Context-aware Access Control in Novel Automotive HMI Systems

Simon Gansel[1], Stephan Schnitzer[2], Ahmad Gilbeau-Hammoud[2], Viktor Friesen[1], Frank Dürr[2], Kurt Rothermel[2], Christian Maihöfer[1], and Ulrich Krämer[3]

[1] System Architecture and Platforms, Mercedes-Benz Cars, Daimler AG, Germany[⋆]
[2] Institute of Parallel and Distributed Systems, University of Stuttgart, Germany[⋆⋆]
[3] Telemotive AG, Germany[⋆⋆⋆]

Abstract. The growing relevance of vehicular applications like media player, navigation system, or speedometer using graphical presentation has lead to an increasing number of displays in modern cars. This effectuates the desire for flexible sharing of all the available displays between several applications. However, automotive requirements include many regulations to avoid driver distraction to ensure safety. To allow for safe sharing of the available screen surface between the many safety-critical and non-safety-critical applications, adequate access control systems are required. We use the notion of *contexts* to dynamically determine, which application is allowed to access which display area. A context can be derived from vehicle sensors (e.g., the current speed), or be an application-specific state (e.g., which menu item is selected). We propose an access control model that is inherently aware of the context of the car and the applications. It provides delegation of access rights to display areas by applications. We implemented a proof-of-concept implementation that demonstrates the feasibility of our concept and evaluated the latency introduced by access control. Our results show that the delay reacting on dynamic context changes is small enough for automotive scenarios.

1 Introduction

Within the last 30 years the development of cars in the automotive industry has increasingly depended on electronics and software instead of mechanics [3]. The growing relevance of graphics functions and applications using integrated displays in modern cars is a good indicator for this trend. For instance, the *Head Unit* (HU) uses displays integrated into the backside of the front seats and center console to display multimedia content, navigation system, and web browser. Additionally, the *Instrument Cluster* (IC) uses the instrument and the head-up displays to show car specific information like speed, warnings, and navigation instructions. In order to support these applications, cars are equipped with a

[⋆] `firstname.lastname <at> daimler.com`
[⋆⋆] `lastname <at> ipvs.uni-stuttgart.de`
[⋆⋆⋆] `firstname.lastname <at> telemotive.de`

growing number of displays of steadily increasing size. The availability of these displays has lead to the desire to use them flexibly, by dynamically mapping applications to display areas. For instance, users desire to customize the *Human-Machine Interface* (HMI), e.g., reduce the size of the speedometer in favor of a larger display area for navigation, or choose between HMI themes.

The dynamic mapping of applications to display areas introduces one great challenge: ensuring safety. It is the responsibility of the *Original Equipment Manufacturers (OEMs)* to ensure that graphical outputs from the various applications do not violate safety requirements. Different standards like [15, ISO 26262] and automotive guidelines (e.g., [7]) address the safety aspects of displaying information in vehicles. For example, as regulated by German law (StVZO §57 [16]), the speedometer must be visible. Additionally, warning messages must be displayed at consistent places on the displays, easily perceivable by the driver (e.g., the brake warning light is statically mapped to a place and guaranteed to be visible).

Most of these requirements apply to specific situations or states of a vehicle, only. For instance, the visibility of the speedometer applies only to moving vehicles, thus the display area of the speedometer could be used for any purpose while the car is parking. Moreover, the display area used by the break warning light could be used for extended radio information. In order to guarantee safe display access, the flexibility must be restricted adequately, using the automotive requirements and guidelines. To this end, we use the notion of *contexts* to dynamically determine at some point in time, which application is allowed to access which display area. A context can be derived from vehicle sensors (e.g., speed, location, or time), or be an application-specific state (e.g., which menu item is selected). Integrating context natively into access control, significantly improves safety, flexibility, and efficiency, since a certified component is in charge.

Moreover, the growing number of applications and the desire to integrate third-party applications (e.g., from Google Play), favors a decentralized developement process. While traditional access control assumes a central authority, e.g., to assign the access control matrix, a decentralized development process requires access control that supports decentralized granting of permissions.

In [10] we proposed an access control model that provides access control to display areas, where access decisions only depend on the applications and reaction on context changes is completely left to the applications in a distributed fashion. However, correctly considering the context of the car or applications often determines whether the automotive requirements are fulfilled or not. Thus, inherent support for contexts is an obvious evolution of the concepts in [10].

In this paper, we extend our approach by concepts for *context-aware* access control, which allow for adapting access permissions based on the context of the car or the applications without compromising on safety. Our model grants permissions to exclusively access certain display areas to applications depending on the current context. In detail, we make the following contributions: (1) A formal definition of the context-aware access control model and the required properties like isolation. (2) A proof-of-concept implementation to show the feasibility of the approach. (3) An evaluation of the performance of our implementation.

The rest of this paper is structured as follows. In Sec. 2 we present our system model. In Sec. 3 we define our context-aware access control model and address correctness in Sec. 4. In Sec. 5 we define the protocol for state transitions and proof the correctness of our protocol. We present our implementation in Sec. 6 and evaluate the latency introduced by access control in Sec. 7. We discuss related work in Sec. 8 and conclude in Sec. 9.

2 System Model

In this section, we introduce our system model (c.f. Fig. 1) for context-based display access control in automotive HMI systems. The *display surface* is a shared resource represented by the set of all available pixels of the connected displays. Applications present their graphical output on *display areas*, which are defined as subsets of the display surface. The mapping between applications and display areas is dynamic and performed by the *Access Control Layer* depending on the current *context* and the *permissions*. Each application authenticates itself to the Access Control Layer. To this end, each application has a Universally Unique Identifier (UUID). A context represents a distinct situation of the car or of an application and can be set only by the responsible application using the *Context Manager*. A *permission* defines which application is allowed to access which display area in which context. To prevent inconsistencies, each pixel must be mapped to at most one application. In case the visibility of an application is restricted, depending on the context permissions must be revoked if this context is active, e.g., context "car in motion" requires that a video playback must not be visible to the driver. To guarantee conflict-freeness either two permissions do not allow access to the same or part or the same display area or their mapped contexts cannot be active at the same time. If the context changes, the application allowed to access a certain display area might also change.

Figure 2 shows an example where a display area located in the center of the IC display is shared between four applications. Based on the automotive requirements, in such a case the decision which of the four applications gets exclusive access to the display area shall be based on the current set of contexts—namely, "Imminent collision", "Incoming phone call", and "Navigation selected".

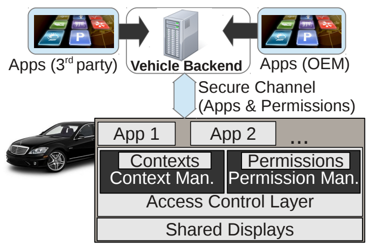

Fig. 1. System model

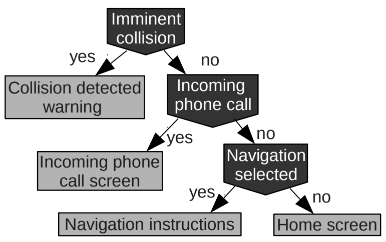

Fig. 2. Context mapping

To facilitate the software development process, the OEM shall be able to pass context-based usage permissions for display areas to software development companies or even individual developers, which again shall be able to pass usage permissions to others in a hierarchical fashion. This allows the OEM to meet all safety-relevant requirements without being a central certification authority for all applications. Nevertheless, for the sake of security, the deployment of applications is centralized using the *Vehicle Backend* (cf., Fig. 1).

3 Context-aware Access Control Model

In this section we present our model for access control to display areas based on contexts. First, we present an overview. Then, we define the entities and describe the granting of permissions—constrained by contexts—between applications.

3.1 Overview

Access control mechanisms determine which subjects are allowed to access which objects. In this work, subjects correspond to *applications* and objects to *display areas* whose pixels are accessed by applications. We use *permissions* that allow applications to access *display areas*. Access to display areas is restricted by *contexts* of the car or applications. An application requires a permission restricted to certain contexts—called *constrained-permission*—to access a dedicated display area. Only if the contexts specified in the constrained-permission match the current contexts of the car and applications, access is granted. An application that owns a constrained-permission *cp* can *grant* another application a constrained-permission not exceeding *cp* in both, size and context. Since our model guarantees that at any point in time each pixel is accessed by exactly one application, an application might need to *revoke* a constrained-permission if it wants to access a certain pixel itself. Providing dynamic granting and revoking of constrained-permissions our model suits to a decentralized development process where the OEM delegates the development of certain application components to a contractor, who delegates parts of the application to subcontractors, etc. Next, we define these concepts in a formal model.

3.2 Objects and Subjects

We define a display area (object) as a set of pixels as depicted in Fig. 3. The smallest display area consists of a single pixel called an *atomic object*. The complete display surface is called the *display surface* and consists of all pixels.

Definition 1. $\boldsymbol{\Lambda} = \{\lambda_1, ..., \lambda_n\}$ is a finite set of pixels (atomic objects). A display area is a subset of the set of pixels, formally a display area o is an object $o \in \boldsymbol{O} = \mathcal{P}(\Lambda)\backslash\emptyset$ with O representing the set of all display areas.

An application (subject) requires a permission to access a display area's content.

Definition 2. $\boldsymbol{S} = \{s_1, ..., s_n\}$ is a set of applications (subjects) with $n \geq 1$.

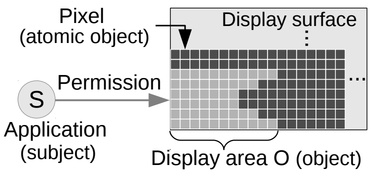

Fig. 3. Subject and objects

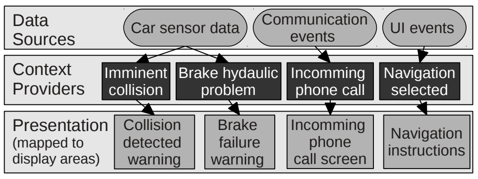

Fig. 4. Contexts

3.3 Contexts

In automotive scenarios setting the graphical content of display areas often depends on context. As depicted in Fig. 4, contexts are identified by three *Data Sources*. The source *car sensor data* provides information about the status of the car (e.g., the RPM of the wheels, or the status of the brakes) or environmental conditions (e.g., the distance to the car in front). The source *communication events* considers events occurring due to incoming information via communication devices, e.g., phone calls, SMS, and mails. Finally, *user interface (UI) events* are triggered by the user input events like selecting the radio in the HU menu. The data from the data sources is interpreted by *Context Providers*, which decide for their contexts whether they are active or inactive. Each context is exclusively mapped to one application which serves as context provider and has an ID, unique within the scope of its application. Next, we define contexts formally.

Definition 3. $\boldsymbol{C} = (S \times \mathbb{N})$ is the set of all contexts, where each context is represented by an application and the context ID. $\boldsymbol{CTX} = \{f : C \rightarrow A\}$ is the set of functions that return for each context its status $\boldsymbol{A} = \{active, inactive\}$.

A context (e.g., with id 1) describes a state like "car is in motion" which is determined by the application speedometer s_{sp}, i.e., $(s_{sp}, 1) \in C$. Hence, in case the current speed is above 0 mph, the speedometer sets the status of its context $(s_{sp}, 1)$ to active otherwise to inactive. For instance, if set $ctx \in CTX$ is a set of contexts and the context $(s_{sp}, 1)$ is active then $((s_{sp}, 1), active) \in ctx$. Let $a \in A$. If $a = active$, the function $inv(a) = a'$ returns $a' = inactive$ else $a' = active$.

3.4 Constrained-Permissions

A constrained-permission represents a permission restricted to contexts. An application is only allowed to access a display area if it has a constrained-permission matching the current contexts. We define a constrained-permission as follows.

Definition 4. $\boldsymbol{CP} : CTX \times S \times O$ is the set of constrained-permissions expressing that an application is allowed to access a display area in certain contexts.

We assume the application speedometer s_{sp} shall only be visible in case the car is in motion (i.e., $c_1 = (s_{sp}, 1)$ is active), as depicted in Fig. 5. To this end, s_{sp} has a constrained-permission cp_1 which allows access to display area o_1, iff the current contexts of $ctx_1 \in CTX$ contains $(c_1, active)$ and $(c_2, inactive)$.

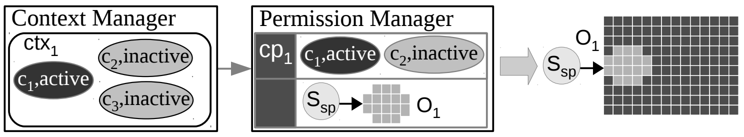

Fig. 5. Example of contexts and constrained-permissions

Next we define *conflict-freeness* of two constrained-permissions, which guarantees that at no point in time a pixel is accessibly by more than one application. Let $cp = (C_1, (s_1, o_1)) \in CP$ and $cp' = (C_2, (s_2, o_2)) \in CP$. We say two constrained-permissions cp and cp' are *conflict-free*, iff either the intersection of the two objects o_1 and o_2 is empty or a context c is in the set of contexts C_1 and in C_2, and the status of both is different.

Definition 5. cp and cp' are ***conflict-free*** $\Leftrightarrow cp \sqcap cp' = \emptyset \Leftrightarrow o_1 \cap o_2 = \emptyset \vee \exists cx = (c, a) \in C_1, \exists cx' = (c', a') \in C_2 : c = c' \wedge a \neq a'$. Let $Im(ctx) = \{ctx(c) | c \in C\}$. We say $cp \sqsubseteq cp' \Leftrightarrow Im(ctx_{cp'}) \subseteq Im(ctx_{cp}) \wedge o_{cp} \subseteq o_{cp'}$.

For instance, let $cp_{sp} = (\hat{C}, s_{sp}, o) \in CP$ and $cp_{acc} = (\tilde{C}, s_{acc}, \hat{o}) \in CP$ be constrained-permissions of speedometer and adaptive cruise control, respectively, where o covers $\hat{o}$. In this case, to guarantee conflict-freeness means that $\hat{C}$ and $\tilde{C}$ cannot both match the same set of current contexts.

To increase flexibility of obtaining and releasing constrained-permissions, we next introduce hierarchical granting and revoking of constrained-permissions.

Hierarchical Granting of Constrained-Permissions An application that received a constrained-permission cp can itself grant a constrained-permission cp' to other applications under the following conditions. First, the display area of cp' is a subarea of the display area of cp. Second, cp' must be at least as constraining (in terms of contexts) as cp. Third, if other constrained-permissions have been granted based on cp, all of them must be conflict-free with cp'. An application is no longer allowed to access any pixels included in one of its granted constrained-permissions which match the current contexts. Formally, we define a set that maps to each application its granted and received constrained-permissions.

Definition 6. $\boldsymbol{CONS} = \{f : S \rightarrow \mathcal{P}(S \times CP) \times \mathcal{P}(S \times CP)\}$ maps to each application two sets $\mathcal{P}(S \times CP)$ that contain mappings of constrained-permissions to applications representing the constrained-permissions an application has *received* and the constrained-permissions it has *granted* to other applications.

Let $cons \in CONS, s \in S$. Set $received_{cp}(cons, s) := \{r | (r, g) \in cons(s)\}$ denotes the set of constrained-permissions s has received. Similarly, set $granted_{cp}(cons, s)$ $:= \{g | (r, g) \in cons(s)\}$ is the set of constrained-permissions s has granted. Accordingly, $(s', cp') \in received_{cp}(cons, s)$ indicates that application s has received a constrained-permission cp' from application s' and $(s', cp') \in granted_{cp}(cons, s)$ indicates that application s' granted cp' to s'.

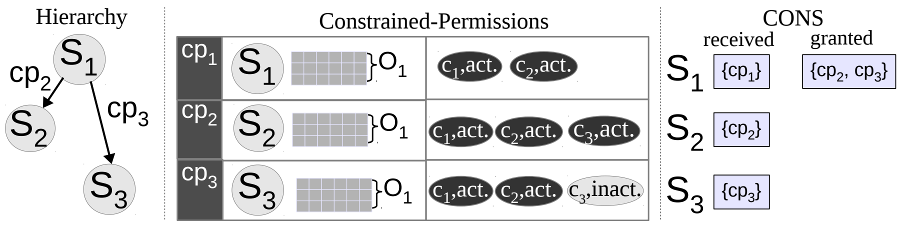

Fig. 6. Example of granting constrained-permissions of S_1 to S_2 and S_3

In Fig. 6 we depict an example of granting constrained-permissions between the applications s_1, s_2, and s_3. We assume set $cons \subseteq CONS$ contains the current constrained-permissions and s_1 received a constrained-permission cp_1 from application $\hat{s}$ (not depicted in Fig. 6) and granted $\{cp_2\}$ and $\{cp_3\}$ to s_2 and s_3, respectively. Thus, the display areas of cp_1, cp_2, and cp_3 are all the same. The constrained-permissions cp_2 and cp_3 are conflict-free ($cp_2 \sqcap cp_3 = \emptyset$) since the context c_3 must be *active* in cp_2 and *inactive* in cp_3 which cannot happen at the same time. Since each context that matches cp_1 also matches either cp_2 or cp_3, s_1 has no longer access to o_1 in any context (as long as it does not revoke cp_2 or cp_3). Hence, the display area an application can use actually depends on the current contexts and the constrained-permissions it granted to other applications.

We define the comparison operator $<_{cp}$ for applications to indicate the chain of dependencies according to constrained-permissions. An application which received a constrained-permission has a dependency to its granting application.

Definition 7. Let $s, s' \in S; cp \in CP; cons \in CONS$. We say $\boldsymbol{s <_{cp} s'} \Leftrightarrow$

$$\exists s_1, ..., s_n \in S; \exists cp_1, ..., cp_{n-1} \in CP : \tag{7.1}$$
$$s_1 = s' \wedge s_n = s \wedge cp \sqsubseteq cp_{n-1} \wedge \tag{7.2}$$
$$\forall i : 1 \leq i < n : (s_i, cp_i) \in received_{cp}(cons, s_{i+1}) \wedge \tag{7.3}$$
$$\forall i : 1 \leq i < n-2 : cp_{i+1} \sqsubseteq cp_i \tag{7.4}$$

If an application s has received a constrained-permission cp from application s' or indirect by using a chain of intermediate applications then $s <_{cp} s'$. Hence, s depends on s' according to the constrained-permission cp.
Formally, with $s, s' \in S$, we say $s \neq_{cp} s' \Leftrightarrow \nexists cp \in CP : s <_{cp} s' \vee s' <_{cp} s$.

4 Safety Property

In this section, we introduce *states* in our model and describe the correctness of our model using a *safety property* defined using states. States are called *safe*, if they fulfill the safety property. A *state* in our model is a set of sets of constrained-permissions and contexts.

Definition 8. $U : CONS \times CTX$ represents the set of contexts, granted and received constrained-permissions. We say $u \in U$ is a *state* in U.

Next, we define a *safety property* which can be satisfied in a state in U or not. States satisfying this property are *safe states* and a sequence of safe states is called *safe state sequence.*

Definition 9. A state satisfies the **Conflict-freeness Property (CFP)**, if each constrained-permission is conflict-free. Let $u = (cons, ctx) \in U$.
u satisfies CFP $\Leftrightarrow$

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

This means, it exists at most one constrained-permission that allows access to a display area for a given set of contexts at a time. This property implies **exclusive access**, i.e., in each context each display area is accessed by at most one application. Thus, an application that has access to a pixel is guaranteed to be visible if the according context is given.

5 Protocol

In this section, we describe the *requests* that can be issued by applications in order to change contexts or constrained-permissions and are performed by *transitions* using *rules.* Moreover, we discuss the verification of our protocol, i.e., that our defined model fulfills the defined safety property using our transitions.

5.1 Requests and Transitions

A transition is triggered by a request to add or delete a constrained-permission, or to set the status of a context.

Definition 10. A request $r \in \boldsymbol{RP} = RA \times S \times S \times CP$ consists of the operation mode $\boldsymbol{RA} = \{append, discard\}$, grantor, grantee, and the constrained-permission. A request $\boldsymbol{RC} = S \times C \times A$ consists of the application, the context, and the desired status. The set of all possible requests is $\boldsymbol{R} = RP \cup RC$.

We define transitions between states. To maintain consistency, these transitions are restricted by rules. If none of the rules apply then the state does not change.

Definition 11. $trans : U \times R \rightarrow U$ is a function which represents the transition from one state to another in U initiated by a request $r \in \boldsymbol{R}$.

The three possible types of request, in particular, grant a constrained-permission, revoke a constrained-permission, and set the status of a context, are described in the following.

To **grant a constrained-permission** cp to another application s', an application s initiates a request $r \in RP$ with $ra = append$. The request is accepted, if s has received a constrained-permission which is an super-set of cp and cp is conflict-free to all granted constrained-permissions of s—expressed by Rule 1.
Rule 1 To satisfy Rule 1, the following condition $cond_1$ has to be fulfilled.

$$cond_1 = (r = (ra, s, s', cp) \in RP \wedge ra = append \wedge cp = (C_1, \hat{s}, \hat{o}) \wedge$$
$$s \neq s' \wedge s' = \hat{s} \wedge [\forall cp' \in \{\hat{cp} \in CP | \exists(\hat{s}, \hat{cp}) \in granted_{cp}(cons, s)\} : cp \sqcap cp' = \emptyset] \wedge$$
$$[\exists \tilde{cp} \in \{\hat{cp} \in CP | \exists(\hat{s}, \hat{cp}) \in received_{cp}(cons, s)\} : cp \sqsubseteq \tilde{cp} = (\tilde{C}, (\tilde{s}, \tilde{o})) \wedge \tilde{s} = s])$$

To perform the grant transition we use the function $add_{cp} : CONS \times S \times S \times CP \rightarrow CONS$ that adds a constrained-permission. Let $s, s' \in S; cp = (C_{cp}, s_{cp}, o_{cp}) \in CP$. We say $[cons' = add_{cp}(cons, s, s', cp)] \Leftrightarrow$

$$cons'(s) = (received_{cp}(cons, s), granted_{cp}(cons, s) \cup \{(s', cp)\}) \wedge \quad (5.1.1.1)$$
$$cons'(s') = (received_{cp}(cons, s') \cup \{(s, cp)\}, granted_{cp}(cons, s')) \wedge \quad (5.1.1.2)$$
$$[\forall s'' \in S \backslash \{s', s\} : cons'(s'') = cons(s'')] \quad (5.1.1.3)$$

If the condition $cond_1$ is fulfilled the function add_{cp} adds cp to the set of $received_{cp}$ and $granted_{cp}$ constrained-permissions for s and s' in $cons$ (5.1.1.1-2). All other applications are not affected by this function (5.1.1.3).

To **revoke a constrained-permission** cp, an application s initiates a request $r \in RP$ with $ra = discard$. If s has granted cp, the request is accepted, and, additionally, constrained-permissions depending on cp will also be revoked—formally expressed by Rule 2.
Rule 2 To satisfy Rule 2, the following condition $cond_2$ has to be fulfilled.

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

To perform the transition we use the function $del_{cp} : CONS \times S \times S \times CP \rightarrow CONS$ that deletes a constrained-permission. Let $s, s' \in S; cp = (C_{cp}, s_{cp}, o_{cp})$; $cp' = (C_{cp'}, s_{cp'}, o_{cp'}) \in CP$. We say $[cons' = del_{cp}(cons, s, s', cp)] \Leftrightarrow$

$$[\forall s_1 \in S : s_1 <_{cp} s \Rightarrow \quad (5.1.2.1)$$
$$received_{cp}(cons', s_1) = received_{cp}(cons, s_1) \backslash \quad (5.1.2.2)$$
$$\{(s_2, cp') \in S \times CP | s_2 <_{cp'} s_1 \wedge cp' \sqsubseteq cp\} \wedge \quad (5.1.2.3)$$
$$granted_{cp}(cons', s_1) = granted_{cp}(cons, s_1) \backslash \quad (5.1.2.4)$$
$$\{(s_2, cp') \in S \times CS | s_1 <_{cp'} s_2 \wedge cp' \sqsubseteq cp\}] \wedge \quad (5.1.2.5)$$
$$cons'(s) = (received_{cp}(cons, s), granted_{cp}(cons, s) \backslash \{(s', cp)\}) \wedge \quad (5.1.2.6)$$
$$[\forall s_3 \in S \backslash \{s', s\}; \forall cp' \in CP : cp' \sqsubseteq cp \wedge s \neq_{cp'} s_3 \vee s <_{cp'} s_3 \quad (5.1.2.7)$$
$$\Rightarrow cons'(s_3) = cons(s_3)] \quad (5.1.2.8)$$

If the condition $cond_2$ is fulfilled, the function del_{cp} removes cp and all depending constrained-permissions from $received_{cp}$ (5.1.2.2-3) and $granted_{cp}$ (5.1.2.4-5). All other constrained-permissions that did not receive or grant a permission depending on cp are not affected by this function (5.1.2.8).
To **change the status of a context** c, an application s initiates a request $r \in RC$. The request is accepted, iff s introduced c.
Rule 3 To satisfy Rule 3, the following condition $cond_3$ has to be fulfilled.

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

To perform the transition, we use the function $set_{ctx} : CTX \times C \times A \to CTX$ which adds a context to the set of current contexts. Formally, $set_{ctx}(ctx, c, a) = ctx' \Leftrightarrow ctx'(c) = a \wedge \forall c' \in C \backslash \{c\} : ctx'(c') = ctx(c')$
Next, we use the three rules to define the function $trans$. Let $u = (cons, ctx) \in U$, and $r \in R$ with $r = (ra, s, s', cp) \in RP$, or $r = (s, c, a) \in RC$. We define

$$trans(u, r) = \begin{cases} (add_{cp}(cons, s, s', cp), ctx), & \text{if } cond_1 \;\; \text{(Rule 1)} \\ (del_{cp}(cons, s, s', cp), ctx), & \text{if } cond_2 \;\; \text{(Rule 2)} \\ (cons, set_{ctx}(ctx, c, a)), & \text{if } cond_3 \;\; \text{(Rule 3)} \\ u, & \text{otherwise} \end{cases}$$

5.2 Protocol Correctness Verification

To verify the correctness of our model we use the Conflict-freeness Property (CFP) (cf., Sec. 4) and a *system* that consists of sequences of states and requests. We use this system to define a proposition which we prove by using complete induction over the states. Finally, we prove that the system is safe if the initial state fulfills the CFP. For instance, using an initial state where a single application has a constrained-permission for the whole screen area for a certain context, is a safe state. Our proof implies that using our protocol, only safe states can be reached. Next, we give the formal definitions.

System In this section, we give the formal definitions for the sequence of states and the system. A *system* consists of all possible *sequences* of requests and the sequence of all states starting from a given initial state. We denote $I^n \subset \mathbb{N}_0$ as a finite set with $I^n = \{0, 1, 2, 3, ..., n\}$.

Definition 12. The set of sequences is a set of n-tuples and defined as $X^{I^n} = \{(x_0, ..., x_i, ..., x_n) | x_i \in X \wedge i \in I^n \wedge x_i = f(i) \text{ with } f : I^n \to X\}$.
We say $(x_0, x_1, ..., x_n) \in X^{I^n}$ is a sequence with $x_0 := x_0 \in X$, $x_1 := x_1' \in X$,..., $x_n := x_n^{(n)} \in X$.
The **operator** $\succ$ indicates whether an element is part of a sequence or not. Let $(x_0, x_1, ..., x_n) \in X^{I^n}$. We define $x \succ (x_0, x_1, ..., x_n) \Leftrightarrow \exists i \in I^n : x = x_i$.

This means, a sequence is an ordered list of elements and the **operator** $\succ$ indicates whether an element is part of that sequence. After the generic definition of sequences we next define sequences of requests and sequences of states.

Definition 13. For a **sequence of requests** $(r_0, ..., r_{n-1}) \in R^{I^{n-1}}$ the **sequence of states** generated by $(r_0, ..., r_{n-1})$ is defined as $(u_0, u_1, ..., u_n) \in U^{I^n}$ with $\forall i \in I^{n-1} : u_{i+1} = trans(u_i, r_i)$.

Using the definition of sequences of requests and states we next define a *system* that consists of an initial state and all possible states that can be reached by using sequences of requests. In addition, we define the **operator** $\gg$ that indicates whether a transition (u, r, u') is part of a *system* or not.

Definition 14. A **system** $\boldsymbol{\Psi(u_{start})} \subset R^{I^n} \times U^{I^n}$ is generated by initial state u_{start}. Let $x_r = (r_0, ..., r_{n-1}) \in R^{I^{n-1}}$; $x_u = (u_0, ..., u_n) \in U^{I^n}$.
We define $(x_r, x_u) \in \Psi(u_{start}) \Leftrightarrow$

$$u_0 = u_{start} \wedge \forall i \in I^n \backslash \{0\} : u_i = trans(u_{i-1}, r_{i-1})$$

Let $(u, r, u') \in U \times R \times U$, $u_0 \in U$. We define $\boldsymbol{(u, r, u') \gg \Psi(u_0)} \Leftrightarrow$

$$\exists x_r \in R^{I^{n-1}}, \exists x_u \in U^{I^n}, \exists i \in I^n \backslash \{0\} : \tag{14.1}$$
$$(x_r, x_u) \in \Psi(u_0) \wedge u_i \succ x_u \wedge u_{i+1} \succ x_u \wedge r_i \succ x_r \wedge \tag{14.2}$$
$$(u, r, u') = (u_{i-1}, r_{i-1}, u_i). \tag{14.3}$$

This means, that (u, r, u') is part of a system if a sequence of requests and states (14.1) exists, of which u, r and u' are part of (14.2) and a transition from state u to u' by request r (14.3) exists.

Next, we define a *safe state* and a *safe system* that consists only of *safe state sequences.*

Definition 15. $u \in U$ is a **safe state** $\Leftrightarrow$ u satisfies the CFP. $(u_0, ..., u_n) \in U^{I^n}$ is a **safe state sequence** $\Leftrightarrow$ $\forall i \in I^n : u_i$ is a safe state. A system $\Psi(u_0) \subset U^{I^n} \times R^{I^n}$ with $x_r \in R^{I^{n-1}}$ and $x_u = (u_0, ..., u_n) \in U^{I^n}$ is a **safe system** $\Leftrightarrow$ $\forall (x_r, x_u) \in \Psi(u_0) : x_u$ is a safe sequence.

By using the definition of a *safe system* we prove in the next section that the CFP defined in Sec. 4 is always satisfied if the initial state satisfies CFP.

5.3 Proof

Since the states and transitions of our model consist of mathematical formulations, we define a proposition that corresponds to the CFP defined in Sec. 4 and helps us to prove the safety of our model. Let $u, u', u_0 \in U$; $u' = (cons', ctx')$; $u = (cons, ctx)$; $r \in R$.

Proposition 1: All sequences in $\Psi(u_0)$ satisfy CFP for all u_0 that satisfy CFP $\Leftrightarrow \forall (u, r, u') \in U \times R \times U : (u, r, u') \gg \Psi(u_0) \Rightarrow u, u' \in U$ satisfy CFP.

Proposition 1 says that all sequences in a system $\Psi(u_0)$ satisfy CFP if, and only if, for all states u' which can be directly generated from any state u with one request, the respective states u and u' also satisfy CFP. If this proposition holds, every system $\Psi(u_0)$ is a safe system if state u_0 satisfies the CFP in Sec. 4.

To prove the correctness of Proposition 1 we define a lemma to prove the proposition using complete induction over the states of the system.
The following Lemma CFP states that a transition from a state which satisfies the CFP will always end in a state which also satisfies the CFP.

Lemma CFP: All sequences in $\Psi(u_0)$ satisfy CFP for all u_0 which satisfy CFP $\Leftrightarrow \forall (u, r, u') \in U \times R \times U : (u, r, u') \gg \Psi(u_0) \Rightarrow u, u' \in U$ satisfy CFP.

Proof: According to Sec. 5, a request $r \in R$ is either in RP or in RC. The case $r \in RC$ is trivial, since Rule 3 (Sec. 5) does not change the set of constrained-permissions *cons* and therefore is not relevant for Lemma CFP. In case $r = (ra, s, s', cp) \in RP$, we have to consider the following three sub-cases:

(R1): Let $ra = append$ and the condition $cond_1$ be fulfilled. We follow that the transition $trans(u, r) = (add_{cp}(cons, s, s', cp), ctx)$ leads to the set of received constrained-permissions $received_{cp}(cons', s') = received_{cp}(cons, s') \cup \{(s, cp)\}$.
We first show that cp is conflict-free with all granted constrained-permissions of s. Then, we show that cp is conflict-free with all constrained-permissions which do not depend on s. Finally, we follow that state u' satisfies the CFP.
In detail, since u satisfies CFP and condition $cond_1$ is fulfilled, we follow $\forall (\tilde{s}, \tilde{cp}) \in granted_{cp}(cons, s) : cp \sqcap \tilde{cp} = \emptyset$ with cp granted by s. In addition, we know that $(s, cp) \notin received_{cp}(cons, s')$ due to $cond_1$. We follow in state u' that cp is conflict-free with all granted constrained-permissions of s.
In addition, $cond_1$ implies $\exists \hat{s} \in S$ with $(\hat{s}, \hat{cp}) \in received_{cp}(cons, s) : cp \sqsubseteq \hat{cp}$. Since state u satisfies CFP, we know that $\hat{cp}$ is conflict-free for all applications and constrained-permissions in the state u. We follow with $cp \sqsubseteq \hat{cp}$ and $(s, cp) \in received_{cp}(cons', s')$ that cp is conflict-free with all constrained-permissions which do not depend on s in state u'.
Finally, due to $\forall cp'' \in CP, \forall s, s_1, s_2 \in S''$: $(s_1, cp'') \in received_{cp}(cons', s'')$ and $(s_2, cp) \in received_{cp}(cons', s')$, it follows $cp \sqcap cp'' = \emptyset$ since $s' \neq_{cp} s''$ and $s' \neq_{cp''} s$.

(R2): Let $ra = discard$ and $cond_2$ be fulfilled. Then it follows $trans(u, r) = (del_{cp}(cons, s, s', cp), ctx)$.
Since u satisfies CFP we know $\forall (\tilde{s}, \tilde{cp}) \in granted_{cp}(cons, s) : cp \sqcap \tilde{cp} = \emptyset$.
Moreover, we know that in the state u the constrained-permission cp received by the application s' ($(s, cp) \in received_{cp}(cons, s')$) is conflict-free to all other constrained-permissions, i.e.,

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

We know that all constrained-permissions in the set of received constrained-permissions are conflict-free in state u and therefore cp is also conflict-free.

After the transition we know that the function $del_{cp}(cons, s, s', cp)$ leads to $(s, cp) \notin received_{cp}(cons, s')$ and additionally removes all constrained-permissions that depend on the constrained-permission cp, i.e., $\forall s_1 \in S : s_1 <_{cp} s \Rightarrow received_{cp}(cons', s_1) = received_{cp}(cons, s_1) \setminus \{(s_2, cp') \in S \times CP | s_2 <_{cp'} s_1 \wedge cp' \sqsubseteq cp\}$. Hence, in state u' the constrained-permission cp and all depending constrained-permissions are removed from the set of received and granted constrained-permissions of all applications, whereas all the other constrained-permissions are not changed, i.e., $\forall s_3 \in S \setminus \{s', s\}; \forall cp' \in CP : cp' \sqsubseteq cp \wedge s \neq_{cp'} s_3 \vee s <_{cp'} s_3 \Rightarrow cons'(s_3) = cons(s_3)$. We follow, that in state u' all remaining constrained-permissions are also conflict-free and state u' satisfies the CFP.

(otherwise): We know that $u' = u$. Since u satisfies the CFP, it follows that u' also satisfies CFP.

Finally, we prove Proposition 1 by complete induction.
Let $(x_r, x_v) \in \Psi(u_{start})$ with $x_r = (r_0, ..., r_{n-1}) \in R^{I^{n-1}}$, $x_u = (u_0, ..., u_n) \in U^{I^n}$. We define $u_0 = u_{start}$ as initial state and generate the states in x_u by using our transition: $\forall i \in I^n \setminus \{0\} : u_i = trans(u_{i-1}, r_{i-1})$. The state u_0 satisfies the CFP (induction base). Let $\forall i \in I^n \setminus \{0\} : (v_{i-1}, r_{i-1}, v_i) \gg \Psi(v_0)$, cf. Def. 14.3.

Base: u_0 satisfies the CFP. With $u_1 = trans(u_0, u_0)$ we conclude u_1 satisfies the CFP, according to Lemma CFP.

Induction hypothesis: u_i satisfies the CFP.

Induction step: Let u_i satisfy the CFP. From the Lemma CFP follows $u_{i+1} = trans(u_i, r_i)$ satisfies the CFP. $\square$

We follow that all systems $\Psi(u_0)$ are safe systems if the state u_0 satisfies the CFP. This means, that our transitions do not violate the CFP. Hence, implementing our access control model by initially assigning all available display area for a certain context to a single application—which is a state that satisfies the CFP—leads to a safe system.

6 Implementation

In this section we describe the proof-of-concept implementation of our access control model for an automotive HMI system. Basically, an application that wants to display content on the screen, first needs to obtain a permission which then is used to create a window. Our compositor copies the content of the windows to the screen-buffer which makes them appear on the displays.

Traditionally, compositors operate on rectangular windows. In contrast, our compositor operates on (typically) non-rectangular windows, since the set of pixels (atomic objects) in a constrained-permission (CP) might not be rectangular. To this end, our implementation uses windows with a bitmap attached that represents the subset of pixels of the windows, the application is actually allowed

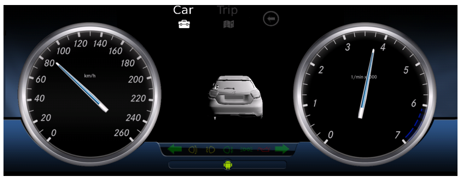

Fig. 7. Automotive Scenario using typical IC applications

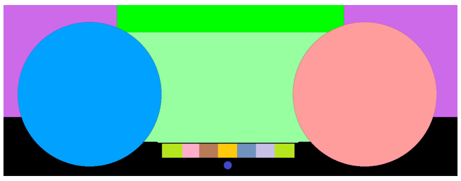

Fig. 8. Bitmasks of the **Automotive Scenario**

to access. In Fig. 7, we depict a typical **Automotive Scenario** for a given set of CPs and active contexts, using applications like speedometer, tachometer, indicators, trip, car status, and menu. In Fig. 8, we depict the bitmasks used in the active CPs of Fig. 7. All CPs are conflict-free and hierarchically granted in the order of their criticality. Besides this depicted scenario, our implementation includes many more scenarios for IC and HU. For instance, we implemented CPs that allow to display 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 navigation system in the middle of the screen.

The implementation consists of the software components depicted in Fig. 9. To isolate the safety-critical applications (e.g., brake failure warning) from the non-safety-critical applications (e.g., media playback) traditionally running physically isolated on IC and HU, we use different virtual machines (VM) running on the virtualization solution PikeOS from Sysgo. We use a dedicated VM—called *Virtualization Manager*—which provides access control and has exclusive access to the hardware components GPU, displays, and input devices.

The *Isolated Communication Channel* provides session-based FIFO communication between the applications in the VMs and the system components of the *Virtualization Manager* by using shared memory. The applications communicate via *Communication Channel* with the Virtualization Manager to get access to the hardware components. The *Communication Manager* stores the certificates of the applications, forwards them to the *Authentication Manager (AM)*, and creates a dedicated communication channel to the system components for each authenticated application. The AM restricts the communication between appli-

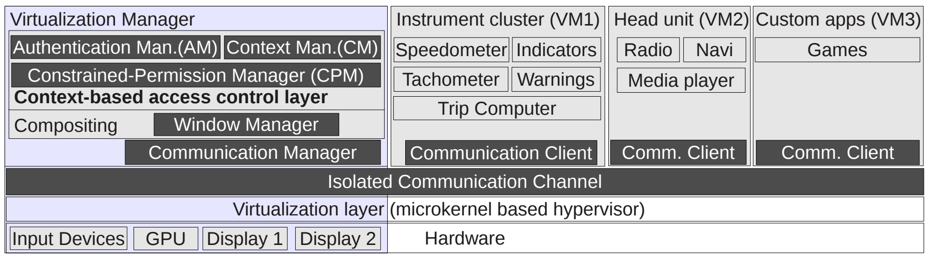

Fig. 9. Implemented architecture

cations and system components based on their identification. The context-based access control layer performs access decisions using contexts. The *Context Manager* (CM) provides context handling for applications which can set the status of contexts by using the API call $setContext(CTX\ c, CA\ a)$ (cf. Rule 3 in Sec. 3) with context c and status a. The CPM handles granting and revoking of CPs and provides access decisions for the *Window Manager* (WM) which is responsible for creating, destroying, and positioning of windows. Applications can grant or revoke CPs by calling $grantConsPerm(CTX[(c_1, a_1), (c_2, a_2), ...], uuid_A, o)$ (cf. Rule 1 in Sec. 3) with the list of contexts $[(c_1, a_1), (c_2, a_2), ...]$, the targeting application $uuid_A$, and the display area o or $revokeConsPerm(ID\ cp_{id})$ (cf. Rule 2 in Sec. 3) with the id cp_{id} of the CP. Access control is enforced through pixel-exact CPs which are implemented using rectangular areas in combination with bitmasks that restrict operations to the allowed pixels. Applications can create, modify, or move a window within the bounds of a received CP. We created a *Compositing Layer* that provides an API for resizing and mapping of windows. Applications directly render into the off-screen buffers of their windows, and the compositing takes care of pixel-exact copying of the window contents to the screen buffers of the two displays. Due to the exclusive access property of our system, the compositing does not need to care about the layering of windows.

We use a cockpit demonstrator (cf. Fig. 10) with two automotive 12" displays each with a resolution of 1440×540 pixels and common input devices like steering wheel buttons and the central control knob to control the applications. The hardware platform is 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 which supports vector graphics with OpenVG 1.1, and the GC320 which provides compositing of framebuffers with a 2D API. We use the OpenGL ES 2.0 API for the rendering of the graphical content into the applications' backbuffers. The compositing layer uses the 2D API of the Image Processing Unit (IPU) for copy operations in framebuffers.

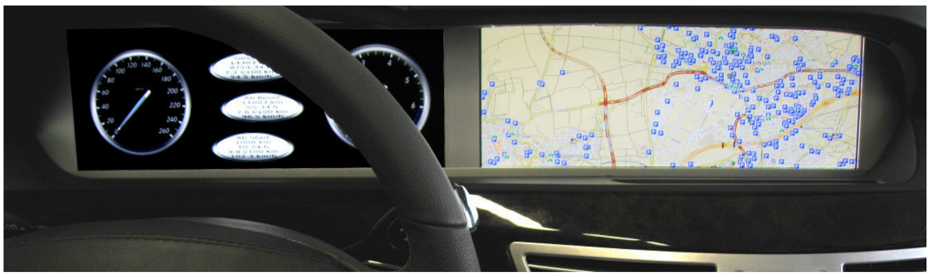
Fig. 10. Cockpit demonstrator

7 Performance Evaluation

We evaluated the performance of our access control model implementation. To this end, we measured the latency introduced by our access control. Since some context changes are safety-critical the time required to change the access to display areas delays the visibility of safety-critical applications. As described in [11], important information shall be visible within given time constraints. The time constraint is not a fixed value, but determined by the OEM based on automotive guidelines, ISO standards, and legal requirements. However, the required maximum latency for graphical output does not exceed 2 seconds. For instance, the image of the rear view camera shall be visible in no more than 2 s after shifting into reverse (as demanded by US National Highway Traffic Safety Administration). Thus, 2 s can be considered as generic upper bound. On the other hand, for time-critical user interaction, typically latency shall not exceed 250 ms [11]. More precisely, the time delay required to change the access to display areas and to allow applications to request new windows is crucial since it delays the visibility of safety-critical applications. Next, we describe the evaluation setup and present the results.

7.1 Setup

The latency introduced by our access control system primarily depends on the number of permissions that need to be changed. A permission can change either by being disabled, so that the application window is no longer visible, or by being enabled, making an application window visible. The latency to *get* a new permission, is denoted as Δt_{get}, i.e., the time between sending the command $setContext(CTX\ c, A\ a)$ to set a context c and receiving the command $confirmWindow(WinID\ id)$ which returns the id of the created window. The latency to *cede* an owned permission, is denoted as Δt_{cede}, i.e., the time between sending $setContext(CTX\ c, A\ a)$ and revoking access by using the command $revoke(o_x)$. Fig. 11 depicts the messages sent between the components of our model, for a scenario where the permissions of two applications (B and C) are affected. In more detail, application A sets the context $c_1 = (A, 1)$ to status a by sending $setContext(c_1, a)$ to the CM. The CM changes the context status and notifies the CPM. The CPM determines the affected permissions and notifies

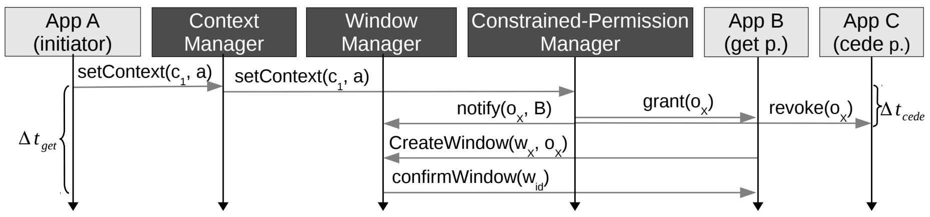

Fig. 11. Scenario, measuring access control latencies

the WM ($nofify(o_x, B)$, where o_x is the new bitmap) and the affected applications. Application C is notified ($revoke(o_x)$), that it has no longer access to o_x. Application B is granted a new permission ($grant(o_x)$) and creates windows w_x with $createWindow(w_x, o_x)$ using the bitmap o_x. Finally, the WM sends $confirmWindow(w_{id})$ to B.

We evaluated two different scenarios consisting of up to 16 applications. In the **first scenario** one application provides up to 15 constraint-permissions to 15 separate display areas, as depicted in Fig 12. By setting a context, access to these dedicated display areas is granted to up to 15 different applications at once. After each applications successfully created a window the application revokes the granted permissions by setting back the context status. We call this *flat-granting* of permissions.

In the **second scenario**, each of the 15 applications provides one constraint-permission to part of its display area. The constrained-permissions are derived from one display area where (except for the last) each application grants exactly one application access to a subset of its display area, as depicted in Fig 13. All granted constrained-permissions depend on the same context, i.e., if this context is changed, all applications eventually need to get or cede their respective window. We call this scenarios *deep-granting* of permissions.

7.2 Results

We measured the latencies Δt_{get} and Δt_{cede} for different numbers of affected applications. The average latencies over 100 runs of the *flat-granting* and of the *deep-granting* of permissions are depicted in Fig. 14 and Fig. 15. In addition, the minimal and maximal latencies are depicted.

We observe, that the latency Δt_{get} linearly depends on the number of affected applications that get a new permission and create new windows. Even with 15 new application windows, the latency did not exceed 1 s in both evaluations. Situations where many new windows are created occur only in mode changes of big screen areas, e.g., if the rear-view camera needs to be switched off, or the IC display switches from full-screen video playback back to driving mode. Thus, the upper bound of 2 s (cf., [11]) applies and our implementation is always

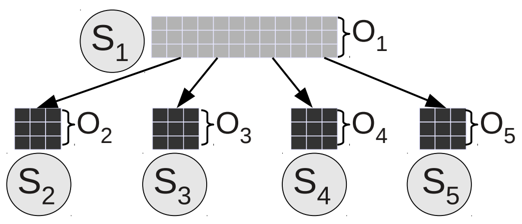

Fig. 12. Example: Flat-granting of display areas to four applications

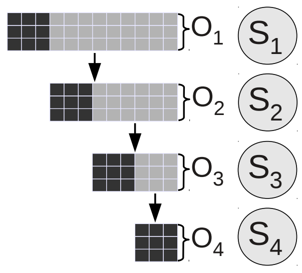

Fig. 13. Example: Deep-granting of display areas

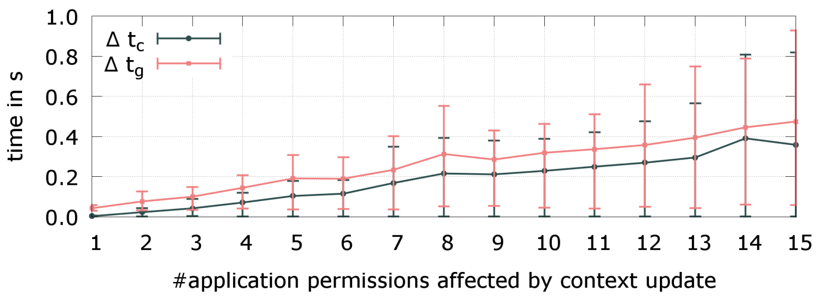

Fig. 14. Latencies of context changes affecting flatly granted permissions

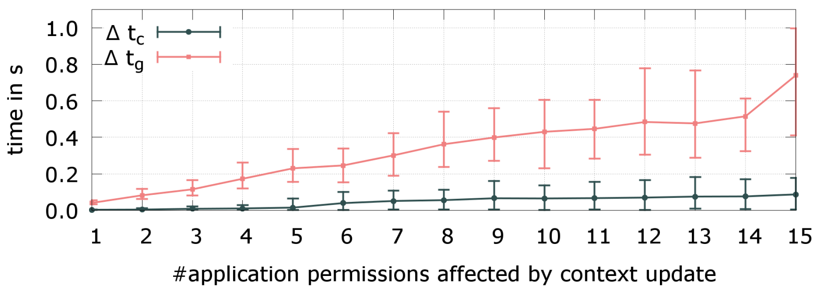

Fig. 15. Latencies of context changes affecting in deeply granted permissions

fast enough. For situations of time-critical user interaction, typically only one application gets a new permission which takes in our implementation up to 58 ms at worst. This is below the 250 ms threshold. In case of sequential granting, the upper bound of ceded permissions Δt_{cede} did not exceed 200 ms, thus staying below the 250 ms threshold. Although flatly granted permissions exceed the 250 ms in case of more than 10 applications, but still stay below 1 s. Thus, our measurement results indicate, that it is fast enough to fulfill automotive requirements for a sufficient amount of applications.

8 Related Work

In [10] we presented an access control concept, a formal model for defining and controlling the access to display areas to guarantee the safe and secure sharing of displays. However, this concept is not aware of contexts, which leaves the applications the complex task to detect combinations of distributed contexts and change the access rights accordingly within tight latency constraints. Window compositing for virtual machines is target of some works (e.g., [8, 9, 12]). However, they do not provide fine-grained access control and assume that the user has full control over window placement which incompatible with automotive requirements. Epstein addresses security issues in the X-server and proposes mechanisms [6] to prevent them. Again, the user controls compositing without restrictions. For context-aware access control, there exists a plethora of work. Schilit and Theimer [18] first introduced *context-awareness* and used *context* as location, identities of nearby people and objects, and changes to those objects. They focus on providing clients with information about located-objects and how those objects change over time. However, they do not consider any access control. The focus of [2, 14, 17, 19] is on role-based access control (RBAC) using contexts to decide which roles are currently active and which according permissions are valid. But either do they consider a system administrator to be responsible for defining the applicable set of permissions for each context or do not provide delegation of access rights. Context-aware access control models which do not rely on RBAC are using context information similar to roles in the access decision process (e.g., [5], [4]). Since they do not provide hierarchically depending permissions the access control prioritized usage of resources is not possible. Herges et al. [13] introduced a generic access control framework which uses an access control model based on context information, a trust model, and the concept of isolation domains to cope with automotive related requirements for infotainment applications. However, they focus on the communication between the components using messages. Our models are state-based systems similar to the Bell and LaPadula model (BLP) [1] which also defines a state machine for enforcing access control. BLP focuses on confidentiality of information and uses an access control matrix for restricting access to data. However, the BLP does neither prevent concurrent access nor allow flexible granting of permission by subjects.

9 Summary and Future Work

Sharing the available screen area between an increasing number of automotive applications becomes more and more important. Due to automotive safety requirements an appropriate access control system is required. Since the context of the car or applications often determines whether the automotive requirements are fulfilled or not it is an obvious evolution to use the context in the access decisions of automotive HMI. In this paper, we present a context-aware access control model that targets safety-critical automotive HMI systems. Our model provides the ability to hierarchically grant access to display areas depending on the contexts of the car and applications and offers dynamic flexibility without compromising on safety. This allows for guaranteed displaying of safety-critical applications and prevents intended or unintended presentation of driver-distracting content while the vehicle is in motion. Our fully formalized model meets the automotive safety requirements which can be formally proved using our defined safety property. To demonstrate the feasibility of our concept we presented a proof-of-concept implementation and evaluated it using an automotive scenario. Our next steps will be the optimization of the graphics forwarding between virtual machines to improve the rendering performance and a GPU scheduler that meets the timing requirements of the safety-critical applications. Additionally, we want to add a virtualized Android VM.

Acknowledgment

This paper has been supported in part by the ARAMiS project of the German Federal Ministry for Education and Research with funding ID 01IS11035.

Bibliography

[1] Bell, D.E., Lapadula, L.J.: Secure Computer System: Unified Exposition and MULTICS Interpretation. Tech. Rep. ESD-TR-75-306 (1976)
[2] Bhatti, R., et al.: A trust-based context-aware access control model for web-services. In: Web Services. Proceedings. International Conference on (2004)
[3] Broy, M., Kruger, I., Pretschner, A., Salzmann, C.: Engineering automotive software. Proceedings of the IEEE 95(2), 356 –373 (Feb 2007)
[4] Corradi, A., et al.: Context-based access control for ubiquitous service provisioning. In: Proceedings of the 28th COMPSAC (2004)
[5] Corradi, A., et al.: Context-based access control management in ubiquitous environments. In: Proceedings of the 3rd NCA (2004)
[6] Epstein, J., et al.: A prototype b3 trusted x window system. In: Proceedings of the 7th Annual Computer Security Applications Conference (1991)
[7] 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)
[8] Feske, N., Helmuth, C.: Overlay window management: User interaction with multiple security domains (2004)
[9] Feske, N., Helmuth, C.: A nitpicker's guide to a minimal-complexity secure gui. In: Proceedings of the 21st ACSAC (Dec 2005)
[10] Gansel, S., et al.: An access control concept for novel automotive hmi systems. In: Proceedings of the 19th SACMAT (2014)
[11] Gansel, S., et al.: Towards Virtualization Concepts for Novel Automotive HMI Systems. In: Proc. of IESS. IFIP LNCS, Springer Berlin (2013)
[12] Hansen, J.G.: Blink: Advanced Display Multiplexing for Virtualized Applications. In: Proceedings of the 17th NOSSDAV (2007)
[13] Herges, D., et al.: Ginger: An access control framework for telematics applications. In: Processing of the 11th TrustCom (2012)
[14] Hong-Yue, L., Miao-Lei, D., Wei-Dong, Y.: A context-aware fine-grained access control model. In: Computer Science Service System (CSSS) (2012)
[15] ISO 26262: Road vehicles – Functional Safety. ISO, Geneva, CH (Nov 2011)
[16] Janker, H.: Straßenverkehrsrecht: StVG, StVO, StVZO, Fahrzeug-ZulassungsVO, Fahrerlaubnis-VO, Verkehrszeichen, Bußgeldkatalog. Beck (2011)
[17] Mostéfaoui, G.K., Brézillon, P.: A generic framework for context-based distributed authorizations. In: Proceedings of the 4th CONTEXT'03 (2003)
[18] Schilit, B., Theimer, M.: Disseminating active map information to mobile hosts. Network, IEEE 8(5), 22–32 (1994)
[19] Strembeck, M., et al.: An integrated approach to engineer and enforce context constraints in rbac environments. ACM Trans. Inf. Syst. Secur. (2004)