen Informationen eines Anwenders in dem Maße offen gelegt werden, wie der Anwender es selbst für die Kontexte festgelegt hat. Dadurch ist neben dem Administrator, welcher die Rechte für die Zugriffe der Ressourcen festlegt, auch der Anwender selbst in der Lage, Einschränkungen abhängig der Kontexte zu spezifizieren. Da es UbiCOSM jedoch nicht ermöglicht, hierarchisch abhängige Berechtigungen bereitzustellen, ist die Verwendung einer priorisierten Zugriffskontrolle für die Benutzung von Ressourcen nicht möglich. Des Weiteren lässt sich dadurch auch kein konfliktfreier Zugriff auf Ressourcen realisieren.

Herges et al. [HAK+12] führen ein Framework für eine generische Zugriffskontrolle ein, welche ein Zugriffskontrollmodell basierend auf Kontextinformationen verwendet. Hierbei soll der Zugriff auf sensible Daten beschränkt werden. Des Weiteren wird die mögliche Ablenkung des Fahrers von den vorgeschlagenen Mechanismen beachtet. Das vorgeschlagene Framework Ginger basiert dabei auf isolierten Domänen, welchen Anwendungen zugeordnet sind, die mittels Nachrichten kommunizieren. Jede Domäne ist einem Level für die Vertrauenswürdigkeit der Anwendungen („nicht vertrauenswürdig“, „vertrauenswürdig“, „hoch privilegiert“) zugeordnet. Das Berechtigungsmodell selbst schränkt die Kommunikation ein. Das bedeutet, wenn eine Entität (z.Ḃ. eine Anwendung) eine Nachricht an eine andere Entität senden möchte, dann benötigt sie dafür eine Berechtigung. Die Berechtigung selbst ist gekoppelt an einen oder mehrere Kontexte, welche

erfüllt sein müssen, damit die Nachricht gesendet bzw. empfangen werden darf. Hierdurch soll verhindert bzw. minimiert werden, dass es z. B. während der Fahrt zu Ablenkungen des Fahrers durch bestimmte Anwendungen kommt. Herges et al. selbst stellen sich die Frage, wie bzw. von wem die Richtlinien für die Autorisierung der Privilegien der Entitäten erstellt werden. Hierfür schlagen sie eine Schnittstelle für ihre Komponente zur Richtlinienverwaltung vor, welche von den zuständigen Personen dazu genutzt werden soll, Richtlinien zu erstellen. Ob und wie dies genau aussehen soll, soll in späteren Arbeiten behandelt werden. Der Fokus ihrer bisherigen Arbeit liegt auf der Kommunikation zwischen den Komponenten unter Verwendung von Nachrichten. Der konfliktfreie Zugriff auf die Ressourcen und eine Berechtigungserstellung, welche auf Basis einer Hierarchie arbeitet, wird hierbei nicht unterstützt.

Sichere Informationsübertragung im Fahrzeug Im Fahrzeug hängen die Kontexte in erster Linie von den Informationen ab, welche von den Sensoren und Steuergeräten des Fahrzeugs ermittelt bzw. ausgewertet und in der Regel über den Fahrzeugbus ausgetauscht werden. Zur Bestimmung der Kontexte im Fahrzeug ist die Sicherheit und Zuverlässigkeit dieser Informationen entscheidend. Zu diesem Thema ist viel Forschung betrieben worden, sodass entsprechende Konzepte zur sicheren und zuverlässigen Informationsübertragung im Fahrzeug existieren.

Müter et al. [MA11] setzen sich mit der entropie-basierten Erkennung von Anomalien in Fahrzeug-Netzwerken auseinander. Hierbei sollen Angriffe auf die Fahrzeug-Netzwerke durch Abweichungen des normalen Verhaltens erkannt werden. Mittels verschiedenen Angriffsszenarien wird der Ansatz für CAN evaluiert. Hoppe et al. [HED11] erstellten eine Machbarkeitsstudie zur Anwendbarkeit signaturbasierter Erkennung von Eindringlingen (engl. „Intrusion Detection“) auf ein Bussystem im Fahrzeug. Hierbei wurde als Beispiel für ein Netzwerk CAN verwendet. Neben den verschiedenen Verfahren zur Erkennung von Angriffen auf ein Netzwerk im Fahrzeug gibt es Ansätze, welche eine sichere und zuverlässige Kommunikation über die Netzwerke im Fahrzeug ermöglichen. Groll et al. [GR09] schlagen einen Ansatz basierend auf vertrauenswürdigen Kommunikationsgruppen vor. Der Ansatz ermöglicht sichere Kommunikation zwischen den Komponenten eines Fahrzeugs und garantiert, dass nur vertrauenswürdige Steuergeräte, die ein signiertes Zertifikat des OEMs besitzen, Teil der geschlossenen Kommunikationsgruppe sind. Hierzu wird ebenfalls CAN als Beispiel für ein Netzwerk im Fahrzeug verwendet. Mit diesem Ansatz können zwar nicht alle Arten von Angriffen verhindert werden (z. B. Fluten des Netzwerkes mit Nachrichten), je-

doch kann ein sicherer und vertrauenswürdiger Nachrichtenaustausch zwischen den Komponenten einer Kommunikationsgruppe gewährleistet werden. Lin et al. [LSV12] schlagen einen Sicherheitsmechanismus vor, welcher sogenannte Cyberangriffe (Maskieren und erneutes Senden) in Fahrzeugen mit CAN verhindern soll. Ziel ist hierbei, die Auslastung des Busses so gering wie möglich zu halten. Es wird anhand von experimentellen Ergebnissen aufgezeigt, dass eine hohe Sicherheit erreicht werden kann, ohne zusätzliche Kommunikation.

3.9. Zusammenfassung

Das Aufteilen der verfügbaren Anzeigebereiche auf eine steigende Anzahl an Anwendungen im Fahrzeug wird zunehmend wichtiger. Aufgrund der unterschiedlichen Anforderungen bzgl. der funktionalen Sicherheit im Fahrzeug ist ein Zugriffskontrollsystem für Anzeigebereiche unumgänglich. Der konfliktfreie Zugriff auf die Anzeigebereiche ist insbesondere für sicherheitskritische Anwendungen unabdingbar, da diese für den Fahrer sichtbar sein müssen, falls es die Situation erfordert. Um dies zu gewährleisten wurde in diesem Kapitel ein Zugriffskontrollmodell vorgeschlagen, das auf Grundlage von Berechtigungen den Zugriff auf Anzeigebereiche gewährt. Hierfür wurden kontext-beschränkte Berechtigungen eingeführt, die abhängig der Kontexte des Fahrzeugs und der Anwendungen den Anwendungen den Zugriff erlauben. Dieses Konzept ist nicht nur auf Anzeigebereiche beschränkt und kann auf beliebige Ressourcen angewandt werden. Beispielsweise können kontext-beschränkte Berechtigungen für Eingabeereignisse erstellt werden, die es erlauben, die Eingaben des Anwenders gezielt an einzelne Anwendungen zu verteilen.

Für eine flexible Nutzung der Anzeigebereiche von verschiedenen Anwendungen mit unterschiedlicher Kritikalität können Anwendungen die kontext-beschränkten Berechtigungen an andere Anwendungen weitervergeben, aber auch wieder entziehen. Erhält eine Anwendung Zugriff auf einen Anzeigebereich, dann kann die Anwendung, die die Berechtigung weitergereicht hat, nicht auf den Anzeigebereich zugreifen. Dies gewährleistet, dass nur eine Anwendung gleichzeitig auf einen Anzeigebereich zugreifen kann. Damit die Vergabe von kontext-beschränkten Berechtigungen nicht willkürlich stattfindet, muss zwischen den Anwendungen, die Berechtigungen austauschen, eine Delegationsbeziehung bestehen. Das Zugriffskontrollmodell wurde als Zustandsmodell definiert, das Zustandsänderungen mittels Anfragen und Transitionen realisiert. Hierfür wurde ein Protokoll vorgestellt, das die Bedingungen und Zustandsänderungen für die Vergabe und das Entziehen

von kontext-beschränkten Berechtigungen, das Ändern der Kontextzustände und das Eingehen und Auflösen von Delegationsbeziehungen beschreibt.

Um zu beweisen, dass das Zugriffskontrollmodell bestimmte Anforderungen erfüllt, wurden drei Korrektheitseigenschaften definiert, die formal beweisen werden können. Dazu wurde ein System definiert, das aus Sequenzen aus Zuständen, Anfragen und Transitionen besteht. Mittels vollständiger Induktion wurde gezeigt, dass jeder Zustand, der unter Einhaltung des Protokolls von einem Startzustand erreichbar ist, die Korrektheitseigenschaften genau dann erfüllt, wenn der Startzustand die Korrektheitseigenschaften erfüllt.

Neben dem formalen Beweis wurde das Zugriffskontrollmodell auf der eingebetteten Plattform i.MX6 von Freescale implementiert und mittels Lastsimulationen evaluiert. Hierzu wurden zwei Szenarien erstellt, die dazu dienten, die zusätzliche Latenz zu messen, welche durch das Zugriffskontrollmodell verursacht wird. Anhand der Lastsimulationen kann gezeigt werden, dass die Latenz bei der Vergabe einer Berechtigung unter 250 ms liegt und somit die Anforderungen bzgl. zeitkritischen Benutzereingaben erfüllbar macht. Bei der Vergabe von 15 Berechtigungen liegt die maximale Latenz bei ca. 1,2 s und damit unter der oberen Schranke von 2 Sekunden, was für die Darstellung der Rückfahrkamera notwendig ist.

Neben der Latenz für die Vergabe bzw. dem Entzug der Berechtigungen gibt er weitere Anforderungen, die für eine Zugriffskontrolle relevant sind. Wie in Abschnitt 2.1 beschrieben, sind für die Fenstersysteme Einschränkungen in der Erstellung und der Positionierung von Fenstern zu berücksichtigen (Anforderung R2). Dies gliedert sich auf in die Anforderungen R2.1 (Einschränkung der Sichtbarkeit von Fenstern), R2.2 (Prioritätsbasiertes Anzeigen von Fenstern) und R2.3 (Zeitliche Beschränkungen). Die Anforderung R2.1 wird durch das Zugriffskontrollmodell gegeben, welches es ermöglicht, mittels den kontext-beschränkten Berechtigungen gezielt das Erstellen bzw. Modifizieren der Fenster einzuschränken. Durch die Möglichkeit der hierarchischen Vergabe entsprechend der Kritikalität bzw. Priorität wird die Anforderung R2.2 erfüllt. Wie durch die Evaluation in Abschnitt 3.7 dargelegt, können zeitliche Beschränkungen für die Vergabe der Berechtigungen abhängig der Anzahl eingehalten werden. Dies bedeutet, unter Berücksichtigung der zusätzlich benötigten Zeit zur Berechnung der grafischen Inhalte der Anwendungen, kann auch die Anforderung R2.3 erfüllt werden.

Des Weiteren erfüllt das Zugriffskontrollmodell die Anforderung R5 (Rekonfiguration der Richtlinien) für grafische Anwendungsfenster. Diese teilt sich auf in die drei Anforderungen R5.1 (Dynamische Zustandsänderungen), R5.2 (Dynamische Richtlinienänderungen) und R5.3 (Gewährleistung der Darstellung). Durch die

Verwendung der Kontexte können sowohl auf Eingabe des Anwenders als auch automatisiert aufgrund von Zustandsänderungen die Berechtigungen der Anwendungen und somit die Darstellung geändert werden, wie in Anforderung R5.1 gefordert. Zusätzlich können Anwendungen abhängig der hierarchischen Vergabe ihre vergebenen Berechtigungen jederzeit entziehen und somit die Darstellung der Informationen verändern. Dies erfüllt die Anforderung R5.2. Durch die Kombination der Kontexte mit den Berechtigungen kann mit der hierarchischen Vergabe die Darstellung rechtlich erforderlicher oder sicherheitskritischer Anwendungen jederzeit gewährleistet werden wie in Anforderung R5.3 gefordert.

Das Zugriffskontrollmodell ermöglicht somit eine sichere und zuverlässige Nutzung der Anzeigebereiche im Fahrzeug. In Ergänzung dazu wird im folgenden Kapitel das Compositing der Anwendungsfenster betrachtet, welches die grafischen Inhalte der Anwendungen in den Anzeigebereichen darstellt.

4. Effizientes Compositing von Anzeigebereichen

In diesem Kapitel wird das Compositing von Anzeigebereichen, die die Anwendungen zur Darstellung ihrer grafischen Inhalte verwenden, beschrieben. Die Anwendungen zeichnen ihre grafischen Inhalte in sogenannte Offscreen-Puffer, die dann mittels Compositing anhand der Fensterinformationen der Anwendungen zu einem Gesamtbild auf der Anzeige zusammengeführt werden.

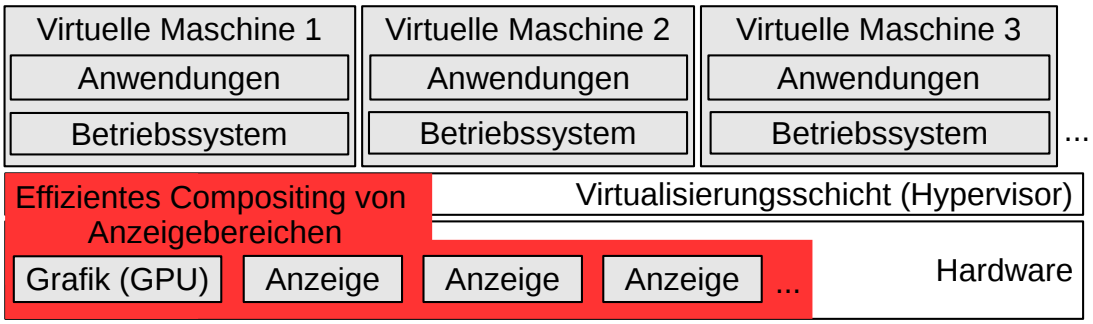

Abbildung 4.1.: Compositing der Anzeigebereiche

Das Compositing für Anzeigebereiche der einzelnen Anwendungen wird unter Verwendung der Grafikhardware (GPU) durchgeführt. Damit liegt das Compositing der Anzeigebereiche thematisch wie in Abbildung 3.1 dargestellt. Sobald eine bzw. mehrere Anwendungen ihre Offscreen-Puffer aktualisiert haben, werden entweder sofort oder in regelmäßigen Abständen (z. B. mit der Bildwiederholrate der Anzeigen) mittels Compositing die Anzeigen aktualisiert.

Im Folgenden wird zunächst ein Systemmodell für das Compositing von Anzeigebereichen festgelegt. Für ein besseres Verständnis werden im Anschluss zuerst die Grundlagen für das Compositing beschrieben, wobei insbesondere die derzeit bekannten Compositing-Verfahren beschrieben werden.

Danach folgt ein Kapitel, in welchem die Konzepte für ein effizientes Compositing beschrieben werden. Hierfür wird zwischen zwei verschiedenen Compositing-Konzepten unterschieden, die sich dadurch unterscheiden, dass ein Konzept

nur rechteckige Fenster darstellen kann, während das andere Konzept auch frei definierte Flächen auf Basis einzelner Pixel darstellt. Es werden zuerst mehrere Compositing-Strategien unter Verwendung der herkömmlichen Definition von rechteckigen Fenstern vorgestellt. Hierfür wird die Architektur für das Compositing und die relevanten Komponenten beschrieben. Des Weiteren wird die API vorgestellt, welche die Anwendungen für das Verwalten der Fenster verwenden. Schließlich werden die beiden vorgeschlagenen Strategien *Hybrid-Compositing* und *Cache-Hybrid-Compositing* vorgestellt, welche ein effizientes Compositing von rechteckigen Fenstern ermöglicht. Danach folgt eine Beschreibung eines Compositing-Konzepts, welches auf pixel-definierte Fensterbeschreibungen basiert.

4.1. Grundlagen

Für das Compositing von Anzeigebereichen ist ein grundlegendes Verständnis über die Funktionsweise der Operation *Bitblitting* notwendig, welche für das Kopieren von Bitmasken verwendet wird. Des Weiteren werden die zwei gängigsten Compositing-Ansätze beschrieben, welche als Grundlage für die, in den folgenden Kapiteln, beschriebenen Konzepten dienen.

4.1.1. Bitblitting

Bitblitting stellt eine Operation aus der Computergrafik dar, welche dazu verwendet wird, um verschiedene Bitmasken auf einer Anzeige kombiniert darzustellen (siehe [Tan09]). Der Begriff Bitblitting ist abgeleitet von dem Begriff „bitblt“, der eine Abkürzung für „bit boundary block transfer“ darstellt und eine allgemeine Operation für Bitmasken ist [Pik83]. Die Operation „bitblt“ ist in der Literatur auch unter dem Begriff „RasterOp“ bekannt [NS79]. In der Regel wird die Operation Bitblitting dazu verwendet, die Bilddaten, welche als Bitmasken im Hauptspeicher liegen, in den Speicher der GPU zu legen, von wo die Bilddaten dann auf der Anzeige dargestellt werden.

Die Operation „bitblt“ wurde 1979 im Computer Xerox Alto [T+79] eingesetzt. Bereits 1975 wurde von Dan Ingalls und Diana Merry (beide waren bei Xerox PARC tätig) eine erste Implementierung der Operation vorgenommen [Ing75].

Die Operation Bitblitting erhält mindestens die zwei Parameter Quelle und Ziel. Die Quelle gibt an, welche Bilddaten als Rechteck in das Ziel transferiert bzw. kopiert werden sollen. Da dies nur das Kopieren von rechteckigen Flächen zulässt, kann der Operation Bitblitting meist noch eine Bitmaske übergeben werden,

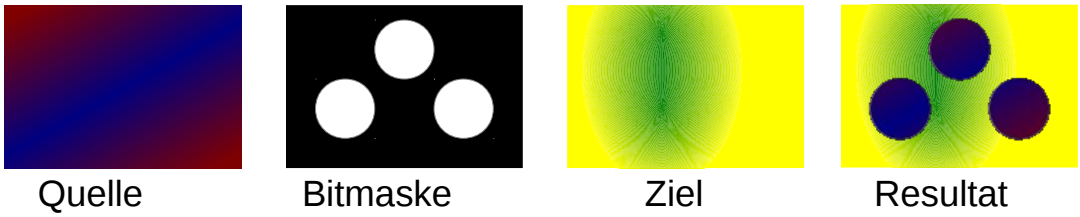

Abbildung 4.2.: Beispiel für das Bitblitting unter Anwendung einer Bitmaske

welche die Pixel angibt, die aus der Quelle kopiert werden sollen. Alle anderen Pixel werden dann im Ziel nicht überschrieben. Für die Definition einer Bitmaske reicht 1 Bit pro Pixel aus. Werden mehr Bits pro Pixel verwendet, dann können damit auch Transparenzstufen verwendet werden, welche die Farbwerte der Quelle und des Ziels entsprechend mischen. Unter Anwendung einer 1-Bit-Bitmaske würde dann das Bitblitting wie folgt aussehen:

$$result = (source\ AND\ bitmask)\ OR\ destination$$

Die bedeutet, dass die Quelle (*source*) und die Bitmaske (*bitmask*) per logischen Operator AND verknüpft wird. Folglich werden nur die Pixel in das Ziel (*destination*) geschrieben, welche auch in der Bitmaske mit 1 markiert sind. Dann wird das Ziel mit dem Ergebnis der AND Operation mittels des logische Operators OR verknüpft und als Resultat (*result*) gespeichert.

In Abbildung 4.2 ist ein Beispiel für das Bitblitting mit Verwendung einer Bitmaske dargestellt. Die Quelle kombiniert mit der Bitmaske per AND Operator drei rot- und blaufarbige Kreise, welche dann mit dem Ziel kombiniert werden. Das Resultat ist das mit drei Kreisen überschriebene Ziel.

4.1.2. Compositing-Ansätze

In diesem Abschnitt werden die beiden meist genutzten Compositing-Strategien *fenster-basiertes Compositing* und *kachel-basiertes Compositing* beschrieben, welche in herkömmlichen grafischen Fenstersystemen verwendet werden, um ein Compositing von rechteckigen Fenstern durchzuführen. Da die in Kapitel 4.4 vorgeschlagenen Ansätze mit diesen beiden Strategien verglichen werden, folgt nun eine ausführliche Beschreibung.

Fenster-basiertes Compositing

Der einfachste Weg, die Fenster auf einer Anzeige zu aktualisieren besteht darin, die Inhalte der Fenster, beginnend mit dem untersten Fenster, unter Verwendung der Operation Bitblitting in den Anzeigespeicher zu kopieren. Dieser Algorithmus wird auch Maleralgorithmus genannt [FvDFH90]. Der Algorithmus für das fenster-basierte Compositing ist in Auflistung 4.1 zu finden.

Auflistung 4.1 Algorithmus für das fenster-basierte Compositing

```
 1: function compose_full (w_list, marks)
 2:   win = w_list.bottom
 3:   while (win)
 4:     if(marks[win.id])
 5:       bitblit(win.x, win.y, win.w, win.h, win.fb)
 6:       mark_dependencies(win.id, marks)
 7:     end if
 8:     win = win.next
 9:   end while
10: end function
```

Die Funktion *compose_full* bekommt eine Liste *w_list*, bestehend aus Fenstern, übergeben. Des Weiteren erhält sie Felder *marks* mit allen Fenstern, welche zur Aktualisierung markiert wurden. Der Algorithmus beginnt mit dem untersten Fenster (Zeile 2) und iteriert durch die Liste der Fenster (Zeile 3). Sobald ein Fenster gefunden wird, welches zur Aktualisierung markiert ist (Zeile 4), dann wird der Inhalt des Fensters (aus dem Bildspeicher *win.fb*) mittels der Operation bitblit (Zeile 5), entsprechend der Position und der Größe des Fensters, in den Anzeigespeicher kopiert. Die Funktion *mark_dependencies* sucht dann alle Fenster, welche das soeben kopierte Fenster überdecken und markiert diese ebenfalls zur Aktualisierung. Dies ist nötig, da überdeckende Fenster durch die unten liegenden Fenster überzeichnet werden und dann folglich wieder aktualisiert werden müssen. Danach wird mit dem nächsten Fenster in der Liste *w_list* fortgesetzt.

Die Anzahl der Bitblit-Operationen entspricht maximal der Anzahl der Fenster. Jedoch können Teile von Fensterinhalten mehrfach überschrieben werden, was ebenfalls einen höheren Aufwand verursacht.

Kachel-basiertes Compositing

Sollen mittels Bitblitting nur die sichtbaren Bereiche eines Fensters in den Bildspeicher kopiert werden, dann ist das Zerlegen des Fensters in seine sichtba-

ren rechteckigen Bereiche, genannt *Kacheln*, erforderlich. Die Strategie *kachel-basiertes Compositing* dient dazu, die sich überdeckenden Fenster in ihre sichtbaren Kacheln aufzuteilen und mittels Bitblitting darzustellen, wie in Auflistung 4.2 beschrieben.

Auflistung 4.2 Algorithmus für das kachel-basierte Compositing

```
 1: function compose_tile (w_list, marks)
 2:   win = w_list.bottom
 3:   while (win)
 4:     if(marks[win.id])
 5:       bitblit_tile(win.tiles, win.fb)
 6:     end if
 7:     win = win.next
 8:   end while
 9: end function
10:
11: function bitblit_tile (tiles, fb)
12:   for (each t in tiles)
13:     if(t.visible)
14:       bitblit(t.x, t,y, t.w, t.h, fb)
15:     else
16:       bitblit_tile (t.tiles, fb)
17:     end if
18:   end for
19: end function
```

Die Funktion *compose_tile* bekommt eine Liste *w_list*, welche die Fenster mit den zugehörigen Kacheln enthält. Des Weiteren erhält sie die Felder *marks*, die die markierten Fenster angibt. Der Algorithmus in Auflistung 4.2 prüft für jedes Fenster, beginnend mit dem untersten Fenster, ob es für eine Aktualisierung markiert wurde (siehe Zeile 4) und ruft in diesem Fall dann die Funktion *bitblit_tile* (siehe Zeile 5) mit den Kacheln des Fensters und dem zugehörigen Offscreen-Puffer, welcher den grafischen Inhalt der Anwendung enthält, auf. Die Funktion *bitblit_tile* (siehe Ziele 11) überprüft für jede Kachel, ob sie sichtbar ist oder von einem weiteren Fenster überdeckt wird. Ist die Kachel sichtbar, dann wird das Bitblitting für sie durchgeführt (siehe Zeile 14) ansonsten wird die Funktion rekursiv mit den sichtbaren Unterkacheln der Kachel aufgerufen.

In Abbildung 4.3 ist ein Beispiel für den Vergleich des *kachel-basierten Compositings* mit dem *fenster-basierten Compositing* für zwei überdeckende Rechte-

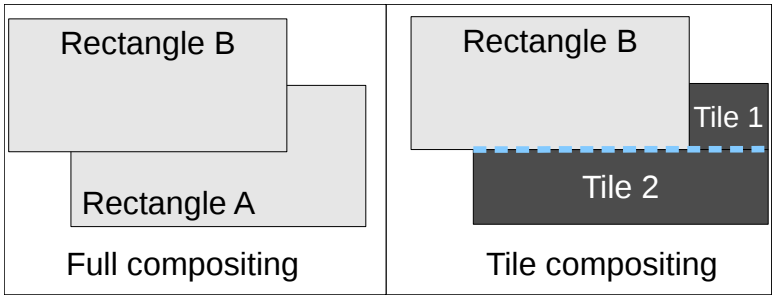

Abbildung 4.3.: Zwei überdeckende Fenster und ihre sichtbaren Kacheln

cke dargestellt. Hierbei überdeckt das Rechteck B das Rechteck A teilweise. Das *fenster-basierte Compositing* kopiert mittels Bitblitting zuerst das Rechteck A und dann das Rechteck B (vgl. linke Seite der Abbildung 4.3) in den Anzeigespeicher. Dies bedeutet, dass ein Bereich des Rechtecks A durch das Rechteck B überschrieben wird. Im Gegensatz dazu teilt das *kachel-basierte Compositing* das Rechteck A in die zwei Kacheln 1 und 2 unter Anwendung der horizontalen Linie (vgl. linke Seite der Abbildung 4.3) auf. Die beiden Kacheln 1 und 2 stellen zusammen die Fläche dar, die nicht durch Rechteck B überdeckt wird.

4.1.3. OpenGL ES und EGL

OpenGL [Ope16a] (engl. **Open G**raphics **L**ibrary) ist eine API zur Berechnung von grafischen Inhalten in 2D und 3D, welche von der Khronos Gruppe, einem Industriekonsortium, als Standard unabhängig von Geräten spezifiziert wird. OpenGL bietet eine flexible und mächtige Schnittstelle zwischen Software und der Grafikbeschleunigung auf unterer Ebene an.

OpenGL ES (engl. **E**mbedded **S**ystems) [Ope16b] ist eine spezielle Variante des OpenGL Standards, welche für eingebettete Systeme ausgelegt ist. Der definierte Befehlssatz der API von OpenGL ES stellt eine Untermenge von OpenGL für Desktopsysteme dar und versucht durch die Einschränkungen eine effizientere Nutzung auf den eingebetteten Systemen zu ermöglichen.

Für die Verwendung von OpenGL ES mit einem darunterliegenden nativen Fenstersystem wird die Schnittstelle EGL [EGL16] (engl. Native Platform Graphics Interface) verwendet. Mittels EGL können die grafischen Kontexte und die Puffer für das Erstellen der grafischen Inhalte verwaltet werden. Des Weiteren bietet die Schnittstelle z. B. Synchronisationsmechanismen und Methoden zur effizienten Datenübertragung zwischen verschiedenen APIs von Khronos an.

4.2. Systemmodell und Anforderungen

In diesem Abschnitt wird das Systemmodell für das Compositing von Anzeigebereichen beschrieben. Das Systemmodell legt zunächst die Schnittstellen zu den anderen Systemkomponenten fest und trifft Annahmen über das Systemverhalten des Bitblittings (schnelle Kopier- bzw. Verschiebeoperationen von Speicherbereichen) von Fensterinhalten. Durch die Systemarchitektur aus Abbildung 1.5 ergeben sich weitere Anforderungen, die im Anschluss beschrieben werden.

4.2.1. Systemmodell

Für das Compositing von Anzeigebereichen in HMI-Systemen sind verschiedene Komponenten, wie in Abbildung 4.4 dargestellt, notwendig.

Anwendungen benötigen für das Darstellen von grafischen Inhalten *Fenster*, die der *Fenster-Manager* verwaltet und auf Anfrage erstellt, löscht oder modifiziert. Ein Fenster gibt den Anzeigebereich an, die es auf der Anzeige einnimmt. Eine Anzeige besteht aus einer Menge an Pixeln, welche eindeutig durch ihre Position mittels der Koordinaten x und y identifiziert werden können. Ein Anzeigebereich besteht aus einer Untermenge einer Anzeige und beschreibt eindeutig die Pixel, die vom grafischen Inhalt der Anwendung belegt werden. Jedes Fenster besitzt eine ID, welche die eindeutige Unterscheidung der Fenster ermöglicht. Nachdem eine Anwendung ein Fenster erstellt hat, kann sie grafische Inhalte erzeugen und in dem Fenster darstellen.

Jedem Fenster ist genau ein *Bildspeicher* zugeordnet. Hierzu wird ein Offscreen-Puffer verwendet, in welchen die Anwendungen ihre grafischen Inhalte zwischenspeichern und aus dem dann für die Darstellung auf der Anzeige der grafische Inhalt entnommen wird. Für die Darstellung auf der Anzeige müssen dann die grafischen Inhalte in den *Anzeigespeicher* kopiert werden. Die *2D/3D Grafik-*

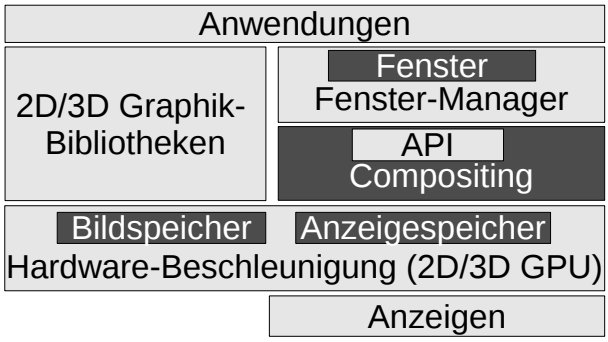

Abbildung 4.4.: Systemmodell für das Compositing von Anzeigebereichen

Bibliotheken (z. B. OpenGL ES) stellen 2D/3D Schnittstellen (APIs) zur Verfügung und können von den Anwendungen für die Generierung ihrer grafischen Inhalte verwendet werden. Wenn ein Bild erzeugt und in den Bildspeicher abgelegt wird, wird das Compositing informiert.

Der Fenster-Manager informiert das Compositing mittels einer API über das *Erstellen*, das *Entfernen* und das *Modifizieren* von Fenstern. Die Grafik-Bibliotheken können das Compositing mittels Anfrage *Markieren* über die Aktualisierung eines Bildspeichers einer Anwendung informieren. Die Anfrage *Compose* führt zu einer Aktualisierung des Anzeigespeichers mit den grafischen Inhalten der Anwendungen und somit zur Aktualisierung der markierten Fenster auf der Anzeige. Im Folgenden werden die fünf Anfragen der API im Detail erläutert:

- Erstellen: Der Fenster-Manager sendet die Anfrage *erstellen(ID id, Anzeigebereich af)* um die Information über den Anzeigebereich *af* eines neu erstellten Fensters mit ID *id* zu den bestehenden Fensterbeschreibungen im Compositing hinzuzufügen.
- Entfernen: Wird ein Fenster im Fenster-Manager entfernt, dann sendet der Fenster-Manager die Anfrage *entfernen(ID id)* um das entsprechende Fenster mit ID *id* im Compositing zu entfernen.
- Modifizieren: Wird ein Fenster im Fenster-Manager modifiziert, d. h. ändert sich der Anzeigebereich eines Fensters, dann sendet der Fenster-Manager die Anfrage *modifizieren(ID id, Anzeigebereich af)*. Die Anfrage enthält die ID *id* und den geänderten Anzeigebereich *af* des betreffenden Fensters.
- Markieren: Wurde der grafische Inhalt einer Anwendung aktualisiert, dann sendet die Grafik-Bibliothek die Anfrage *markieren(ID id)* und informiert über die benötigte Aktualisierung des zugehörigen Fensters mit ID *id*.
- Compose: Die Anfrage *compose()* führt zu einer Aktualisierung des Anzeigespeichers. Hierbei wird der Bildspeicher jedes markierten Fensters in den Anzeigespeicher kopiert, so dass die grafischen Inhalte aktualisiert werden.

Das Compositing führt das Bitblitting der Bildspeicher der Anwendungen in den Anzeigespeicher entsprechend der Fensterinformationen mittels der Hardwarebeschleunigung der GPU durch. Hierbei lässt sich das Compositing in zwei Schritte untergliedern. Dies ist zuerst die Verarbeitung auf der CPU, welche der Compositing-Algorithmus benötigt, um die benötigten Bitblitting-Operationen zu ermitteln. Dann folgt die Verarbeitung der Bitblitting-Operationen auf der GPU. Dies bedeutet, die Zeit t_{comp}, welche das Compositing benötigt, um die Anzeige zu aktualisieren, setzt sich aus der Verarbeitungsdauer auf der CPU t_{CPU} und der Verarbeitungsdauer des Bitblittings t_{BB} zusammen.

Definition 22 *Sei* $\boldsymbol{BS}$ *die Menge aller Bildspeicher, welche mittels Bitblittings in den Anzeigespeicher übertragen werden müssen. Sei* $\boldsymbol{c_{commit}}$ *die konstant benötigte Zeit für das Absenden eines bzw. mehrerer Bitblitting-Operationen und* $\boldsymbol{c_{bitblit}}$ *die konstant benötigte Zeit für das Abarbeiten einer Bitblitting-Operation mit dem variablen Anteil* $\boldsymbol{t_{bitblit}(bs)}$*, welcher von der Größe des Bildspeichers* $\boldsymbol{bs}$ *abhängt. Die Verarbeitungsdauer des Bitblittings* $\boldsymbol{t_{BB}}$ *ist dann wie folgt definiert:*

$$t_{BB} = c_{commit} + \sum_{bs \in BS} \left(c_{bitblit} + t_{bitblit}(bs) \right) \tag{4.1}$$

Dies bedeutet, die Verarbeitungsdauer des Bitblittings setzt sich aus einem konstanten Anteil c_{commit} und der Summe der einzelnen Verarbeitungszeiten der Bitblitting-Operationen zusammen. Die gesamte Verarbeitungsdauer des Compositings t_{comp} ist dann wie folgt definiert:

Definition 23 *Sei für das Compositing die Verarbeitungsdauer auf der CPU* $\boldsymbol{t_{CPU}}$ *und die Verarbeitungsdauer des Bitblittings* $\boldsymbol{t_{BB}}$*. Dann gilt:*

$$t_{comp} = t_{CPU} + t_{BB} \tag{4.2}$$

Dies bedeutet, die Verarbeitungsdauer t_{comp} wird geringer, wenn die Verarbeitungsdauer des Bitblittings t_{BB} sinkt. Dies kann erreicht werden, wenn man die Anzahl der Bitblitting-Operationen senkt oder den variablen Anteil, der von der Größe der Bildspeicher abhängt, reduziert.

Die Grafik-Bibliotheken verwenden die Hardwarebeschleunigung der GPUs, um den 2D/3D Inhalt der Anwendungen zu erstellen, welcher dann in die Bildspeicher gespeichert wird. Damit eine effiziente Verarbeitung der Bitblitting-Operationen möglich ist, verwendet das Compositing hierfür auch die Hardwarebeschleunigung der GPUs, um das Bitblitting zwischen den Bildspeichern und dem Anzeigespeicher durchzuführen. Hierbei kann eine Bitblitting-Operation entweder vollständig oder nur bestimmte Teile des Bildspeichers in den Anzeigespeicher kopieren. Die Bitblitting-Operationen arbeiten dann direkt auf dem Speicher der GPU.

4.2.2. Anforderungen

Für das Compositing ergeben sich in erster Linie abgeleitete Anforderungen aus Kapitel 2, welche die Darstellung von Fenstern betrifft. Dies ist insbesondere die Anforderung *R2.3 – Zeitliche Beschränkungen*. Diese Anforderung stellt die obere Schranke dar, welche das Compositing nicht überschreiten darf. Dies bedeutet,

das Compositing darf eine gegebene zeitliche Beschränkung für die Aktualisierung der Anzeige nicht überschreiten. Diese hängt in erster Linie von der Bildwiederholrate der Anzeige ab, kann jedoch auch abweichend davon seitens des OEMs festgelegt werden.

Für diese Arbeit wird für einen qualitativen Vergleich ein Wert von 16,6 ms festgelegt, welcher einer Aktualisierungsrate von 60 Hz entspricht und somit zu der Bildwiederholrate der meisten derzeitig verbauten Anzeigen im Fahrzeug passt. Dies bedeutet, das Compositing muss gewährleisten, dass nach der Aktualisierung eines Bildspeichers dessen Inhalt innerhalb von 16,6 ms auf der Anzeige dargestellt werden kann. Abhängig von der verwendeten Hardware kann sich somit folglich auch eine maximale Anzahl an Fenstern ergeben, welche zur selben Zeit aktualisiert werden können, ohne die zeitliche Beschränkung zu verletzen.

Neben den abgeleiteten Anforderungen aus Kapitel 2 ergeben sich noch weitere Anforderungen an ein Compositing, welche sich aus dem Zugriffskontrollmodell für Anzeigebereiche aus Kapitel 3 ergeben. Die klassische Definition eines Fensters ist die Beschreibung in rechteckiger Form. Dies bedeutet, die Position wird über die Koordinaten x, y und z beschrieben und die Größe lässt sich mittels Höhe und Breite ausdrücken. Um auch nicht rechteckige Fensterdefinitionen zu ermöglichen, wie es für eine flexible Definition von Berechtigungen in Kapitel 3 nötig ist, soll die Beschreibung eines Fensters auf Ebene einzelner Pixel möglich sein. Dies bedeutet, ein Fenster kann auch aus einer Menge aus Pixeln bestehen, welche nicht zusammenhängend sind. Daraus ergibt sich eine neue Form der Fensterbeschreibung, die das Compositing unterstützen soll.

Im Folgenden wird diese Anforderung gesondert betrachtet und ausgewertet.

4.3. Compositing-Architektur

Für das Compositing sind verschiedene Komponenten und deren Interaktion relevant. Die benötigte Architektur dafür ist in Abbildung 4.5 dargestellt. Im Folgenden werden die wesentlichen Komponenten im Detail erklärt.

4.3.1. Anwendungen

Wie bereits im Abschnitt 4.2 beschrieben, nutzen Anwendungen Grafik-Bibliotheken wie OpenGL ES 2.0 [Ope16b], um grafische 2D/3D-Inhalte zu erzeugen und diese in ihre Bildspeicher abzulegen. Hat eine Anwendung ein Bild in ihren Bildspeicher bzw. Offscreen-Puffer geschrieben, dann wird ein Befehl für den

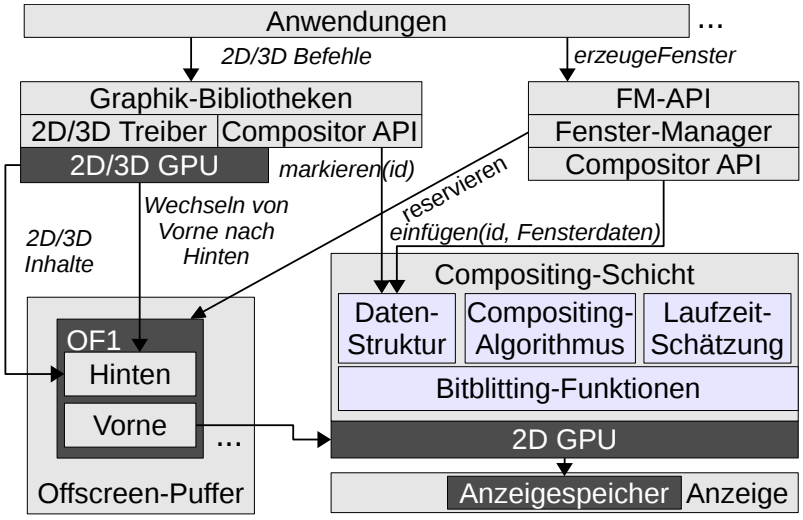

Abbildung 4.5.: Architektur für das Compositing

Wechsel (z. B. eglSwapBuffers) des hinteren Offscreen-Puffers zum vorderen Offscreen-Puffer durchgeführt, wodurch das Bild dann mittels Compositing in den Anzeigespeicher kopiert werden und auf der Anzeige erscheinen kann. Zusätzlich wird beim Aufruf des Wechsels der Offscreen-Puffer der Befehl *markieren* aufgerufen, welcher die *Compositing-Schicht* über die Aktualisierung des Bildspeichers der Anwendung informiert und die somit erforderliche Aktualisierung des Anzeigespeichers durchführt. Die Anwendung bzw. das Erzeugen von neuen grafischen Inhalten wird dann solange blockiert, bis der Anzeigespeicher aktualisiert wurde.

4.3.2. Fenster-Manager

Der Fenster-Manager ist für die Verwaltung der Fenster zuständig und stellt den Anwendungen dazu eine API zur Verfügung. Eine Anwendung kann initial mittels des Befehls *erzeugeFenster* ein Fenster erzeugen. Der Fenster-Manager registriert dann mittels des Befehls *einfügen* das Fenster in der Compositing-Schicht und *reserviert* die nötigen Offscreen-Puffer. Mittels des Befehls *modifiziereFenster* kann ein Fenster in seiner Größe und Position verändert werden. Der Fenster-Manager aktualisiert dann die entsprechenden Informationen in der Compositing-Schicht und passt die Größe der Offscreen-Puffer an. Eine Anwendung kann mit dem Befehl *löscheFenster* ein Fenster entfernen. Der Fenster-Manager informiert die Compositing-Schicht entsprechend und gibt die zugehörigen Offscreen-Puffer frei.

4.3.3. Compositing-Schicht

In der Compositing-Schicht werden die Informationen über die Fenster, welche der Fenster-Manager liefert, in einer Datenstruktur gespeichert. Wird seitens des Fenster-Managers ein neues Fenster erzeugt, ein bestehendes Fenster gelöscht oder modifiziert, dann aktualisiert die Compositing-Schicht den Anzeigespeicher. Ansonsten wird eine Aktualisierung des Anzeigespeichers durchgeführt, wenn ein Fenster zum Aktualisieren markiert wurde. Das Compositing verwendet für das Kopieren der graphischen Inhalte der Offscreen-Puffer die Bitblitting-Funktionen der 2D-GPU. Es gibt zwei Befehle, die sich hierzu eignen. Während der eine Befehl reine Kopieroperationen durchführt, verwendet der andere Befehl zusätzlich eine Bitmaske. Der Befehl ohne Bitmaske (genannt $2D\_Blit$) ermöglicht das schnelle Bitblitting der Inhalte zwischen den Puffern. Hierzu wird ein rechteckiger Bereich für das Kopieren vorgegeben. Der andere Befehl (genannt $2D\_MaskBlit$) verwendet zusätzlich eine Bitmaske, die die Pixel markiert, welche kopiert werden sollen. Somit lassen sich beliebige Flächen bzw. einzelne Pixel im Gegensatz zu dem Befehl $2D\_Blit$ kopieren, welcher nur rechteckige Flächen kopieren kann. Meist bedeutet diese Flexibilität eine höhere Ausführungszeit für das Bitblitting.

Die verwendeten Compositing-Algorithmen unterscheiden sich abhängig der Compositing-Strategien. Dies gilt auch für die Laufzeitschätzung, welche es ermöglicht, die Verarbeitungsdauer einer Bitblit-Operation abzuschätzen.

Im Folgenden werden die Compositing-Strategien für rechteckige Fenster beschrieben und evaluiert, welche für das Bitblitting den Befehl $2D\_Blit$ verwenden. Anschließend wird für das Compositing pixel-definierter Fenster ein Konzept beschrieben und evaluiert, das für das Bitblitting den Befehl $2D\_MaskBlit$ mit Bitmasken verwendet.

4.4. Compositing-Strategien für rechteckige Fenster

Dieser Abschnitt behandelt die Compositing-Strategien, die auf Grundlage von rechteckigen Fenstern ein Bitblitting mittels des 2D-Befehls $2D\_Blit$ durchführen. Die Position eines Fensters wird über die Koordinaten x, y, z und die Größe mittels Höhe und Breite beschrieben. Dies lässt das Überdecken von Fenstern zu. Die z-Koordinate ermöglicht dabei eine eindeutige Zuordnung von Pixeln zu Fensterinhalten und gibt an, ob Fenster über- oder untereinander liegen.

Die beiden Strategien *fenster-basiertes Compositing* und *kachel-basiertes Com-*

positing haben Vor- und Nachteile beim Compositing von rechteckigen Fenstern. Die Strategie *fenster-basiertes Compositing* benötigt zwar nur eine minimale Anzahl an Bitblit-Operationen, jedoch werden die überdeckten Bereiche der Fenster mehrfach überschrieben. Dies kann die Verarbeitungsdauer der Bitblit-Operationen deutlich erhöhen, da diese von der Größe der darzustellenden Fenster abhängt. Im Gegensatz dazu werden bei der Strategie *kachel-basiertes Compositing* mittels Bitblitting nur die sichtbaren Kacheln kopiert und somit das mehrfache Überschreiben von Fensterbereichen verhindert. Jedoch kann dies bis zu 400 % mehr Bitblit-Operationen erfordern, um die deutlich höhere Anzahl an sichtbaren Kacheln im Vergleich zu den gesamten Fenstern zu kopieren. Dies bedeutet ebenfalls einen Mehraufwand und erhöht die Verarbeitungsdauer für das Bitblitting abhängig davon, wie die Fenster sich überdecken. Beispielsweise benötigt das Bitblitting für ein Rechteck mit einer Größe von 1000×1000 Pixel weniger Zeit als das Bitblitting für zwei Rechtecke mit je einer Größe von 1000×500 Pixel, welche in Summe dieselbe Fläche haben wie das große Rechteck.

Um diese Nachteile zu vermeiden, wird in der Strategie *Hybrid-Compositing* nach einer Kombination an Bitblit-Operationen für die rechteckigen Flächen gesucht, welche die Verarbeitungsdauer des Bitblittings reduziert. Damit eine solche Kombination gefunden werden kann, muss die Verarbeitungsdauer einer Bitblit-Operation im Voraus genau geschätzt werden können. Zu diesem Zweck wurde ein Modell erstellt, welches die Verarbeitungsdauer des Bitblittings abhängig der Fenstergrößen und der Anzahl der Bitblit-Operationen vorhersagen kann. Die Vorhersage der Verarbeitungsdauer erfordert jedoch eine Kalibrierung auf der Hardwareplattform, auf welcher das Modell eingesetzt werden soll.

Für die *Hybrid-Compositing* Strategie ist eine effiziente Datenstruktur notwendig, die die relevanten Informationen über die Fenster und deren sichtbaren Kacheln speichert. Dies ermöglicht eine Reduzierung der Verarbeitungsdauer des Compositings für die Aktualisierung eines Anzeigespeichers. Da die Verarbeitungsdauer, welche für die Suche einer Kombination von Bitblit-Operationen auf der CPU benötigt wird, signifikant höher ausfallen kann als beim *fensterbasierten Compositing* und beim *kachel-basierten Compositing*, kann ein zusätzlicher Cache die CPU-Verarbeitungsdauer reduzieren. Diese Erweiterung wird *Cache-Hybrid-Compositing* Strategie genannt. Der Cache dient dazu, das bereits ermittelte Wissen über Kombinationen von rechteckigen Flächen abzuspeichern und für zukünftige Entscheidungen einen Geschwindigkeitsvorteil zu bekommen, da dieses Wissen nur noch abgerufen werden muss.

Im Folgenden wird zunächst ein Konzept für das Erstellen der sichtbaren Ka-

cheln eines Fensters beschrieben.

4.4.1. Konzept zur Berechnung sichtbarer Kacheln

In diesem Abschnitt wird ein Konzept zur effizienten Berechnung der sichtbaren Kacheln für überdeckende Fenster beschrieben. Hierzu werden zuerst die möglichen Fälle klassifiziert und zu vier Fällen vereinigt. Wie in Abbildung 4.6 dar-

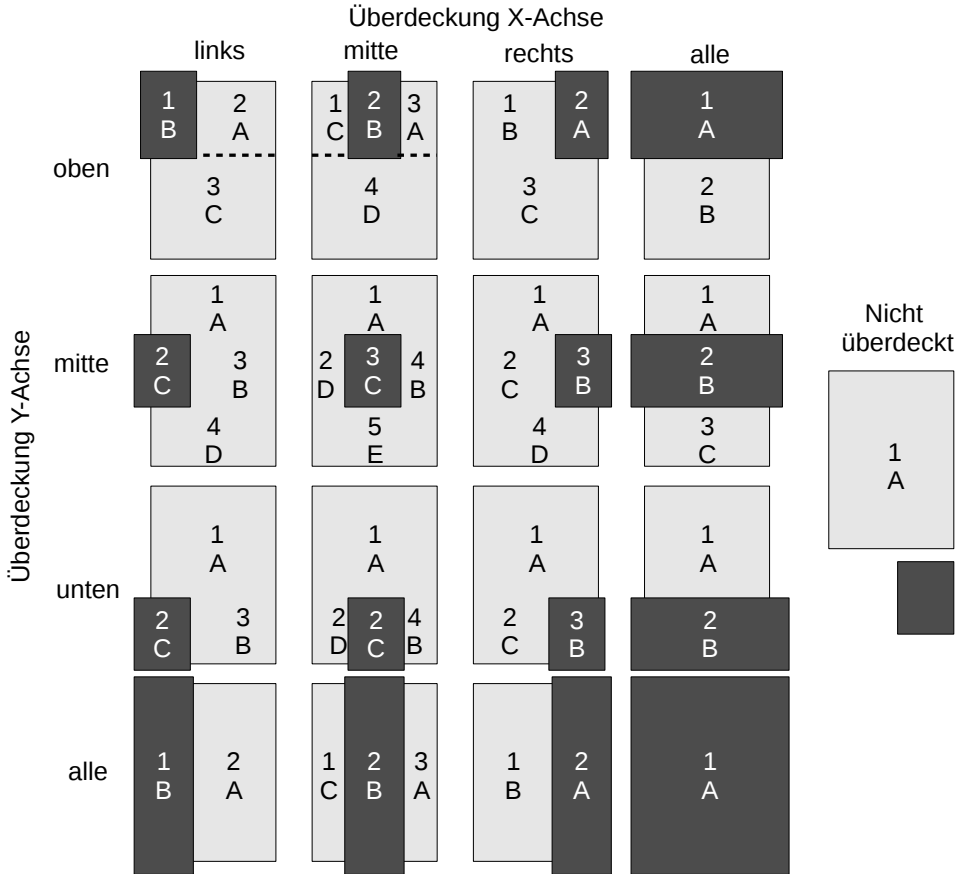

Abbildung 4.6.: Alle möglichen Fälle für eine Überdeckung zweier Fenster [Mye88]

gestellt, wurden in [Mye88] 17 verschiedene Fälle für eine Überdeckung zweier Fenster identifiziert. Dies sind 16 Fälle, die die unterschiedlichen Fälle der Überdeckung der Kanten eines Rechteckes darstellen und ein weiterer Fall ohne Überdeckung durch Kanten.

Die 16 Fälle können fünf Äquivalenzklassen zugeordnet werden. Der Fall ohne Überdeckung ist außen vor, da dies beim *fenster-basiertem Compositing* und beim *kachel-basiertem Compositing* identisch ist und es keine Optimierung gibt.

In Abbildung 4.7 sind die möglichen Fälle der Überdeckungen in vier Äquivalenzklassen dargestellt. Fall M_3 tritt ein, wenn das Rechteck B drei Kanten des

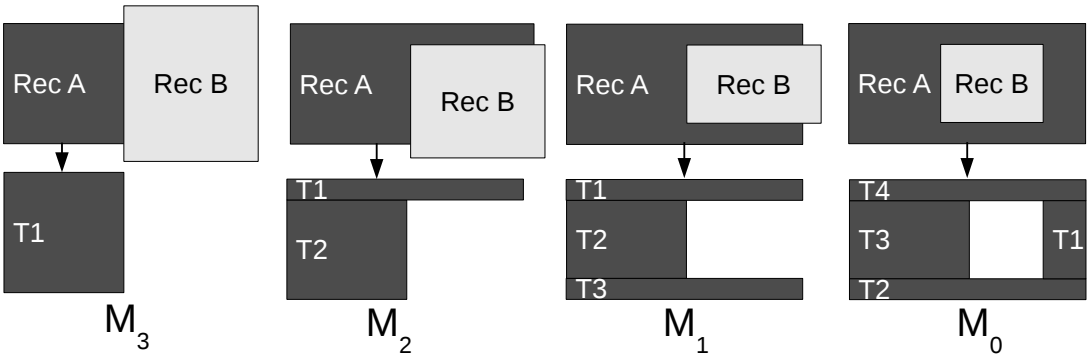

Abbildung 4.7.: Vier Äquivalenzklassen für überdeckende Fenster mit Kacheln

Rechteckes B überdeckt. Dies führt zu einer sichtbaren Kachel $T1$ von Rechteck A. In Fall M_2 überdeckt das Rechteck B zwei Kanten des Rechteckes A, wodurch zwei sichtbare Kacheln $T1$ und $T2$ entstehen. In Fall M_1 werden durch die Überdeckung des Rechteckes A durch das Rechteck B die drei sichtbaren Kacheln $T1$, $T2$ und $T3$ erzeugt. Der Fall M_0 erzeugt die maximale Anzahl an Kacheln, welche durch Überdeckung zweier Fenster möglich ist. Hierbei liegt das Rechteck B vollständig innerhalb der Kanten des Rechteckes A und erzeugt damit vier sichtbare Kacheln $T1$, $T2$, $T3$ und $T4$. Der Fall einer kompletten Überdeckung des Rechteckes A durch das Rechteck B ist nicht dargestellt, da hierbei keine sichtbaren Kacheln von Rechteck A entstehen.

Auf Grundlage dieser vier Fälle M_0 bis M_3 wurde der Algorithmus, dargestellt in Auflistung 4.3, entworfen. Dieser Algorithmus erzeugt die sichtbaren Kacheln im Falle einer Überdeckung von zwei Fenstern.

Auflistung 4.3 Algorithmus zur Berechnung der sichtbaren Kacheln

```
 1: function tiling (windows) // in z-order
 2:   while (not windows.isempty)
 3:     win = windows.pop
 4:     rectangles.push(win)
 5:     for each w in windows do
 6:        r_tmp.clear
 7:        for each r in rectangles do
 8:           if (not overlap(w, r))
 9:             r_tmp.push(r)
10:           else
11:             r_tmp.push(cut_new_tiles(w, r))
12:        end do;
13:        rectangles = r_tmp
```

```
14:      end for
15:      resultlist.append({win,rectangles})
16:      rectangles.clear
17:    end while
18:    return resultlist // windows with visible tiles
19: end function
```

Die Funktion *tiling(rectangles)* iteriert über die Liste der, entsprechend ihres z-Wertes geordneten, Fenster und berechnet die sichtbaren Kacheln für jedes Fenster. Der Algorithmus beginnt mit dem untersten Fenster und speichert es in die Liste der Rechtecke *rectangle* (vgl. Zeile 3 und 4 in Auflistung 4.3). Anschließend iteriert der Algorithmus über die verbleibenden Fenster (vgl. Zeile 5) und prüft für jedes Rechteck in der Liste *rectangle*, ob es durch das Fenster überdeckt wird (vgl. Zeile 8 bis 11). Wenn es keine Überdeckung des Rechteckes gibt, dann kommt das Rechteck in die temporären Liste r_tmp. Im Falle einer Überdeckung des Rechteckes durch ein Fenster wird die Funktion $cut\_new\_tiles(t, v)$ aufgerufen, welche die sichtbaren Kacheln entsprechend der vier Fälle in Abbildung 4.7 bestimmt und in die temporären Liste r_tmp speichert. Eine sichtbare Kachel, die teilweise durch ein Fenster verdeckt wird, wird dann ebenfalls in weitere sichtbare Kacheln unterteilt. Nachdem jedes Rechteck in der Liste *rectangles* überprüft wurde, wird die Liste *rectangles* mit der temporären Liste r_tmp überschrieben. Anschließend überprüft der Algorithmus, ob das nächste Fenster in der Liste *window* ein Rechteck in der Liste *rectangles* überdeckt. Abschließend werden für jedes Fenster die sichtbaren Kacheln in der Liste *resultlist* gespeichert und als Rückgabewert der Funktion zurückgegeben.

4.4.2. Compositing-Strategien

Wie bereits in Abschnitt 4.1 beschrieben, besitzen die beiden gängigen Compositing-Strategien *fenster-basiertes Compositing* und *kachel-basiertes Compositing* entscheidende Nachteile, die die Ausführungszeit für das Compositing erhöhen. Das *fenster-basierte Compositing* benötigt zwar die minimale Anzahl an Bitblit-Operationen, aber überschreibt dafür Fensterbereiche teils mehrfach und erhöht dadurch die Ausführungszeit für das Bitblitting. Das *kachel-basierte Compositing* hingegen vermeidet das mehrfache Überschreiben von Fensterbereichen, benötigt jedoch eine höhere Anzahl an Bitblit-Operationen, wodurch ebenfalls ein Mehraufwand für das Bitblitting entsteht. Im Folgenden werden die beiden Compositing-Strategien *Hybrid-Compositing* und *Cache-Hybrid-Compositing* im Detail

vorgestellt. Beide Ansätze benötigen eine Datenstruktur, die die Informationen über die Fenster und ihre sichtbaren Kacheln, die mittels des Algorithmus in Auflistung 4.3 berechnet werden, speichert. Da die CPU-Ausführungszeit t_{CPU} der beiden Ansätze *Hybrid-Compositing* und *Cache-Hybrid-Compositing* stark von der verwendeten Datenstruktur abhängt, wird zunächst eine effiziente Datenstruktur beschrieben, welche die API des Compositings, beschrieben in Abschnitt 4.2.1, unterstützt.

Datenstruktur

Die Datenstruktur hat fünf Anfragen, die zur API des Compositings passen. Dies sind *insert*, *remove*, $modify$, *mark* und *compose*. Da die CPU-Ausführungszeit t_{CPU} von der Anzahl der ausgeführten Anfragen abhängt, ist die Datenstruktur dafür ausgelegt, die häufigsten Anfragen effizient zu verarbeiten.

Die Datenstruktur, dargestellt in Abbildung 4.8, basiert auf einem Graphen, welcher die Informationen über die Fenster und deren sichtbaren Kacheln speichert. Die Anfragen *insert* und *remove* werden aufgerufen, wenn Fenster erstellt oder entfernt werden, was vergleichsweise selten stattfindet. Speziell im Fahrzeug gilt, dass der Fahrer nicht durch das Infotainmentsystem abgelenkt werden darf. Laut den Richtlinien im Fahrzeug (vgl. [AAM06]) darf die Eingabe, die aus mehreren einzelnen Eingabeschritten bestehen kann, für das Erreichung eines bestimmten Ziels (z. B. das Wechseln des Radiosenders) nicht länger als 20 s dauern. Hierbei darf der Blick des Fahrers nicht länger als zwei Sekunden auf die Anzeige gerichtet sein. In [LFG+13] werden maximal 10 einzelne Eingabeschritte für das Erreichen des Ziels empfohlen. Hiermit schränkt sich die maximale Anzahl an Anfragen für das Erstellen bzw. das Entfernen von Fenstern ein. Dies gilt ebenfalls für die Anfrage $modify$, welche die Position oder die Größe eines Fensters ändert. Im Vergleich dazu werden die Anfragen *mark* und *compose* wesentlich öfters aufgerufen. Die Anfrage *mark* wird jedes Mal aufgerufen, wenn der Inhalt eines Fensters aktualisiert wird. Da dies in der Regel an die Aktualisierungsrate des Bildschirms gekoppelt ist (z. B. 60 Hz), kann folglich jedes Fenster pro Aktualisierung des Bildschirms diese Anfrage einmal senden. Die Anfrage *compose* wird ebenfalls häufig aufgerufen, nämlich wenn mindestens ein Fenster seinen Fensterinhalt aktualisiert hat, d. h. die Anfrage *mark* aufgerufen hat. Da einige Anwendungen wie beispielsweise Tachometer, Navigation, Videodarstellung oder 360°-Kamera dies mit hoher Frequenz tun, ist für *compose* mit einer Frequenz von 60 Hz auszugehen.

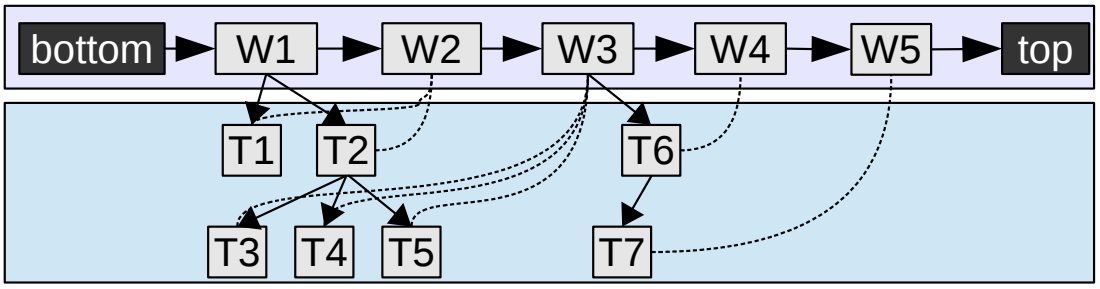

Abbildung 4.8.: Datenstruktur für Fenster, sichtbare Kacheln und Verknüpfungen

Folglich ist die Datenstruktur darauf optimiert die Fenster und Kacheln entsprechend vorzuhalten, dass die Anfragen *mark* und *compose* effizient verarbeitet werden können. Ein Beispiel für die Datenstruktur mit fünf Fenstern ist in Abbildung 4.8 dargestellt. Die sichtbaren Kacheln für das Fenster $W1$ und die Kachel $T2$ ist in der Abbildung 4.9 dargestellt. Für jedes Fenster werden die Informationen über die Position, Breite und Höhe in der Datenstruktur gespeichert. Zusätzlich werden die Fenster, beginnend mit dem untersten Fenster, entsprechend ihren z-Werten, in aufsteigender Folge verknüpft (z. B. das Fenster $W4$ hat einen niedrigeren z-Wert als Fenster $W5$ und liegt in der Verknüpfung folglich vor $W5$). Wenn ein Fenster durch ein anderes Fenster überdeckt wird, dann wird der Algorithmus zur Erzeugung der sichtbaren Kacheln (vgl. Abschnitt 4.4.1) verwendet und die resultierenden Kacheln mit ihrem zugehörigen Fenster verknüpft. Beispielsweise entstehen durch die Überdeckung des Fensters $W1$ durch das Fenster $W2$ die beiden Kacheln $T1$ und $T2$ (vgl. Abbildung 4.9), welche mit $W1$ verknüpft werden. Zusätzlich wird jede Kachel mit dem Fenster verknüpft, welche durch Überdeckung für die Erzeugung der Kachel verantwortlich ist. Diese Verknüpfung stellt die Abhängigkeit zwischen der sichtbaren Kachel und dem überdeckenden Fenster dar und wird deshalb *Abhängigkeitsverknüpfung* genannt. Wie in der Abbildung 4.8 dargestellt, sind die beiden Kacheln $T1$ und $T2$ des Fensters $W1$, welches durch das Fenster $W2$ überdeckt wird, mittels einer Abhängigkeitsverknüpfung (gepunktete Linie) mit dem Fenster $W2$ verbunden. Die

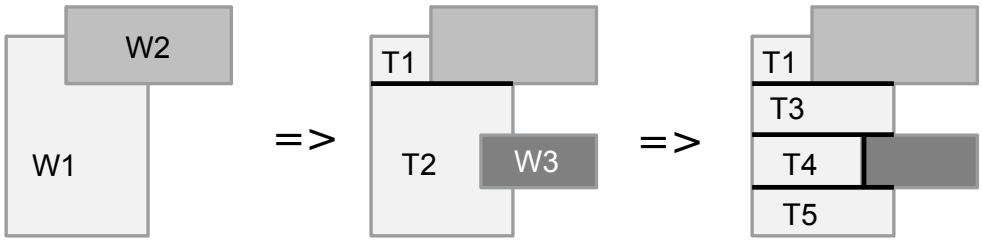

Abbildung 4.9.: Beispiel für die Überdeckung von Fenstern und Kacheln

Abhängigkeitsverknüpfung wird auch im Falle der Überdeckung von Kacheln angewandt. Dies bedeutet, die sichtbaren Kacheln einer überdeckten Kachel werden ebenfalls mittels Abhängigkeitsverknüpfungen mit dem überdeckenden Fenster verknüpft. Ein Beispiel dafür ist die Überdeckung der Kachel $T2$ durch das Fenster $W3$ (vgl. Abbildung 4.9). Hierbei entstehen die drei sichtbaren Kacheln $T3$, $T4$ und $T5$, welche jeweils mit dem Fenster $W3$ verknüpft sind.

Im Folgenden werden die beiden Compositing-Strategien *Hybrid-Compositing* und *Cache-Hybrid-Compositing* im Detail vorgestellt.

Hybrid-Compositing

Die *Hybrid-Compositing*-Strategie soll die Nachteile der beiden Ansätze *fensterbasiertes Compositing* und *kachel-basiertes Compositing* beseitigen, indem zur Laufzeit für jedes Fenster und jede Kachel entschieden wird, ob es schneller ist das Fenster bzw. die Kachel gesamt oder nur die sichtbaren Kacheln mittels Bitblit-Operationen zu kopieren. Jedes Mal, wenn die Anfrage *compose* aufgerufen wird, kopiert der Compositor mittels Bitblit alle Fenster, die zur Aktualisierung markiert wurden. Zu diesem Zweck durchläuft der Algorithmus die Datenstruktur und verwendet die Informationen über die Fenster bzw. Kacheln, um den Inhalt der zugehörigen Bildspeicher in den Anzeigespeicher zu kopieren.

Der Algorithmus der *Hybrid-Compositing*-Strategie berechnet die benötigte Ausführungszeit für ein Fenster und dessen sichtbare Kacheln basierend auf den geschätzten Ausführungszeiten und führt die Bitblit-Operationen mit den ermittelten Fenstern bzw. sichtbaren Kacheln durch. Da diese Optimierung auf der Granularität von Kacheln anstatt auf der Granularität von Fenstern durchgeführt wird, kann die Ausführungszeit für das Bitblitting deutlich verringert werden.

Auflistung 4.4 Algorithmus für das Hybrid-Compositing

```
 1: function compose_hybrid (graph, marks)
 2:   win = graph.bottom
 3:   while (win)
 4:     if(marks[win.id])
 5:       bitblit_hybrid(win, marks, win.fb)
 6:     end if
 7:     win = win.next
 8:   end while
 9: end function
10:
11: function bitblit_hybrid (n, marks, fb)
```

```
12:   if (n.tiles && t_full(n) > t_tiled(n))
13:     for (each t in n.tiles)
14:       bitblit_hybrid(t, marks, fb)
15:     end for
16:   else
17:     bitblit(n.x, n.y, n.w, n.h, fb)
18:     mark_dependencies(win.id, marks)
19:   end if
20: end function
21:
22: function t_full (n)
23:   time_full = predicted_time (n.size)
24:   for (each d in n.dependencies)
25:     time_full += min(t_full(d), t_tiled(d))
26:   end for
27:   return time_full
28: end function
29:
30: function t_tile (n)
31:   time_tile = 0
32:   if(n.visible)
33:     time_tile = predicted_time (n.size)
34:   else
35:     for (each t in n.tiles)
36:       time_tile += min(t_full(t), t_tiled(t))
37:     end for
38:   end if
39:   return time_tile
40: end function
```

Der Algorithmus für das *Hybrid-Compositing* ist in Auflistung 4.4 dargestellt. Die Funktion *compose_hybrid* durchläuft die Liste der Fenster, gespeichert in *graph* und überprüft, ob ein Fenster zur Aktualisierung markiert wurde (vgl. Zeile 3 in Auflistung 4.4). Für jedes Fenster, welches zur Aktualisierung markiert wurde, wird die Funktion *bitblit_hybrid* aufgerufen (vgl. Zeile 5). Die Funktion *bitblit_hybrid* überprüft, ob ein anderes Fenster das markierte Fenster überdeckt und ob die geschätzte Ausführungszeit für das Bitblitting unter Verwendung des *fenster-basierten Compositings* höher ist als unter Verwendung des *kachel-basierten Compositings* (vgl. Zeile 12). Trifft dies zu, dann wird die Funktion

bitblit_hybrid rekursiv für alle sichtbaren Kacheln des markierten Fensters (vgl. Zeile 14) aufgerufen. Ansonsten führt der Algorithmus das Bitblitting für das markierte Fenster und alle überdeckenden Fenster durch (vgl. Zeile 17 und 18). Die Funktionen t_full und t_tile werden für die Abschätzung der benötigen Ausführungszeit für das Bitblitting mittels *fenster-basiertem Compositing* und *kachel-basiertem Compositing* benötigt. In Funktion t_full wird die Ausführungszeit für das Bitblitting eines Fensters (oder einer Kachel) n abgeschätzt und als Rückgabewert der Funktion zurückgegeben. Hierbei wird die Funktion $predicted\_time$ aufgerufen, welche ein Modell zur Abschätzung der Ausführungszeit verwendet (siehe Abschnitt 4.5.2), um die abgeschätzte Zeit, abhängig der Größe und Anzahl der Fenster, zu liefern (vgl. Zeile 21). Dann wird für jedes Fenster, welches das Fenster (oder die Kachel) n überdeckt, unter Anwendung der Abhängigkeitsverknüpfungen die Ausführungszeit für das Bitblitting mittels *fenster-basiertem Compositing* und *kachel-basiertem Compositing* berechnet und die kleinere Ausführungszeit zu der Variable $time\_full$ addiert (vgl. Zeile 25). Schließlich wird die Summe aller minimalen Ausführungszeiten als Rückgabewert der Funktion zurückgegeben (vgl. Zeile 27). Die Funktion t_tile ist ähnlich zu der Funktion t_full, jedoch wird zusätzlich überprüft, ob entweder ein Fenster oder eine Kachel sichtbar ist oder nicht. Wenn eine Kachel sichtbar ist, dann wird die geschätzte Ausführungszeit zurückgegeben (vgl. Zeile 33 und 39). Ansonsten wird für jede sichtbare Kachel die Ausführungszeit für *fenster-basiertes Compositing* und *kachel-basiertes Compositing* berechnet und die kleinere Ausführungszeit zu der Variable $time\_tile$ (vgl. Zeile 36) addiert, welche schließlich als Rückgabewert zurückgegeben wird.

Cache-Hybrid-Compositing

Das *Hybrid-Compositing*, kann im Vergleich zu den beiden anderen Strategien aufgrund der höheren Ausführungszeit t_{CPU} auf der CPU in manchen Fällen schlechter sein. Unter Verwendung eines Caches kann die Ausführungszeit t_{CPU} reduziert werden. Unter Anwendung einer Caching-Strategie, welche die optimalen Kombinationen aus Fenstern und Kacheln speichert, kann das *Hybrid-Compositing* optimiert werden. Dieser Ansatz wird *Cache-Hybrid-Compositing* genannt und reduziert die benötige Ausführungszeit t_{CPU} auf der CPU, wodurch ebenfalls die Ausführungszeit t_{comp} für das Compositing reduziert wird. Für die Verwendung eines Caches wird angenommen, dass die Position und die Größe der Fenster sich selten ändert. Dadurch können bereits berechnete Sequenzen aus Fenstern

und Kacheln mehrmals verwendet werden. Insbesondere im Falle von Szenarien, wie sie im Fahrzeug bei den Kombiinstrumenten vorkommen können, ist diese Annahme gültig. Auf der anderen Seite kann die Aktualisierungsrate der Fenster unterschiedlich sein. Deshalb wird für den Cache eine Hashtabelle verwendet, welche die zur Aktualisierung markierten Fenster als Schlüssel für das Auffinden der bereits berechneten Sequenzen aus Fenstern und Kacheln nutzt. In dem seltenen Fall, dass die Fenster ihre Position oder Größe ändern, wird der Cache invalidiert.

Auflistung 4.5 Algorithmus für das Cache-Hybrid-Compositing

```
 1: function compose_cache_hybrid (graph, marks)
 2:   key = HASH(marks)
 3:   if(match(cache[key], marks))
 4:     for (each r in cache[key])
 5:       bitblit(r.x, r.y, r.w, r.h, r.fb)
 6:     end for
 7:   else
 8:     cache_enable(key, marks)
 9:     compose_hybrid(graph, marks)
10:     cache_disable()
11:   end if
12: end function
```

In Auflistung 4.5 ist der Algorithmus für das *Cache-Hybrid-Compositing* abgebildet. Er berechnet die Hashwerte aus den Feldern der zur Aktualisierung markierten Fenster und überprüft, ob für die markierten Fenster bereits im Cache eine Sequenz gespeichert wurde oder nicht (vgl. Zeile 3 in Auflistung 4.5). Falls für die markierten Fenster bereits eine optimale Sequenz aus Fenstern und Kacheln im Cache gespeichert wurde, dann wird diese Sequenz für das Bitblitting verwendet (vgl. Zeile 5). Ansonsten wird der Cache aktiviert (vgl. Zeile 8). Hierzu erstellt der Algorithmus einen neuen Eintrag in der Hashtabelle und aktiviert das Aufzeichnen der Bitblit-Operationen, so dass jedes Fenster bzw. jede Kachel, welche mittels Bitblitting kopiert wird, dem neuen Eintrag hinzugefügt wird. Um die Größe des Caches niedrig zu halten, wird dabei nur eine Verknüpfung auf das Fenster bzw. auf die Kachel in der Datenstruktur gespeichert. Für das Bitblitting wird der Algorithmus des *Hybrid-Compositings* (siehe Auflistung 4.4) verwendet. Schließlich wird der Cache wieder deaktiviert und der Aufzeichnungsmodus beendet. Der neue Cacheeintrag steht somit für zukünftige Anfragen zur Verfügung.

4.5. Evaluation der Compositing-Strategien für rechteckige Fenster

In diesem Abschnitt wird die Evaluation der Compositing-Strategien für rechteckige Fenster, basierend auf der Compositing-Architektur aus Abschnitt 4.3, beschrieben. Zuerst wird die verwendete Plattform- und Messanordnung, sowie die Szenarien beschrieben. Anschließend wird die Auswertung des Modells zur Abschätzung der Ausführungszeit des Bitblittings und dessen Kalibrierung dargestellt. Dann wird die Evaluation der verschiedenen Compositing-Strategien präsentiert und diskutiert. Hierzu wird ein Vergleich der beiden Strategien *Hybrid-Compositing* und *Cache-Hybrid-Compositing* mit den beiden Strategien *fensterbasiertes Compositing* und *kachel-basiertes Compositing* durchgeführt. Zusätzlich wird für das *Cache-Hybrid-Compositing* eine Auswertung für eine optimale Berechnung der sichtbaren Kacheln im Vergleich zu dem Algorithmus in Auflistung 4.3 dargestellt.

4.5.1. Plattform und Evaluationsumgebung

Für die Evaluation wurde die eingebettete Plattform i.MX6 SABRE Automotive von Freescale verwendet, die für das Infotainment im Fahrzeug ausgelegt ist und vier CPU-Kerne aufweist. Die Plattform bietet drei verschiedene GPUs von Vivante [Fre15]. Die 3D-GPU GC2000 unterstützt die Grafikschnittstelle OpenGL ES 2.0 [Ope16b]. Die GPU GC320 ist ausgelegt für das Compositing und wird zu diesem Zweck in der Evaluation verwendet. Die 2D-Grafikschnittstelle OpenVG [Ope16c] wird mit der GPU GC355 unterstützt. Für das Bitblitting wurde der Vivante Treiber für den *Bildspeicher* mit dem Befehl „$gco2D\_Blit$“ verwendet, welcher dem Befehl $2D\_Blit$ entspricht. Für die Messungen der Ausführungszeit der Compositing-Strategien wurde die POSIX-Funktion $clock\_gettime$ verwendet. Die Ausführungszeit t_{comp} für eine Compositing-Strategie besteht aus der Ausführungszeit auf der CPU, t_{CPU}, und der Ausführungszeit des Bitblittings, t_{BB}, die von der GPU benötigt wird. Um die Anteile dieser beiden Zeiten besser darzustellen, wurden diese Ausführungszeiten separat gemessen.

Zur Messung der Ausführungszeit des Compositings wurde auf der Plattform (siehe Abbildung 4.10) der Linuxkern 3.10.17 betrieben, welcher vollständig präemptiv ist. Hierbei wurde die Echtzeitpriorität der Testanwendung auf 99 gesetzt, um eine präzise Messung zu ermöglichen. Alle anderen Prozesse behalten die Priorität 0, sodass die Testanwendung die höchste Priorität hat. Zusätzlich wurden

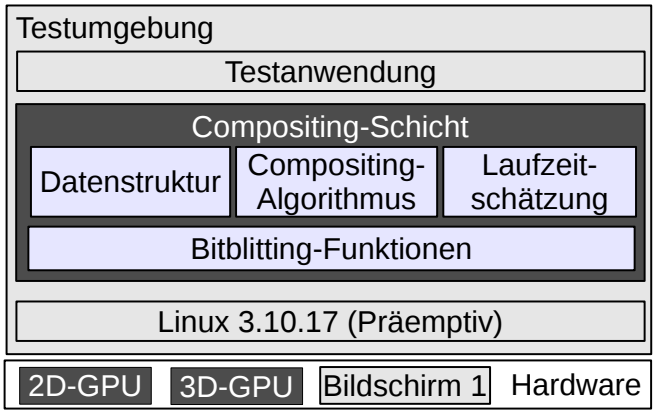

Abbildung 4.10.: Testarchitektur für das Compositing unter Verwendung eines vollständig präemptiven Linuxkerns

sowohl der Thread für das Compositing als auch die Threads des GPU-Treibers im Kern auf denselben CPU-Kern gelegt, sodass keine zusätzliche Zeit für Wechsel zwischen den CPU-Kernen in die Messungen einfließt. Die Compositing-Schicht stellt die Datenstruktur für die Speicherung der Informationen der Kacheln und Fenster zur Verfügung. Des Weiteren beinhaltet sie die Compositing-Algorithmen, welche gemessen werden sollen, und die Komponente Laufzeitschätzung für das Abschätzen der Ausführungszeit des Compositings.

Für eine Evaluation des Compositings in der Zielarchitektur aus Abbildung 2.2, beschrieben in Abschnitt 2.2, wurde eine entsprechende Testumgebung mit 3D-Anwendungen aufgebaut. Hierzu wurde als Virtualisierungslösung PikeOS [Pik15] von Sysgo verwendet und zwei Partitionen für jeweils den Virtualisierungsmanager und die Testanwendungen aufgebaut (siehe Abbildung 4.11).

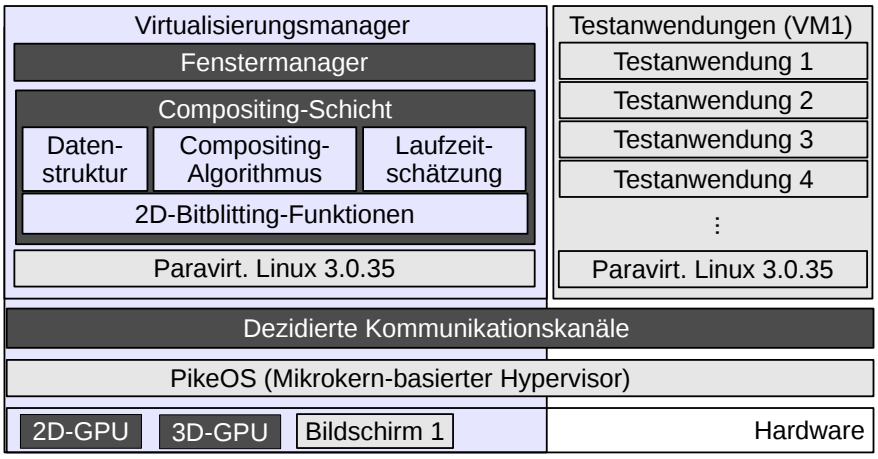

Abbildung 4.11.: Testarchitektur für das Compositing in einer virtualisierten Umgebung

Die Testanwendungen kommunizieren mit dem Fenstermanager und registrieren ihre Fenster. Sobald sie für ihr Fenster eine ID erhalten, können die Testanwendungen einen 3D-Kontext erstellen. Hierzu werden alle EGL-Befehle an den Virtualisierungsmanager gesendet, der diese an die GPU weiterleitet. Sobald ein gültiger Kontext vorliegt, senden die Testanwendungen die OpenGL ES 2.0-Befehle ebenfalls an den Virtualisierungsmanager. Die OpenGL ES 2.0-Befehle werden dann von der GPU ausgeführt und das resultierende Bild in den Bildspeicher der entsprechenden Testanwendung geschrieben. Dann wird das entsprechende Fenster zur Aktualisierung markiert. Das Compositing wird mit der Bildwiederholrate der Anzeige synchronisiert, die 60 Hz beträgt, und führt das Bitblitting der markierten Fenster mittels des 2D-Befehle „$gco2D\_Blit$“ des Vivante Treibers durch. Zusätzlich zu der Ausführungszeit des Bitblittings wird die Anzahl der Bilder pro Sekunde der einzelnen Testanwendungen gemessen.

4.5.2. Kalibrierung des Abschätzungsmodells für die Ausführungszeit

Für die Berechnung einer optimalen Kombination an Bitblitting-Operationen wird ein Modell zur Abschätzung der Ausführungszeit des Bitblittings, das für die gegebene Hardwareplattform kalibriert werden muss, benötigt. Dazu wurden für eine große Menge an verschiedenen Rechtecksgrößen und einer großen Anzahl an Bitblitting-Operationen Messungen durchgeführt. Wie in Kapitel 4.2 beschrieben, hängt die Ausführungszeit des Bitblittings t_{BB} von der Anzahl der Bitblitting-Operationen und der Größe der Bildspeicher ($w_{fb} \times h_{fb}$) ab. Hierzu wurden für eine große Anzahl an Bildspeichern $n{=}|FB|$ und vielen Kombinationen an Fensterdimensionen mit Breite w_{fb} und Höhe h_{fb} die Ausführungszeit t_{BB} gemessen.

In Abbildung 4.12 ist die Ausführungszeit t_{BB} abhängig der Anzahl der Bitblitting-Operationen n und der Fensterdimensionen mit $w_{fb} \times h_{fb}$ dargestellt. Jeder Punkt ist ein gemittelter Wert aus 100 Messungen. Unter Verwendung von *Octave* [GNU15], eine Programmiersprache zum Lösen numerischer Probleme, wurde eine *bilineare Funktion* mittels der Methode der kleinsten Quadrate berechnet und diese als Abschätzungsfunktion Eq.1 für das Bitblitting abgeleitet.

$$t_{BB} = a + \sum_{fb \in FB} b + c * w_{fb} + d * h_{fb} + e * w_{fb} * h_{fb} \qquad \text{(Eq.1)}$$

Für die Parameter wurden die folgenden Werte ermittelt: $a{=}67{,}2\,\mu$s, $b{=}9{,}03\,\mu$s, $c{=}1{,}29 \times 10^{-4}\,\mu$s, $d{=}6{,}71 \times 10^{-4}\,\mu$s und $e{=}1{,}67 \times 10^{-3}\,\mu$s.

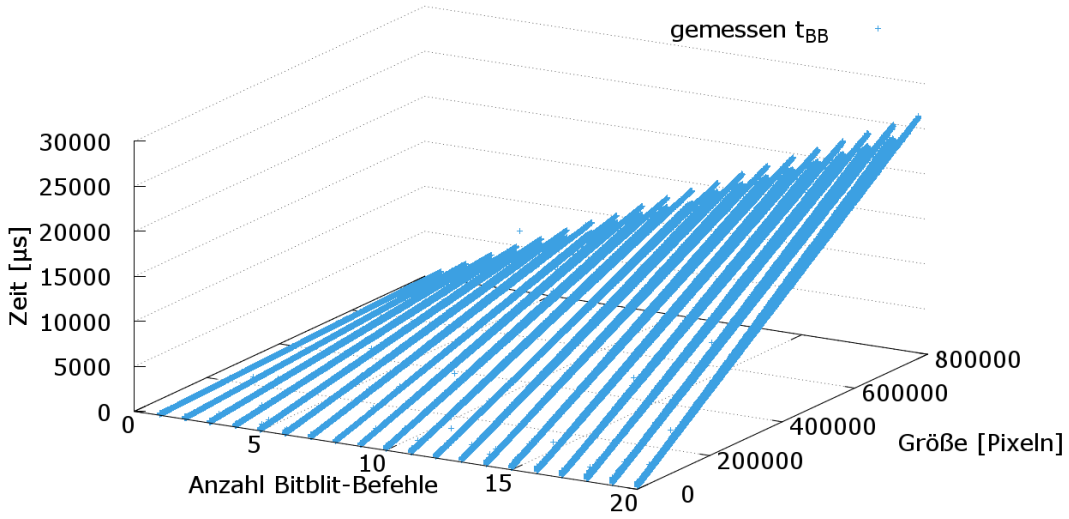

Abbildung 4.12.: Ausführungszeit für das Bitblitting für verschiedene Operationen

Obwohl in Abbildung 4.12 die Fensterdimensionen $w_{fb} \times h_{fb}$ als kombinierter Wert dargestellt sind, werden in der Abschätzungsfunktion Eq.1 die Parameter für Breite w_{fb} und Höhe h_{fb} als unabhängige Terme betrachtet, was die Genauigkeit für die Abschätzung erhöht. Im Vergleich zu der Abschätzungsfunktion weichen die gemessenen Werte nach Bereinigung von Messfehlern im Schnitt nur um 1,9 % ab, womit der Fehler bei der Abschätzung sehr gering gehalten werden kann.

4.5.3. Szenarien

Um die Leistung der verschiedenen Compositing-Strategien vergleichen zu können, wurden verschiedene Szenarien definiert. Jedes Szenario besteht aus einer Menge an Fenstern, die als Rechtecke durch Breite, Höhe und der Position mit den Koordinaten (x, y, z) beschrieben werden. Dadurch können Fenster andere Fenster teilweise oder vollständig überdecken. Des Weiteren wurden Fenster zur Aktualisierung markiert, die mittels Bitblitting in den Anzeigespeicher kopiert wurden. Für die Evaluation wurde ausgewertet, wie sich die Leistung der Compositing-Strategien verhält, wenn immer alle Fenster markiert wurden. Dies bedeutet, dass die Anzahl an Fenstern bzw. Kacheln, die mittels des Compositings in den Anzeigespeicher kopiert werden immer gleich ist. In realistischen Szenarien sind jedoch die Aktualisierungsraten der Anwendungen – diese beeinflusst

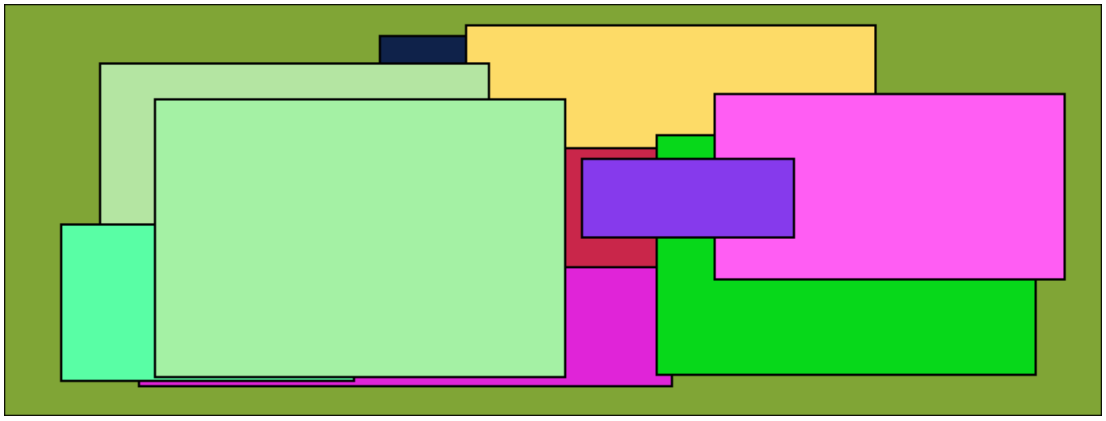

Abbildung 4.13.: Beispiel für die Fensterverteilung in einem *Zufallsszenario*

direkt die Ausführungszeit des Bitblittings – häufig nicht gleich. Dies kann der Fall sein, wenn Anwendungen ihre Fensterbereiche nicht aktualisieren müssen, da keine Änderungen vorliegen oder die Berechnung des grafischen Inhaltes, z. B. auf der GPU, nicht für alle Anwendungen gleich schnell durchgeführt werden kann. Zu diesem Zweck wurde zusätzlich der Einfluss der Aktualisierungsrate der Fenster unter Verwendung eines zufällig gewählten Intervalls für die Markierung der Fenster untersucht. Das heißt, aus den möglichen Aktualisierungsraten von 10, 15, 30, 45, 50 und 60 Bilder pro Sekunde wurde jedem Fenster für eine bestimmte Periode ein Wert zugeordnet und entsprechend zur Aktualisierung markiert.

Es wurden zwei Arten an Szenarien definiert. Zum einen *Zufallsszenarien* und zum anderen ein fest definiertes Szenario, genannt *Fahrzeugszenario*, welches realitätsnah ein dynamisches Kombiinstrument darstellen soll. Um ein *Zufallsszenario* zu erstellen, wie in Abbildung 4.13 dargestellt, wurde ein Generator implementiert, der zufällig die Breite im Bereich zwischen 10 bis 1440 Pixel und die Höhe im Bereich zwischen 10 bis 540 Pixel wählt. Zusätzlich wurde die Position des Fensters gewählt, die jedoch eingeschränkt wurde, sodass das Fenster vollständig im Bereich der Bildschirmauflösung mit 1440 auf 540 liegt.

Das *Fahrzeugszenario* ist in Abbildung 4.14 dargestellt und repräsentiert Anwendungen, wie sie typischerweise in einem Kombiinstrument verwendet werden, z. B. Geschwindigkeitsanzeige, Drehzahlmesser, Warnlampen, Warnmeldungen.

4.5.4. Evaluation der Compositing-Strategien

In diesem Abschnitt werden die Messungen und Auswertungen der vier Compositing-Strategien beschrieben. Für die Messungen wurde der Messaufbau und die Szenarien aus Abschnitt 4.5.1 und Abschnitt 4.5.3 verwendet.

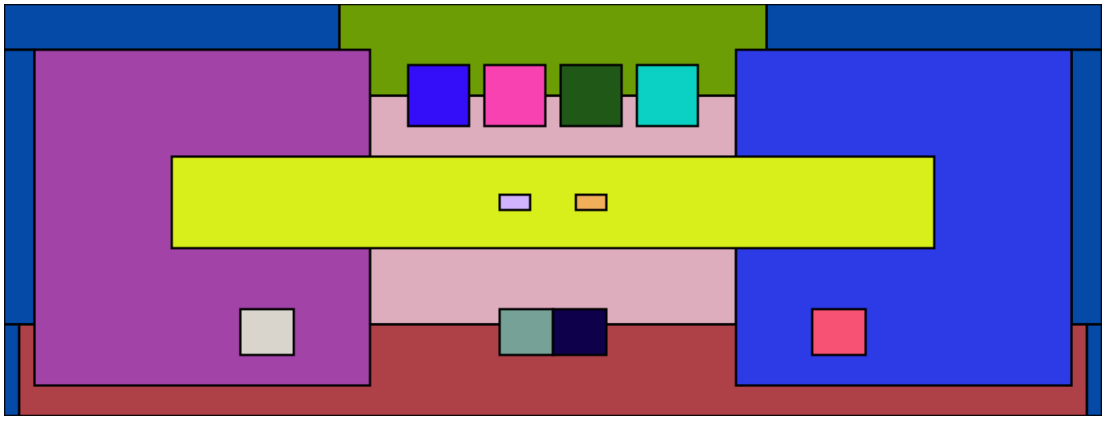

Abbildung 4.14.: Fensterverteilung in einem *Fahrzeugszenario*

Ausführungszeit des Compositings für Fenster mit fester Aktualisierungsrate

Für einen Vergleich der Compositing-Strategien wurden die Ausführungszeiten des Compositings von mehreren Fenstern mit den Strategien *fenster-basiertes Compositing*, *kachel-basiertes Compositing*, *Hybrid-Compositing* und *Cache-Hybrid-Compositing* gemessen. Hierzu wurden jeweils die *Zufallsszenarien* (siehe Abschnitt 4.5.3) und das *Fahrzeugszenario* (siehe Abschnitt 4.5.3) angewandt.

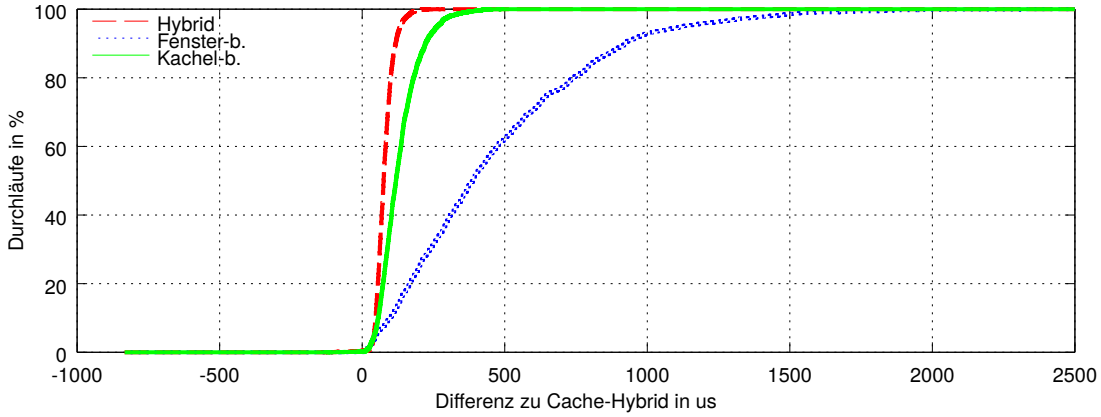

Abbildung 4.15.: Differenz der Ausführungszeit t_{comp} für **Zufallsszenarien** mit immer alle Fenster markiert und 20000 Durchläufen

Zuerst wurden 1000 *Zufallsszenarien* mit 10 bis 17 verschiedenen Fenstern verwendet. Für jede der vier Compositing-Strategien wurden dann 20 Durchläufe mit jeweils 120 Bildern je *Zufallsszenario* und unterschiedlicher Anzahl an Fenstern gemessen, sowie der Durchschnitt für 120 Bilder der Ausführungszeit für das Compositing t_{comp} berechnet.

Die Ergebnisse sind in Form einer kumulativen Verteilungsfunktion in Abbil-

dung 4.15 dargestellt. Die y-Achse repräsentiert die Anzahl der Durchläufe in %, in diesem Fall sind dies 20000 Durchläufe aufgrund der 1000 Zufallsszenarien mit je 20 Durchläufen, und die x-Achse stellt die Differenz der Ausführungszeiten t_{comp} der verschiedenen Compositing-Strategien in μs dar. Die drei Kurven stellen die Differenzen der Ausführungszeiten t_{comp} der drei Strategien *fenster-basiertes Compositing* (blau-gepunktete Kurve), *kachel-basiertes Compositing* (grüne Kurve) und *Hybrid-Compositing* (rot-gestrichelte Kurve) im Vergleich zu der Ausführungszeit für die Strategie *Cache-Hybrid-Compositing* dar. Wie in der Abbildung 4.15 ersichtlich, ist in der überwiegenden Anzahl der Fälle das *Cache-Hybrid-Compositing* effizienter im Ausführen des Compositings für die Szenarien als die anderen Strategien. Die Reduzierung der Ausführungszeit t_{comp}, welche in den 1000 Szenarien unter Verwendung des *Cache-Hybrid-Compositings* im Vergleich zu dem *fenster-basierten Compositing* erreicht werden kann, ist im Durchschnitt 27 %. Dies entspricht einer durchschnittlich 467 μs geringeren Ausführungszeit t_{comp}, was eine deutliche Reduzierung darstellt. Im Vergleich zu dem *kachel-basierten Compositing* kann die Ausführungszeit t_{comp} im Durchschnitt um 9 % reduziert werden.

Das *Cache-Hybrid-Compositing* reduziert primär die Ausführungszeit t_{CPU} des Compositings, welche für das Berechnen der optimalen Sequenz der Bitblitting-Operationen benötigt wird. In diesem Szenario kann damit eine Verbesserung von 5 % gegenüber dem *Hybrid-Compositing* erreicht werden.

In einer weiteren Auswertung wurde das *Fahrzeugszenario* für 10000 Durchläufe ausgeführt. Hierbei wurden die 17 verwendeten Fenster immer alle zur Aktualisierung markiert, sodass alle die gleiche Aktualisierungsrate hatten. Die durchschnittliche Ausführungszeit für die vier Strategien ist in Abbildung 4.16 dargestellt. Die x-Achse stellt die vier Strategien dar und die y-Achse repräsentiert die Ausführungszeit der einzelnen Compositing-Strategien in μs.

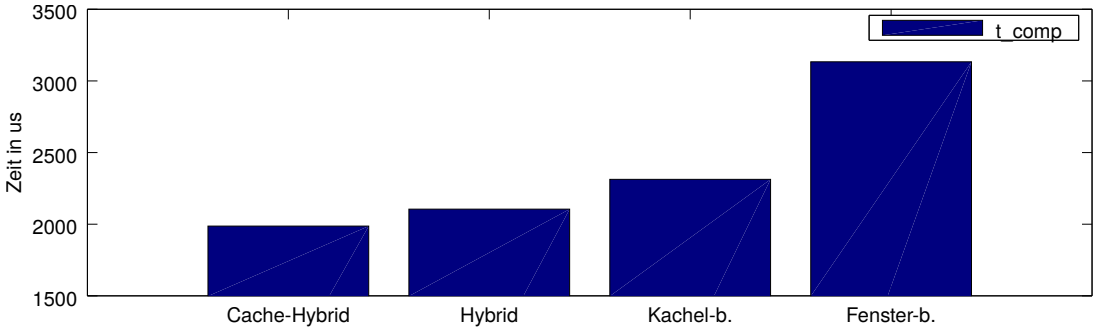

Abbildung 4.16.: Ausführungszeit t_{comp} für das **Fahrzeugszenario** mit immer alle Fenster markiert und 10000 Durchläufen

Das *Hybrid-Compositing* ist hierbei im Durchschnitt 34 % (bzw. 1117 μs) schneller als das *fenster-basierte Compositing* und 17 % (bzw. 474 μs) schneller als das *kachel-basierte Compositing*. Im Vergleich zu dem *Hybrid-Compositing* kann durch das *Cache-Hybrid-Compositing* eine Verbesserung der Ausführungszeit von 6 % im Durchschnitt erreicht werden.

Ausführungszeit des Compositings für Fenster mit zufälliger Aktualisierungsrate

In weiteren Evaluationen wurde die Aktualisierungsrate der Fenster mittels eines Intervalls zufällig gewählt (vgl. Abschnitt 4.5.1). Hierzu wurden 1000 *Zufallsszenarien* mit 10 bis 17 Fenstern für je 2400 Bilder ausgeführt und ausgewertet. Zusätzlich wurden 1000 Durchläufe für das *Fahrzeugszenario* mit jeweils 2400 Bildern durchgeführt und ebenfalls ausgewertet. Dabei wurden die markierten Fenster nach jeweils 120 Bildern mittels einem zufällig gewählten Intervall neu markiert und der Durchschnitt der Ausführungszeit t_{comp} für diese 120 Bilder berechnet. Die Ergebnisse sind in Abbildung 4.17 und 4.18 in Form einer kumulativen Verteilungsfunktion dargestellt. Ähnlich zu Abbildung 4.15 repräsentiert die y-Achse die Anzahl der Durchläufe in % und die x-Achse stellt die Differenz der Ausführungszeit des *Cache-Hybrid-Compositings* zu den drei Strategien *fenster-basiertes Compositing*, *kachel-basiertes Compositing* und *Hybrid-Compositing* in μs dar.

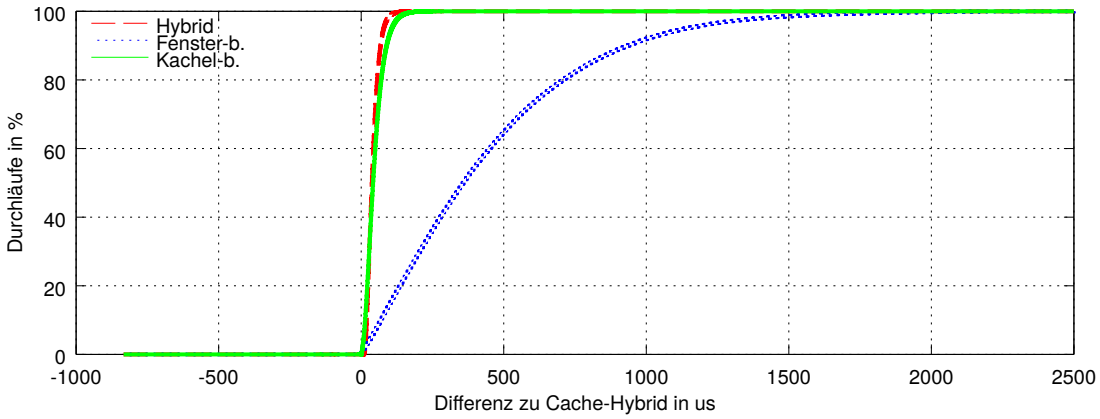

Abbildung 4.17.: Differenz der Ausführungszeit t_{comp} für **Zufallsszenarien** mit zufällig gewählter Aktualisierungsrate und 160000 Durchläufen

Wie in Abbildung 4.17 dargestellt, ist das *Cache-Hybrid-Compositing* deutlich schneller als das *fenster-basierte Compositing*. Dies bedeutet im Durchschnitt eine Optimierung von 42 % (bzw. 635 μs). Die Ausführungszeit des *Hybrid-Compositings*

ist in den meisten Fällen besser oder gleich der Ausführungszeit des *kachel-basierten Compositings*. Im Durchschnitt wurde eine Verbesserung um 1 % erreicht. Dies liegt in erster Linie an der höheren CPU-Ausführungszeit des *Hybrid-Compositings* im Vergleich zu dem *kachel-basierten Compositing*. Dieser Nachteil wird durch das *Cache-Hybrid-Compositing* jedoch ausgeglichen und somit kann eine Optimierung von 6 % im Vergleich zu dem *kachel-basierten Compositing* erreicht werden.

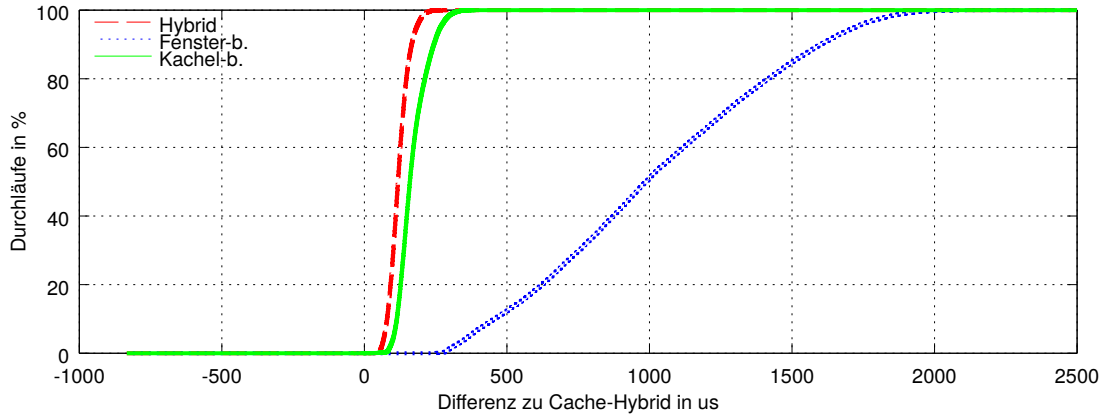

Abbildung 4.18.: Differenz der Ausführungszeit t_{comp} für das **Fahrzeugszenario** mit zufällig gewählter Aktualisierungsrate und 20000 Durchläufen

Wie in Abbildung 4.18 dargestellt, ist das *kachel-basierte Compositing* (gestrichelte blaue Kurve) im *Fahrzeugszenario* weniger effizient als die anderen Strategien. Die Ausführungszeit für das *kachel-basierte Compositing* ist im Durchschnitt 1017 μs (ungefähr 41 %) höher als beim *Cache-Hybrid-Compositing*. Ähnlich wie bei den *Zufallsszenarien* ist die Ausführungszeit des *Hybrid-Compositings* in allen Fällen besser oder gleich der Ausführungszeit des *kachel-basierten Compositings*.

Jedoch ist das *Hybrid-Compositing* im Durchschnitt nur 3 % schneller als das *kachel-basierte Compositing*. Dies liegt in erster Linie daran, dass das *Hybrid-Compositing* mehr Ausführungszeit auf der CPU benötigt als das *kachel-basierte Compositing*. Dies kann dazu führen, dass in manchen Fällen die Ausführungszeit für das *Hybrid-Compositing* höher ist als die des *kachel-basierten Compositings*. Dieser Nachteil wird mittels des *Cache-Hybrid-Compositings* aufgehoben, wodurch im Durchschnitt eine Verbesserung der Ausführungszeit um 7 % im Vergleich zu dem *kachel-basierten Compositing* erreicht wird.

Optimale Aufteilung der Fenster in Kacheln

Die Ausführungszeit des *Hybrid-Compositings* hängt von dem Algorithmus in Auflistung 4.3 ab, welcher die sichtbaren Kacheln erzeugt.

Während der Algorithmus sehr schnell in der Berechnung der sichtbaren Kacheln ist, kann es jedoch in manchen Fällen zu einer sub-optimalen Aufteilung der Fenster in Kacheln kommen, da nicht alle Kombinationen an möglichen Kacheln berücksichtigt werden. Beispielsweise startet der Algorithmus bei der Traversierung immer mit dem obersten Rechteck und ignoriert als Startpunkt die möglichen anderen Rechtecke. Dies kann zu Fällen führen, in welchen z. B. eine Kachel durch ein Fenster vollständig verdeckt wird, wenn die Aufteilung in die sichtbaren Kacheln in einer anderen Reihenfolge ermittelt wird. Dadurch kann sich die Ausführungszeit des Bitblittings reduzieren. Dazu werden alle möglichen Kombinationen für eine Aufteilung in Kacheln berechnet, um die optimale Kombination zu ermitteln. Die Anzahl an Kombinationen hängt dabei exponentiell von der Anzahl an überdeckenden Fenstern ab. Beispielsweise ist die maximale Anzahl an Kombinationen für zwei überdeckende Rechtecke die Permutation der vier möglichen Schnittkanten (vgl. M_0 Fall in Abbildung 4.3). Dies führt zu 24 verschiedenen Fällen der Aufteilung und somit zu 16 Kombinationen an unterschiedlichen Kacheln. Für die Evaluation wurde für verschiedene Szenarien die optimale Kombination an Kacheln ermittelt und die abgeschätzte Ausführungszeit für das Bitblitting mit der des Algorithmus in Auflistung 4.3 verglichen.

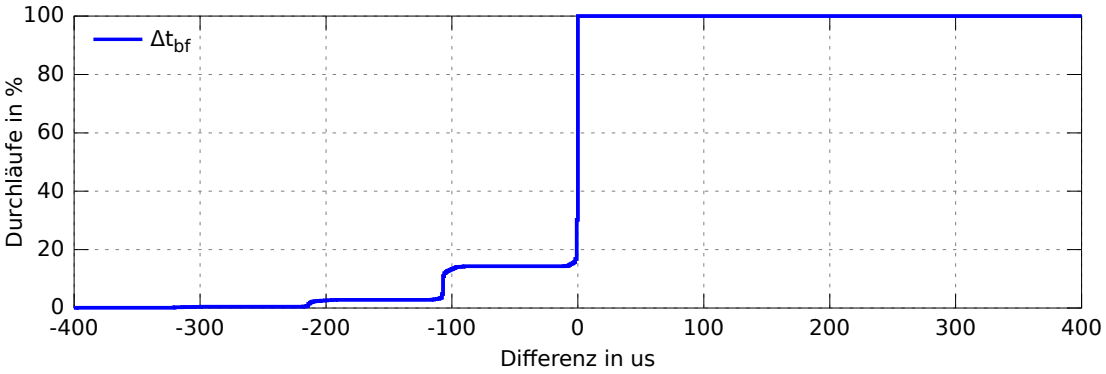

Abbildung 4.19.: Differenz zur Ausführungszeit t_{BB} zwischen der optimalen Kombination an Kacheln und der mittels Algorithmus in Auflistung 4.3 berechneten Kombination mit 6000 Durchläufen

Die Evaluation wurde für zwei bis sieben Fenster mit jeweils 1000 zufällig gewählten Szenarien durchgeführt. Es wurde dann die Ausführungszeit gemessen, welche sich sowohl für das Bitblitting mit optimaler Kachelaufteilung (t_{bf}), als

auch die durch den Algorithmus berechnete Kachelaufteilung (t_{ta}) ergab. Die Ergebnisse sind in Abbildung 4.19 dargestellt. Die y-Achse repräsentiert die Anzahl der Durchläufe in % und die x-Achse stellt die Differenz der minimalen Ausführungszeit des Bitblittings t_{bf} zu der Ausführungszeit t_{ta} in μs als kumulative Verteilungsfunktion dar. Dies bedeutet, die blaue Kurve ist die zeitliche Differenz $\Delta t_{bf} = t_{ta} - t_{bf}$. Wie in Abbildung 4.19 zu sehen ist, ist in etwa 17 % der Durchläufe die berechnete Aufteilung der Kacheln mittels des Algorithmus in Auflistung 4.3 nicht optimal. Das bedeutet, durch die ermittelte Aufteilung der Fenster in sichtbare Kacheln hat der Algorithmus sub-optimale Kombinationen erzeugt, welche eine höhere Ausführungszeit für das Bitblitting zur Folge hatten. Jedoch ist im Durchschnitt die Ausführungszeit für das Bitblitting nur um ca. 1 % gestiegen. Andererseits ist der Berechnungsaufwand für die Ermittlung der optimalen Kachelaufteilung um mehrere Größenordnungen höher als der Algorithmus in Auflistung 4.3 benötigt. Beispielsweise benötigt die Berechnung der optimalen Kachelaufteilung von fünf Fenstern 10^6 mal länger als die Berechnung mittels des Algorithmus.

4.5.5. CPU-Ausführungszeit

Da der Berechnungsaufwand des *Hybrid-Compositings* auf der CPU höher ist als im Vergleich zu den beiden Ansätzen *fenster-basiertes Compositing* und *kachel-basiertes Compositing* wurde für alle Strategien die Ausführungsdauer auf der CPU unter verschiedenen Szenarien evaluiert. Hierzu wurden das *Fahrzeugszenario* (*Auto_X* und *Auto_C*) und die *Zufallsszenarien* (*Rand_X* und *Rand_C*) aus Abschnitt 4.5.3 verwendet. Des Weiteren wurden die Szenarien jeweils mit der gleichen Aktualisierungsrate, d. h. es wurden immer alle Fenster markiert (*Auto_C* und *Rand_C*), als auch mit einer zufällig gewählten Aktualisierungsrate, d. h. die Fenster werden zufällig mittels einem Intervall markiert (*Rand_X* und *Rand_X*), ausgewertet. In Abbildung 4.20 ist die durchschnittliche Ausführungszeit auf der CPU von 1000000 Durchläufen dargestellt.

Wie erwartet, benötigt das *fenster-basierte Compositing* die geringste Ausführungszeit auf der CPU im Vergleich zu den anderen Strategien, wie in Abbildung 4.20 zu sehen ist. Dies liegt in erster Linie daran, dass nur die Liste der Fenster traversiert werden muss, um das Bitblitting durchzuführen. Im Falle des *kachel-basierten Compositings* muss zusätzlich die Liste der Kacheln pro Fenster traversiert werden, was entsprechend mehr Ausführungszeit auf der CPU benötigt.

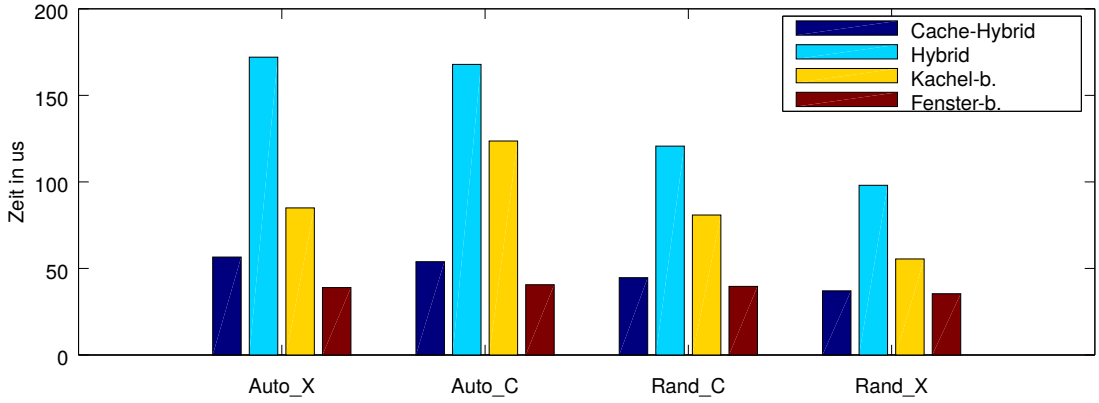

Abbildung 4.20.: Ausführungszeit der vier Strategien auf der CPU

Da das *Hybrid-Compositing* aufgrund der höheren Ausführungszeit t_{CPU} auf der CPU in manchen Fällen schlechter sein kann als die anderen Strategien, senkt die Verwendung eines Caches die Ausführungszeit auf der CPU deutlich. Somit kann die Effizienz auf das Niveau des *fenster-basierten Compositings* und des *kachel-basierten Compositings* gebracht werden, wie in der Abbildung 4.20 im Falle des *Cache-Hybrid-Compositings* zu sehen ist.

4.5.6. Leistungsvergleich mit 3D-Anwendungen

Nachdem die Strategien bzgl. der Ausführungszeiten im Detail evaluiert wurden, wird in der folgenden Evaluation ein Leistungsvergleich auf Basis der Aktualisierungsraten von 3D-Anwendungen aufgezeigt. Zu diesem Zweck wurde für den Vergleich der Compositing-Strategien mit Anwendungen, welche mittels OpenGL ES 2.0 und EGL 3D-Inhalte in ihre Bildspeicher ablegen, das *Fahrzeugszenario* (vgl. Kapitel 4.5.3) verwendet, um eine möglichst fahrzeugnahe Evaluation des

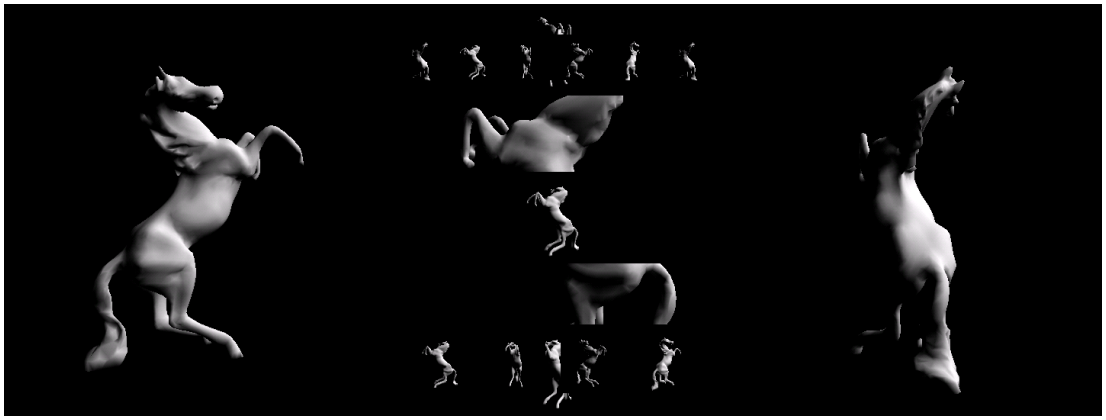

Abbildung 4.21.: Fahrzeugszenario mit 17 Instanzen der Anwendung „glmark2“

Compositings zu ermöglichen. Des Weiteren wurde die Testarchitektur aus Abbildung 4.11 verwendet, um eine Evaluation im Rahmen der Zielarchitektur aus Abbildung 2.2 zu gewährleisten.

Es wurden 12 bis 17 Anwendungen verwendet, die jeweils eine Instanz der Bewertungsanwendung „glmark2-es“ [Glm16] laufen lassen, wie in Abbildung 4.21 dargestellt. Dabei entspricht die Interaktion zwischen den Anwendungen und dem Compositing der Beschreibung aus Abschnitt 4.3. Die Anwendungen erhalten als Eingabe das 3D-Modell „horse“, verarbeiten dieses mittels OpenGL ES 2.0 Befehlen und zeichnen das resultierende Bild in ihre Bildspeicher. Mittels des EGL Befehls „eglSwapBuffers“ wird dann das Compositing über die Aktualisierung des Bildspeichers informiert. Der Inhalt der Bildspeicher wird jeweils unter Verwendung einer der Compositing-Strategien in den Anzeigespeicher kopiert. Für die Auswertung werden pro Strategie jeweils 28000 Bilder berechnet und die Anzahl der Aktualisierungen in Bildern pro Sekunde gemessen. Die Ergebnisse sind in Abbildung 4.22 dargestellt. Die x-Achse stellt die Anzahl an Anwendungen bzw. Fenster dar, die pro Durchlauf dargestellt wurden. Die rechte y-Achse repräsentiert die Ausführungszeit des Compositings in μs und die linke y-Achse ist die Anzahl an Bildern pro Sekunde, welche die Anwendungen im Durchschnitt haben.

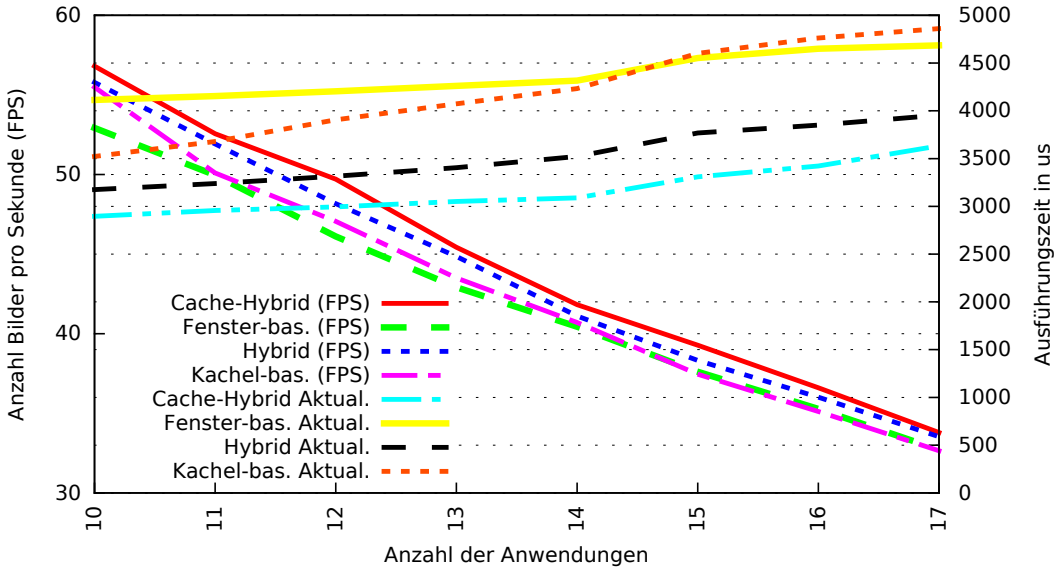

Abbildung 4.22.: Leistungsvergleich der Compositing-Strategien mit 12 bis 17 3D-Anwendungen

Wie in der Abbildung 4.22 zu sehen, steigt die Ausführungszeit für das Compositing mit Zunahme der Anzahl der Anwendungen an. Hierbei zeigt sich, dass das *Cache-Hybrid-Compositing* deutlich weniger Ausführungszeit für das Compo-

siting benötigt als die anderen Strategien. Im Durchschnitt liegt die Optimierung im Vergleich zum dem *fenster-basierten Compositing* bzw. dem *kachel-basierten Compositing* bei ca. 26 %. Dies wirkt sich auch direkt auf die Aktualisierungsrate der Anwendungen aus. Das *Cache-Hybrid-Compositing* ermöglicht eine Optimierung der Anzahl an Bildern pro Sekunde, welche die Anwendungen darstellen können, im Durchschnitt von ca. 4 % bzw. 5 % im Vergleich zu dem *fenster-basierten Compositing* bzw. dem *kachel-basierten Compositing*. Dies sind knapp 2 Bilder pro Sekunde, welche im Durchschnitt je Anwendung dazugewonnen werden. Dies zeigt, dass die Optimierung der Ausführungszeit für das Compositing auch die Aktualisierungsrate der Anwendungen verbessert.

4.5.7. Zusammenfassung der Auswertung

Es wurde gezeigt, dass die Ausführungszeit des Bitblittings durch die Verwendung des *Hybrid-Compositings* im Vergleich zu den existierenden Ansätze in verschiedenen Szenarien deutlich gesenkt wird. Obgleich die Strategie *Hybrid-Compositing* in manchen Fällen eine höhere Ausführungszeit auf der CPU im Vergleich zu den anderen Ansätzen zur Folge hat, kann dieser Nachteil durch das *Cache-Hybrid-Compositing* ausgeglichen werden. Der Ansatz *Cache-Hybrid-Compositing* übertrifft dabei die existierenden Ansätze und optimiert die Ausführungszeit für das Bitblitting erheblich, wie durch Evaluationen gezeigt wurde. Einzig durch eine optimale Aufteilung der sichtbaren Kacheln kann die Ausführungszeit des Bitblittings noch weiter reduziert werden. Hierbei konnte jedoch gezeigt werden, dass dies im Durchschnitt unter 1 % liegt, aber die dazu benötigte Berechnungszeit auf der CPU t_{CPU} um einige Größenordnungen höher liegt, sodass dieser Aufwand nicht im Verhältnis steht. Schließlich konnte gezeigt werden, dass das *Cache-Hybrid-Compositing* auch in einem Szenario mit 3D-Anwendungen eine Optimierung der Aktualisierungsraten der Anwendungen herbeiführen kann.

4.6. Compositing-Konzept für pixel-definierte Fenster

In diesem Abschnitt wird das Compositing behandelt, welches auf Grundlage von pixel-definierten Fenstern ein Bitblitting mittels des 2D-Befehls $2D\_MaskBlit$ durchführt. Hierbei werden die Fenster nicht mittels Höhe und Breite definiert, sondern jedes Fenster ist mittels einer Bitmaske beschrieben, welche pro Pixel

angibt, ob ein Pixel zu einem Fenster gehört oder nicht.

Im Compositing werden mittels Bitblitting die Pixel, die in der Bitmaske markiert wurden, in den Anzeigespeicher kopiert. Die Bitmasken ermöglichen es, die Form der Fenster beliebig zu gestalten, im Gegensatz zu rechteckigen Fenstern.

Für die Anwendung des Zugriffskonzeptes für Anzeigebereiche, das in Kapitel 3 beschrieben wird, sind pixel-definierte Fenster und Berechtigungen erforderlich, falls die Berechtigungen nicht als rechteckige Bereiche definiert werden sollen. Das heißt, für die Definition der Pixel einer Berechtigung wird eine Bitmaske verwendet, die die Pixel markiert, für welche der Zugriff gewährt wird. Dies ermöglicht es auf Granularität einzelner Pixel den Zugriff der Anwendungen zu regeln.

Im Folgenden werden im Detail die pixel-definierten Fenster und das Compositing-Konzept beschrieben.

4.6.1. Pixel-definierte Fenster

Im Gegensatz zu rechteckigen Fenstern ist bei pixel-definierten Fenstern die Form des Fensters beliebig, da auf Grundlage einzelner Pixel das Fenster definiert werden kann. Hierbei kann ein Fenster aus einer beliebigen Anzahl an Pixeln bestehen, welche nicht zwangsläufig zusammenhängend sein müssen und die „Fläche“ des Fensters somit Lücken enthalten kann. Dies ermöglicht es, einzelne Pixel nur mittels x- und y-Koordinate zu identifizieren, da ein Überdecken von Fenstern nicht notwendig ist und somit auf die z-Koordinate verzichtet werden kann.

Für die Definition eines Fensters auf Basis einzelner Pixel wird eine Bitmaske verwendet, die als rechteckiger Rahmen definiert ist und alle relevanten Pixel umschließt. Es wird zusätzlich eine Startposition für die Bitmasken gegeben, damit das Compositing nur den relevanten Bereich betrachten muss. Für die Festlegung der relevanten Pixel wird eine monochrome Bitmaske verwendet, die allen Pixeln, welche zu dem Fenster gehören, den Monochromwert weiß (d. h. den Wert 1) und allen nicht zugehörigen Pixel den Monochromwert schwarz (d. h. den Wert 0) gibt. Das heißt, nur die mit 1 markierten Pixel werden durch das Compositing kopiert.

In Abbildung 4.23 ist ein Beispiel für drei pixel-definierte Fenster dargestellt. Das Fenster A besitzt eine rechteckige Fläche, welche zwei Flächen in Form eines Kreises und eines Sechsecks besitzt. In diesen freien Flächen werden die Fenster B und C dargestellt, welche sich nicht mit Fenster A überdecken. Rechts in der Abbildung sind die verwendeten Bitmasken dargestellt. Die rechteckigen Bitmasken markieren die zugehörigen Pixel mit schwarz bzw. wegen der besseren Lesbarkeit in Grautönen, während die nicht zugehörigen Pixel weiß sind.

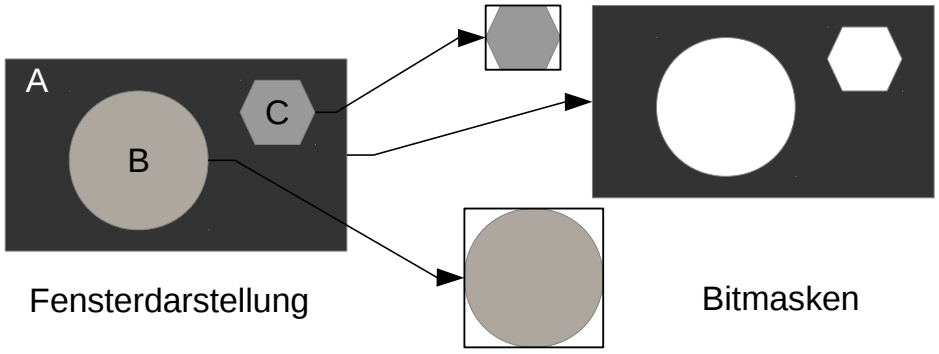

Abbildung 4.23.: Beispiel für pixel-definierte Fenster

Die Verwendung von Bitmasken zur Beschreibung der Fenster bietet jedoch nicht nur Vorteile im Vergleich zu den rechteckigen Fenstern. Die Definition eines rechteckigen Fensters besteht im Wesentlichen aus der Beschreibung der Größe und der Position. Im Vergleich dazu muss für ein pixel-definiertes Fenster eine Bitmaske in der Größe der Fläche, welche alle Pixel umschließt, gespeichert werden. Zudem erhöht sich der Aufwand, welcher für das Bitblitting eines Fensters erforderlich ist, da nun zusätzlich die Bitmaske berücksichtigt wird. Dies bedeutet, die Flexibilität der pixel-definierten Fenster erfordert einen höheren Aufwand im Vergleich zu den rechteckigen Fenstern.

4.6.2. Compositing-Konzept

Das Compositing unter Verwendung von Bitmasken gestaltet sich bei der Verwendung von kontext-beschränkten Berechtigungen (siehe Kapitel 3) deutlich einfacher, da Überdeckungen von Fenstern durch überschneidungsfreie Bitmasken verhindert werden können. In Abbildung 4.5 ist die Architektur für das Compositing dargestellt, welche für das Bitblitting mit Bitmasken ebenfalls relevant ist. Im Unterschied zu den rechteckigen Fenstern werden bei der Fensteranmeldung Bitmasken verwendet, welche der Fenstermanager dann als Fensterdaten der Compositing-Schicht zur Verfügung stellt. Für jedes registrierte Fenster wird ebenfalls ein Offscreen-Puffer angelegt, welches die grafischen Inhalte der Anwendungen enthält. Wird ein Fenster zur Aktualisierung markiert, dann kann die Compositing-Schicht sofort das Bitblitting des zugehörigen Offscreen-Puffers unter Verwendung der Bitmaske beginnen, da es zu keinem Schreibkonflikt der Pixel kommen kann. Das heißt, der entsprechende Offscreen-Puffer wird mittels des 2D-Befehls $2D\_MaskBlit$ kopiert, wobei im Anzeigespeicher nur die Pixel geschrieben werden, welche in der zugehörigen Bitmaske markiert sind.

Wie die Leistung des Compositings sich unter Verwendung von Bitmasken verhält, wird im folgenden Kapitel evaluiert.

4.7. Evaluation der Compositing-Strategien für pixel-definierte Fenster

In diesem Abschnitt wird das Compositing unter Verwendung von Bitmasken evaluiert. Zuerst wird die Plattform- und Messanordnung gefolgt von den verwendeten Szenarien beschrieben. Für das Compositing unter Verwendung des 2D-Befehls $2D\_MaskBlit$ mittels Bitmasken wird zuerst eine Evaluation der Ausführungszeiten des Compositings mit verschieden großen Bitmasken beschrieben. Anschließend wird das Compositing mittels 2D-Befehl $2D\_MaskBlit$ mit dem Compositing unter Verwendung des 2D-Befehls $2D\_Blit$ und der Compositing-Strategie *Cache-Hybrid-Compositing* für rechteckige Fenster aus dem Abschnitt 4.4 verglichen.

4.7.1. Plattform und Testumgebung

Für die Evaluationen wurde die eingebettete Plattform i.MX6 SABRE Automotive wie bei den Evaluationen aus Abschnitt 4.5 verwendet. Deren Beschreibung ist in Abschnitt 4.5.1 zu finden. Für das Bitblitting mit Bitmasken wurde der Vivante Treiber für den *Bildspeicher* verwendet, in diesem Fall der Befehl „$gco2D\_MonoBlit$“, welcher dem Befehl $2D\_MaskBlit$ entspricht.

Die Messung der Ausführungszeit des Compositings t_{comp} unter Verwendung von Bitmasken wurde mittels der POSIX-Funktion $clock\_gettime$ durchgeführt. Des Weiteren wurden für die Testanwendungen jeweils die Anzahl der Bilder pro Sekunde ausgewertet. Ähnlich den Evaluationen aus Abschnitt 4.5 wurde eine virtualisierte Umgebung für die Evaluationen verwendet, welcher der Zielarchitektur, beschrieben in Abschnitt 2.2, entspricht. Dazu wurde die Virtualisierungslösung PikeOS von Sysgo mit zwei Partitionen für jeweils den Virtualisierungsmanager und die 3D-Testanwendungen verwendet, wie in Abbildung 4.24 dargestellt.

Die Testanwendungen kommunizieren mittels dedizierter Kommunikationskanäle mit dem Virtualisierungsmanager bzw. mit den einzelnen Komponenten. Bevor eine Testanwendung ein Fenster beim Fenstermanager registrieren kann, muss über das Berechtigungs- und Richtlinien-Management eine kontext-beschränkte Berechtigung für einen Anzeigebereich bezogen werden. Die kontext-beschränkten

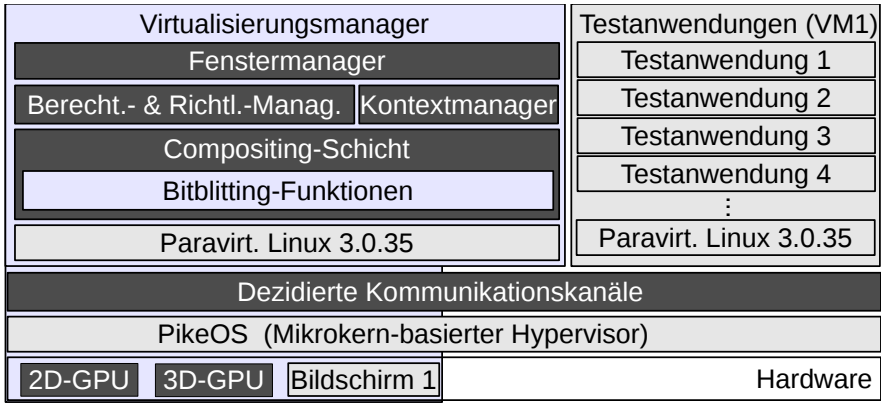

Abbildung 4.24.: Testarchitektur für das Compositing mittels Bitmasken in einer virtualisierten Umgebung

Berechtigungen für Anzeigebereiche, wie in Kapitel 3 beschrieben, werden dazu in Form von XML-Dateien definiert und über die Testanwendungen an das Berechtigungs- und Richtlinien-Management übertragen. Dieses vergibt dann die Berechtigungen entsprechend der Kontexte, welcher der Kontextmanager dem Berechtigungs- und Richtlinien-Management übermittelt. Liegt einer Testanwendung eine Berechtigung vor, kann sie ein Fenster entsprechend der Bitmaske der Berechtigung beim Fenstermanager registrieren und einen EGL-Kontext erzeugen. Liegt ein gültiger EGL-Kontext vor, dann kann die Testanwendung die OpenGL ES 2.0-Befehle über einen dezidierten Kommunikationskanal an den Virtualisierungsmanager übertragen, welcher die Befehle entsprechend an die 3D-GPU weiterleitet. Die resultierenden Bilder werden in die jeweiligen Bildspeicher der Testanwendungen geschrieben und zur Aktualisierung markiert. Das Compositing wird mit der Bildwiederholrate der Anzeige synchronisiert, welche 60 Hz beträgt. Das Bitblitting kopiert dann die markierten Fenster entsprechend der Bitmaske mittels des 2D-Befehls „$gco2D\_MonoBlit$“ des Vivante Treibers in den Anzeigespeicher. Zusätzlich zu der Ausführungszeit des Bitblittings wird die Anzahl der Bilder pro Sekunde der einzelnen Testanwendungen gemessen.

4.7.2. Szenarien

Für die Evaluationen wurden zwei verschiedene Szenarien mit kontext-beschränkten Berechtigungen definiert, welche die Leistungsfähigkeit des Bitblittings mittels Bitmasken evaluieren sollen.

Dies ist zum einen eine Anordnung verschiedener rechteckiger Bitmasken, welche sich sowohl in Anzahl als auch in der Größe verändern können. Das *Skalierbarkeitsszenario* soll in erster Linie die Skalierbarkeit des Bitblittings mittels

Abbildung 4.25.: Fensterverteilung für 10 Anwendungen mit je einer Bitmaske der Größe 170 × 170 Pixel

Bitmasken aufzeigen. Die Größe der Bitmasken variiert dabei von 10 × 10 Pixel, 20 × 20 Pixel, usw. bis auf 170 × 170 Pixel. Die einzelnen Bitmasken liegen dabei alle nebeneinander, sodass es zu keinen Überdeckungen kommt und jedes Fenster sichtbar auf der Anzeige mit 1440 × 540 Pixel ist. In Abbildung 4.25 ist eine Verteilung von 10 Fenstern mit je einer Größe von 170 × 170 Pixel dargestellt.

Zum anderen wurde ein *Bitmasken-Fahrzeugszenario* definiert, das dem *Fahrzeugszenario* aus Abbildung 4.14 ähnlich ist. Es wurde vollständig mit Bitmasken erstellt, die in kontext-beschränkten Berechtigungen definiert wurden, wie in Abbildung 4.26 für die Anzeige mit 1440 × 540 Pixel dargestellt. Dies demonstriert ein typisches Szenario im Fahrzeug.

4.7.3. Skalierbarkeit des Compositings mittels Bitmasken

Für die Evaluation der Skalierbarkeit des Compositings mittels Bitmasken wurde eine Lastsimulation mit bis zu 15 Anwendungen durchgeführt. Diese Lastsimulation soll zeigen, wie sich eine zunehmende Zahl an Bitmasken mit unterschiedlichen

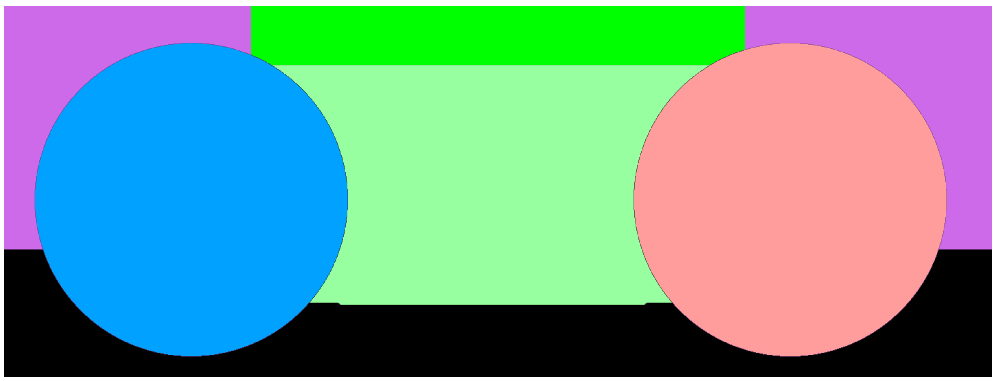

Abbildung 4.26.: Bitmasken-Fahrzeugszenario

Größen auf die Ausführungszeit des Compositings auswirkt. Für das Compositing wurde der 2D-Befehl „*gco2D_MonoBlit*“ mit den Bitmasken aus den Berechtigungen verwendet. Es wurde die Ausführungszeit des Compositings und die Anzahl der Bilder pro Sekunde von 3D-Testanwendungen in Abhängigkeit der Anzahl der Anwendungen und der Größe der Bitmasken bzw. der Fenster evaluiert. Die Größe des Bitmasken entspricht der Fenstergröße. Es wurden für je 1 bis 15 Anwendungen Lastsimulationen mit dem *Skalierbarkeitsszenario*, d. h. mit an Größe zunehmenden Bitmasken, durchgeführt. Dazu wurden je Anzahl Fenster acht verschiedene Fenstergrößen verwendet und jeweils 100 mal ausgeführt, wobei je 10000 Bilder für jeden Durchgang dargestellt wurden.

Als Testanwendungen wurden mehrere Instanzen der Anwendung „glmark2-es“ mit dem Modell „Horse“, das aus ca. 7250 Dreiecken besteht, gestartet. Ein Beispiel für die Verteilung von zehn Fenstern mit je einer Instanz „glmark2-es“ ist in Abbildung 4.27 dargestellt. In Abbildung 4.28 ist die durchschnittliche Ausführungszeit des Compositings in Abhängigkeit der Bitmaskengröße und der Anzahl der Fenster bzw. Anwendungen dargestellt. Die x-Achse repräsentiert die Größe der Bitmaske (Höhe × Breite) in Pixel. Die z-Achse stellt die Anzahl der Fenster und die y-Achse die Ausführungszeit des Compositings t_{comp} in ms dar.

Wie in Abbildung 4.28 zu sehen, ist die Ausführungszeit des Compositings bei nur einer Anwendung sehr gering und steigt nur minimal mit Zunahme der Fenstergröße an. Das heißt, die Fenstergröße beeinflusst bei Verwendung von Bitmasken nur minimal die Ausführungszeit des Compositings. Nimmt die Anzahl der Fenster bzw. der Bitmasken zu, dann erhöht sich die Ausführungszeit des Compositings erheblich. Die gemessenen Ausführungszeiten schwanken ab 10 Fenstern relativ stark, da die Messung aufgrund der parallelen Ausführung zunehmend durch das Scheduling der CPU beeinflusst wird. Im Wesentlichen beeinflusst jedoch die

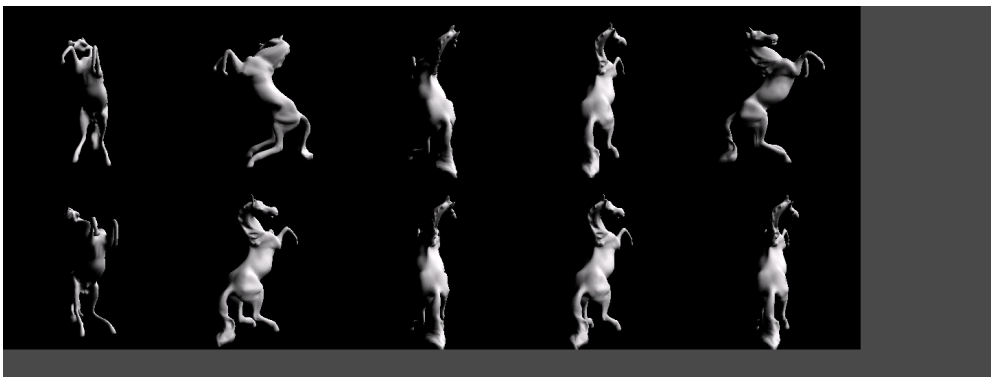

Abbildung 4.27.: Fensterverteilung für zehn Anwendungen „glmark2-es“ der Größe 170 auf 170 Pixel

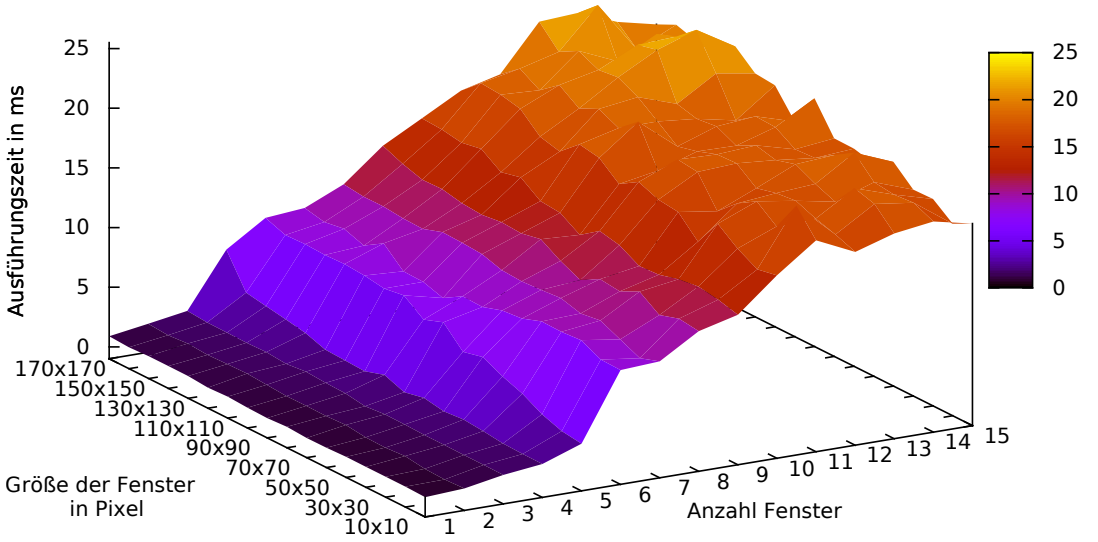

Abbildung 4.28.: Ausführungszeit des Compositings unter Verwendung von Bitmasken in Abhängigkeit der Größe und der Anzahl der Fenster

Anzahl der Bitmasken bzw. der Fenster beim Compositing mit Bitmasken die Ausführungszeit. Die Anzahl der im Compositing gleichzeitig verwendeten Bitmasken stellt eine obere Schranke für die i.MX6 Plattform dar, da bereits bei 10 Bitmasken bis zu 16 ms für die Aktualisierung der Anzeige benötigt wird und eine Aktualisierungsrate von 60 Bilder pro Sekunde bei mehr Bitmasken nicht für alle Anwendungen möglich ist. Sollten auf dieser Plattform mehr Anwendungen parallel ausgeführt werden, dann wird die Aktualisierungsrate für die Anwendungen niedriger als 60 Bilder pro Sekunde sein.

4.7.4. Ausführungszeiten des Compositings für das *Bitmasken-Fahrzeugszenario*

In dieser Evaluation wurde das *Bitmasken-Fahrzeugszenario* verwendet, um die Kombiinstrumente zu evaluieren. In diesem Fall wird die gesamte Anzeige auf fünf Anwendungsfenster überdeckungsfrei aufgeteilt, sodass das Compositing folglich bei einer Aktualisierung aller Fenster die gesamte Anzeige mit 1440×540 Pixel mittels Bitblitting überschreibt.

Für die Evaluation wurden typische Anwendungen der Kombiinstrumente verwendet. Wie in Abbildung 4.29 dargestellt, handelt des sich um 3D-Anwendungen wie Tachometer und Statusleiste[1]. In der Evaluation wurden Bitmasken mit dem 2D-Befehl „$gco2D\_MonoBlit$“ verwendet. Für einen Vergleich zu dem Compo-

[1] von Fujitsu [Fuj16] in Rahmen von ARAMiS zur Verfügung gestellt

Abbildung 4.29.: Szenario mit fünf Anwendungen in einem Kombiinstrument

siting mit rechteckigen Fenstern aus Abschnitt 4.4 wurden rechteckige Bereiche verwendet. Die Fenster überdecken sich dann stellenweise (z. B. die Drehzahlmesser mit der Statusleiste) und werden deshalb in z-Richtung geordnet. Für das Compositing wurde die *Cache-Hybrid-Compositing*-Strategie unter Anwendung des 2D-Befehls „*gco2D_Blit*“ verwendet.

Die Evaluation wurde für 1000 Bilder durchgeführt und der Durchschnitt der Ausführungszeit für das Bitblitting und die Anzahl der Bilder pro Sekunde gemessen. In Abbildung 4.30 sind die Ergebnisse der Evaluation dargestellt. Die x-Achse stellt das verwendete Compositing dar. Die linke y-Achse bildet die Anzahl der Bilder pro Sekunde ab und die rechte y-Achse stellt die Ausführungszeit für das Compositing t_{comp} in μs dar. Für das Szenario wurden jeweils die Anzahl der Bilder pro Sekunde (roter Balken) und die Ausführungszeit des Compositings (blaukarierter Balken) dargestellt.

Wie allgemein in Abbildung 4.30 zu sehen ist, sinkt die Anzahl der Bilder pro Sekunde deutlich bei der Verwendung von Bitmasken im Vergleich zu den recht-

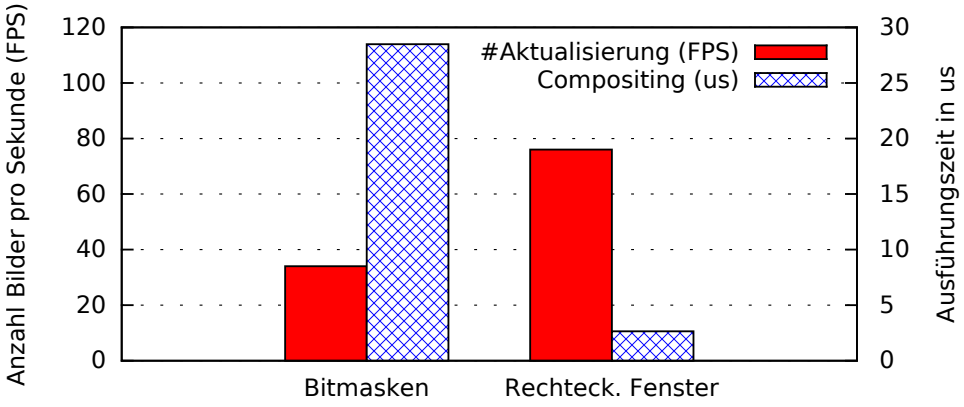

Abbildung 4.30.: Vergleich des Compositings von Fenstern mit Bitmasken und rechteckigen Fenstern im fahrzeugnahen Szenario

eckigen Fenstern mit der *Cache-Hybrid-Compositing*-Strategie. Der Aufwand für das Bitblitting mittels Bitmasken ist hierbei weitaus größer im Vergleich zu den rechteckigen Fenstern. Im Verhältnis benötigt das Bitblitting mit Bitmasken im Schnitt ca. das Zehnfache an Ausführungszeit. Jedoch sinkt die Anzahl der Bilder pro Sekunde bei der Verwendung von Bitmasken nur in etwa um die Hälfte. Dies hängt an der Berechnung der 3D-GPU, welche den grafischen Inhalt der Fenster berechnet. Das Compositing wird im Anschluss daran auf der 2D-GPU ausgeführt, jedoch parallel zur Berechnung der nächsten Fensterinhalte auf der 3D-GPU. Während im Fall der Bitmasken die Bildwiederholrate durch die Geschwindigkeit der 2D-GPU begrenzt wird, wird bei rechteckigen Fenstern die 3D-GPU zum Flaschenhals.

Somit lässt sich sagen, dass die gewonnene Flexibilität in einem fahrzeugnahen Szenario durch die Verwendung von Bitmasken im Gegensatz zu einfachen rechteckigen Fenstern einen deutlich höheren Aufwand zur Folge hat. Jedoch sind die Leistungseinbußen, welche dadurch verursacht werden, insbesondere bei der Verwendung von 3D-Anwendungen, zwar nicht unerheblich, aber sie hängen letztendlich von der benötigten Berechnungszeit der Anwendungen auf der 3D-GPU ab.

4.7.5. Zusammenfassung der Evaluation

Der Befehl „$gco2D\_Blit$“ hat unter Verwendung von rechteckigen Fenstern einen deutlichen Geschwindigkeitsvorteil gegenüber dem Befehl „$gco2D\_MonoBlit$“ mit Bitmasken. Ist vor allem Geschwindigkeit in der Darstellung der Anwendungsfenster bzw. eine große Anzahl an Fenster gefragt, dann sind rechteckige Fenster besser geeignet. Dies ist in erster Linie abhängig des Treibers und der verwendeten Hardwareplattform. Mittels einer Auswertung zur Skalierbarkeit kann in Abbildung 4.28 gezeigt werden, dass das Compositing mit Bitmasken nur bis ca. 10 Fenster bzw. Bitmasken eine Aktualisierungsrate von 60 Bildern pro Sekunde gewährleisten kann.

Dennoch bietet der Befehl „$gco2D\_MonoBlit$“ mit Bitmasken eine bessere Flexibilität. Insbesondere mit der kontext-basierten Zugriffskontrolle für Anzeigebereiche (vgl. Kapitel 3) ergibt sich die Möglichkeit der feingranularen Gestaltung von Szenarien, die durch die Bitmasken ein hohes Maß an Flexibilität erhalten. Die Zuordnung der Pixel auf Anwendungen lässt sich dann in Abhängigkeit von Kontexten beliebig ändern. Im Falle von bis zu 10 Anwendungsfenster mit Bitmasken in einem fahrzeugnahen Szenario sinkt auf der i.MX6 Plattform

die Leistung um ca. Faktor 2 im Vergleich zu dem Befehl „*gco2D_Blit*“, wie in Abbildung 4.30 gezeigt. Zukünftige leistungsfähigere Plattformen werden diese Einschränkung mit großer Sicherheit aufheben.

4.8. Verwandte Arbeiten

Seit der Erfindung der Rastergrafik gibt es eine große Anzahl an grafischen Fenstersystemen, die die grafische Ausgabe der Anwendungen verwalten und auf der Anzeige zusammenstellen. Die Fenstersysteme lassen in der Regel nur rechteckige Fenster zu. Hierbei sind mittlerweile Fenstersysteme dominierend, die das Überdecken von Fenstern zulassen. Wie von Bly und Rosenberg [BR86] analysiert, kann die Verwendung von Fenstermanagern, die das Überdecken von Fenster nicht zulassen und folglich eine kachel-basierte Darstellung der vollständig sichtbaren Fenster wählen, in manchen Situationen übersichtlicher sein. Kandogan und Shneiderman [KS97] verwenden in ihrer Fensterverwaltung ebenfalls eine kachel-basierte Darstellung der vollständig sichtbaren Fenster, versuchen diese aber hierarchisch zu gruppieren, sodass der Anwender bestimmte Operationen schneller ausführen kann. Des Weiteren ist der Anwender in der Lage, Operationen auf mehrere Fenster anzuwenden. Sie zeigen mittels Evaluationen auf, dass für bestimmte Operationen der Anwender in der Durchführung mittels ihrer Fensterverwaltung wesentlich schneller ist als in Fensterverwaltungen mit überdeckenden Fenstern. Dennoch hat die höhere Flexibilität der überdeckenden Fenster seinen Siegeszug unter den Fenstersystemen erfolgreich durchgeführt.

Die Verwendung von Fenstern in beliebiger Form ist bei weitem weniger häufig verbreitet. Neben dem Ansatz aus Kapitel 4, das die Definition beliebiger Fensterformen auf Basis einzelner Pixel zulässt, gibt es ein paar Arbeiten dazu. Im Folgenden werden zuerst verwandte Arbeiten für Fenstermanager mit rechteckigen Fenstern vorgestellt. Anschließend folgen die Ansätze für beliebige Fensterformen.

4.8.1. Fenstermanager mit rechteckigen Fenstern

Meyrowitz und Moser [MM81] entwickelten einen Fenstermanager, genannt BRUWIN, dessen Fokus auf der Adaptabilität des grafischen Systems zu einer breiten Masse an Geräten und Betriebssystemen liegt. Wichtige Bedingungen bei BRUWIN sind, neben der vollständigen Kontrolle des Anwenders über die Größe und die Position der Fenster, dass der Zustand eines Fensters immer aktualisiert ist, auch wenn es nicht sichtbar auf der Anzeige ist. BRUWIN bietet eine Menge

von Befehlen an, die dazu genutzt werden, Fenster zu erzeugen, zu zerstören, zu ändern und zu bewegen. Die Komponente Anzeigemanager ist dafür verantwortlich, die Fenster auf der Anzeige darzustellen und bei Änderungen der Fenster die Anzeige zu aktualisieren. Der Anzeigemanager ist auch für das Zusammenstellen der z-geordneten Fenster verantwortlich. Da der Inhalt eines Fensters nur aktualisiert wird, wenn es sichtbar ist, muss der Anzeigemanager entsprechend unterscheiden, ob ein Fenster vollständig verdeckt ist oder nicht. Hierfür wird ein *fenster-basiertes Compositing* vorgeschlagen, das alle vollständig verdeckten Fenster überspringt. Ansonsten verwendet BRUWIN für teilweise verdeckte Fenster entsprechend der Reihenfolge *fenster-basiertes Compositing*, womit auch bei mehrfach überdeckten Fenstern entsprechend die Effizienz sinkt.

Myers [Mye84a, Mye84b] entwickelte den Fenstermanager Sapphire, der abgekürzt für Anzeigenallokationspaket zur Bereitstellung von nützlichen Piktogrammen und rechteckigen Umgebungen (engl. Screen Allocation Package Providing Helpful Icons and Rectangular Environments) steht. Sapphire unterstützt z-geordnete rechteckige Fenster, die sich überdecken können. In [Mye86] wird vorgeschlagen, den sichtbaren Teil der Fenster nur im Anzeigespeicher und die verdeckten Teile der Fenster in den Offscreen-Puffern zu speichern. Dies minimiert den benötigten Speicherbedarf, da nicht der gesamte Inhalt der Fenster zwischengespeichert werden muss. Sapphire verwaltet pro Fenster eine Liste der Rechtecke, die einen sichtbaren Teil ähnlich zu der Graphenstruktur in Abschnitt 4.4.2 besitzen. Da Sapphire nicht den gesamten Inhalt der Fenster in den Offscreen-Puffern speichert, führt eine Positionsänderung der Fenster dazu, dass die Anwendungen ihre Fensterinhalte neu zeichnen müssen. Dies kann zu kurzzeitigen Artefakten auf der Anzeige führen, wenn die Inhalte nicht schnell genug neu gezeichnet werden.

Pike et al. [Pik83] entwickelten eine Datenstruktur zur Speicherung verdeckter Teile eines Fensters (von den Autoren Schichten genannt), um die Latenz für die Fensteraktualisierungen zu reduzieren. Vollständig sichtbare Fenster oder sichtbare Teile von Fenstern werden nicht gespeichert, da diese sich im Anzeigespeicher befinden. Deshalb wurde das Anwendungsgebiet des Operators für Bitmaps *bitblt* – definiert in [GS82] – erweitert, um vollständig oder teilweise verdeckte Fenster einbeziehen zu können. Im Wesentlichen wird das *kachel-basierte Compositing* verwendet, wobei die sichtbaren Kacheln nur im Anzeigespeicher gehalten werden. Obwohl dies den Speicherbedarf etwas reduziert, ist es wesentlich aufwändiger, die Aufteilung in Kacheln zu ändern, was notwendig ist, sobald sich beispielsweise die Position der Fenster ändert. Dies erfordert dann, dass die Daten zwischen dem Anzeigespeicher und der Offscreen-Datenstruktur kopiert werden müssen.

Chapuis und Roussel [CR05] entwickelten eine Fensterverwaltung, genannt Metisse, die dazu dient, das Design, die Implementierung und die Evaluation von innovativen Techniken der Fensterverwaltung zu evaluieren. Metisse basiert auf dem Fenstersystem X und verwendet einen modifizierten X-Server. Es gibt eine klare Trennung zwischen dem Erstellen der grafischen Inhalte der Anwendungen und dem Compositing. Der modifizierte X-Server verwendet Offscreen-Puffer für die grafischen Inhalte und nutzt für das Compositing eine leicht modifizierte Version des Fenstermanagers FVWM [Fvw16]. Für die Darstellung der Fenster werden mittels OpenGL transformierte Polygone verwendet, die mit 3D-Koordinaten beliebig platziert werden können. Dabei kann es zu Überdeckungen kommen, aber nicht zu einem Schnitt einzelner Fenster, da großer z-Werte für die Fenster gewählt werden. Dennoch erfordert jegliche Änderung eines Fensterinhaltes die gesamte Aktualisierung mittels des Compositings unter Beachtung der z-Werte. Da Anwendungen in Metisse, die OpenGL verwenden, keine Hardwarebeschleunigung mittels GPU nutzen können, kann es zu deutlichen Leistungseinbußen kommen.

Eines der bekanntesten grafischen Systeme ist das Fenstersystem X (X11) [SG86], das Grafikberechnung in 3D unterstützt z. B. OpenGL ES 2.0 mit EGL. Der X-Server verwaltet die Fenster und verwendet für das Compositing die Erweiterungen *Xdamage* und *Xcomposite*. *xcompmgr* [xco15] ist ein Fenstermanager, der das Compositing für X unterstützt und die Erweiterung Xdamage [GP04] verwendet, die es ermöglicht, X-Ereignisbenachrichtigungen über die Bereiche der Fenster zu erhalten, die modifiziert wurden und eine Aktualisierung benötigen. Da keine zusätzlichen Puffer für das Zwischenspeichern der grafischen Inhalte der Anwendungen verwendet werden, müssen die betroffenen Anwendungen ihre grafischen Inhalte auf Anfrage in z-Ordnung aktualisieren. Mit der Komponente *Xcomposite* des X-Servers werden alle Fensterinhalte in Offscreen-Puffern gehalten, aus welchen das Compositing mittels Bitblitting die Inhalte in den Anzeigespeicher kopiert. Wenn ein Fenster aktualisiert wird, dann wird das Compositing informiert und aktualisiert die Anzeige. Für das Compositing in X11 wird das *kachel-basierte Compositing* verwendet. Während dies für Systeme, in denen der Fensterinhalt in leistungsfähigem Speicher gehalten wird (z. B. dezidierter GDDR5 Speicher), ein gutes Konzept sein mag, ist dies für eingebetteten Systeme nicht der Fall.

Ein Nachfolger von X11 ist *Wayland* [Way15], das weniger komplex als X11 ist und dessen Funktionalitäten sich auf das Verwalten und das Compositing von Fenstern beschränkt. Im Gegensatz zu X11 trennt Wayland nicht zwischen Anzeigeserver, Fenstermanager und Compositing. Eine effiziente Implementierung

von Wayland ist *Weston*, welche die Bibliothek *Pixman* [Pix15] für das Erstellen und Verwalten der sichtbaren Kacheln der Fenster verwendet. Pixman betrachtet die Überdeckungen von Fenstern und berechnet Listen der resultierenden sichtbaren Kacheln der Fenster. Das heißt, das Compositing minimiert die Bereiche, die mittels Bitblitting kopiert werden müssen und verhindert das unnötige Aktualisieren von Fenstern, die andere Fenster überdecken. Unter Verwendung von Pixman gelten für das Compositing in Weston dieselben Vor- und Nachteile wie bei der Verwendung der Strategie *kachel-basiertes Compositing*.

Im Betriebssystem Android wird ein Flächenmanager (engl. genannt Surface Manager) für das Verwalten der *Flächen* der Anwendungen verwendet. Flächen sind *Widgets* oder *Views*, welche Komponenten der Benutzeroberfläche darstellen und deshalb nicht Fenster genannt werden, da Fenster in Android normalerweise die gesamte Anzeige überdecken. Das Compositing der Fenster, die durch die Flächen repräsentiert werden, wird von der Komponente *SurfaceFlinger* [Sur15] durchgeführt. SurfaceFlinger führt das Compositing ähnlich wie in X11 durch. Es werden die Rechtecke zur Laufzeit verglichen, um festzustellen, welche Bereiche aktualisiert wurden und welche Flächen zuoberst erscheinen müssen [Sur15]. Dennoch verwendet Android eine leicht optimierte Variante des *fenster-basierten Compositings*. Da in typischen Anwendungsfällen Fenster sich nicht überdecken, erscheint dies eine begründete, wenn auch nicht sehr allgemeingültige Wahl.

4.8.2. Fenstermanager für frei definierte Fensterformen

Neben den Fenstermanagern mit rechteckigen Fenstern gibt es verwandte Arbeiten, welche beliebige Formen für die Fensterinhalte verwenden.

Waldner et al. [WSGS11] stellen eine Fensterverwaltung vor, die das Compositing nach Wichtigkeit durchführt. Die Fenster werden zwar in rechteckiger Form definiert, jedoch wird die Wichtigkeit des Inhaltes mittels eines visuellen Aufmerksamkeitsmodells integriert, sodass die Fenster bei Überdeckung auch nicht rechteckige Formen annehmen können. Dies soll die räumliche Anordnung der Fenster optimieren, sodass eine maximale Sichtbarkeit und Interaktivität möglich ist. Dies wird durch eine Kombination aus verdecktem Fensterinhalt und durchsichtigen Fenstern ermöglicht. Der Fenstermanager greift auf die Offscreen-Puffer der Anwendungen zu und analysiert den Inhalt, um festzustellen, welche Bereiche wichtig sind. Wichtige Bereiche werden in einem Abbild der Wichtigkeit gespeichert, das angibt, in welchem Bereich des Fensters diese liegen. Dieses Abbild verwendet der Fenstermanager beim Compositing, um die verdeckten wichti-

ge Bereiche, z. B. durch Transparenz oder Ausschneiden, hervorzuheben. Ziel ist es, dass wichtige Teile des Inhalts, welche durch das Erscheinen neuer Fenster plötzlich verdeckt werden, ohne Interaktion mit den Fenstern dennoch sichtbar sind. In dieser Arbeit erfolgt die Definition der Fenster zuerst in rechteckiger Form, die spätere Darstellung der Fenster kann dann hingegen in beliebiger Form stattfinden. Da die Abbildung der Wichtigkeit aber zur Laufzeit und abhängig des Inhaltes generiert wird, ist eine gezielte Verwendung der Fenster in beliebiger Form schwer möglich. Des Weiteren wird die Wichtigkeit der Inhalte eines Fensters in dieser Arbeit sehr generisch ermittelt und befasst sich in erster Linie mit textuellen oder grafischen Dingen, welche durch Überdeckung von Fenstern nicht sichtbar wären. Hierbei kann jedoch z. B. bei mehrfacher Überdeckung nicht garantiert werden, dass sicherheitskritische Darstellungen sichtbar bleiben.

In einer weiteren Arbeit von Waldner et al. [WGSS11] werden rechteckige Fenster auf irreguläre Flächen dargestellt. Ziel ist es, hierbei beliebige Flächen für die Darstellung von Fenstern zu verwenden, wie es zum Beispiel mittels Projektoren möglich ist. Hierbei werden rechteckige Fenster in ihrer Form verändert, sodass sie auf den irregulären Flächen dennoch in rechteckiger Form wahrgenommen werden können. Des Weiteren werden die Fenster semi-automatisch auf die verfügbare irreguläre Fläche verteilt, sodass die Fläche möglichst optimal genutzt wird. Die Fenstermanipulationen ermöglichen zwar verschiedene Formen für Fenster, jedoch entspringen diese immer aus einem rechteckigen Fenster.

4.9. Zusammenfassung

Durch die flexible Nutzung der Anzeigen im Fahrzeug sind Anwendungen nicht mehr an eine statische Position gebunden. In Verbindung mit z-geordneten Fenstern, die sich gegenseitig überdecken können, kann das grafische System im Fahrzeug so flexibel wie ein gewöhnliches Desktopsystem sein. Unglücklicherweise sind die eingebetteten Plattformen im Fahrzeug wesentlich stärker in der Leistung beschränkt, was maßgeblich dem geringen Energiebedarf geschuldet ist. Effizienz ist eine Schlüsselanforderung in einem grafischen Fenstersystem im Fahrzeug.

Normalerweise zeichnen die Anwendungen im Fahrzeug, z. B. die Geschwindigkeitsanzeige, ihren grafischen Inhalt in Offscreen-Puffer, die dann vom grafischen Fenstersystem wie X11 in die Anzeigespeicher mittels Bitblitting kopiert werden. Die flexible Nutzung der Anzeigen erfordert es jedoch, dass jede Anwendung entsprechend ihrer Kritikalität dargestellt wird und es nicht zu unbeabsichtigten oder böswilligen Überdeckungen von Fenstern kommt. Dazu werden Berechtigungen

benötigt, wie sie in Kapitel 3 abhängig des Kontexts definiert werden. Dies muss das Compositing eines grafischen Fenstersystems im Fahrzeug berücksichtigen.

Für das Compositing wurden verschiedene Ansätze betrachtet. Während ein Teil der Ansätze nur rechteckige Fenster darstellen können (siehe Kapitel 4.4), wurde ein weiterer Ansatz vorgestellt, der frei definierbare Flächen auf Basis von Pixeln als Fensterdefinition zulässt und sich für die Verwendung von Berechtigungen eignet, die den Zugriff auf Basis einzelner Pixel regelt (siehe Kapitel 4.6).

Im Falle von rechteckigen Fenstern, die sich überdecken können, gibt es entweder die Möglichkeit in z-Ordnung alle Fenster mittels Bitblitting in den Anzeigespeicher zu kopieren – *fenster-basiertes Compositing* – was zu mehrfachem Kopieren von überdeckten Fensterbereichen führen kann oder es werden nur die sichtbaren Kacheln der Fenster mittels Bitblitting in den Anzeigespeicher kopiert – *kachel-basiertes Compositing* – was eine höhere Anzahl an Bitblitting-Operationen erfordern kann. Um die Verarbeitungsdauer des Bitblittings zu verringern, wurde die Strategie *Hybrid-Compositing* vorgestellt, welche eine Kombination aus Rechtecken sucht, die zu einer geringeren Ausführungszeit für das Bitblitting führt. Dazu muss die Verarbeitungsdauer der Bitblitting-Operationen abgeschätzt werden, sodass eine optimale Kombination gewählt werden kann. Es wurde ein Modell erstellt, welches die Verarbeitungsdauer des Bitblittings, abhängig der Fenstergrößen und der Anzahl der Bitblit-Operationen, vorhersagen kann. Zudem wurde der Ansatz *Cache-Hybrid-Compositing* vorgestellt, welcher einen Cache für das Speichern bereits ermittelter Kombinationen aus Bitblitting-Operationen für die zukünftige Verwendung nutzt und somit die Verarbeitungsdauer auf der CPU senkt.

Mittels verschiedener Evaluationen konnte gezeigt werden, dass die Strategie *Cache-Hybrid-Compositing* schneller als die Standardansätze *fenster-basiertes Compositing* und *kachel-basiertes Compositing* ist. Die Verarbeitungsdauer für das Compositing ist in zufällig gewählten Szenarien, in welchen alle Fenster gleichzeitig markiert werden, im Durchschnitt um bis zu 5 % schneller im Vergleich zur Strategie *fenster-basiertes Compositing*. Unter Verwendung zufällig gewählter Aktualisierungsraten für die Fenster ist die Strategie *Cache-Hybrid-Compositing* sogar 51 % schneller als das *fenster-basierte Compositing*. Mit verschiedenen Evaluationen konnte gezeigt werden, dass ein signifikanter Leistungszuwachs mittels *Cache-Hybrid-Compositing* bei überdeckenden rechteckigen Fenstern möglich ist.

Bei nicht rechteckigen Fenstern werden mit Bitmasken die Pixel festgelegt, welche das Compositing mittels Bitblitting in die Anzeigespeicher kopiert. Hierdurch können beliebige Fensterformen definiert werden, die auch aus nicht zusammen-

hängenden Pixeln bestehen können. Die Bitmasken geben dabei an, welche Pixel zu einem Fenster gehören. Durch die Verwendung pixel-definierter Fenster kann das Zugriffskonzept aus Kapitel 3 ebenfalls auf nicht rechteckige Flächen angewandt werden. Hierdurch können kontext-beschränkte Berechtigungen definiert werden, die ebenfalls mittels Bitmasken festlegen, auf welche Pixel eine Anwendung zugreifen darf. Werden für die Fenster Bitmasken verwendet, die auf Berechtigungen beruhen, dann kann der geforderte exklusive Zugriff (vgl. Definition 15, Konfliktfreiheitseigenschaft in Abschnitt 3.5.1) auf Pixelebene durchgängig bis zur Anzeige gewährleistet werden.

Die gewonnene Flexibilität erfordert jedoch einen höheren Aufwand für das Bitblitting. Anhand von Evaluationen wurde gezeigt, dass das Compositing eine um den Faktor 10 höhere Ausführungszeit in zufällig gewählten Szenarien haben kann. Dennoch wirkt sich dies nicht in gleichem Maße auf die Aktualisierungsrate der Anwendungen aus. In einem fahrzeugnahen Szenario sinkt die Aktualisierungsrate der Anwendungen um ca. Faktor zwei. Letztlich bedeutet der Zugewinn an Flexibilität einen Mehraufwand, der von der eingesetzten Hardware abhängt und sich auf zukünftigen Hardwaregenerationen minimieren lassen wird. Im nächsten Kapitel folgt die Beschreibung eines Demonstrators, der die beschriebenen Konzepte demonstriert.

5. Demonstrator Virtualized-Car-Telematics

Der Demonstrator Virtualized-Car-Telematics (VCT) [RAL+15] wurde im Rahmen des Projektes ARAMiS erstellt und dient in erster Linie dazu, die im Projekt entwickelten Architekturen und Methoden bzw. Konzepte für die Multicore-basierte Hard- und Software zu validieren. Der Demonstrator VCT fokussiert auf die Domäne Infotainment und soll die Leistungsfähigkeit von Multicore-Steuergeräten und Virtualisierung mittels Anwendungsfällen demonstrieren. Es soll gezeigt werden, wie mittels Virtualisierung in einem Multicore-Steuergerät die unterschiedlichen Funktionalitäten von bisher getrennten Steuergeräten zusammengeführt werden können. Dies soll konkret anhand der beiden Steuergeräte der Kombiinstrumente – teils sicherheitskritisch – und der Headunit – nicht sicherheitskritisch – gezeigt werden. Mit einem zentralen grafischen Anzeigesystem sollen, abhängig des Kontexts, die unterschiedlichen Inhalte dem Fahrer auf den verschiedenen Anzeigen dargestellt werden. Hierbei spielt vor allem die flexible Zuordnung der grafischen Inhalte der Anwendungen eine besondere Rolle.

Es wurden zwei Varianten des Demonstrators VCT erstellt, die von der BMW Forschung und Technik GmbH (Plattform A) in ein Fahrzeug und der Daimler AG (Plattform B) in ein Cockpitaufbau integriert wurden.

Die Plattform A verwendet eine Intel-Plattform (CPU mit vier Kernen, basierend auf der „Sandy-Bridge-Architektur“) mit einer GPU. Es handelt sich um eine Plattform, die in Desktopsystemen Anwendung findet und nicht für den Bereich Automobil spezialisiert ist, was sich insbesondere im Stromverbrauch widerspiegelt. Diese Plattform wurde in ein Fahrzeug von BWM eingebaut und neben der Anbindung an die fahrzeugspezifischen Schnittstellen wie CAN, wurden auch die Anzeigen im Fahrzeug angeschlossen. Die Virtualisierungslösung wurde von Windriver gestellt. Im Unterschied dazu wurde bei der Plattform B das eingebettete System i.MX6 SABRE [Fre15] von Freescale verwendet, das eine CPU mit vier Kerne aufweist und die GPUs ebenfalls integriert sind. Die Plattform wurde in einen Cockpitdemonstrator eingebaut, welcher, neben zwei Anzeigen, über CAN

die Eingabetasten des Lenkrads und den Dreh-Drück-Steller anbindet.

Im Folgenden werden die Ziele und der Aufbau des Demonstrators beschrieben.

5.1. Ziele des Demonstrators

Neben den bereits in Abschnitt 1.1 genannten allgemeinen Zielen des Demonstrators hat die Plattform B weitere Ziele. Neben der Machbarkeit soll anhand des Demonstrators die Effektivität der implementierten Konzepte mit realitätsnahen Anwendungen, in verschiedenen Anwendungsfällen geordnet, gezeigt werden. Des Weiteren wird gezeigt, dass durch die Umsetzung der Komponenten, die nicht im Fokus dieser Arbeit stehen, eine funktionsfähige Gesamtarchitektur entsteht. Dies sind insbesondere die Komponenten für die Virtualisierung, das Weiterleiten von Grafikbefehlen mittels gemeinsamen Speichers und das Anbinden von CAN-basierten Eingabegeräten. Aus der Sicht des Kunden ist in erster Linie der zusätzliche Nutzen, welcher sich durch einen Infotainment-Domänenserver ergibt, relevant. Hierbei können im Wesentlichen die drei folgenden Ziele genannt werden:

1. Flexible Nutzung der Anzeigen der Kombiinstrumente und der HU: Bei statischer Zuweisung der Anwendungen auf die Anzeigebereiche wird bei den Kombiinstrumenten viel Fläche für Anwendungen wie Kontroll- und Warnleuchten reserviert, die nur in den seltenen Fällen von Fehlern benötigt wird. Des Weiteren kann nur bei bestimmten Anwendungen der HU die Darstellung auf der Anzeige der Kombiinstrumente erfolgen, z. B. Richtungsanweisungen der Navigationssoftware. Durch die flexible Nutzung der Anzeigen kann der Kunde beliebige Inhalte der Kombiinstrumente und der HU auf den verfügbaren Anzeigen darstellen. Beispielsweise kann die Anzeige der Kombiinstrumente für das Abspielen von DVDs genutzt werden oder die Darstellung der Geschwindigkeit und der Drehzahl kann verkleinert werden, um der Navigation in der Mitte der Anzeige der Kombiinstrumente einen größeren Anzeigebereich zu geben. Eingeschränkt wird die flexible Nutzung nur durch gesetzliche und sicherheitskritische Anforderungen (z. B. würde das Abspielen einer DVD während der Fahrt den Fahrer ablenken).
2. Nutzung von Drittanbietersoftware in einer abgeschotteten Partition: Die Anwendungen der Kombiinstrumente und der HU kommen in der Regel vom OEM und sind für die Hardwareplattform optimiert. Hierbei wird vor allem auf den sicheren und zuverlässigen Betrieb der Anwendungen geachtet, welcher eine Beeinträchtigung von sicherheitskritischen Anwendungen durch nicht-sicherheitskritische Anwendungen verhindern soll. Durch die Be-

reitstellung einer Partition (d. h. einer virtuellen Maschine) kann der Kunde Anwendungen betreiben, die nicht explizit durch den OEM geprüft wurden. Dadurch kann der Kunde Anwendungen von Drittanbietern nutzen, wie sie beispielsweise auf dem Smartphone angeboten werden.

3. Geringere Kosten der Hardware und reduzierter Stromverbrauch: Durch die Konsolidierung der Hardware, wie bei den Kombiinstrumenten und der HU, können die Kosten für die Hardware gesenkt werden. Dies ermöglicht es dem OEM entweder die Preise zu senken oder mehr Funktionalitäten zur Verfügung zu stellen. Insbesondere ergeben sich durch die Konsolidierung ein niedrigerer Stromverbrauch und ein geringerer Platzbedarf für den Einbau in das Fahrzeug. Meist wird dadurch auch Gewicht eingespart.

5.2. Aufbau des Demonstrators

In diesem Abschnitt wird der Aufbau des Demonstrators beschrieben. Dieser unterteilt sich in den Hardwareaufbau des Demonstrators, basierend auf dem Cockpit einer S-Klasse 221, und die Softwarearchitektur, die drei virtuelle Maschinen bereitstellt, in welchen verschiedene Komponenten und Anwendungen laufen.

5.2.1. Hardwareaufbau

Der Hardwareaufbau des Demonstrators besteht aus einem Cockpit einer S-Klasse 221, in die zwei Anzeigen mit einer Größe von je 10 Zoll und einer Auflösung

Abbildung 5.1.: Vorderseite des Demonstrators

von je 1440×540 Pixel eingebaut sind, wie in Abbildung 5.1 zu sehen ist. Die Anzeigen sind mittels der Low Voltage Differential Signaling (LVDS)-Schnittstelle angebunden. Des Weiteren sind die beiden Steuerkreuze mit den Tasten auf dem Lenkrad mittels des Steuergerätes Mantelrohr-Schaltmodul (MRSM) über die CAN-Schnittstelle mit 500 Kbit/s angebunden. Dies gilt auch für den Dreh-Drück-Steller, welcher mit dem Steuergerät Zentrales-Bedienelement (ZBE) ebenfalls über die CAN-Schnittstelle angebunden ist, jedoch mit 125 Kbit/s.

Wie in Abbildung 5.2 zu erkennen, sind auf der Rückseite des Cockpits die Hardwarekomponenten angebracht. Das eingebettete System i.MX6 SABRE von Freescale ist für die Verwendung im Fahrzeug ausgelegt. Die Plattform bietet eine CPU mit vier Kernen, die mit 800 kHz betrieben werden. Des Weiteren stehen 2 GB DDR3 Arbeitsspeicher zur Verfügung. Die Plattform bietet drei verschiedene GPUs von Vivante [Fre15]. Die 3D-GPU GC2000 unterstützt die Grafikschnittstelle OpenGL ES 2.0 [Ope16b]. Die GPU GC320 ist für das Compositing ausgelegt. Zusätzlich gibt es die GPU GC355, die die 2D-Grafikschnittstelle OpenVG [Ope16c] unterstützt. Die Plattform stellt die Schnittstelle MIPI-CSI2 für Kameras zur Verfügung. An diese Schnittstelle wurde eine Rückfahrkamera angeschlossen, welche mit der Auflösung 640×540 Pixel Bilder aufnehmen kann.

Für die Anbindung der beiden LVDS-Anzeigen an die i.MX6 Plattform sind drei Adapter verwendet worden. Die Plattform besitzt zwei LVDS-Schnittstellen,

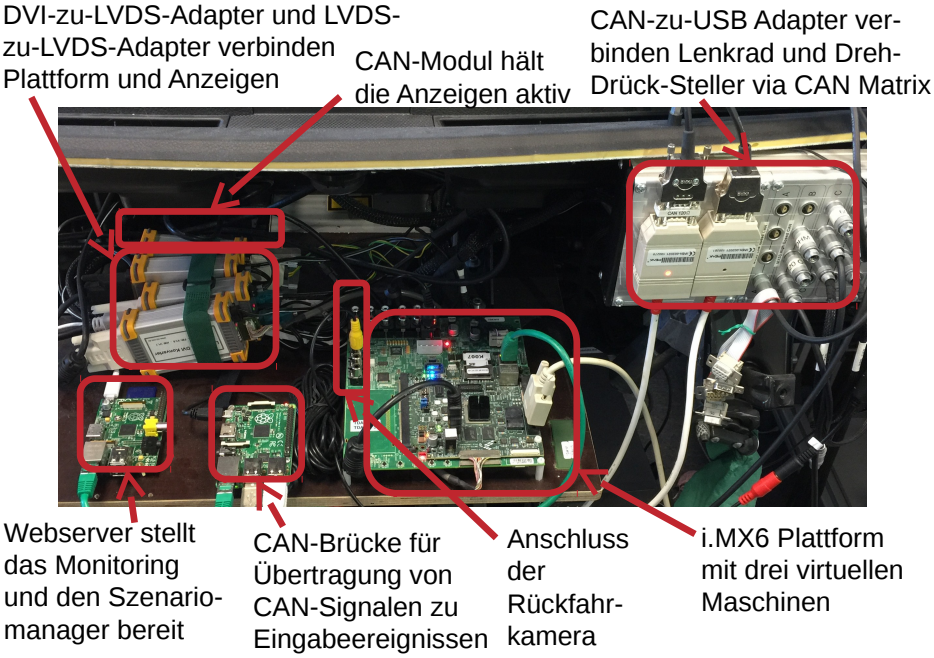

Abbildung 5.2.: Komponenten des Demonstrators

welche jedoch nicht kompatibel zu den LVDS-Schnittstellen der beiden Anzeigen sind. Deshalb wurde per Digital Visual Interface (DVI)-zu-LVDS-Adapter eine der Anzeigen an die einzig verfügbare High Definition Multimedia Interface (HDMI)-Schnittstelle – diese ist auch DVI-kompatibel – der Plattform angeschlossen. Die andere Anzeige wurden mit zwei weiteren Adaptern angeschlossen. Ein LVDS-zu-DVI-Adapter konvertiert das LVDS-Signal auf die DVI-Schnittstelle und ein weiterer DVI-zu-LVDS Adapter konvertiert das DVI-Signal dann auf die LVDS-Schnittstelle der Anzeige. Die beiden Anzeigen werden durch CAN-Signale aktiviert und schalten sich bei Inaktivität des CAN-Busses nach einer gewissen Zeit ab. Deshalb wurde ein CAN-Modul angeschlossen, das kontinuierlich CAN-Signale über den CAN-Bus sendet und die Anzeigen aktiv hält.

Die Plattform besitzt zwei CAN-Schnittstellen, von welchen aber nur eine gleichzeitig zur Ethernet-Schnittstelle aktiv sein kann. Deshalb wurden die beiden Steuergeräte ZBE und MRSM mittels zwei CAN-zu-Universal Serial Bus (USB)-Adaptern angeschlossen. Die Steuergeräte wurden an eine CAN-Matrix angeschlossen, die mit den ebenfalls angeschlossenen CAN-zu-USB-Adaptern verschalten wurden. Die CAN-zu-USB-Adapter sind per USB an ein Raspberry Pi – dieses fungiert als CAN-Brücke – angeschlossen, das die via USB übertragenen CAN-Signale empfängt. Die CAN-Signale werden dann in Eingabecodes, ähnlich denen von X11, umgewandelt und an die i.MX6 Plattform per Ethernet gesendet.

Für die Steuerung der Demonstration wurde als weitere Plattform ein Raspberry Pi verwendet, welche eine auf HTLM5 basierte Oberfläche mittels Apache Webserver zur Auswahl der Szenarien bereitstellt. Auf dieser Plattform befindet sich auch ein Server, der mit der i.MX6 Plattform kommuniziert, um Statusinformationen zu empfangen und Steuerungssignale für die Szenarien zu senden. In der Tabelle 5.1 sind alle Hardwarekomponenten aufgelistet und beschrieben.

Im folgenden Abschnitt werden die Softwarearchitektur und die Softwarekomponenten des Demonstrators im Detail beschrieben.

5.2.2. Softwarearchitektur

Für den Demonstrator wurden Softwarekomponenten implementiert, die die wesentlichen Konzepte dieser Arbeit umsetzen. Hierzu wurde die Softwarearchitektur aus Abbildung 5.3 auf der i.MX6 Plattform implementiert, welche aus einzelnen Softwarekomponenten besteht. Die Softwarearchitektur entspricht der Systemarchitektur eines Infotainment-Domänenservers (vgl. Abbildung 2.2).

Als Virtualisierungslösung wurde der mikrokern-basierte Hypervisor PikeOS

Tabelle 5.1.: Auflistung der Hardwarekomponenten des Demonstrators

Komponente	**Beschreibung**
Hauptplattform	Plattform i.MX6 Freescale mit Virtualisierungslösung
IC-Anzeige	Vollgrafische Anzeige der Kombiinstrumente
HU-Anzeige	Zentrale Anzeige der HU
CAN-Modul	Modul zum Aktivhalten der Anzeigen via CAN
Dreh-Drück-Steller (ZBE)	Zentrales-Eingabegerät für die Steuerung der HU
Lenkradtasten (MRSM)	Eingabetasten für die Steuerung der Kombiinstrumente
Rückfahrkamera	Kamera für die Erfassung der Umgebung im Rückwärtsgang
Webserver	Raspberry Pi Plattform (Modell A) mit Apache Webserver für die Steuerung der Demonstration
CAN-Brücke	Raspberry Pi Plattform (Modell B)

[Pik15] von Sysgo eingesetzt, der die Isolation der drei virtuelle Maschinen gewährleistet. Die drei virtuellen Maschinen stellen den Virtualisierungsmanager (VM0), die Kombiinstrumente (VM1) und die HU (VM2) dar. Alle drei virtuelle Maschinen verwenden Linux mit der Kernelversion 3.0.35. Jedoch hat einzig die virtuelle Maschine des Virtualisierungsmanagers Zugriff auf die GPUs und die Anzeigen, weshalb die Treiber der GPUs nur in dieser virtuellen Maschine laufen.

Da PikeOS nur für bis zu vier virtuelle Maschinen einen virtuellen Ethernet-Treiber bereitstellt, jedoch bereits eine virtuelle Maschine zu Kontroll- und Wartungszwecken von PikeOS selbst benötigt wird, musste beim Demonstrator auf eine dedizierte virtuelle Maschine für benutzerdefinierte Anwendungen verzichtet werden. Dafür wurde die virtuelle Maschine für die HU zu Demonstrationszwecken auch mit benutzerdefinierte Anwendungen wie einem Spiel verwendet. Des Weiteren wurde auf eine virtuelle Maschine für die Kommunikation über CAN mit einer Bussimulation des Fahrzeugs aus Aufwandsgründen verzichtet. Die benötigten Informationen (z. B. zur Bestimmung der Kontexte wie „Fahrzeug parkt“) werden den Anwendungen per TCP/IP-Verbindung zur Verfügung gestellt.

Im Folgenden werden die Komponenten der virtuellen Maschinen beschrieben.

Anwendungen der Kombiinstrumente und der HU

Zur Demonstration geeigneter Szenarien im Fahrzeug wurden 12 Anwendungen verwendet, die mittels einer ID eindeutig identifizierbar sind. Auf der virtuellen Maschine für die HU wurde als benutzerdefinierte Anwendung das Spiel „Quake

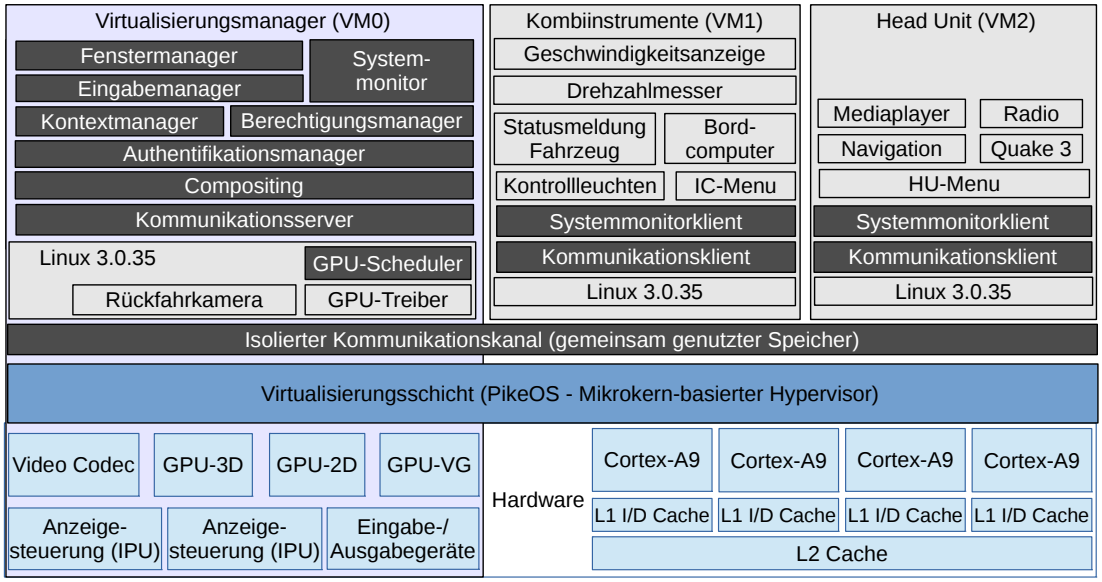

Abbildung 5.3.: Architektur des Demonstrators auf dem i.MX6

3“ verwendet, das gleichzeitig auch zur Demonstration der grafischen Leistungsfähigkeit des Systems dient. Wie das Spiel „Quake 3“ verwenden alle Anwendungen außer der Rückfahrkamera OpenGL ES für die grafische Darstellung ihrer Inhalte. Des Weiteren enthält die virtuelle Maschine für die HU noch die Anwendung „Mediaplayer“ (basierend auf VLC [VLC16]), „Navigation“ (Navit [Nav16]), „Radio“ und ein Menü „HU-Menü“ zum Auswählen der Anwendungen.

Die virtuelle Maschine für die Kombiinstrumente stellt typische Anwendungen für den Status des Fahrzeugs zur Verfügung. Dies sind die „Geschwindigkeitsanzeige“ und der „Drehzahlmesser“, die die Geschwindigkeit und die Drehzahl darstellen. Des Weiteren werden die Anwendungen „Kontrollleuchten“, „Statusmeldungen des Fahrzeugs“ (in Form eines sich rotierenden 3D-Modells eines Fahrzeugs), und „Bordcomputer“ für die Darstellung von Fahrzeuginformationen verwendet. Das Menü „IC-Menü“ dient zur Auswahl von Anwendungen. Die „Rückfahrkamera“ wird über einen Treiber im Virtualisierungsmanager angesteuert und speichert die Bilder in einen dedizierten Offscreen-Puffer, da aus Leistungsgründen eine Übertragung des Videobildes via Ethernet nicht ausreichend schnell ist.

Es gibt „Kommunikationsklienten“ auf beiden virtuellen Maschinen, die den Anwendungen eine Schnittstelle für die Kommunikation mit dem Virtualisierungsmanager auf Basis von gemeinsam genutztem Speicher bereitstellen. Hierbei werden alle Befehle für OpenGL ES mit den Grafikdaten, sowie EGL übertragen und auch die Befehle für den Fenstermanager und den Eingabemanager gesendet. Die Kommunikation findet über einen isolierten Kanal statt, der vom „Kommunika-

tionsserver“ des Virtualisierungsmanagers verwaltet wird und jeder Anwendung einen dedizierten Kanal zuweist, sodass kein unbefugter Zugriff möglich ist.

Für die Überwachung des Zustandes der virtuellen Maschinen gibt es einen „Systemmonitorklient“, der die Statusinformationen der Anwendungen an den „Systemmonitor“ des Virtualisierungsmanagers übersendet. Dadurch lassen sich Ausfälle der Anwendungen und der virtuellen Maschinen überwachen. Der Systemmonitor sammelt die Zustände der Kontexte. Somit lassen sich zu jedem Zeitpunkt die Zustände aller Kontexte abrufen.

Softwarekomponenten des Virtualisierungsmanager

Die virtuelle Maschine des Virtualisierungsmanagers ist als einzige privilegiert auf die Ressourcen GPUs, Rückfahrkamera und Anzeigen zuzugreifen. Alle anderen virtuellen Maschinen müssen für den Zugriff die Befehle und Daten an den Virtualisierungsmanager via isoliertem Kommunikationskanal schicken. Die Komponenten „Fenstermanager“, „Eingabemanager“, „Kontextmanager“, „Berechtigungsmanager“, „Authentifikationsmanager“, „Systemmonitor“, „GPU-Scheduler“, „Compositing“ und „Kommunikationsserver“ kontrollieren und regulieren diesen Zugriff.

Bevor eine Anwendung Befehle an eine der Komponenten senden darf, muss sie sich zuerst beim Authentifikationsmanager authentifizieren. Ist dies erfolgt, dann erstellt der Kommunikationsserver einen dedizierten Kanal für die Anwendung zur Kommunikation mit dem Fenstermanager, dem Kontextmanager und dem Berechtigungsmanager, sowie einen weiteren dedizierten Kanal zur Kommunikation mit dem GPU-Scheduler. Die Anwendung kann nun Befehle für den Zugriff auf die Anzeige bzw. GPU senden.

Damit eine Anwendung auf eine Anzeige zugreifen kann, benötigt sie eine Berechtigung. Hierfür sind der Kontextmanager und der Berechtigungsmanager zuständig, die das Zugriffskontrollmodell aus Kapitel 3 umsetzen. Der Kontextmanager stellt die Verwaltung der Kontexte bereit, welche Anwendungen nutzen, um den Status der Kontexte zu setzen. Die Kontextzustände werden dann an den Berechtigungsmanager gesendet, der abhängig von den Kontexten und den vorliegenden Berechtigungen Zugriff auf die Anzeigen gewährt. Die Zugriffskontrolle erfolgt auf Basis von Bitmasken, die es erlauben, auf Grundlage einzelner Pixel, den Zugriff zu gewähren. Für die Verwaltung der kontext-beschränkten Berechtigungen ermöglicht der Berechtigungsmanager den Anwendungen, kontext-beschränkte Berechtigungen an andere Anwendungen weiterzugeben und auch wieder zu entziehen. Zur Vergabe von kontext-beschränkten Berechtigun-

gen werden einzelne XML-Dateien verwendet, die die ID der Anwendung und die Bitmaske enthalten. Dies ermöglicht es, Szenarien mittels der Definition von kontext-beschränkten Berechtigungen in XML-Dateien aufzubauen. Die Anwendungen vergeben dann die kontext-beschränkten Berechtigungen entsprechend den XML-Dateien. Hat der Berechtigungsmanager einer Anwendung aufgrund einer Berechtigung den Zugriff auf einen Anzeigebereich gewährt, dann kann die Anwendung ein Fenster erstellen. Dazu sendet sie eine Anfrage an den Fenstermanager. Dieser prüft, ob eine entsprechende Berechtigung vorliegt. Das Löschen und Modifizieren von Fenstern wird vom Fenstermanager ebenfalls unterstützt. Wurde ein Fenster erfolgreich angelegt, kann die Anwendung mittels EGL Ressourcen allokieren und Befehle für OpenGL ES an die 3D-GPU senden. Der GPU-Scheduler ist dafür verantwortlich, dass die Grafikbefehle der unterschiedlichen Anwendungen entsprechend ihrer Anforderungen abgearbeitet werden. Die Anwendungen bekommen Prioritäten entsprechend ihrer Kritikalität zugewiesen, die die Abarbeitung ihrer Grafikbefehle beeinflussen. Neben den Prioritäten besitzt jede Anwendung eine Aktualisierungsrate, die angibt, in welchen zeitlichen Abständen Bilder auf der GPU berechnet werden sollen. Dies dient in erster Linie dazu, dass Anwendungen mit einer hohen Priorität ihre erforderliche Aktualisierungsrate erreichen und gleichzeitig noch Ressourcen für niedrig priorisierte Anwendungen zur Verfügung stehen. Der GPU-Scheduler ist dafür verantwortlich, die Befehle entsprechend einzureihen. Um die Befehle in die richtige Reihenfolge zu bringen, muss die Verarbeitungsdauer der Befehle auf der GPU bestimmt werden. Dazu wird die Ausführungszeit der Befehle abgeschätzt und eine optimale Reihenfolge der Befehle bestimmt, die den Anforderungen der Anwendungen entspricht.

Sobald die Grafikbefehle auf der GPU abgearbeitet und die Bilder in die Offscreen-Puffer der Anwendungen abgelegt wurden, werden die Fenster der Anwendungen im Anzeigespeicher mittels des Compositings aktualisiert. Hierzu werden die Bitmasken der Fenster verwendet, die das Bitblitting auf Basis einzelner Pixel erlauben. Der Anzeigespeicher muss ebenfalls aktualisiert werden, wenn der Fenstermanager Änderungen an den Fenstern vornimmt, wie das Anlegen eines neuen oder das Verkleinern eines bestehenden Fensters. Dazu wurde das Compositing aus Kapitel 4 implementiert. Das Compositing nutzt für das Kopieren der Offscreen-Puffer die Bitblitting-Funktionen der GPU GC320.

Die Eingabeereignisse, die von den Lenkradtasten und dem Dreh-Drück-Steller über die CAN-Brücke an den Eingabemanager via TCP/IP gesendet werden, werden vom Eingabemanager an die entsprechenden Anwendungen verteilt. Da nach der Anforderung R1 aus Kapitel 2 die Eingabeereignisse ebenfalls einer Zugriffs-

kontrolle unterliegen müssen, benötigt eine Anwendung eine Berechtigung bevor sie Eingabeereignisse empfangen kann. Hierzu wurde dasselbe Prinzip wie bei dem Zugriffskontrollmodell für Anzeigebereiche verwendet. Ein Eingabeereignis stellt ähnlich einem Pixel ein atomares Objekt im Zugriffskontrollmodell dar. Eine Menge aus Eingabeereignissen repräsentiert ein Objekt. Das heißt, ähnlich den Anzeigebereichen können Anwendungen für Eingabeereignisse kontext-beschränkte Berechtigungen weitergeben, welche dann abhängig des Kontexts geschaltet werden. Somit können Eingabeereignisse nur an genau eine Anwendung zur selben Zeit vergeben werden. Der Eingabemanager ist dafür verantwortlich, dass die Eingabeereignisse entsprechend den geschalteten Berechtigungen an die Anwendungen verteilt werden. Erhält der Eingabemanager ein Eingabeereignis, dann fragt er den Berechtigungsmanager, welche Anwendung die Berechtigung für das Eingabeereignis besitzt. Dieser sendet ihm die ID der Anwendung, woraufhin der Eingabemanager das Eingabeereignis an die betreffende Anwendung weiterleitet.

Der Systemmonitor sammelt die Statusinformationen aller Anwendungen und Kontexte. Dies ermöglicht es, den Ausfall einer Anwendung bzw. einer virtuellen Maschine zu erkennen und den Zustand der Kontexte darzustellen.

5.3. Demonstration verschiedener Szenarien

Für die Demonstration wurde ein Aufbau gewählt, welcher mit typischen Anwendungen aus Kombiinstrumente und HU verschiedene Szenarien darstellen kann. Die Anwendungen Geschwindigkeitsanzeige, Tachometer, Warnanzeigen, Reisecomputer, Fahrzeugstatus und IC-Menü werden auf der Anzeige dargestellt, wie in Abbildung 5.4 zu sehen ist. Die Anwendungen Navigation, DVD-Wiedergabe, Radio und ein Spiel verwenden die Anzeige der HU wie in Abbildung 5.5 mit dem Spiel „Quake 3“ zu sehen ist.

Die Szenarien werden mittels kontext-beschränkten Berechtigungen definiert.

Abbildung 5.4.: **Fahrzeugszenario** (Anwendungen Kombiinstrument)

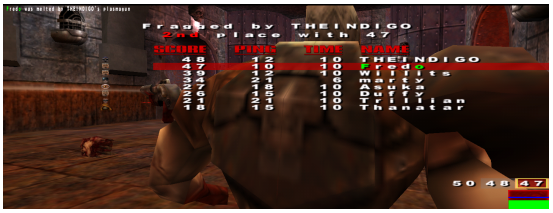

Abbildung 5.5.: **Fahrzeugszenario** (Headunit mit Spiel „Quake 3“)

Dazu werden kontext-beschränkte Berechtigungen in den XML-Dateien festgelegt, die abhängig des Kontexts geschaltet werden. Die Anwendungen lesen die XML-Dateien während dem Start ein und senden die zu vergebenen kontext-beschränkten Berechtigungen entsprechend an den Berechtigungsmanager. Dieser vergibt die kontext-beschränkten Berechtigungen, sobald die Voraussetzungen für die Vergabe der kontext-beschränkten Berechtigungen erfüllt sind (vgl. Abschnitt 3.4). Durch das Schalten von Kontexten kann dann der Zugriff auf die Anzeigebereiche geändert und somit auch verschiedene Szenarien dargestellt werden.

Im Folgenden wird die webbasierte Steuerung der Szenarien beschrieben, die das Schalten der Kontexte über einen webbasierten Dienst ermöglicht.

5.3.1. Webbasierte Steuerung der Szenarien

Die Kontexte stellen teils fahrzeuginterne Zustände dar, die z. B. über Sensoren ermittelt und dann über Schnittstellen wie CAN an den Infotainment-Domänenserver übermittelt werden. Für die Demonstration wurde eine webbasierte Steuerung implementiert, die das Aktivieren und Deaktivieren von Kontexten zulässt. Dies erlaubt es, die Zustände der Kontexte zu simulieren, indem die Auswahl der Kontexte über eine Webseite erfolgt und der Zustand ausgewählt wird (siehe Abbildung 5.6). Dazu wurde eine HTLM5-basierte Webapplikation implementiert, die auf einer Raspberry Pi Plattform mittels eines Apache Webservers betrieben wird und die Statusinformationen und Steuerbefehle per TCP/IP-Verbindung mit der i.MX6 Plattform austauscht. Auf der i.MX6 Plattform werden die Informationen mittels des Systemmonitors im Virtualisierungsmanager gesammelt und an die Webapplikation gesendet, die die Informationen entsprechend darstellt. Neben den Informationen über die Kontexte werden auch Informationen über

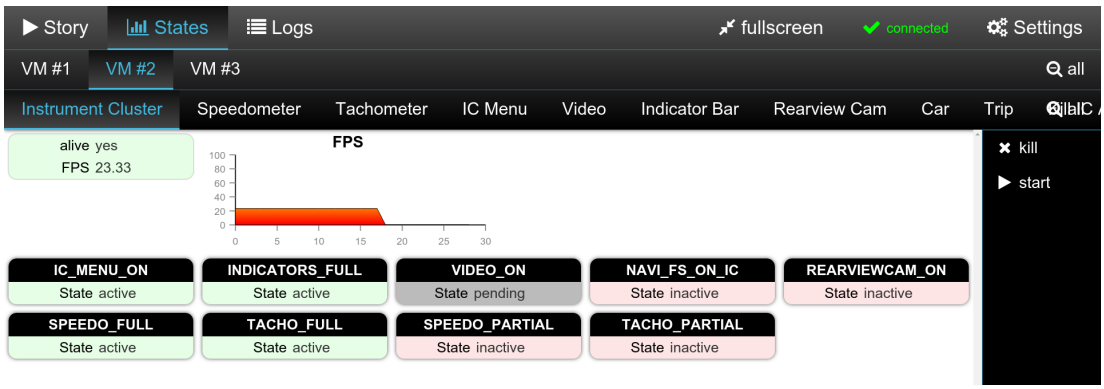

Abbildung 5.6.: Webbasiertes Schalten der Kontexte

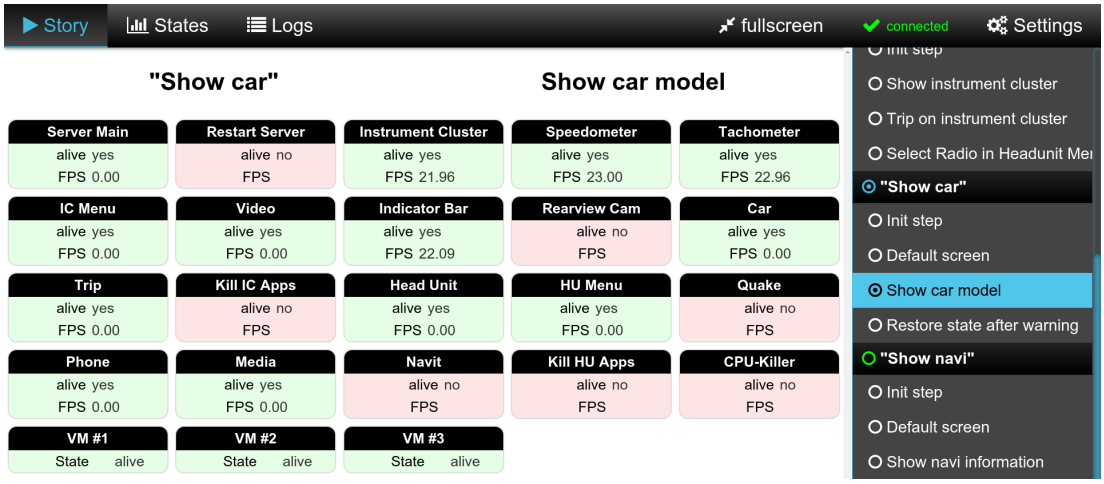

Abbildung 5.7.: Auswahl und Zustände von Anwendungen und Szenarien

den Zustand der Anwendungen und den virtuellen Maschinen übermittelt. Dies bedeutet, es kann ebenfalls dargestellt werden, ob eine Anwendung bzw. eine virtuellen Maschine läuft oder abgestürzt ist. Außerdem werden Informationen, wie die Aktualisierungsrate der Anwendungen, dargestellt. Zudem können die Anwendungen über die Webapplikation gestartet und beendet werden. Dazu werden die Steuerbefehle an den entsprechenden Systemmonitorklient in der virtuellen Maschine gesendet, welcher dann die Anwendung startet oder beendet.

Da für die Steuerung der Szenarien das Schalten einzelner Anwendungen und Kontexte zu aufwändig wäre, wurde eine Steuerung der Szenarien mittels einer Auswahlliste erstellt, wie in Abbildung 5.7 rechts dargestellt. Das heißt, per Auswahl in der Webapplikation kann ein Szenario gewählt werden, was zur Folge hat, dass die für das Szenario benötigten Anwendungen gestartet werden. Mittels mehrerer Unterschritte können für das Szenario die Zustände der Kontexte geschaltet werden, sodass eine Demonstration unterschiedlicher Fensterkonfigurationen möglich ist. Auf der linken Seite der Abbildung 5.7 sind die Anwendungen, die virtuellen Maschinen und die Zustände (aktiv oder inaktiv) dargestellt. Des Weiteren wird die Aktualisierungsrate der Anwendungen gezeigt.

Im folgenden Abschnitt werden die Szenarien in einer Übersicht dargestellt.

5.3.2. Szenarien

Der Demonstrator beinhaltet Konzepte, die die funktionale Sicherheit betreffen und durch die Verwendung einer breit gefächerten Auswahl an Anwendungen und Situationen demonstriert werden sollen. Die Evaluation der Kernkonzepte, wie die kontext-basierte Zugriffskontrolle für Anzeigebereiche in Kapitel 3 und das effizi-

ente Compositing von Anzeigebereichen in Kapitel 4 ist in den jeweiligen Kapiteln beschrieben. Um das Zusammenspiel der Konzepte und die flexible Nutzung der Anzeigen zu demonstrieren, wurde eine Handlung geschrieben, die die Nutzung eines Infotainment-Domänenservers beschreibt und aus einzelnen Situationen bzw. Szenarien besteht. Ein Szenario besteht aus bestimmten Anwendungen, zugehörigen kontext-beschränkten Berechtigungen, Kontexten und Eingabeereignisse, die durch die Eingabegeräte wie Lenkradtasten erzeugt werden. Die Handlung beinhaltet 7 Situationen, in welchen der Fahrer mit dem System interagiert. Zuerst wird die Handlung, gefolgt von den Szenarien, beschrieben.

„Adam und ein Kollege müssen geschäftlich mit dem Flugzeug verreisen und fahren dazu zum Flughafen. Nach dem Verlassen des Hauses begibt Adam sich zu seinem Fahrzeug und setzt sich auf den Fahrersitz. Nachdem per Start/Stopp-Knopf die Verbraucher im Fahrzeug mit Strom versorgt werden, startet das Infotainmentsystem des Fahrzeugs und stellt das Menü der Kombiinstrumente dar. Des Weiteren wird der letzte Zustand der HU dargestellt. Dies war die Wiedergabe eines Videos, die automatisch fortgesetzt und auf der Anzeige der HU dargestellt wird. Adam startet den Motor. Die Kombiinstrumente werden eingeblendet. Dann legt er den Rückwärtsgang ein und die Rückfahrkamera wird automatisch in der Mitte der Anzeige der Kombiinstrumente eingeblendet. Die Geschwindigkeitsanzeige und der Tachometer werden an den rechten bzw. linken Rand verschoben und sind zur Hälfte sichtbar. Dies vergrößert die Darstellung der Rückfahrkamera deutlich. Des Weiteren wird das Video beendet. Nachdem Adam ausgeparkt hat, legt er den Vorwärtsgang ein und fährt zu seinem Kollegen, um ihn abzuholen. Wenige Minuten vor seiner Ankunft öffnet er im Menü der Kombiinstrumente die Kontakte, da er seinen Kollegen über seine Ankunft per Telefon informieren möchte. Für einen besseren Überblick verschiebt er per Menü die Darstellung der Kontakte auf die Anzeige der HU. Kurz vor Ankunft wird eine Warnmeldung auf der Anzeige der Kombiinstrumente angezeigt. Dazu wird ein rotierendes Fahrzeug dargestellt, welches den Fehler anzeigt. Der Luftdruck der Reifen ist niedrig. Adam nimmt die Warnung zur Kenntnis und die Meldung wird wieder ausgeblendet. Der Kollege wartet bereits vor der Haustüre auf Adam und steigt zu. Adam startet das Radio, um die Verkehrsnachrichten zu hören. Zusätzlich startet Adam die Navigation auf der Anzeige der Kombiinstrumente. Daraufhin fährt er nach den Navigationsanweisungen Richtung Flughafen. Um sich die Zeit zu vertreiben, startet der Kollege auf der nur für ihn einsehbaren Anzeige (siehe Dual-View-Display [Spl16]) ein Spiel, welches er über ein Portal eines Drittanbieters bezogen hat. Durch das Starten des Spiels kommt es zu keinen Verzögerungen in der Darstel-

lung der Kombiinstrumente, welche auf derselben Hardware ausgeführt werden. Nach wenigen Minuten spielen, stürzt das Spiel ab. Die anderen Anwendungen bleiben jedoch davon unbeeinflusst. Nachdem die Anwendung wieder zur Verfügung steht, spielt der Kollege bis sie den Flughafen erreichen. Adam parkt das Auto und beide begeben sich in Richtung Terminal."

Die in dieser Geschichte enthaltenen 7 Szenarien werden im Folgenden beschrieben. Je Szenario werden die Situationen aufgelistet, welche die Anwendungen und die Konfiguration zu Beginn und nach Durchführung einer Aktion beschreiben. Es werden die Aktionen, die durchgeführt werden, und das Schalten der entsprechenden Kontexte aufgelistet. Übersicht der 7 Szenarien:

(S0) Ausgangssituation: *Anwendungen:* Kombiinstrumente und HU.
Situation: Das System ist gestartet und die virtuellen Maschinen für die Kombiinstrumente und der HU stehen zur Verfügung (siehe Abbildung 5.8).

Abbildung 5.8.: Ausgangssituation (S0) mit gestarteten virtuellen Maschinen für die Kombiinstrumente und der HU

(S1) Anzeigen des letzten Zustandes: *Anwendungen:* Video und Menü der Kombiinstrumente.
Beginn: Kombiinstrumente und HU sind gestartet (vgl. Abbildung 5.8).
1. Aktion: Zuletzt gewählte Anwendungen werden dargestellt.
Situation: Das Video wird auf der Anzeige der HU angezeigt und das Menü auf der Anzeige der Kombiinstrumente (siehe Abbildung 5.9).
Beschreibung: Szenario S1 demonstriert das Wiederherstellen des letzten Zustandes mittels Kontexten. Die vormals durch Benutzereingaben gewählten Kontexte werden gespeichert und bei Neustart verwendet, um die letzten Berechtigungen wiederherzustellen.

(S2) Starten des Motors und Ausparken des Fahrzeugs: *Anwendungen:* Geschwindigkeitsanzeige, Tachometer, Menü, Kontrollleuchten, Rückfahrkamera.

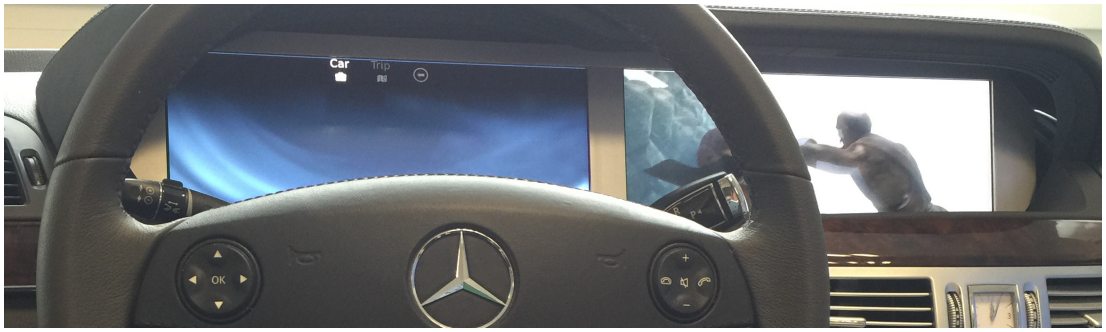

Abbildung 5.9.: Szenario (S1) mit Darstellung des Videos auf der Anzeige der HU

Beginn: Das Video und das Menü werden dargestellt.
1. Aktion: Anwender startet den Motor.
Situation: Kombiinstrumente werden eingeblendet.
2. Aktion: Anwender legt den Rückwärtsgang ein.
Situation: Rückfahrkamera wird auf der Anzeige der Kombiinstrumente eingeblendet, Kombiinstrumente sind nur teils sichtbar (siehe Abbildung 5.10), das Video wird beendet und das Menü auf der Anzeige der HU dargestellt.
3. Aktion: Anwender legt den Vorwärtsgang ein.
Endzustand: Kombiinstrumente sind sichtbar, Menü ist auf Anzeige HU.
Beschreibung: Szenario S2 demonstriert die Verwendung von Kontexten, welche durch den Zustand des Fahrzeuges erzeugt werden. Das Starten des Motors und das Einlegen des Rückwärtsgangs sind Kontextänderungen, die das Darstellen der Kombiinstrumente bzw. der Rückfahrkamera erforderlich machen und das Schalten entsprechender kontext-beschränkter Berechtigungen. Die Darstellung von bewegten Bildern wie Videos ist während der Fahrt verboten und macht das Entziehen der Berechtigung erforderlich.

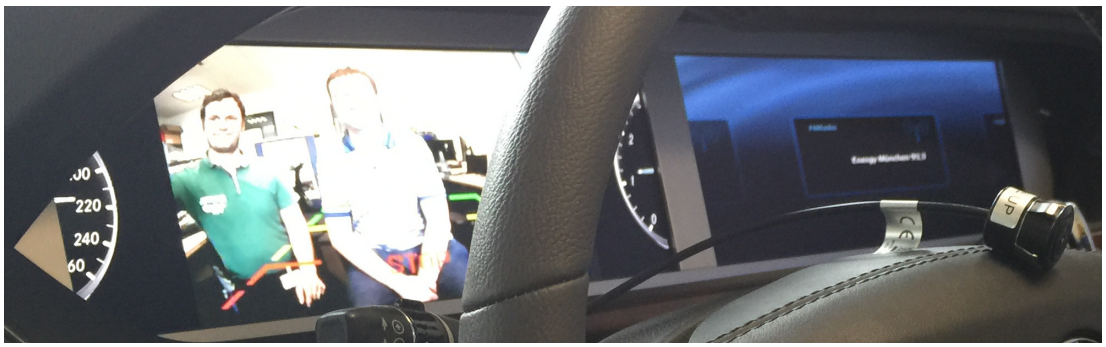

Abbildung 5.10.: Szenario (S2) mit eingeblendeter Rückfahrkamera auf der Anzeige der Kombiinstrumente

(S3) Auswahl der Kontakte: *Anwendungen:* Kontakte.
Beginn: Kombiinstrumente sind sichtbar, Menü ist auf Anzeige HU.
1. Aktion: Anwender wählt Kontakte über IC-Menü aus.
Situation: Kontakte werden auf der Anzeige der Kombiinstrumente gezeigt.
2. Aktion: Anwender schiebt Kontakte auf Anzeige HU mittels IC-Menü.
Situation: Kontakte werden zusätzlich auf Anzeige HU dargestellt.
Endzustand: Kontakte werden auf der Anzeige der HU und der Kombiinstrumente dargestellt (siehe Abbildung 5.11).
Beschreibung: Szenario S3 demonstriert die flexible Nutzung der Anzeigen. Durch die gemeinsame Nutzung der Anzeigen kann mittels Austausch der Berechtigung die Anwendung von einer Anzeige auf der anderen Anzeige zusätzlich dargestellt bzw. auf diese verschoben werden. Die Eingabeereignisse werden ebenfalls über kontext-beschränkte Berechtigungen verteilt.

(S4) Starten der Radioanwendung: *Anwendungen:* Radioanwendung.
Beginn: Kombiinstrumente sind sichtbar, Kontakte sind auf Anzeige HU.
1. Aktion: Anwender wählt Radioanwendung über das HU-Menü aus.
Situation: Die Radioanwendung wird auf der Anzeige der HU dargestellt.
Endzustand: Kombiinstrumente und Reisecomputer sind sichtbar, die Radioanwendung ist auf der Anzeige der HU sichtbar (siehe Abbildung 5.12).
Beschreibung: Szenario S4 demonstriert die Auswahl der Anwendungen mittels Kontexten. Die Benutzereingaben im Menü stellen ebenfalls Kontextänderungen dar. Eine Anwendung, welche im Menü ausgewählt wird, erhält eine Berechtigung durch die Änderung des Kontexts. Einer bereits dargestellten Anwendung wird die Berechtigung entzogen.

(S5) Warnmeldung liegt vor: *Anwendungen:* Warnmeldung mittels 3D-Fahrzeug.
Beginn: Kombiinstrumente sind sichtbar, Radio ist auf Anzeige HU.

Abbildung 5.11.: Szenario (S3) mit Kontakten auf der Anzeige der HU

Abbildung 5.12.: Szenario (S4) mit Radioanwendung auf der Anzeige der HU

1. Aktion: Warnmeldung liegt vor.
Situation: Warnmeldung mittels 3D-Fahrzeug wird zwischen Geschwindigkeitsanzeige und Tachometer eingeblendet (siehe Abbildung 5.13).
2. Aktion: Warnmeldung liegt nicht mehr vor.
Situation: Warnmeldung blendet sich aus.
Endzustand: Kombiinstrumente sind sichtbar, Radio auf Anzeige HU.
Beschreibung: Szenario S5 demonstriert die priorisierte Darstellung von Anwendungen. Da die Warnmeldungen eine höhere Priorität im Vergleich zu Anwendungen wie Reisecomputer haben, wird bei Eintreten einer Warnung mittels Kontextwechsel die Berechtigung der dargestellten Anwendung entzogen und der Warnmeldung zugewiesen.

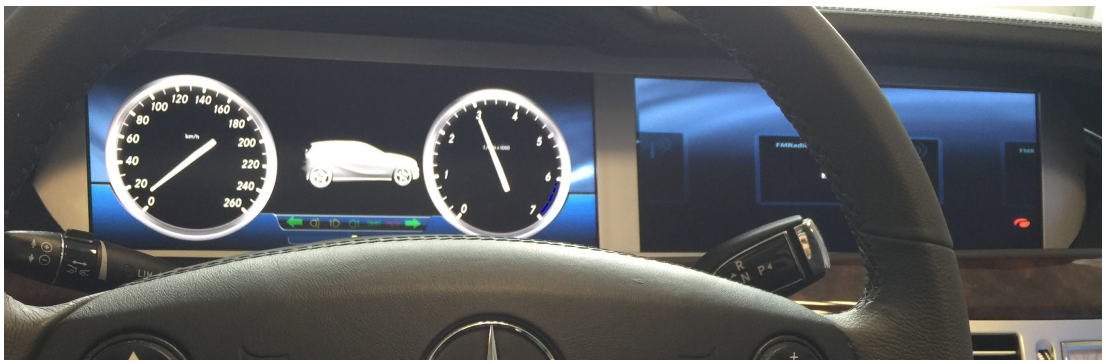

Abbildung 5.13.: Szenario (S5) mit Warnmeldung mittels 3D-Fahrzeug

(S6) Starten der Navigation: *Anwendungen:* Navigation.
Beginn: Kombiinstrumente sind sichtbar, Radioanwendung wird auf der Anzeige der HU dargestellt.
1. Aktion: Anwender wählt die Navigation über das IC-Menü aus.
Situation: Navigation wird auf der Anzeige der Kombiinstrumente gezeigt.
Endzustand: Kombiinstrumente sind sichtbar, Navigation wird auf der Anzeige der Kombiinstrumente dargestellt, Radioanwendung wird auf der An-

zeige der HU dargestellt (siehe Abbildung 5.14).

Beschreibung: Szenario S6 demonstriert die pixel-definierten Fenster und den konfliktfreien Zugriff auf die Anzeigen. Die Anwendung Navigation erhält den Zugriff auf eine nicht rechteckige Fläche, welche zwischen dem Tachometer und der Geschwindigkeitsanzeige liegt. Durch die Definition der kontext-beschränkten Berechtigungen mittels Bitmasken ist eine Zugriffsberechtigung auf Basis einzelner Pixel möglich. Das Compositing lässt zudem die Definition von Fenstern auf Basis von Bitmasken zu und gewährleistet, dass nur die berechtigten Pixel geschrieben werden. Dies ermöglicht den konfliktfreien Zugriff auf die Anzeigen.

(S7) Spielen eines Spieles: *Anwendungen:* Spiel „Quake 3“.

Beginn: Kombiinstrumente sind teils sichtbar, Radioanwendung wird auf der Anzeige der HU dargestellt.

1. Aktion: Anwender startet über das HU-Menü das Spiel.

Situation: Spiel wird auf der Anzeige der HU gezeigt (siehe Abbildung 5.15).

2. Aktion: Spiel bringt Partition HU (VM3) zum Absturz.

Situation: Partition und Spiel sind abgestürzt.

Endzustand: Kombiinstrumente sind sichtbar, Partition der Kombiinstrumente laufen weiter, Partition der HU muss neu gestartet werden.

Beschreibung: Szenario S7 demonstriert die Verwendung der offenen Plattform, die mittels Virtualisierung die sichere Ausführung der sicherheitskritischen Anwendungen gewährleistet. Dies wird durch 3D-GPU-Scheduling unterstützt, das die sichere Ausführung von sicherheitskritischen und nicht-sicherheitskritischen Anwendungen auf der GPU anhand von Prioritäten ermöglicht. Das Spiel wird in einer virtuellen Maschine ausgeführt, die nicht-sicherheitskritischen und nicht-vertrauenswürdigen Anwendungen beinhaltet. Der Absturz des Spiels beeinflusst die Kombiinstrumente nicht. Durch

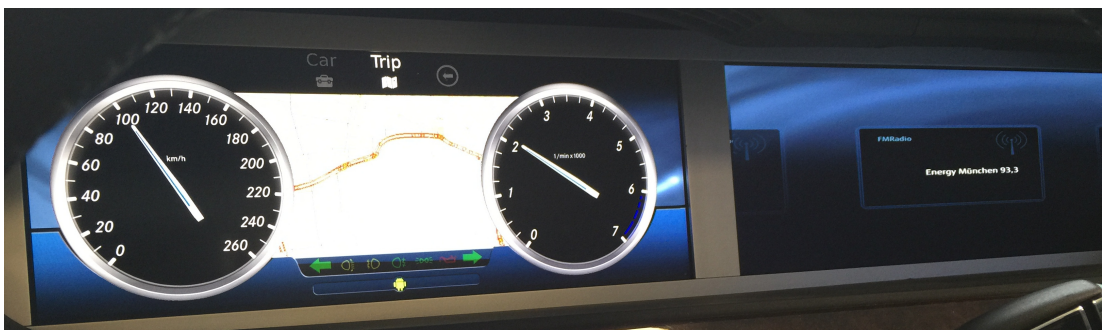

Abbildung 5.14.: Szenario (S6) mit Navigation auf Kombiinstrumente-Anzeige

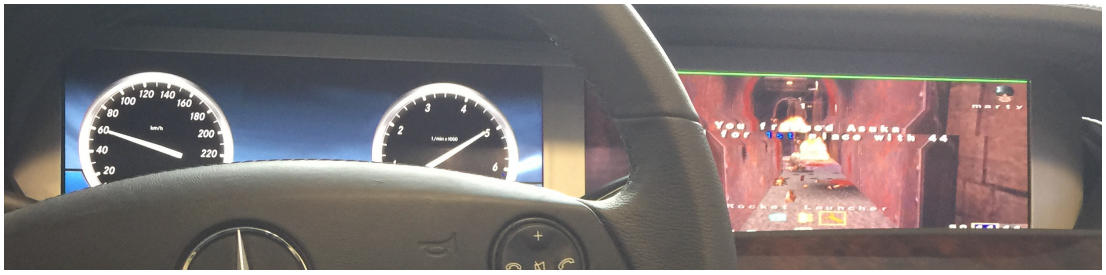

Abbildung 5.15.: Szenario (S7) mit Spiel „Quake 3“ auf der Anzeige der HU

die Verwendung kontext-beschränkter Berechtigungen kann die Ausführung bzw. die Sichtbarkeit für den Fahrer während der Fahrt eingeschränkt werden. Zum Beispiel können die Beifahrer nur das Dual-View-Display oder die Rücksitzanzeigen nutzen. Dies ermöglicht die Installation von Anwendungen seitens des Anwenders, die nicht konform der Automobil-Richtlinien sind. Die Beschreibung der virtuellen Plattform ist in Kapitel 2.1 zu finden.

5.4. Evaluation

Der Demonstrator zielt in erster Linie auf die Anforderungen R1, R2, R3, R4 und R5 aus Abschnitt 2.1 ab. Während die Anforderung R6 (Zertifizierbarkeit) den Entwicklungsprozess betrifft, welcher in dieser Arbeit nicht im Fokus steht, wurde die Anforderung R7 (Systemüberwachung) nur in einem eingeschränkten Rahmen betrachtet. Wie bereits in der Systemarchitektur eines Infotainment-Domänenservers (vgl. Abbildung 2.2) aufgelistet, sind für die Anforderung R7 die Komponenten „Systemmonitor“, „Watchdog“ und „Auditing“ notwendig. Im Rahmen der Demonstration wurde jedoch nur die Komponente „Systemmonitor“ implementiert, welche verschiedene Statusinformationen des Systems empfängt und Reaktionen seitens einer Eingabemöglichkeit über eine Webapplikation ausführt. Die vollständige Systemüberwachung stand hier nicht im Vordergrund.

Da der Demonstrator die Konzepte bzw. die relevanten Komponenten für das Zugriffskontrollmodell für Anzeigebereiche (siehe Kapitel 3) und das effiziente Bitblitting (siehe Kapitel 4) verwendet, sind die Anforderungen R2 und R5 erfüllt. Die Anforderung R3 (Vertrauenswürdiger Kanal) wurde in Form der isolierten Kanäle auf Basis gemeinsam genutzten Speichers umgesetzt. Der Speicher, den die Anwendungen für ihre Kanäle nutzen, ist im Adressbereich jeder Anwendungen isoliert abgebildet, sodass jede Anwendung nur auf ihren Kanal zugreifen kann. Dies impliziert die Integrität und Vertraulichkeit (R3.1) der Kommunikation zwischen den Anwendungen und dem Virtualisierungsmanager. Des Weiteren

ist durch die dedizierten Kanäle, welche jede Anwendung zu dem Virtualisierungsmanager unterhält, die Authentifikation und Nachweisbarkeit (R3.2) gegeben.

Die Anforderungen an die Eingabeverarbeitung (R1) wird mit Hilfe der Komponente Eingabemanager und den kontext-beschränkten Berechtigungen für die Eingabeereignisse erfüllt. Durch die Verwendung kontext-beschränkter Berechtigungen ist die Zugriffskontrolle (R1.1) für Eingabeereignisse gegeben. Die zeitlich beschränkte Verarbeitungsdauer (R1.2) wurde anhand von Messungen ermittelt. Die maximale Latenz für die Übertragung mittels CAN und die Verarbeitung der Eingabeereignisses lag unter 25 ms und damit unter den geforderten 250 ms.

Des Weiteren ist die Anforderung R4 für die virtualisierte Grafikberechnung mittels der Komponente „GPU-Scheduler“ umgesetzt. Dies bedeutet, jede Anwendung hat eine Priorität (R4.1), welche sich aufgrund ihrer Kritikalität ableiten lässt. Für die zeitliche Beschränkung bei der Erstellung der grafischen Inhalte (R4.2) ist das Schalten eines grafischen Kontexts entscheidend, welcher für die Erstellung der grafischen Inhalte nötig ist und innerhalb der geforderten zeitlichen Beschränkung stattfindet. Um eine Isolation der GPU-Ressourcen (R4.3) zu gewährleisten, verteilt der GPU-Scheduler entsprechend den Prioritäten den Zugriff auf die GPU-Ressourcen und gewährleistet damit, dass die sicherheitskritischen Anwendungen die benötigten Ressourcen bekommen. Da die virtualisierte Grafikberechnung nicht im Fokus dieser Arbeit steht, sei an dieser Stelle auf die Arbeiten von Schnitzer et al. [SGDR14, SGDR16] und weitere zukünftige Veröffentlichungen verwiesen.

5.5. Zusammenfassung

Im Rahmen des Projektes ARAMiS ist ein VCT-Demonstrator entstanden, welcher die Konsolidierung der Steuergeräte der Kombiinstrumente und der HU in Form eines Infotainment-Domänenservers demonstriert. Mittels der Technologie Virtualisierung wird die Isolation der sicherheitskritischen Anwendungen von den nicht-sicherheitskritischen und unsicheren Anwendungen gewährleistet. Die Konzepte für das sichere und zuverlässige Teilen von Ressourcen, welche im Rahmen von ARAMiS entwickelt wurden, wurden im VCT-Demonstrator implementiert und anhand geeigneter Szenarien demonstriert. Hierbei konnte gezeigt werden, dass sich die Konzepte für die Umsetzung eines Infotainment-Domänenservers mittels Virtualisierung eignen und auf einer realen fahrzeugtauglichen Plattform umgesetzt werden können. Die Leistungsfähigkeit des Systems konnte anhand der unterschiedlichen Szenarien gezeigt werden.

6. Zusammenfassung und Ausblick

Dieses Kapitel fasst die wesentlichen Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf mögliche Erweiterungen.

6.1. Zusammenfassung

Diese Arbeit behandelt den sicheren Zugriff auf die gemeinsam genutzten grafischen Hardwareressourcen von sicherheitskritischen und nicht-sicherheitskritischen Anwendungen im Fahrzeug, welche durch eine Konsolidierung der HU und der Kombiinstrumente auf derselben Hardwareplattform ausgeführt werden. Mittels der Technologie Virtualisierung wird eine Isolation zwischen den Anwendungen, die sich in verschiedenen virtuellen Maschinen befinden, gewährleistet. Diese Isolation bezieht sich jedoch in der Regel nur auf die CPU und den Speicher. Der konfliktfreie Zugriff auf die Hardwareressourcen der GPU und der Anzeigen im Fahrzeug wird bislang nicht abgedeckt und soll im Rahmen dieser Arbeit betrachtet werden.

Im ersten Teil der Arbeit wurden die Anforderungen an ein grafisches System im Fahrzeug analysiert, welche für eine Konsolidierung von HU und Kombiinstrumente mittels Virtualisierung notwendig ist. Dazu wurden verschiedene Richtlinien wie ISO-Standards und gesetzliche Anforderungen analysiert und Anforderungen, die für ein grafisches System im Fahrzeug relevant sind, abgeleitet. Besonders im Fokus stand hierbei das Begrenzen der Ablenkung des Fahrers durch die Anwendungen von HU und Kombiinstrumente, sowie die funktionale Sicherheit, welche insbesondere für die sicherheitskritischen Anwendungen relevant ist. In dieser Arbeit wurden sieben Kategorien von Anforderungen aufgestellt, die sich aus den unterschiedlichen Richtlinien ableiten lassen und jeweils detaillierte Anforderungen enthalten. Auf Grundlage dieser Anforderungen wurde eine Systemarchitektur für einen Domänen-Server vorgeschlagen, der aus mehreren Komponenten besteht und die Anforderungen erfüllt.

Im zweiten Teil der Arbeit wurde auf Grundlage dieser Systemarchitektur ein Zugriffskontrollmodell für die Anzeigen im Fahrzeug vorgestellt. Da eine flexible Nutzung der Anzeigebereiche seitens der sicherheitskritischen und nicht-sicherheitskritischen Anwendungen möglich sein soll, muss der Zugriff anhand von Berechtigungen beschränkt werden. Hierbei hängt die Berechtigung für einen Anzeigebereich nicht nur von der Anwendung selbst ab, sondern auch von der Situation, in welcher sich das Fahrzeug und die Anwendungen befinden. Dies wird durch Kontexte repräsentiert, welche dynamisch aufgrund von Sensorinformationen des Fahrzeugs (z. B. die Geschwindigkeit) oder aus Zuständen der Anwendungen (z. B. welcher Eintrag wurde in der Auswahlliste ausgewählt) ermittelt werden. Der Zugriff auf einen Anzeigebereich wird nur dann gewährt, wenn eine Berechtigung für die aktuell gegebenen Kontexte existiert. Für eine flexible Nutzung der Anzeigen durch Drittanbieter von Anwendungen im Fahrzeug wurde eine Möglichkeit zur Delegation von Berechtigungen zwischen den Anwendungen vorgeschlagen. Das Zugriffskontrollmodell selbst ist vollständig formal definiert und erlaubt dadurch die formale Beweisführung von Eigenschaften im Modell. Es wurde gezeigt, dass beispielsweise der Zugriff auf einen Anzeigebereich mittels des Zugriffskontrollmodells auch bei Vergabe neuer Berechtigungen immer konfliktfrei erfolgt, wenn die initiale Konfiguration diese Eigenschaft erfüllt. Anhand einer Implementierung des Zugriffskontrollmodells wurde die Latenz analysiert, die sich durch die Zugriffsberechtigung ergibt. Hierbei wurde gezeigt, dass in realistischen Szenarien die Latenz für eine bestimmte Anzahl an Anwendungen, welche gleichzeitig den Zugriff erhalten, die Anforderungen erfüllt.

Im dritten Teil der Arbeit wurde das Compositing der Anwendungsfenster auf die Anzeigen betrachtet. Nachdem eine Anwendung mittels Berechtigung Zugriff auf einen Anzeigebereich erhalten hat, kann sie ein Fenster für diesen Bereich anmelden. Das Compositing ist dann dafür verantwortlich, dass die Fenster aller Anwendungen entsprechend ihrer Position und Größe auf der Anzeige dargestellt werden. Im Rahmen dieser Arbeit wurden zwei verschiedene Möglichkeiten betrachtet, wie Fenster definiert werden können. Entweder die Fenster werden anhand eines Rechtecks definiert, welches mittels Position, Höhe und Breite eindeutig auf den Anzeigen positioniert werden kann, oder das Fenster wird mittels Bitmasken in beliebiger Form definiert. Die rechteckig definierten Fenster finden in den meisten Fenstersystemen Anwendung. Es existieren zwei Ansätze für das Compositing mit rechteckigen Fenstern, die sich auch gegenseitig überdecken können. Auf Grundlage dieser Ansätze wurde ein Hybridansatz vorgestellt, welcher die Vorteile der zwei Ansätze nutzt, um ein effizientes Compositing durchführen

zu können, was anhand von Evaluationen aufgezeigt wird. Das Compositing mit Fenstern, die mittels Bitmasken in beliebiger Form definiert sind, eignet sich ideal, um die durch das Zugriffskontrollmodell unterstützten pixel-definierten Berechtigungen durchgängig zu verwenden. Es wurde ein Ansatz vorgestellt, bei dem das Compositing gewährleistet, dass nur die Pixel auf der Anzeige geschrieben werden, welche in der Berechtigung für den Zugriff mittels Bitmaske entstanden sind. Mittels Evaluationen wurde gezeigt, dass das Compositing mittels Bitmasken zwar einen Mehraufwand im Vergleich zu den rechteckigen Fenstern darstellt, aber dennoch für Szenarien im Fahrzeug anwendbar ist.

Im vierten Teil der Arbeit wurde der Demonstrator in Form eines Cockpits beschrieben, welcher im Rahmen des Projektes ARAMiS entstanden ist. Dieser enthält die wesentlichen Konzepte der zuvor beschrieben Teile dieser Arbeit und demonstriert sie anhand von Szenarien im Fahrzeug. Anhand dieser Implementierung kann die Funktionalität und Leistungsfähigkeit der beschrieben Konzepte mit Anwendungen gezeigt werden, wie sie auch im Fahrzeug verwendet werden.

6.2. Ausblick

Im Rahmen dieser Arbeit wurde eine Systemarchitektur für einen Domänen-Server vorgestellt, wie er im Fahrzeug als Konsolidierung der Steuergeräte HU und Kombiinstrumente eingesetzt werden könnte. Hierbei wurden Anforderungen aufgestellt und beschrieben, welche durch die konzeptionellen Teile dieser Arbeit erfüllt werden können. In mehreren Bereichen sind jedoch noch Arbeiten sinnvoll, die im Rahmen dieser Arbeit nicht betrachtet werden konnten.

Ein Bereich stellt die Kommunikation zwischen den virtuellen Maschinen dar. Im Unterschied zu der herkömmlichen IPC zwischen virtuellen Maschinen müssen bei der vorgestellten Systemarchitektur eines Domänen-Servers für jede Anwendung die grafischen Befehle an den Virtualisierungsmanager gesendet werden. Dies wurde im Rahmen des Demonstrators mittels gemeinsam genutzten Speicher durchgeführt, da die grafischen Daten und Befehle in hoher Anzahl und unterschiedlicher Größe übertragen werden müssen. Hierbei muss ein Kompromiss zwischen dem für die Übertragung bereitgestellten Speicher und der Latenz in der Übertragung gefunden werden. Für eine minimale Latenz kann im Idealfall so viel Speicher zur Verfügung gestellt werden, wie die maximale Größe der Befehle ist. Dadurch können die Daten mit wenigen Kopieroperationen übertragen werden. Der Nachteil daran ist, dass der begrenzte Speicher der eingebetteten Plattform dann nicht mehr für andere Zwecke zur Verfügung steht. Durch geeignete Konzep-

te könnte der Speicher optimal genutzt werden, z. B. durch dynamisches Zuweisen je nach Bedarf.

Ein weiterer Bereich, welcher nicht durchgängig betrachtet wurde, sind die Anforderungen der Systemüberwachung (R7) in Abschnitt 2.1. Bis auf den Systemmonitor, welcher im Rahmen des Demonstrators die verschiedenen Statusinformationen des Systems empfängt und Reaktionen seitens einer Eingabemöglichkeit über eine Webapplikation ausführt, wurde keine vollständige Systemüberwachung implementiert. Hierbei sollte der Systemmonitor noch weitere Zustände überwachen, die für die Darstellung von sicherheitskritischen Anwendungen notwendig sind. Neben den Berechtigungen und den Kontexten für die Anzeigebereiche sollte der Zugriff auf die GPU ebenfalls überwacht werden. Hierbei sind vor allem die Aktualisierungsraten der Anwendungen, aber auch Fehlerzustände in der Ausführung wichtig. Des Weiteren fehlen die Komponenten „Auditing“ und „Watchdog“, welche für eine Nachweisbarkeit die Zustände aufzeichnen bzw. entsprechende Aktionen im Fehlerfall durchführen sollten.

Eine ebenfalls nicht in dieser Arbeit im Fokus stehende Anforderung betrifft die Zertifizierbarkeit (R6) des Systems (siehe Abschnitt 2.1). Für den Einsatz im Fahrzeug wird seitens des OEMs in der Regel eine Zertifizierung nach ISO 26262 [ISO11] angestrebt. Die Verwendung der Methode FMEA [Sta03] dient zur Ermittlung der Kritikalitätslevels der Funktionalitäten und legt den Grad der notwendigen Absicherung fest. Durch die Virtualisierung kann eine Abstraktion in der Zertifizierung vorgenommen werden, die vor allen im Bereich der gegenseitigen Beeinflussung eine Vereinfachung bedeutet. Dennoch müssen die verwendeten Komponenten und die Architektur zertifiziert werden. Hierzu ist insbesondere eine Verifikation der Implementierung gegenüber der formalen Spezifikation hilfreich (z. B. mittels Model Checking [BBF+13]).

A. Mathematische Grundlagen

In diesem Abschnitt werden die mathematische Grundlagen beschrieben, die für ein Verständnis des formal beschrieben Zugriffskontrollmodells und für die Beweise der Korrektheitseigenschaften des Modells notwendig sind.

Das Zugriffskontrollmodell wird unter Verwendung mathematischer Begriffe formal definiert, womit es möglich ist, Beweise über formal definierte Eigenschaften des Modells zu führen. Hierzu wird insbesondere die Aussagenlogik verwendet, welche eine exakte und eindeutige Beschreibung eines Modells zulässt. Des Weiteren sind Grundlagen der Mengenlehre notwendig, da die Aussagen auf diesen aufbauen.

A.1. Aussagenlogik

Die Aussagenlogik ist ein Bereich der Logik und dient dazu, Aussagen einen Wahrheitswert zuzuordnen. Hierzu werden im Folgenden zuerst die Grundbegriffe nach [Sch00] und dann komplexere Operationen nach [Sch13, Hof11] beschrieben.

Grundbegriffe

Eine *Aussage A* ist ein Satz und hat entweder den *Wahrheitswert wahr* oder *falsch*. Eine Aussage kann weder gleichzeitig wahr und falsch sein, noch kann sie nur „halb“ wahr oder „halb“ falsch sein. Ein Beispiel für eine Aussage ist: A=„Berlin ist eine Stadt“. Durch die Interpretation des Inhaltes kann man herleiten, dass dies wahr ist. Gleichzeitig ist dies eine atomare Aussage, die nicht mehrere Teilaussagen hat. Mehrere Teilaussagen lassen sich durch einfache Verknüpfungen wie *und*, *oder* und *nicht* zusammensetzen bzw. negieren. Damit lässt sich z. B. Aussage A mit Aussage B=„Pferde können fliegen“ verknüpfen.

Da die Aussage B falsch ist, ist die Verknüpfung *(A und B)* auch falsch. Wohingegen *(A oder B)* wahr ist, da die Aussage A wahr ist. Für *nicht* gilt bei *nicht A*, dass die Aussage *nicht A* falsch ist, da der Wahrheitswert von A negiert wird.

Operationen

Diese logischen Verknüpfungen sind logische Operationen, auch *Junktoren* genannt, auf Aussagen. Die *Negation* der Aussage A, also *nicht A*, wird mittels des Symbols $\neg$ als $\neg A$ angegeben. Die *Konjunktion* zweier Aussagen, z. B. *A und B*, wird mittels des Symbols $\wedge$ als $A \wedge B$ angegeben. Die *Disjunktion* zweier Aussagen, z. B. *A oder B*, wird mittels des Symbols $\vee$ als $A \vee B$ angegeben.

Des Weiteren gibt es noch die Operationen *Implikation* und die *Äquivalenz*. Die Implikation wird mit dem Symbol $\Rightarrow$ dargestellt, z. B. A impliziert B mit $A \Rightarrow B$. Die Implikation kann auch mit Hilfe der Operationen Konjunktion und Negation dargestellt werden. Hierbei ist $A \Rightarrow B$ aussagenlogisch gleichbedeutend mit $\neg(A \wedge \neg B)$. Die Äquivalenz wird mittels des Symbols $\Leftrightarrow$ dargestellt, z. B. A ist äquivalent zu B mit $A \Leftrightarrow B$. Ähnlich zu der Implikation kann die Äquivalenz mit Hilfe von Disjunktion und Negation dargestellt werden. Hierbei ist $A \Leftrightarrow B$ aussagenlogisch gleichbedeutend mit $\neg A \vee B$. In Tabelle A.1 wird die Semantik aller logischen Operationen dargestellt.

Tabelle A.1.: Semantik der logischen Operationen für Aussagen

A	**B**	$\neg$ A	A $\wedge$ B	A $\vee$ B	A $\Rightarrow$ B	A $\Leftrightarrow$ B
falsch	**falsch**	wahr	falsch	falsch	wahr	wahr
falsch	**wahr**	wahr	falsch	wahr	wahr	falsch
wahr	**falsch**	falsch	falsch	wahr	falsch	falsch
wahr	**wahr**	falsch	wahr	wahr	wahr	wahr

Für die logischen Operationen gibt es verschiedene Rechenregeln bzw. Gesetze, die auf Aussagen angewandt werden können. Hierbei werden nur die für diese Arbeit relevanten Gesetze beschrieben. Für beliebige Aussagen *A, B, C* gelten die folgenden Äquivalenzen:

- Idempotenzgesetz: $A \wedge A \Leftrightarrow A$ bzw. $A \vee A \Leftrightarrow A$
- Kommutativgesetz: $A \wedge B \Leftrightarrow B \wedge A$ bzw. $A \vee B \Leftrightarrow B \vee A$
- Assoziativgesetz: $(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$ bzw.
  $(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$
- Distributivgesetz: $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$ bzw.
  $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$

Aussageformen und Quantoren

In der Mathematik werden in Aussagen häufig Variablen verwendet, welche in *Aussageformen p(x)* betrachtet werden. Die *Aussageformen p(x)* hängen dabei

Tabelle A.2.: Erklärung der Quantoren anhand des Beispiels Gl.1 (siehe [Hof11])

Symbolisch	gelesen	Beispiel Gl.1
$\forall x \in \mathbb{R} : p(x)$	Für alle Zahlen $x \in \mathbb{R}$ gilt $p(x)$	Für alle $x \in \mathbb{R}$ gilt $x^2 - 2x + 1 = 0$
$\exists x \in \mathbb{R} : p(x)$	Es existiert (mindestens) ein $x \in \mathbb{R}$, für das $p(x)$ gilt	Es existiert (mindestens) ein $x \in \mathbb{R}$, für dass $x^2 - 2x + 1 = 0$ gilt

von der Variable x ab. Ein Beispiel einer Aussageform (vgl. [Hof11]) sieht wie folgt aus:

$$p(x) : \text{Es gilt } x^2 - 2x + 1 = 0 \tag{Gl.1}$$

Bei Gl.1 hat die Aussageform keinen festen Wahrheitswert. Damit ein eindeutiger Wahrheitswert zugeordnet werden kann, muss hierbei für die Aussageform $p(x)$ entweder ein bestimmtes Objekt für x eingesetzt werden oder x muss mit einem *Quantor* gebunden werden. Beispielsweise würde die Aussageform $p(x)$ mit $x = 1$ folgende Aussage ergeben: $p(1) : \text{Es gilt } 1^2 - 2 \cdot 1 + 1 = 0$.

Die Aussage $p(1)$ ist wahr, beispielsweise $p(2)$ jedoch falsch (da $2^2 - 2 \cdot 2 + 1 \neq 0$).

Verwendet man die Quantoren $\forall$ (Allquantor oder auch Universalaussagen) und $\exists$ (Existenzquantor oder auch Existenzaussagen), dann werden die Variablen gebunden. Dies soll mit Beispiel Gl.1 in Tabelle A.2 erläutert werden.

Aus Aussageform $p(x)$ entstehen mit den Quantoren $\forall$ und $\exists$ die Aussagen u, v:

$$u : (\forall x \in \mathbb{R} : p(x))$$
$$v : (\exists x \in \mathbb{R} : p(x))$$

Für das Beispiel Gl.1 ist die Aussage u falsch, da es für x Werte gibt (z. B. $x = 0$), für welche die Aussage $x^2 - 2x + 1 = 0$ nicht gilt. Die Aussage v ist wahr, da $p(x)$ z. B. für $x = 1$ wahr ist.

A.2. Mengenlehre

In diesem Abschnitt wird ein Überblick über die Konzepte der Mengenlehre gegeben werden. Diese Grundlagen sind aus [Hof11] und aus [Sch13] entnommen.

Menge und Element

Der Begriff der *Menge* ist ein zentraler Begriff der Mathematik und wird von Hofmann [Hof11] wie folgt definiert bzw. erklärt:

> „Unter einer *Menge* versteht man in der Mathematik eine Zusammenfassung gewisser Dinge zu einem neuen einheitlichen Ganzen. Die hierbei zusammengefassten Dinge heißen die *Elemente* der betreffenden Menge."

Ein „Ding" a kann also folglich Element einer Menge M oder kein Element einer Menge sein. Dies wird wie folgt beschrieben:

- $a \in M$: a ist Element von M (Kurzform: a Element M)
- $a \notin M$: a ist nicht Element von M (Kurzform: a nicht Element M)

Besitzt eine Menge keine Elemente, dann wird sie *leere Menge* genannt und als $\emptyset$ notiert. Formal gilt für die leere Menge: $M = \emptyset \Leftrightarrow \forall a : \neg a \in M$.

Für zwei Mengen M_1 und M_2 kann eine Aussage bzgl. den enthaltenen Elementen getroffen werden.

- $M_1 = M_2$: Menge M_1 ist gleich der Menge M_2, d. h. die beiden Mengen M_1 und M_2 enthalten dieselben Elemente. Formal bedeutet dies: $M_1 = M_2 \Leftrightarrow \forall a : a \in M_1$ genau dann wenn $a \in M_2$.
- $M_1 \subset M_2$: M_1 ist Teilmenge von M_2, d. h. die Elemente von Menge M_1 befinden sich alle in Menge M_2. Formal bedeutet dies: $M_1 \subset M_2 \Leftrightarrow \forall a : a \in M_1$ wenn $a \in M_2$.

Außerdem gilt, dass jede Menge M auch Teilmenge von sich selbst ist, also $M \subset M$, und die leere Menge immer Teilmenge ist, also $\emptyset \subset M$. Ist eine Menge N identisch mit einer Menge M, also $N = M$, dann ist N eine *unechte* Teilmengen von M mit $N \subseteq M$.

Unter der *Potenzmenge* $\mathcal{P}(M)$ der Menge M versteht man die Menge aller Teilmengen von M. Jede Kombination und Anzahl an Elementen der Menge M befindet sich als Menge in der Potenzmenge $\mathcal{P}(M)$. Dies gilt auch für die leere Menge und die Menge M. Formal gilt für die Potenzmenge: $\mathcal{P}(M) = \{A | A \subseteq M\}$.

Die Angabe bzw. die Festlegung einer Menge kann explizit stattfinden. Die Elemente der Menge werden z. B. mit $M = \{1, 2, 4, 7\}$ angegeben. Es können auch die charakteristischen Eigenschaften der Elemente angegeben werden. Das heißt, es befinden sich nur Elemente in der Menge, die dieselbe Eigenschaft besitzen. Geschrieben: $\{x | x$ besitzt eine bestimmte Eigenschaft$\}$.

Operationen auf Mengen

Auf Mengen können verschiedene Operationen angewandt werden. Seien die beiden Mengen M_1 und M_2 gegeben, dann gibt es die folgenden Operationen:

- $M_1 \cap M_2$: Dies ist die *Schnittmenge* der beiden Mengen M_1 und M_2. Formal definiert ist $M_1 \cap M_2 = \{x | x \in M_1 \wedge x \in M_2\}$. Gilt $M_1 \cap M_2 = \emptyset$, dann sind die Mengen *disjunkt*.
- $M_1 \cup M_2$: Dies ist die *Vereinigung* der beiden Mengen M_1 und M_2. Formal definiert ist $M_1 \cup M_2 = \{x | x \in M_1 \vee x \in M_2\}$.
- $M_1 \backslash M_2$: Dies ist die *Differenz* der beiden Mengen M_1 und M_2. Formal definiert ist $M_1 \backslash M_2 = \{x | x \in M_1 \wedge x \notin M_2\}$.
- $M_1 \times M_2$: Dies ist das *kartesische Produkt* der beiden Mengen M_1 und M_2. Formal definiert ist $M_1 \times M_2 = \{(a, b) | a \in M_1, b \in M_2\}$. Das heißt, dass die Elemente der beiden Mengen paarweise zu einem *geordneten Paar* (z. B. (a, b)) verknüpft sind. Ein *ungeordnetes Paar* $\{a, b\}$ ist identisch mit $\{b, a\}$.

Eine spezielle Form der Vereinigungsmenge M stellt $\bigcup M$ dar. Formal definiert ist die Vereinigung $\bigcup M$ wie folgt:

$$\bigcup M = \{x | \exists A : A \in M \text{ mit } x \in A\}$$

$\bigcup M$ besteht aus allen Elementen, die nicht Teilmengen in M sind bzw. sich in Teilmengen von M befinden. $\bigcup M$ enthält keine Mengen bzw. Teilmengen.

Ähnlich den logischen Operationen gibt es bei den Operationen auf Mengen Gesetze. Seien die drei Mengen M_1, M_2 und M_3 gegeben, dann gelten die Gesetze:

- Kommutativitätsgesetz:
  Durchschnitt: $M_1 \cap M_2 = M_2 \cap M_1$
  Vereinigung: $M_1 \cup M_2 = M_2 \cup M_1$
- Assoziativitätsgesetz:
  Durchschnitt: $(M_1 \cap M_2) \cap M_3 = M_1 \cap (M_2 \cap M_3)$
  Vereinigung: $(M_1 \cup M_2) \cup M_3 = M_1 \cup (M_2 \cup M_3)$
  Differenz: $(M_1 \backslash M_2) \backslash M_3 = M_1 \backslash (M_2 \backslash M_3)$
- Distributivitätsgesetz:
  $(M_1 \cup M_2) \cap M_3 = (M_1 \cap M_3) \cup (M_2 \cap M_3)$
  $(M_1 \cap M_2) \cup M_3 = (M_1 \cup M_3) \cap (M_2 \cup M_3)$
  $(M_1 \cap M_2) \backslash M_3 = (M_1 \backslash M_3) \cap (M_2 \backslash M_3)$
  $(M_1 \cup M_2) \backslash M_3 = (M_1 \backslash M_3) \cup (M_2 \backslash M_3)$

Des Weiteren gilt:

- $(M_1 \backslash M_2) \cap M_2 = \emptyset$

Beweis: $(M_1 \backslash M_2) \cap M_2 = \{x | (x \in M_1 \wedge x \notin M_2) \wedge x \in M_2\}$
$= \{x | x \in M_1 \wedge x \notin M_2 \wedge x \in M_2\}$
$= \emptyset$

- Sei $M_2 \subseteq M_1 \Rightarrow (M_1 \backslash M_2) \cup M_2 = M_1$

  Beweis: $(M_1 \backslash M_2) \cup M_2 = \{x | (x \in M_1 \wedge x \notin M_2) \vee x \in M_2\}$
  $= \{x | (x \in M_1 \vee x \in M_2) \wedge (x \notin M_2 \vee x \in M_2)\}$
  $= \{x | (x \in M_1 \vee x \in M_2)\}$
  $= \{x | x \in M_1\}$ (da $M_2 \subseteq M_1$)

Relationen

Eine Relation $\mathcal{R}$ ist eine *Beziehung* zwischen „Dingen“. Aus Sicht der Mengentheorie stellt dies eine Verknüpfung von Elementen dar. Ein geordnetes Paar (a, b), also eine Verknüpfung zweier Elemente a und b, die beispielsweise in Menge M_1 und M_2 mit $a \in M_1$ und $b \in M_2$ sind, stellen eine zweistellige Relation dar. Eine zweistellige Relation $\mathcal{R} \subseteq M_1 \times M_2$ stellt eine Teilmenge des kartesischen Produkts zweier Mengen M_1 und M_2 dar.

Werden mehrere Elemente miteinander verknüpft, dann nennt man das ein *n-Tupel*, wobei das n für die Anzahl der verknüpften Elemente steht. Zum Beispiel stellt das 4-Tupel (a, b, c, d) eine Relation zwischen den vier Elementen a, b, c und d dar. Eine n-stellige Relation $\mathcal{R}$ stellt dann eine Teilmenge des kartesischen Produkts von n Mengen $M_1, M_2, ..., M_n$ mit $\mathcal{R} \subseteq M_1 \times M_2 \times ... \times M_n$ dar.

Funktionen

Funktionen stellen eine spezielle Form der Relationen dar. Die zweistellige Relation $f \subseteq M_1 \times M_2$, welche aus geordneten Paaren (a, b) besteht, ist eine Funktion. Dies bedeutet, jedem Element $a \in M_1$ ist eindeutig ein Element $b \in M_2$ zugeordnet. Mehrdeutige Funktionen werden hier nicht betrachtet, da sie für diese Arbeit nicht relevant sind. Eine Funktion $f \subseteq M_1 \times M_2$ wird auch geschrieben als: $f : M_1 \to M_2$. Ein geordnetes Paar (a, b), welches Element einer Funktion ist, kann wie folgt geschrieben werden: $(a, b) \in f$ oder $f(a) = b$.

Glossar

Application Programming Interface (API)

Die Applikationsprogrammierschnittstelle (API, engl. Application programming interface) dient zur Anbindung anderer Programme oder Systeme mittels eines Programmteils an das System, welches die API zur Verfügung stellt.

Auditing

Das Auditing protokolliert Benutzer- oder Systemaktivitäten und dient damit zur Nachweisbarkeit von Vorgängen im System.

Bildwiederholrate

Die Bildwiederholrate bezeichnet die Anzahl der Einzelbilder (engl. frames), welche pro Sekunde (engl. FPS) auf einem Bildschirm angezeigt werden.

Bitblitting

Bitblitting bezeichnet ein Verfahren, welches Speicherbereiche kopiert, wie es vor allem bei der Darstellung von grafischen Fensterinhalten auf Anzeigen verwendet wird.

Bitmaske

Eine Bitmaske ist ein Verbund aus Bits, welche für jede Binärstelle ein Flag (d. h. einen Statusindikator) repräsentiert. Dies ermöglicht beispielsweise das Einblenden bzw. Ausblenden bestimmter Pixel eines Bildes.

Cache

Ein Cache ist ein Zwischenspeicher, welcher Daten ablegt und sie für weitere schnelle Zugriffe vorhält.

CE-Industrie

Mit CE-Industrie (engl. consumer electronics) wird ein Industriezweig bezeichnet, welcher elektronische Geräte wie Fernseher und Smartphone für den Endkunden produziert.

Compositing

Compositing bezeichnet ein Verfahren, bei dem mehrere Bildelemente zu einem Gesamtbild zusammengeführt werden.

Controller Area Network (CAN)

CAN (engl. Controller Area Network) ist ein Fahrzeugbusstandard, welcher die Steuergeräte in einem Fahrzeug miteinander vernetzt.

Cyber Physical Systems

Cyber Physical Systems sind Systeme, welche aus zusammenarbeitenden rechenintensiven Elementen bestehen, die physikalische Entitäten kontrollieren.

Denial of Service (DoS)

DoS (Verweigerung des Dienstes, engl. Denial of Service) stellt einen Angriff auf ein System dar, welches durch eine böswillig erzeugte enorme Anzahl an Anfragen seinen Aufgaben nicht mehr nachkommen kann und die eigentlichen Anfragen nicht mehr abarbeiten kann.

Digital Visual Interface (DVI)

DVI ist eine Schnittstelle für die Übertragung von digitalen Videodaten.

Embedded-System Graphics Library (EGL)

EGL (Abkürzung engl. Embedded-System Graphics Library) ist eine Programmierschnittstelle, die zwischen den Khronos APIs (z. B. OpenGL ES 2.0) und dem Fenstersystem der nativen Plattform liegt.

Extensible Markup Language (XML)

XML (Erweiterbare Auszeichnungssprache, engl. Extensible Markup Language) ist eine Beschreibungssprache für die Darstellung von Daten in einer Textdatei. Hierbei können Hierarchien zur Beschreibung der Daten verwendet werden.

First in first out (FIFO)

FIFO (Abkürzung engl. First in, first out) ist eine Festlegung der Reihenfolge, in welcher ein Bearbeitungsprozess Anfragen abarbeitet. Bei FIFO wird die zuerst eingehende Anfrage immer zuerst abgearbeitet. Das bedeutet, in der Reihenfolge der eingehenden Anfragen werden diese auch abgearbeitet.

Grafikprozessoreinheit (GPU)

Eine Grafikprozessoreinheit (GPU, engl. Graphics processing unit) ermöglicht die schnelle und effiziente Berechnung von grafischen Inhalten in 2D und 3D.

Hashtabelle

Eine Hashtabelle ist eine spezielle Indexstruktur, welche das schnelle Auffinden von gespeicherten Daten mittels eines Schlüssels ermöglicht.

Headunit (HU)

Die Headunit dient im Fahrzeug maßgeblich als Unterhaltungs- und Informationsquelle. Dazu werden Anwendungen wie Radio, TV, DVD-Wiedergabe, Navigation, aber auch Internetzugriff von der Headunit bereitgestellt.

High Definition Multimedia Interface (HDMI)

HDMI ist eine Schnittstelle, welche digitale Bild- und Tondaten ermöglicht.

Human-Machine-Interface (HMI)

Die Mensch-Maschine-Schnittstelle (HMI, engl. Human-Machine-Interface) dient zur Steuerung von Anwendungen mittels einer Bedienoberfläche und Eingabegeräten.

Hypervisor

Ein Hypervisor ist eine Software, Hardware oder Firmware, welche dazu dient den Betrieb von virtuellen Maschinen sicherzustellen. Hierbei ist es die Aufgabe des Hypervisors die Isolation bei den Zugriffen auf die gemeinsam genutzte Hardware sicherzustellen.

Identifikator (ID)

Ein Identifikator dient zur eindeutigen Bestimmung eines Objektes bzw. zur Unterscheidung von mehreren Objekten.

Inter-Partition Communication (IPC)

Die Inter-Partition Communication (auf deutsch Interpartitionskommunikation) bietet Kommunikation zwischen verschiedenen Partitionen.

International Organization for Standardization (ISO)

ISO (engl. International Organization for Standardization) ist eine Organisation, welche internationale Standards festlegt.

Kombiinstrumente

Die Kombiinstrumente stellen den Status des Fahrzeugs dar. Dazu werden Informationen wie die Geschwindigkeit, Drehzahl oder der Tankfüllstand in Form digitaler oder analoger Anzeigen dem Fahrer angezeigt.

Local Interconnect Network (LIN)

LIN (engl. Local Interconnect Network) ist ein serielles Netzwerkprotokoll, das im Fahrzeug eingesetzt wird. LIN dient unter anderem zur Vernetzung von Sensoren und Aktoren.

Low Voltage Differential Signaling (LVDS)

LVDS (engl. Low Voltage Differential Signaling) ist eine Schnittstelle zur Datenübertragung mit Hochgeschwindigkeit wie es z. B. für die Darstellung des Bildes auf der Anzeige eines Notebooks benötigt wird.

Mantelrohr-Schaltmodul (MRSM)

Das MRSM ist ein Steuergerät, welches die Eingabetasten und -hebel mittels CAN an den Fahrzeugbus anschließt.

Offscreen-Puffer

Offscreen-Puffer stellen Bildspeicher dar, in welchen Anwendungen ihre grafischen Inhalte speichern.

OpenGL ES 2.0

OpenGL ES 2.0 ist eine API der Khronos Group für 3D-Grafik, welche vor allem auf eingebetteten System verwendet wird.

Original Equipment Manufacturer (OEM)

Automobilhersteller werden als OEMs (Erstausstatter, engl. Original Equipment Manufacturer) bezeichnet, da sie direkt unter eigenem Namen produzieren und an die Kunden verkaufen.

Prozessor

Der Prozessor (engl. Central Processing Unit (CPU)) ist die zentrale Verarbeitungseinheit eines Computers.

Steuergeräte

Steuergeräte stellen Funktionalitäten (z. B. die Regelung des Motor mittels Motorsteuergerät) im Fahrzeug bereit. Sie sind untereinander vernetzt und tauschen Informationen mittels eines Fahrzeugbusses wie CAN aus.

Thread

Ein Thread ist ein leichtgewichtiger Prozess, welcher die Abarbeitungsreihenfolge eines Programms darstellt. Ein Thread stellt einen Teil eines Prozesses dar.

Universal Serial Bus (USB)

USB ist ein serielles Bussystem für das Verbinden von externen Geräten.

Virtual Machine Monitor (VMM)

Ein Virtual Machine Monitor (auf deutsch virtueller Maschinenüberwacher) stellt eine Abstraktionsebene zwischen der Software und der Hardware dar, welche die Zugriffe der Software auf die Hardware überwacht und beschränkt. Hierbei sind meistens mehrere verschiedene Betriebssysteme in der Softwareschicht, welche durch den Virtual Machine Monitor kontrolliert werden.

Virtualisierung

Virtualisierung ist eine Technologie, die zur Abstraktion von Software und Hardware eingesetzt wird. Sie ermöglicht den isolierten Betrieb von mehreren virtuellen Maschinen, welche sich die Ressourcen der Hardware, kontrolliert durch einen Hypervisor, teilen.

Virtuelle Maschine

Eine virtuelle Maschine ist ein Container, in welchem, von anderen virtuellen Maschinen isoliert, ein Betriebssystem ausgeführt wird.

Watchdog

Ein Watchdog (auf deutsch Wachhund) überwacht die Zustände in einem System und erkennt das Eintreten in einen Fehlerzustand.

Zentrale Bedienelement (ZBE)

Das ZBE ist ein Steuergerät, welches den Dreh-Drück-Steller mittels CAN an den Fahrzeugbus anschließt.

Abkürzungsverzeichnis

API

Application Programming Interface

CAN

Controller Area Network

CPU

Central Processing Unit

DoS

Denial of Service

DVI

Digital Visual Interface

EGL

Embedded-System Graphics Library

FIFO

first in, first out

GPU

Graphics Processing Unit

HDMI

High Definition Multimedia Interface

HMI

Human-Machine-Interface

HU

Headunit

ID

Identifikator

IPC

Inter-Partition Communication

ISO

International Organization for Standardization

LIN

Local Interconnect Network

LVDS

Low Voltage Differential Signaling

MRSM

Mantelrohr-Schaltmodul

OEM

Original Equipment Manufacturer

USB

Universal Serial Bus

VMM

Virtual Machine Monitor

XML

Extensible Markup Language

ZBE

Zentrales-Bedienelement

Literaturverzeichnis

[AAM06] AAM. *Statement of Principles, Criteria and Verification Procedures on Driver Interactions with Advanced In-Vehicle Information and Communication Systems.* Alliance of Automotive Manufacturers, Juli 2006. (Zitiert auf den Seiten 41, 44, 99 und 129)

[Ara15] Das Projekt ARAMiS. `http://www.projekt-aramis.de/`, 2015. (Zitiert auf Seite 36)

[Aut] Autosar. `http://www.autosar.org/`. (Zitiert auf Seite 38)

[BAK+02] Ali Raza Butt, Sumalatha Adabala, Nirav H. Kapadia, Renato J. O. Figueiredo und José A. B. Fortes. Fine-Grain Access Control for Securing Shared Resources in Computational Grids. In *16th International Parallel and Distributed Processing Symposium (IPDPS 2002), USA*, 2002. (Zitiert auf Seite 105)

[BBF+13] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci und Philippe Schnoebelen. *Systems and software verification: model-checking techniques and tools.* Springer Science & Business Media, 2013. (Zitiert auf Seite 188)

[BBG04] Rafae Bhatti, Elisa Bertino und Arif Ghafoor. A trust-based context-aware access control model for Web-services. In *Web Services. Proceedings. International Conference on*, 2004. (Zitiert auf den Seiten 18, 106 und 107)

[BDC08] Mikhail Bautin, Ashok Dwarakinath und Tzicker Chiueh. Graphic engine resource management. In Proceedings of Multimedia Computing and Networking (MMCN), 2008. (Zitiert auf Seite 55)

[BL73] David E. Bell und Leonard J. Lapadula. Secure Computer Systems: Volume I – Mathematical Foundations, Volume II – A Ma-

thematical Model, Volume III – A Refinement of the Mathematical Model. Technical Report MTR-2547, The MITRE Corporation, Bedford, Massachusetts, 1973. (Zitiert auf Seite 60)

[BL76] David E. Bell und Leonard J. Lapadula. Secure Computer System: Unified Exposition and MULTICS Interpretation. Technical Report ESD-TR-75-306, 1976. (Zitiert auf Seite 104)

[BR86] S. A. Bly und J. K. Rosenberg. A Comparison of Tiled and Overlapping Windows. In *Proc. of CHI-86*, Seiten 101–106, Boston, MA, 1986. (Zitiert auf Seite 158)

[BZNR01] J.-C. Birget, X. Zou, G. Noubir und B. Ramamurthy. Hierarchy-based access control in distributed environments. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 1, Seiten 229 –233 vol.1, Juni 2001. (Zitiert auf Seite 105)

[Cec14] Riccardo Cecolin. Compositing Concepts for the Presentation of Graphical Application Windows on Embedded Systems. Master thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Deutschland, April 2014. (Zitiert auf Seite 35)

[CMT04a] A. Corrad, R. Montanari und D. Tibaldi. Context-based access control management in ubiquitous environments. In *Proceedings of the 3rd NCA*, 2004. (Zitiert auf den Seiten 19 und 107)

[CMT04b] A. Corradi, R. Montanari und D. Tibaldi. Context-based access control for ubiquitous service provisioning. In *Proceedings of the 28th COMPSAC*, 2004. (Zitiert auf den Seiten 19 und 108)

[Cor09] Intel Corporation. Intel developer forum. In *https://newsroom.intel.com/press-kits/intel-developer-forum-idf-san-francisco-2009/*, 2009. (Zitiert auf Seite 36)

[CR05] Olivier Chapuis und Nicolas Roussel. Metisse is not a 3D desktop! In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, Seiten 13–22, New York, NY, USA, 2005. ACM. (Zitiert auf Seite 160)

[Dey01] Anind K. Dey. Understanding and Using Context. *Personal Ubiquitous Comput.*, 5(1):4–7, Januar 2001. (Zitiert auf Seite 62)

[EGL16] Die Programmierschnittstelle EGL für die Anbindung an das native Fenstersystem. `https://www.khronos.org/egl/`, Januar 2016. (Zitiert auf den Seiten 96 und 118)

[EJ09] C. Ebert und C. Jones. Embedded software: Facts, figures, and future. *Computer*, April 2009. (Zitiert auf den Seiten 17, 25, 46, 48 und 62)

[EMP+91] J. Epstein, J. McHugh, R. Pascale, H. Orman, G. Benson, C. Martin, A. Marmor-Squires, B. Danner und M. Branstad. A prototype B3 trusted X Window System. In *Proceedings of the 7th Annual Computer Security Applications Conference*, Seiten 44–55, Dezember 1991. (Zitiert auf den Seiten 53 und 104)

[EP91] J. Epstein und J. Picciotto. 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*, volume 1, Oktober 1991. (Zitiert auf den Seiten 18, 53 und 103)

[Eps96] Jeremy Epstein. A high-assurance hardware-based high-performance trusted windowing system. Proceedings of the 19th Annual National Information System Security Conference, Baltimore MD. Cordant, Inc., Dezember 1996. (Zitiert auf Seite 104)

[ESO08] 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. (Zitiert auf Seite 41)

[FH03] Norman Feske und Helmuth Hartig. Dope – a window server for real-time and embedded systems. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, Seiten 74–77, Dez. 2003. (Zitiert auf Seite 54)

[FH04] Norman Feske und Christian Helmuth. Overlay Window Management: User Interaction With Multiple Security Domains. Technical Report TUD-FI04-02, Technical University Dresden, März 2004. (Zitiert auf den Seiten 18, 54 und 103)

[FH05] Norman Feske und Christian Helmuth. A Nitpicker's guide to a minimal-complexity secure GUI. In *Proceedings of the 21st Computer Security Applications Conference*, Seiten 85–94, Dez. 2005. (Zitiert auf den Seiten 18, 54 und 103)

[For13] Ford. Software-Development-Kit (SDK). `https://developer.ford.com/develop/openxc/`, Feb. 2013. (Zitiert auf Seite 28)

[Fre] Freescale. i.MX 6Dual/6Quad Applications Processor Reference Manual, Document Number: IMX6DQRM Rev. 3, Juli 2015. `http://www.nxp.com/files/32bit/doc/ref_manual/IMX6DQRM.pdf`. (Zitiert auf Seite 31)

[Fre15] Freescale i.MX 6Quad Processors. `http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX6Q`, März 2015. (Zitiert auf den Seiten 34, 99, 135, 165 und 168)

[Fuj16] Fujitsu CGI Studio - 2D/3D HMI Software Development Platform for Automotive Systems. `http://www.fujitsu.com/us/products/devices/semiconductor/gdc/products/cgistudio.html`, Mai 2016. (Zitiert auf Seite 155)

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner und John F. Hughes. *Computer Graphics: Principles and Practice (2Nd Ed.)*. Addison-Wesley, Boston, MA, USA, 1990. (Zitiert auf Seite 116)

[Fvw16] Der virtuelle Fenstermanager FVWM für das X Fenstersystem. `http://www.fvwm.org/`, Januar 2016. (Zitiert auf Seite 160)

[GH13] Ahmad Gilbeau-Hammoud. Erstellung und Evaluation einer Zugriffskontrolle für die Darstellung graphischer Applikationen im Fahrzeug. Diploma thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, Mai 2013. (Zitiert auf Seite 35)

[Glm16] Bewertungsprogramm GLmark2-es2 für OpenGL E2 2.0. `https://github.com/glmark2/glmark2`, Januar 2016. (Zitiert auf Seite 147)

[GM09] Eimear Gallery und Chris J. Mitchell. Trusted computing: Security and applications. *Cryptologia*, 33(3):217–245, 2009. (Zitiert auf Seite 51)

[GNU15] GNU Octave. `https://www.gnu.org/software/octave/`, März 2015. (Zitiert auf Seite 137)

[GP04] James Gettys und Keith Packard. The (re) architecture of the x window system. *In Proceedings of the 2004 Linux Symposium*, volume 1:227–237, 2004. (Zitiert auf den Seiten 19, 30 und 160)

[GR09] A. Groll und C. Ruland. Secure and authentic communication on existing in-vehicle networks. In *Intelligent Vehicles Symposium, 2009 IEEE*, Seiten 1093–1097, Juni 2009. (Zitiert auf Seite 109)

[GS82] L. J. Guibas und J. Stolfi. A Language for Bitmap Manipulation. *ACM Trans. Graph.*, 1, Juli 1982. (Zitiert auf den Seiten 19 und 159)

[GSC+15] Simon Gansel, Stephan Schnitzer, Riccardo Cecolin, Frank Dürr, Kurt Rothermel und Christian Maihöfer. Efficient compositing strategies for automotive HMI systems. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, Seiten 1–10, Juni 2015. (Zitiert auf Seite 35)

[GSD+13] Simon Gansel, Stephan Schnitzer, Frank Dürr, Kurt Rothermel und Christian Maihöfer. Towards Virtualization Concepts for Novel Automotive HMI Systems. In *Proceedings of IESS*, IFIP LNCS. Springer Berlin Heidelberg, 2013. (Zitiert auf den Seiten 17, 35 und 99)

[GSGH+14] Simon Gansel, Stephan Schnitzer, Ahmad Gilbeau-Hammoud, Viktor Friesen, Frank Dürr, Kurt Rothermel und Christian 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, Seiten 17–28, New York, NY, USA, 2014. ACM. (Zitiert auf Seite 35)

[GSGH+15] Simon Gansel, Stephan Schnitzer, Ahmad Gilbeau-Hammoud, Viktor Friesen, Frank Dürr, Kurt Rothermel, Christian Maihöfer und Ulrich Krämer. Context-Aware Access Control in Novel Automotive HMI Systems. In Sushil Jajodia und Chandan Mazumdar, Hrsg., *Information Systems Security: 11th International Conference, ICISS 2015, Kolkata, Indien, 16-20. Dezember, 2015.*

Proceedings, Seiten 118–138, Cham, 2015. Springer International Publishing. (Zitiert auf Seite 35)

[HAK+12] D. Herges, N. Asaj, B. Könings, F. Schaub und M. Weber. Ginger: An Access Control Framework for Telematics Applications. In *Processing of the 11th TrustCom*, 2012. (Zitiert auf Seite 108)

[Han07] Jacob G. Hansen. 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*, 2007. (Zitiert auf den Seiten 18, 56 und 103)

[HED11] Tobias Hoppe, Frederik Exler und Jana Dittmann. IDS-Signaturen für automotive CAN-Netzwerke. *Peter Schartner, Jürgen Taeger (Hrsg.), D·A·CH Security 2011 - Bestandsaufnahme, Konzepte, Anwendungen, Perspektiven*, Seiten 55–66, September 2011. (Zitiert auf Seite 109)

[Hei08] Gernot Heiser. The Role of Virtualization in Embedded Systems. In *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems*, IIES '08, Seiten 11–16, New York, NY, USA, 2008. ACM. (Zitiert auf Seite 52)

[Hof11] Gerald Hofmann. *Ingenieurmathematik für Studienanfänger: Formeln - Aufgaben - Lösungen.* Vieweg+Teubner Verlag, 2011. (Zitiert auf den Seiten 189, 191 und 192)

[Hoh02] Michael Hohmuth. *The Fiasco kernel: System Architecure.* Technical report: TUD-FI02-06-Juli-2002, Technical University Dresden, 2002. (Zitiert auf Seite 54)

[HYMLWD12] Liu Hong-Yue, Deng Miao-Lei und Yang Wei-Dong. A Context-Aware Fine-Grained Access Control Model. In *Computer Science Service System (CSSS)*, 2012. (Zitiert auf den Seiten 18, 106 und 107)

[Ing75] Dan Ingalls. Description of the BIT BLT routines. `ftp://bitsavers.informatik.uni-stuttgart.de/pdf/xerox/alto/BitBLT_Nov1975.pdf`, November 1975. (Zitiert auf Seite 114)

[ISO96] ISO 11428. *Ergonomics – Visual danger signals – General requirements, design and testing.* ISO, Genf, Schweiz, Dez. 1996. (Zitiert auf den Seiten 42, 45 und 47)

[ISO02] ISO 15005. *Road vehicles – Ergonomic aspects of transport information and control systems – Dialogue management principles and compliance procecdures.* ISO, Genf, Schweiz, Juli 2002. (Zitiert auf den Seiten 42, 43, 44, 45, 46 und 47)

[ISO04] 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, Genf, Schweiz, 2004. (Zitiert auf den Seiten 42, 43 und 45)

[ISO08] ISO 15408-2. *Information technology – Security techniques – Evaluation criteria for IT security – Part 2: Security functional components.* ISO, Genf, Schweiz, Aug. 2008. (Zitiert auf den Seiten 42, 44, 45, 46, 47, 48 und 53)

[ISO10] ISO 2575. *Road vehicles – Symbols for controls, indicators and tell-tales.* ISO, Genf, Schweiz, Juli 2010. (Zitiert auf den Seiten 42 und 47)

[ISO11] ISO 26262. *Road vehicles – Functional Safety.* ISO, Genf, Schweiz, Nov. 2011. (Zitiert auf den Seiten 28, 31, 38, 42, 46 und 188)

[JAM04] JAMA. *Guideline for In-vehicle Display Systems – Version 3.0.* Japan Automobile Manufacturers Association, Aug. 2004. (Zitiert auf Seite 41)

[Jan11] Helmut Janker. *Straßenverkehrsrecht: StVG, StVO, StVZO, Fahrzeug-ZulassungsVO, Fahrerlaubnis-VO, Verkehrszeichen, Bußgeldkatalog.* C.H. Beck, 2011. (Zitiert auf den Seiten 30, 41 und 43)

[KAE+10] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch und Simon Winwood. seL4: Formal verification of an OS kernel. *Communications of the ACM*, 53(6):107–115, Juni 2010. (Zitiert auf Seite 52)

[Kai07] Robert Kaiser. Combining Partitioning and Virtualisation for Safety-critical Systems. In *Embedded World Conference 2007, Nuremberg*, Februar 2007. (Zitiert auf Seite 29)

[KLIR11] S. Kato, K. Lakshmanan, Y. Ishikawa und R. Rajkumar. Resource Sharing in GPU-Accelerated Windowing Systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, Seiten 191–200, April 2011. (Zitiert auf Seite 55)

[KLRI11] S. Kato, K. Lakshmanan, R. Rajkumar und Y. Ishikawa. TimeGraph: GPU scheduling for real-time multi-tasking environments. In *Proceedings of USENIX Annual Technical Conference*, Berkeley, CA, USA, 2011. USENIX Association. (Zitiert auf Seite 55)

[KNP04] Ingolf H. Krüger, Edward C. Nelson und K. Venkatesh Prasad. Service-Based Software Development for Automotive Applications. In *Proceedings of the CONVERGENCE 2004. CTEA*, Januar 2004. (Zitiert auf Seite 63)

[KS97] Eser Kandogan und Ben Shneiderman. Elastic windows: Evaluation of multi-window operations. In *Human Factors in Computing Systems, CHI '97 Conference Proceedings, Atlanta, Georgia, USA, 22-27. März, 1997.*, Seiten 250–257, 1997. (Zitiert auf Seite 158)

[Lam74] Butler W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1):18–24, Januar 1974. (Zitiert auf Seite 104)

[LCTSdL07] H. Andres Lagar-Cavilla, Niraj Tolia, M. Satyanarayanan und Eyal de Lara. VMM-independent graphics acceleration. In *Proceedings of the 3rd international conference on Virtual execution environments*, Seiten 33–43, New York, NY, USA, 2007. ACM. (Zitiert auf Seite 55)

[LDE+01] Jochen Liedtke, Uwe Dannowski, Kevin Elphinstone, Gerd Liefländer, Espen Skoglund, Volkmar Uhlig, Christian Ceelen, Andreas Haeberlen und Marcus Völp. The L4Ka Vision, April 2001. University of Karlsruhe, Germany. (Zitiert auf Seite 53)

[LFG+13] K. Lee, J. Flinn, T.J. Giuli, B. Noble und C. Peplin. Amc: Verifying user interface properties for vehicular applications. In *Pro-*

ceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '13, Seiten 1–12, New York, NY, USA, 2013. ACM. (Zitiert auf Seite 129)

[LH02] Gabriel Leen und Donal Heffernan. Expanding Automotive Electronic Systems. *Computer*, 35(1):88–93, Januar 2002. (Zitiert auf Seite 25)

[LHH97] Jochen Liedtke, Hermann Härtig und Michael Hohmuth. OS-Controlled Cache Predictability. In *Proceedings of the 3rd IEEE Real-time Technology and Applications Symposium (RTAS)*, Montreal, Canada, Juni 1997. (Zitiert auf Seite 53)

[Lie93] Jochen Liedtke. Improving IPC by kernel design. *SIGOPS Oper. Syst. Rev.*, 27:175–188, Dezember 1993. (Zitiert auf Seite 53)

[Lie95] Jochen Liedtke. On Micro-kernel Construction. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, SOSP '95, Seiten 237–250, New York, NY, USA, 1995. ACM. (Zitiert auf Seite 53)

[LSV12] Chung-Wei Lin und Alberto Sangiovanni-Vincentelli. Cyber-Security for the Controller Area Network (CAN) Communication Protocol. In *Proceedings of the 2012 International Conference on Cyber Security*, CYBERSECURITY '12, Seiten 1–7, Washington, DC, USA, 2012. IEEE Computer Society. (Zitiert auf Seite 110)

[MA11] M. Müter und N. Asaj. Entropy-based anomaly detection for in-vehicle networks. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, Seiten 1110–1115, Juni 2011. (Zitiert auf Seite 109)

[MB03] Ghita Kouadri Mostéfaoui und Patrick Brézillon. A Generic Framework for Context-Based Distributed Authorizations. In *Proceedings of the 4th CONTEXT'03*, 2003. (Zitiert auf den Seiten 18 und 106)

[MGR+14] B. Mencher, W. Gollin, F. Reiter, A. Glaser, F. Landhäußer, K. Lerchenmüller, D. Boebel, M. Hamm, T. Spingler, F. Niewels, T. Ehret, G. Nenninger, P. Knoll und A. Kuttenberger. Overview of electrical and electronic systems in the vehicle. *Fundamentals of Automotive and Engine Technology. Standard Drives,*

Hybrid Drives, Brakes, Safety Systems., Seiten 158–160, 2014. (Zitiert auf Seite 25)

[MKS12] Hans-Ulrich Michel, Dirk Kaule und Martin Salfer. Vision einer intelligenten Vernetzung. *Elektronik automotive*, April 2012. (Zitiert auf den Seiten 17 und 26)

[MM81] Norman Meyrowitz und Margaret Moser. BRUWIN: An Adaptable Design Strategy for Window Manager/Virtual Terminal Systems. *SIGOPS Oper. Syst. Rev.*, 15(5):180–189, 1981. (Zitiert auf den Seiten 19 und 158)

[Mul11] Arbeitskreis Multicore. Relevanz eines Multicore-Ökosystems für künftige Embedded Systems. Positionspapier zur Bedeutung, Bestandsaufnahme und Potentialermittlung der Multicore-Technologie für den Industrie- und Forschungsstandort Deutschland. *BIC-Cnet Innovationszirkel Embedded Systems*, Seite 48, München, 2011. (Zitiert auf den Seiten 17 und 25)

[Mye84a] Brad A. Myers. A Complete Implementation of Covered Windows for a Heterogeneous Environment. Technical report, University of Toronto, Computer Science Dept., 1984. (Zitiert auf den Seiten 19 und 159)

[Mye84b] Brad A. Myers. The user interface for Sapphire. *IEEE Computer Graphics and Applications*, 12:12–23, Dez. 1984. (Zitiert auf den Seiten 19 und 159)

[Mye86] Brad A Myers. A Complete and Efficient Implementation of Covered Windows. *Computer*, 19(9):57–67, Sep. 1986. (Zitiert auf den Seiten 19 und 159)

[Mye88] Brad A. Myers. A taxonomy of window manager user interfaces. *j-IEEE-CGA*, 8(5):65–84, 1988. (Zitiert auf den Seiten 10 und 126)

[Nat85] National Computer Security Center. *Trusted Computer System Evaluation Criteria*, Dezember 1985. 5200.28-STD, DoD. (Zitiert auf den Seiten 53, 60 und 104)

[Nav16] Fahrzeugnavigationssoftware Navit. `http://www.navit-project.org/`, Januar 2016. (Zitiert auf Seite 171)

[Nou16] Grafikkartentreiber Nouveau. `http://nouveau.freedesktop.org/wiki/`, Januar 2016. (Zitiert auf Seite 55)

[NS79] William M. Newman und Robert F. Sproull, Hrsg. *Principles of Interactive Computer Graphics (2Nd Ed.)*. McGraw-Hill, Inc., New York, NY, USA, 1979. (Zitiert auf Seite 114)

[Nus98] Stefan Nusser. *Sicherheitskonzepte im WWW*. Vieweg Verlag, 1998. (Zitiert auf Seite 60)

[OKL15] Der OKL4-Microvisor, ein sicherer Kernel für Virtualisierung. `https://gdmissionsystems.com/cyber/products/trusted-computing-cross-domain/microvisor-products/`, 2015. (Zitiert auf Seite 53)

[Ope16a] Die offene Grafikschnittstelle OpenGL. `https://www.khronos.org/opengl/`, Januar 2016. (Zitiert auf den Seiten 96 und 118)

[Ope16b] Die offene Grafikschnittstelle OpenGL ES für eingebettete Systeme. `https://www.khronos.org/opengles/`, Januar 2016. (Zitiert auf den Seiten 118, 122, 135 und 168)

[Ope16c] Die Programmierschnittstelle OpenVG für Vektorgrafik. `https://www.khronos.org/openvg/`, Januar 2016. (Zitiert auf den Seiten 135 und 168)

[Pik83] Rob Pike. Graphics in Overlapping Bitmap Layers. *ACM Trans. Graph.*, 2(2):135–160, 1983. (Zitiert auf den Seiten 19, 114 und 159)

[Pik15] Der PikeOS Hypervisor. `https://www.sysgo.com/products/pikeos-rtos-and-virtualization-concept/`, 2015. (Zitiert auf den Seiten 29, 34, 53, 136 und 170)

[Pix15] Pixman: Die Pixel-Manipulationsbibliothek für X und Cairo. `http://cgit.freedesktop.org/pixman/`, März 2015. (Zitiert auf den Seiten 19 und 161)

[RAL+15] Dominik Reinhardt, Daniel Adam, Enno Luebbers, Rakshith Amarnath, Rolf Schneider, Simon Gansel, Stephan Schnitzer, Christian Herber, Timo Sandmann, Hans-Ulrich Michel, Dirk Kaule, Damla Olkun, Matthias Rehm, Jens Harnisch, Andre Richter, Steffen Baehr, Oliver Sander, Juergen Becker, Uwe Baumgarten und Henrik Theiling. Embedded Virtualization Approaches for Ensuring Safety and Security within E/E Automotive Systems. 2015. (Zitiert auf den Seiten 34, 35 und 165)

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein und Charles E. Youman. Role-Based Access Control Models. *Computer*, 29(2):38–47, Februar 1996. (Zitiert auf Seite 61)

[Sch00] Uwe Schöning. *Logik für Informatiker*. Spektrum Akademischer Verlag, 5. a. edition, 2000. (Zitiert auf Seite 189)

[Sch13] Michael Schenke. *Logic calculi in informatics. How is logic used by the computer? (Logikkalküle in der Informatik. Wie wird Logik vom Rechner genutzt?)*. Wiesbaden: Springer Vieweg, 2013. (Zitiert auf den Seiten 189 und 191)

[SG86] Robert W. Scheifler und Jim Gettys. The X Window System. *ACM Trans. Graph.*, 5(2):79–109, April 1986. (Zitiert auf den Seiten 19, 53, 55 und 160)

[SGDR14] Stephan Schnitzer, Simon Gansel, Frank Dürr und Kurt Rothermel. Concepts for execution time prediction of 3D GPU rendering. In *Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on*, Seiten 160–169, Juli 2014. (Zitiert auf den Seiten 35 und 184)

[SGDR16] Stephan Schnitzer, Simon Gansel, Frank Dürr und Kurt Rothermel. Real-time scheduling for 3D GPU rendering. In *11th IEEE Symposium on Industrial Embedded Systems, SIES 2016, Krakau, Polen, 23-25. Mai, 2016*, Seiten 45–54, 2016. (Zitiert auf den Seiten 35, 36 und 184)

[SK10] Udo Steinberg und Bernhard Kauer. NOVA: a microhypervisor-based secure virtualization architecture. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, Seiten 209–222, New York, NY, USA, 2010. ACM. (Zitiert auf Seite 52)

[SN04] Mark Strembeck und Gustaf Neumann. An integrated approach to engineer and enforce context constraints in RBAC environments. *ACM Trans. Inf. Syst. Secur.*, 2004. (Zitiert auf den Seiten 18, 106 und 107)

[Spl16] Mercedes Benz Splitview. `http://m.mercedes-benz.de/techcenter/splitview/detail.html`, Januar 2016. (Zitiert auf Seite 177)

[ST94] Bill N. Schilit und Marvin M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, 1994. (Zitiert auf den Seiten 18 und 106)

[Sta03] D.H. Stamatis. *Failure Mode and Effect Analysis: FMEA from Theory to Execution.* ASQ Quality Press, 2003. (Zitiert auf den Seiten 46 und 188)

[Sur15] SurfaceFlinger. `https://android.googlesource.com/platform/frameworks/native/+/android-cts-4.1_r1/services/surfaceflinger/SurfaceFlinger.cpp`, März 2015. (Zitiert auf den Seiten 20 und 161)

[SVNC04] Jonathan S. Shapiro, John Vanderburgh, Eric Northup und David Chizmadia. Design of the EROS trusted window system. In *Proceedings of the 13th conference on USENIX Security Symposium – Volume 13*, Berkeley, CA, USA, 2004. USENIX Association. (Zitiert auf Seite 54)

[T+79] Charles P. Thacker et al. Alto–a personal computer. Technical report, Xerox PARC, August 1979. (Zitiert auf Seite 114)

[Tan09] Andrew S. Tanenbaum. *Moderne Betriebssysteme (3. Aufl.)*. Pearson Studium, 2009. (Zitiert auf den Seiten 58 und 114)

[Viv16] Vivante Corporation. `http://www.vivantecorp.com/`, Januar 2016. (Zitiert auf Seite 99)

[VLC16] Videowiedergabesoftware VLC. `http://www.videolan.org/vlc/`, Januar 2016. (Zitiert auf Seite 171)

[vSH07] Carl van Schaik und Gernot Heiser. High-performance microkernels and virtualisation on ARM and segmented architectures.

In *Proceedings of the 1st International Workshop on Microkernels for Embedded Systems (MIKES)*, Sydney, Australien, 2007. (Zitiert auf Seite 53)

[Way15] Wayland (Display-Server-Protocol). `http://wayland.freedesktop.org/docs/html/`, März 2015. (Zitiert auf den Seiten 19 und 160)

[WGSS11] Manuela Waldner, Raphael Grasset, Markus Steinberger und Dieter Schmalstieg. Display-adaptive window management for irregular surfaces. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '11, Seiten 222–231, New York, NY, USA, 2011. ACM. (Zitiert auf Seite 162)

[WN05] Erik Wilde und Nick Nabholz. Access Control for Shared Resources. In *Proceedings of the CIMCA-IAWTIC'06*, Seiten 256–250, 2005. (Zitiert auf Seite 105)

[WSGS11] Manuela Waldner, Markus Steinberger, Raphael Grasset und Dieter Schmalstieg. Importance-driven compositing window management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, Seiten 959–968, New York, NY, USA, 2011. ACM. (Zitiert auf Seite 161)

[WZT09] Jue Wang, Li Zhou und Chengxiang Tan. A BLP-Based Model for Hierarchical Organizations. In *Proc. of the 2009 Second IWCSE*, Seiten 456–459, 2009. (Zitiert auf Seite 105)

[xco15] Der Compositing-Manager xcompmgr. `http://freedesktop.org/xapps/release/xcompmgr-1.1.tar.gz`, März 2015. (Zitiert auf den Seiten 19 und 160)

Alle URLs wurden zuletzt am 24.10.2016 geprüft.

Erklärung

Ich erkläre hiermit, dass ich, abgesehen von den ausdrücklich bezeichneten Hilfsmitteln und den Ratschlägen von jeweils namentlich aufgeführten Personen, die Dissertation selbstständig verfasst habe.

(Simon Gansel)