Technical Report 2013/02

An Access Control Concept for Novel Automotive HMI Systems

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

[1] System Architecture and Platforms Department
Daimler AG
Benzstraße
71063 Sindelfingen
Germany

[2] Institute of Parallel and Distributed Systems
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

October 2013

An Access Control Concept for Novel Automotive HMI Systems

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

[1] System Architecture and Platforms Department, Daimler AG, Germany*
[2] Institute of Parallel and Distributed Systems, University of Stuttgart, Germany**

Abstract. The relevance of graphical functions in vehicular applications has increased significantly during the last years. Modern cars are equipped with multiple displays used by different applications such as speedometer, navigation system, or media players. However, so far applications are restricted to using dedicated displays. In order to increase flexibility, the requirement of sharing displays between applications has emerged. Sharing displays leads to safety and security concerns since safety-critical and trusted applications as the dashbord warning lights share the same displays with uncritical or untrusted applications like the navigation system or third-party applications. To guarantee the safe and secure sharing of displays, we present a formal model for defining and controlling the access to display areas in this paper. We proof the validity of this model, and present a proof-of-concept implementation to demonstrate the feasibility of our concept.

1 Introduction

Innovations in cars are mainly driven by electronics and software today [6]. In particular, graphical functions and applications enjoy growing popularity as shown by the increasing number of displays integrated into cars. For instance, the head unit (HU)—the main electronic control unit (ECU) of the infotainment system—uses the center console screen to display the navigation system, or displays integrated into the backside of the front seats together with the center console screen to display multimedia content. Displays connected to the instrument cluster (IC) replace analog indicators displaying speed information or warnings. Moreover, head-up displays are used for displaying navigation instructions or assistance messages on the windshield.

Moreover, as demonstrated by advanced used cases already implemented in concept cars, there is a trend to *share* the different available displays *flexibly* by displaying content from different applications on dynamically defined display areas. For instance, while parking, applications can output information on any display including, in particular, the IC display. For example, this allows for playing fullscreen videos on the IC display while the car is not moving. Moreover,

* firstname.lastname <at> daimler.com
** lastname <at> ipvs.uni-stuttgart.de

the window size can be configured dynamically, for instance, to reduce the size of the speedometer in favor of a larger display area of the navigation software. Note that the flexible and dynamic usage of displays allows applications running on *different* ECUs to access all available displays. Even third-party applications downloaded from an app store [2, 12] and running in isolated execution environments—e.g., an Android partition within QNX [2]—or on the mobile phone of the user [1] can be granted access to these displays.

Although these uses cases are very attractive for the user, they come with a great challenge: ensuring safety. Different standards and guidelines consider the safety aspect of displaying information in vehicles. For instance, current standards regulate that the level of safety criticality of each functionality has to be assessed and suitable methods must be implemented to minimize the risks caused by malfunctions [16, ISO 26262]. Moreover, automotive design guidelines [3, 9, 17] require that safety-critical content must be displayed at defined display areas and for all content that is visible to the driver while driving, the potential of distraction must be restricted. For instance, video playback must not be visible to the driver while the vehicle is in motion. Additionally, country-specific laws must be fulfilled, e.g., as regulated by German law (StVZO §57 [18]), the speedometer must always be visible while the car is moving.

Since display sharing results in scenarios where safety-critical applications like the brake warning light share the same display with uncritical applications, concepts for the safe sharing of displays between applications are required. In more detail, display areas have to be isolated such that it is guaranteed that the output of different applications does not interfere. For instance, the break warning light may indicate a potentially severe hydraulic problem in the break system and being covered by other application windows could be critical to the safety of the driver. Therefore, the output of safety-critical applications must be guaranteed to be always visible if required by the status of the car or traffic conditions. Since this also applies to third-party applications, it is the responsibility of the OEMs to ensure that graphical outputs from different applications do not interfere. One approach to ensure this is to test and certify the correct behavior of all applications by the OEM. However, such a certification process is expensive and cumbersome. Therefore, technical solutions are required to ensure the isolation of display areas.

A naïve approach to provide isolation for window placement is to define a static mapping in which the IC applications only access the IC display and the HU applications access the HU display and, additionally, the IC display within a reserved area. However, this approach lacks flexibility since only a predefined number of applications can be supported, and sharing is restricted to pre-defined display areas. Therefore, more flexible methods for dynamic display sharing are required.

In this paper, we present a novel access control model for sharing graphical displays to offer flexibility without compromising on safety. Basically, our model grants applications dynamic permissions to draw into certain display areas. To support the decentralized software development process involving OEMs

and sub-contractors, and possibly third-party developers like app developers, permissions are managed based on a delegation hierarchy such that applications can pass permissions to sub-components, e.g., third-party or sub-contractor components. In detail, we make the following contributions in this paper: 1. A formal definition of the access control model and the required properties such as isolation. 2. A formal proof of correctness of this model. 3. A proof-of-concept implementation to show the feasibility of the approach.

The rest of this paper is structured as follows. In Sec. 2.1 we present our system model and requirements. In Sec. 3 we define our access control model and present properties for the model and proof them in Sec. 4. We present our implementation in Sec. 5, and discuss related work in Sec. 6. We conclude this paper with a summary and an outlook on future work in Sec. 7.

2 System Model and Requiremens

In this section, we describe the components, assumptions and requirements of our system for display access control in an automotive HMI system.

2.1 System Model

The components of our system are depicted in Fig. 1. First of all, we assume that the available *display surface* consists of multiple **displays**. The display surface is shared between all applications. We define a *display area* as a subset of the pixels of the display surface. Each pixel of the display surface is indexed by x and y coordinates and is unambiguously identifiable by its position.

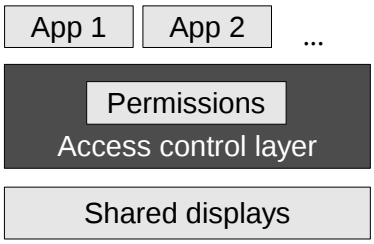

Fig. 1. System model

Applications communicate with the Access Control Layer to get access to display areas. We assume that all applications can be unambiguously identified, e.g., by using Universally Unique Identifiers (UUID). Applications can be deployed or removed dynamically during runtime. Moreover, the usage of the displays by applications is restricted by the context of the car. For instance, video playback is permitted only if the car is not in motion.

The **Access Control Layer** restricts access to the display surface. Since there is no static mapping between applications and display areas, the mapping is performed by the Access Control Layer. Before an application can access a display area, it needs permission from the Access Control Layer, which manages all permissions. All granted permissions are stored within the Access Control Layer so that it is any time decidable if an application get access to a display area or not.

2.2 Requirements

In the following, we present requirements targeting access control of the display surface which need to be fulfilled to ensure safety in automotive HMI systems.

Req. 1 – Dynamic permissions: An application shall be allowed to access a display area if and only if there exists a corresponding permission. A permission shall be grantable and revokable during runtime of the system to meet the different demands of the applications which can be influenced by the status of the car or traffic conditions.

Req. 2 – Priorities: Applications shall have priorities assigned. Priorities can depend on importance, urgency, criticality, and legal requirements for displaying graphical content (cf. [13], Req. 2.2 – Priority-based Displaying of Windows). If multiple applications want to access the same display area, access shall depend on their priorities.

Req. 3 – Safe access: Each pixel shall be mapped to *exactly* one application. This requirement consists of the following two sub requirements.

Req. 3.1 – Exclusive access: Each pixel shall be mapped to *at most* one application. Thus, an application that has access to a pixel is guaranteed to be visible there. This effectively prevents competing access to display areas caused by malicious or malfunctioning applications.

Req. 3.2 – Completeness: Each pixel shall be mapped to *at least* one application. For each pixel there exists an application that has a permission to set its content and can grant access to it, and dead pixels are avoided.

Req. 4 – Delegation: Also for the sake of flexibility, the OEM may pass usage permissions for display areas to software development companies or even individual developers, which can likewise pass usage permissions to others. Passing usage permissions must happen in a way that the OEM can ensure to meet all safety-relevant requirements without being a central certification authority for all applications. Therefore, the applications must have *priorities*. For instance, as depicted in Fig. 2, the OEM passes different usage permissions to Company 1. Company 1 decides to pass a subset of its permissions to Company 2. Therefore, the installation of third-party applications does not require the involvement of the OEM if the applications are only permitted while the car is not in motion

and therefore cannot violate safety-relevant requirements. But this requires a *delegation relation* between the parties for exchanging permissions.

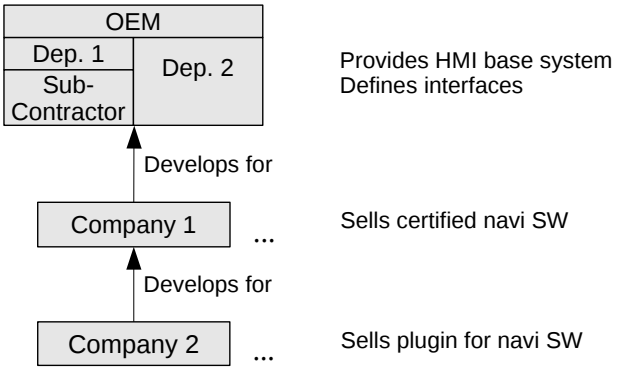

Fig. 2. Example for software development hierarchy

A *decentralized development process* is commonly applied in today's car industry, e.g., service-based software development [19]. This process involves different departments of the OEM, sub-contractors, as well as third-party developers like application developers for a (future) app store for vehicular apps. An possible scenario is depicted in Figure 2.

In this scenario, the OEM software development is done by different departments and subcontractors. The OEM provides the HMI base system, interface definitions, and certification policy. Company 1 is independent from the OEM and sells navigation software which is compatible and certified for the OEM's HMI system. This navigation software uses display areas dedicated by the OEM to display navigation information to the driver. Company 2 sells an application that enhances the features of this navigation system. For instance, an application could display relevant information about points of interest to the driver at its current position.

As becomes obvious from this scenario, our system has to support the delegation of permissions to access display areas between the involved parties to facilitate the decentralized development process. For instance, Company 1 should be able to delegate a permission for part of its display area to Company 2 for displaying the plugin content.

3 Access Control Model

In order to restrict access of applications to shared display areas, an access control layer is required. To guarantee safety, the access control layer must conform to a well-defined mathematical model that fulfills the requirements of Sec. 2.1. In this section we present our formal state-based model. We define the entities

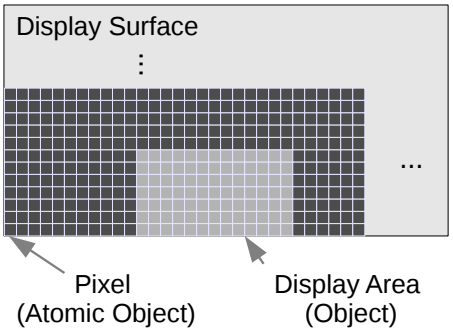

Fig. 3. Entities: Pixels and display areas

and states of our model, the rules for the transitions to change states, and finally present important properties of our system.

3.1 Entities

In general, an access control mechanism controls which *subjects* can access which *objects*. In our context, subjects correspond to *applications* and objects to *display areas*. A display area is defined as a set of pixels as depicted in Fig. 3. The smallest area is one pixel corresponding to an *atomic object*. The complete display area consists of all pixels and is called the *display surface*.

Definition 1. $AO = \{ao_1, ..., ao_n\}$ 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 $o \in O = \mathcal{P}(AO) \backslash \emptyset$ with O representing the set of all display areas.

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

3.2 States

Each *state* of our model is represented by the currently valid *permissions* and *delegation relations*. The delegation relationships that are defined between applications determine to which other applications a subset of a permission can be passed.

Definition 3. A permission grants an application access to a certain display area. Formally, $P = \mathcal{P}(S \times O)$ represents the set of sets of permissions.
$B = \{f : S \rightarrow P \times P\}$ maps to each application in S two sets of sets of permissions $(P \times P)$ representing the permissions in P an application has *received* from other applications and permissions in P it has *granted* to other applications.

Let $b \in B, s \in S$. The set

$$received(b, s) := \{r | (r, g) \in b(s)\}$$

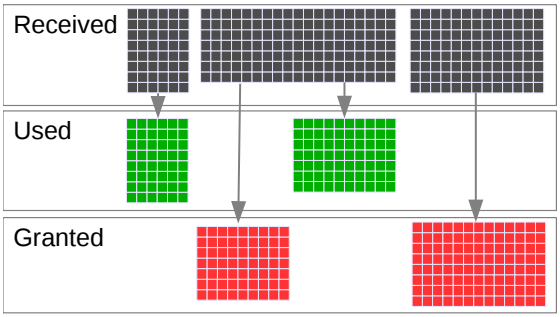

Fig. 4. Example for a set of received, granted, and used display areas of an application

denotes the set of permissions application s has received.
And the set

$$granted(b, s) := \{g|(r, g) \in b(s)\}$$

denotes the set of permissions application s has granted. $(s, o) \in received(b, s')$ indicates that application s' has received the permission to access display area o from application s. $(s, o) \in granted(b, s')$ indicates that application s' has granted the permission to access display area o to application s.
The function

$$\Lambda(b, s) := \bigcup\{o \in O|\exists s' \in S : (s', o) \in received(b, s)\}$$

returns a union of sets of received display areas of an application in b.
The function

$$\Gamma(b, s) := \bigcup\{o \in O|\exists s' \in S : (s', o) \in granted(b, s)\}$$

returns a union of sets of granted display areas of an application in b.

Permissions for display areas can be dynamically granted and revoked by applications that have received a permission. If an application has granted a permission, it can no longer set the graphical content for the display area contained in that permission. However, it can at any time revoke the granted permission.

Definition 4. We distinguish between received permissions for display areas and actually *used* display areas. A display area o is used by an application if it is setting the graphical content of o. $used : B \times S \to \mathcal{P}(O)$ is a function which returns the set of display areas used by an application.
Let $o \in O, s \in S, b \in B$. We define $o \in used(b, s) \Leftrightarrow$

$$\exists(\hat{s}, \hat{o}) \in S \times O : (\hat{s}, \hat{o}) \in received(b, s) \wedge o \subseteq \hat{o} \wedge \tag{4.1}$$

$$\forall(s', o') \in S \times O : (s', o') \in granted(b, s) \Rightarrow o \cap o' = \{\} \tag{4.2}$$

$\Omega_{used} : B \to \mathcal{P}(O)$ is a function which returns a set of all used objects according to b. Let $b \in B$. We define:

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

In Fig. 4 we depicted an example of the sets of received, granted, and used display areas. A display area is in the set of used display areas of an application if the following two conditions hold. Condition (4.1) states that the display area needs to be in the set of received display areas and Condition (4.2) states that it must not be granted to other applications. $\Omega_{used}(b)$ is the union set of all pixels in the sets of used display areas in b. The set $\Phi(b,s) := \bigcup used(b,s)$ denotes the union set of all pixels in the sets of used display areas of an application s in b.

Definition 5. We define the transitive operator $<_o$ denoting whether an application has granted a given display area to another application.
Let $s, s' \in S; o \in O;\ b \in B$. We define $s <_o s' \Leftrightarrow$

$$\exists s_1, ..., s_n \in S; \exists o_1, ..., o_{n-1} \in O : \tag{5.1}$$
$$s_1 = s' \wedge s_n = s \wedge o \subseteq o_{n-1} \wedge \tag{5.2}$$
$$\forall i : 1 \leq i < n : (s_i, o_i) \in received(b, s_{i+1}) \wedge \tag{5.3}$$
$$\forall i : 1 \leq i < n-2 : o_{i+1} \subseteq o_i \tag{5.4}$$

Let $s, s' \in S$. We define $s \neq_o s' \Leftrightarrow \nexists o \in O : s <_o s' \vee s' <_o s$. That is, if $s <_o s'$ then s has received o either directly from s' or by using a chain of intermediate applications, i.e., s depends on s' according to o.

Definition 6. We map to each application the set of applications it wants to delegate permissions to. This mapping is performed by a function $dr \in DR = \{f : S \to \mathcal{P}(S)\}$. Applications will only grant permissions to or receive permissions from applications if they are in a delegation relation. Application s and $\tilde{s}$ are in a delegation relation if and only if $s \in dr(\tilde{s})$ and $\tilde{s} \in dr(s)$.

The delegation relations correspond to the development hierarchy, cf. Sec. 2.1. Hence, the delegation relationship between applications restricts the propagation of permissions.

Delegation relations can be represented as a directed graph (*delegation graph*, Fig. 5) in which each application can have multiple edges to other applications. To support dynamic deployment of applications during runtime, applications are allowed to dynamically declare which applications they want to be in a delegation relation with.

Definition 7. A *state* consists of permissions and delegation relations between applications. Formally, $v \in V$ is a state in $V = B \times DR$.

3.3 Transitions

To support dynamic state changes our model supports transitions from one state to another state using *requests*.

Definition 8. A transition uses a request to add or delete permissions or delegation relations. The operation mode is determined by $RA = \{append, discard\}$. The request to alter permissions consists of operation mode, grantor, grantee

and the display area; formally, $RO = RA \times S \times S \times O$. The request to alter delegation relations consists of operation mode and the two applications of which the first wants to establish a delegation relation with the second; formally, $RD = RA \times S \times S$. The set of all possible requests is $R = RO \cup RD$.

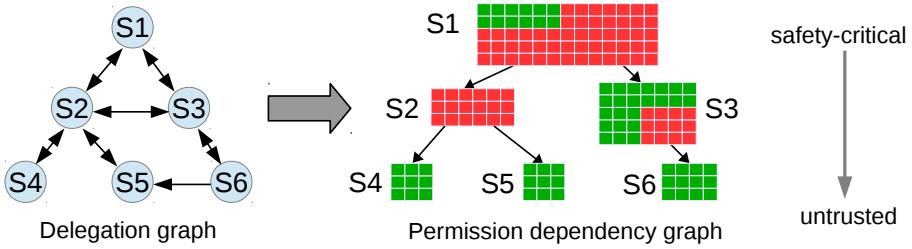

Fig. 5. Example: Delegation graph and permission dependency graph

Definition 9. $trans : V \times R \to V$ is a function which represents the transition from one state to another state initiated by a request.

In the following, we define Rule 1 to describe how permissions can be changed, and Rule 2 to describe how delegation relations can be changed using $trans$.

Rule 1. If application s' wants to have a permission for display area o from application s, this is expressed by the request $r \in RO$ with $r = (append, s, s', o)$. If application s decides to grant the permissions, $trans(v, r)$ is executed. In detail, $trans$ calls $add_{so}(b, s, s', o)$, which adds o to the set of granted permissions of s and adds it to the set of received permissions of s'.

Formally, function $add_{so} : B \times S \times S \times O \to B$ is defined as follows. Let $b, b' \in B; s, s' \in S; o \in O$. We define
$b' = add_{so}(b, s, s', o) \Leftrightarrow$

$$b'(s) = (received(b, s), granted(b, s) \cup \{(s', o)\}) \wedge \quad (9.1.1)$$

$$b'(s') = (received(b, s') \cup \{(s, o)\}, granted(b, s')) \wedge \quad (9.1.2)$$

$$[\forall s_3 \in S \backslash \{s', s\} : b'(s_3) = b(s_3)] \quad (9.1.3)$$

An application s can revoke a permission for display area o from another application s'. This is expressed by the request $r \in RO$ with $r = (discard, s, s', o)$. In this case, $trans$ calls $del_{so}(b, s, s', o)$, which removes o from the set of granted permissions of s and removes it from the set of received permissions of s'. If s' has granted permissions that contain part of o, function del_{so} will recursively revoke all these permissions. Formally, function $del_{so} : B \times S \times S \times O \to B$ is defined

as follows. Let $b, b' \in B; s, s' \in S; o \in O$. We define $b' = del_{so}(b, s, s', o) \Leftrightarrow$

$$[\forall \hat{s} \in S : \hat{s} <_o s \Rightarrow \tag{9.1.4}$$

$$received(b', \hat{s}) = received(b, \hat{s}) \backslash \{(\tilde{s}, o') \in S \times O | o' \subseteq o \wedge \hat{s} <_{o'} \tilde{s}\} \wedge \tag{9.1.5}$$

$$granted(b', \hat{s}) = granted(b, \hat{s}) \backslash \{(\tilde{s}, o') \in S \times O | o' \subseteq o \wedge \tilde{s} <_{o'} \hat{s}\}] \wedge \tag{9.1.6}$$

$$b'(s) = (received(b, s), granted(b, s) \backslash \{(s', o)\}) \wedge \tag{9.1.7}$$

$$\forall s'' \in S \backslash \{s', s\}; \forall o' \in O : o' \subseteq o \wedge s'' \neq_{o'} s \vee s <_o s'' \Rightarrow b'(s'') = b(s'') \tag{9.1.8}$$

In $(9.1.5 - 6)$ all permissions with display areas that are part of display area o are removed from the sets of granted and received permission if the applications depend on application s (9.1.4). In (9.1.7) the permission which application s granted to s' is removed from the set of granted permissions of s. The sets of permission of all applications that do not depend on s will not be changed (9.1.8).

Rule 2. The permission dependency graph based on granted permissions is a sub graph of the delegation graph and, therefore, restricts granting of permissions only between applications which are in a delegation relation. For instance, in Fig. 5 application S_2 is in a delegation relation with application S_3 but no permissions were granted between them. All applications which granted a permission to other applications are in a delegation relation as depicted in the delegation graph.

The set of delegation relations DR can also be changed by using transitions. If application s wants to be in a delegation relation with application s', this is expressed by the request $r \in RD$ with $r = (append, s, s')$. Then $trans$ calls $add_{dr}(dr, s, s')$, which adds the new relation $s \rightarrow s'$ to dr.

Formally, Function $add_{dr} : DR \times S \times S \rightarrow DR$ is defined as follows.
Let $dr, dr' \in DR; s, s' \in S$. We define $dr' = add_{dr}(dr, s, s') \Leftrightarrow$

$$dr'(s) = dr(s) \cup \{s'\} \wedge \forall \hat{s} \in S : \hat{s} \neq s \Rightarrow dr'(\hat{s}) = dr(\hat{s}) \tag{9.2.1}$$

Similarly, if application s no longer wants to be in a delegation relation with application s', this is expressed by the request $r \in RD$ with $r = (append, s, s')$. In this case, $trans$ calls $del_{dr}(dr, s, s')$, which removes the delegation relation $s \rightarrow s'$ from dr. Formally, function $del_{dr} : DR \times S \times S \rightarrow DR$ is defined as follows.
Let $dr, dr' \in DR; s, s' \in S$. We define $dr' = del_{dr}(dr, s, s') \Leftrightarrow$

$$dr'(s) = dr(s) \backslash \{s'\} \wedge \forall \hat{s} \in S : \hat{s} \neq s \Rightarrow dr'(\hat{s}) = dr(\hat{s}) \tag{9.2.2}$$

Next, we formally define $trans$ using the two rules. Let $v = (b, dr) \in V$; and $r \in R$ with $r = (ra, s, s', o) \in RO$ or $r = (ra, s, s') \in RD$. We define

$$trans(v, r) = \begin{cases} (add_{so}(b, s, s', o), dr) & \text{if } cond_1 \quad \text{(R1.1)} \\ (del_{so}(b, s, s', o), dr) & \text{if } cond_2 \quad \text{(R1.2)} \\ (b, add_{dr}(dr, s, s')) & \text{if } cond_3 \quad \text{(R2.1)} \\ (b, del_{dr}(dr, s, s')) & \text{if } cond_4 \quad \text{(R2.2)} \\ v & \text{otherwise} \end{cases}$$

with

$cond_1 = r \in RO \wedge ra = append \wedge s \neq s' \wedge$
$s' \in dr(s) \wedge s \in dr(s') \wedge \exists \hat{o} \in O : \hat{o} \in used(b, s) \wedge o \subseteq \hat{o} \wedge$
$\nexists(\tilde{s}, \tilde{o}) \in granted(b, s') : o \subseteq \tilde{o}$

$cond_2 = r \in RO \wedge ra = discard \wedge s \neq s' \wedge$
$(s', o) \in granted(b, s) \wedge (s, o) \in received(b, s')$

$cond_3 = r \in RD \wedge ra = append \wedge s \neq s'$

$cond_4 = r \in RD \wedge ra = discard \wedge s \neq s' \wedge \forall o \in O : s \neq_o s'$

Condition $cond_1$ ensures that the application s and s' are in a delegation relation. Furthermore, $cond_1$ prevents an application s' from receiving a permission for a display area which is part of a display area granted by s', thus, preventing cyclic grants. Additionally, application s can only grant a display area o to s' if o is in its set of used objects. Condition $cond_2$ ensures that the revoking of a display area o only takes place if it previously was granted by application s to s'.
The conditions $cond_3$ and $cond_4$ reject self-referencing delegation relations, and $cond_4$ additionally ensures that if application s no longer wants to be in a delegation relation with s', it must revoke or return all permissions granted between s and s' beforehand. If none of the conditions of Rule 1 and 2 is fulfilled, then the state v does not change.

3.4 System Consistency

We define three *properties* that define the consistency of our *system* and correspond directly to three requirements introduced in Sec 2.1. A system consists of all possible *sequences* of requests and the sequence of all states starting from initial state v_{start}. States satisfying all properties are *safe states* and a sequence of safe states is called *safe state sequence*. If all possible state sequences of a system are safe state sequences, the system is called *safe system*.

Exclusive Access Property (EAP): In a state that satisfies EAP, each display area is used by at most one application. Let $v = (b, dr) \in V$.
v satisfies EAP $\Leftrightarrow \forall s, s' \in S : s \neq s' \Rightarrow \Phi(b, s) \cap \Phi(b, s') = \emptyset$.

The sets of used display areas of all applications are intersection-free. Therefore, the setting of content of a display area is restricted in a way that no concurrent writing or unintended overlapping is possible and Req. 3.1 is fulfilled.

Completeness Property (CP): In a state that satisfies CP, each pixel is used by at least one application. Let $v = (b, dr) \in V$.
v satisfies CP $\Leftrightarrow \Omega_{used}(b) = AO$.

All pixels need to be in $\Omega_{used}(b)$ which represents all pixels of all sets of used display areas. This means, that the complete display surface is covered by dis-

play areas and assigned to at least one application and Req. 3.2 is fulfilled.

Delegation Property (DP): In a state that satisfies DP, permissions are only granted between applications with according delegation relations. Let $v = (b, dr) \in V$.
v satisfies the DP $\Leftrightarrow$

$$\forall s, s' \in S, \forall o \in O : s \neq s' \wedge s <_o s' \Rightarrow \quad \text{(DP.1)}$$

$$\exists s_0, ..., s_{n+1} \in S : s_0 = s \wedge s_{n+1} = s' \wedge \quad \text{(DP.2)}$$

$$\forall i \in \{0, ..., n\} \subset \mathbb{N}_0 : s_i <_o s_{i+1} \wedge s_i \in dr(s_{i+1}) \wedge s_{i+1} \in dr(s_i) \quad \text{(DP.3)}$$

Each application that received a permission is in a delegation relation with the permission grantor (Req. 4). (DP.1) implies that for all applications which depend on another application according to a display area a chain of applications exists (DP.2) for which every application is in a delegation relation with its predecessor and successor (DP.3).

Next, we give the formal definitions for the sequence of states and the system. We first define $I^n \subset \mathbb{N}_0$ as a finite set with $I^n = \{0, 1, 2, 3, ..., n\}$.

Definition 10. Operator $\succ$ indicates whether an element is part of a sequence of states. The set of sequences of states is a set of n-tuples and defined as $X^{I^n} = \{(x_0, ..., x_i, ..., x_n) | x_i \in X \wedge i \in I^n \wedge x_i = f(i) \text{ with } f : I^n \rightarrow X\}$.
$(x_0, x_1, ..., x_n) \in X^{I^n}$ is a sequence with $x_0 := x_0 \in X$, $x_1 := x_1' \in X$,..., $x_n := x_n^{(n)} \in X$. Let $(x_0, x_1, ..., x_n) \in X^{I^n}$.
We define $x \succ (x_0, x_1, ..., x_n) \Leftrightarrow \exists i \in I^n : x = x_i$.

Based on Def. 10, we next define the sequences of requests, the sequences of states and a distinct mapping between these sequences by using transitions.

Definition 11. 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 given as $(v_0, v_1, ..., v_n) \in V^{I^n}$ with $\forall i \in I^{n-1} : v_{i+1} = trans(v_i, r_i)$.

Recall that a system is a sequence of requests and states initiated by v_{start}. The operator $\gg$ indicates whether a sequence of requests and states is part of it.

Definition 12. A system generated by $v_{start} \in V$ is stated as $\Psi(v_{start}) \subset R^{I^n} \times V^{I^n}$. Let $x_r = (r_0, ..., r_{n-1}) \in R^{I^{n-1}}$, $x_v = (v_0, ..., v_n) \in V^{I^n}$.
We definee $(x_r, x_v) \in \Psi(v_{start}) \Leftrightarrow v_0 = v_{start} \wedge \forall i \in I^n \backslash \{0\} : v_i = trans(v_{i-1}, r_{i-1})$.
Let $(v, r, v') \in V \times R \times V$, $v_0 \in V$ we define $(v, r, v') \gg \Psi(v_0) \Leftrightarrow$

$$\exists x_r \in R^{I^{n-1}}, \exists x_v \in V^{I^n}, \exists i \in I^n \backslash \{0\} : \quad (12.1)$$

$$(x_r, x_v) \in \Psi(v_0) \wedge v_i \succ x_v \wedge v_{i+1} \succ x_v \wedge r_i \succ x_r \wedge \quad (12.2)$$

$$(v, r, v') = (v_{i-1}, r_{i-1}, v_i). \quad (12.3)$$

We define (v, r, v') is part of a system if a sequence of requests and states (12.1) exists of which v, r and v' are part of (12.2) and furthermore a valid state transition from state v to state v' by request r (12.3) exists.

In the following we use the properties EAP, CP and DP to define a safe state, a safe state sequence and a safe system which consists only of safe state sequences.

Definition 13. $v \in V$ is a safe state $\Leftrightarrow$ v satisfies EAP and CP and DP.
$(v_0, ..., v_n) \in V^{I^n}$ is a safe state sequence $\Leftrightarrow \forall i \in I^n : v_i$ is a safe state.
A system $\Psi(v_{start}) \subset V^{I^n} \times R^{I^n}$ with $x_r \in R^{I^{n-1}}$ and $x_v = (v_0, ..., v_n) \in V^{I^n}$ is a safe system $\Leftrightarrow \forall (x_r, x_v) \in \Psi(v_0) : v_{start} = v_0 \wedge x_v$ is a safe sequence.

4 System Verification

In this section, we verify our access control model against the requirements in Sec. 2.1. Req. 1 and Req. 2 are direclty given by the model. A path in the permission graph represents presentation priorities between the applications for dedicated display areas. The properties defined in Sec. 3.4 correspond to Req. 3.1, Req. 3.2 and Req. 4. While Req. 1 and Req. 2 are given by the way the model is used, fulfilling Req. 3 and 4 are properties of which we have to proof that our system fulfills them.

Next, we define three propositions that correspond to the properties and help us to proof the safety of our model. Let $v, v', v_0 \in V$; $v' = (b', dr')$; $v = (b, dr)$ and $r \in R$.

Proposition 1: All sequences in $\Psi(v_0)$ satisfy EAP for any v_0 which satisfies EAP $\Leftrightarrow \forall (v, r, v') \in V \times R \times V : (v, r, v') \gg \Psi(v_0) \Rightarrow v, v'$ satisfies EAP

Proposition 2: All sequences in $\Psi(v_0)$ satisfy CP for any v_0 which satisfies CP $\Leftrightarrow \forall (v, r, v') \in V \times R \times V : (v, r, v') \gg \Psi(v_0) \Rightarrow v, v'$ satisfies CP

Proposition 3: All sequences in $\Psi(v_0)$ satisfy DP for any v_0 which satisfies DP $\Leftrightarrow \forall (v, r, v') \in V \times R \times V : (v, r, v') \gg \Psi(v_0) \Rightarrow v, v'$ satisfies DP

To proof the correctness of the propositions EAP, CP and DP, we define a lemma for each of them. Furthermore, we define Lemma 1 and 2 for the similar proofs of Lemma EAP and CP. Finally, we use complete induction over the system states to proof the propositions. Due to space restrictions we only present the proof of Lemma EAP. The proofs for Lemma 1, 2, CP and DP can be found in [14].

We first define Lemma 1, which states that after a transition using rule (R1.1) with $add_{so}(b, s, s', o)$ the display area o has moved from the sets of used display areas of application s to application s'.
Lemma 1: Let $v_0 \in V$, $\Psi(v_0)$ be a system and $(v, r, v') \in V \times R \times V$ with $(v, r, v') \gg \Psi(v_0)$. Let $v = (b, dr)$ satisfy EAP and $ra = append$ and $cond_1 \Rightarrow trans(v, r) = (b', dr) \in V$ with $b' = add_{so}(b, s, s', o)$:

$$\Phi(b', s) = \Phi(b, s) \backslash o \tag{L1.1}$$

$$\Phi(b', s') = \Phi(b, s') \cup \{o\} \tag{L1.2}$$

Proof for Lemma 1
We first show (L1.1): Let $\hat{o} \in used(b,s)$ be the display area in $(cond_1)$ with $o \subseteq \hat{o}$. We have to prove:

$$o \not\subseteq \Phi(b',s) \quad \text{(i)}$$
$$(\hat{o} \backslash o) \subseteq \Phi(b',s) \quad \text{(ii)}$$

Due to $\hat{o} \cap o = o$, for (i) we only have to show that $\hat{o} \not\subseteq \Phi(b',s)$ is true.

(i): Due to (9.1.1), we know $(s',o) \in granted(b',s)$ after a transition to state v'. Since $o \subseteq \hat{o}$ is true, it is obviously that $\hat{o} \cap o = o \neq \emptyset$ follows. Hence the condition in (4.2) is no longer valid for b' which proves (i). This means, all subsets of $\hat{o}$ are not in the set of used display areas of application s in b'.

(ii): To prove statement (ii) we have to show that display area $(\hat{o} \backslash o)$ fulfills the condition of Def. 4. Therefore we prove the following two conditions:

$$(\hat{o} \backslash o) \subseteq \Lambda(b',s) \quad \text{(a)}$$
$$(\hat{o} \backslash o) \cap \Gamma(b',s) = \emptyset \quad \text{(b)}$$

(a): Due to $\hat{o} \subseteq \Phi(b,s)$, we know $\hat{o} \subseteq \Lambda(b,s)$. After a transition with add_{so} to v' we conclude with (9.1.1) $\Lambda(b,s) = \Lambda(b',s)$. Hence, $(\hat{o} \backslash o) \subseteq \hat{o} \subseteq \Lambda(b,s) = \Lambda(b',s)$ and therefore (a) is true.
(b): We know that $\hat{o} \in used(b,s)$ is true. Since we assume EAP is satisfied in $v = (b, dr)$ we conclude $\hat{o} \cap \Gamma(b,s) = \emptyset$. With (9.1.1) we conclude $\Gamma(b',s) = \Gamma(b,s) \cup o$ and therefore (b) is true in state v'.

$$\begin{aligned}
(\hat{o} \backslash o) \cap \Gamma(b',s) &= (\hat{o} \backslash o) \cap (\Gamma(b,s) \cup o) && \text{(9.1.1)} \\
&= ((\hat{o} \backslash o) \cap (\Gamma(b,s))) \cup ((\hat{o} \backslash o) \cap o) && \text{(Distr. law)} \\
&= \emptyset \cup ((\hat{o} \backslash o) \cap o) && (\hat{o} \cap \Gamma(b,s) = \emptyset) \\
&= \emptyset
\end{aligned}$$

Therefore, we conclude statement (ii) $(\hat{o} \backslash o) \subseteq \Phi(b',s)$.
We show (L1.2): We need to prove that the display area o fulfills the conditions of Def. 4 for b'. Therefore we show in a similar approach like in the proof of L1.1 that the following two conditions are satisfied:

$$o \subseteq \Lambda(b',s') \quad \text{(i)}$$
$$o \cap \Gamma(b',s') = \emptyset \quad \text{(ii)}$$

(i): We directly follow $received(b',s') = received(b,s') \cup \{o\}$ due to (9.1.2).

(ii): We know that $\Gamma(b,s) \cap o = \emptyset$ is valid in b. We conclude $\Gamma(b,s') \cap o = \emptyset$ from the following: We assume $\Gamma(b,s') \cap o \neq \emptyset$ and conclude $o \subseteq \Lambda(b,s')$ which leads to $\Gamma(b,s) \cap o \neq \emptyset$. But this is in contradiction to $\Gamma(b,s') \cap o = \emptyset$. Hence, $\Gamma(b,s') \cap o = \emptyset$ is valid. With (9.1.2) we know $\Gamma(b',s') = \Gamma(b,s')$ and we conclude $\Gamma(b',s') \cap o = \Gamma(b,s') = \emptyset \cap o$. The sets of granted and received display areas of all other applications are unmodified due to (9.1.3).

Hence, the conditions of Def. 4 and therefore (L1.2) are satisfied. □

The following Lemma 2 says that after a transition using rule (R1.2) with del_{so} each display area o' which is a subset of the display area o moves from the set of

used display areas of the according application to the set of used display areas of application s. Hence, a previously granted display area can be revoked to be able to set the content of this display area.

Lemma 2: Let $v_0 \in V$, $\Psi(v_0)$ be a system and $(v, r, v') \in V \times R \times V$ with $(v, r, v') \gg \Psi(v_0)$. Let $v = (b, dr)$ satisfies EAP and $ra = discard$ and $cond_2 \Rightarrow trans(v, r) = (b', dr) \in V$ with $b' = del_{so}(b, s, s', o)$:

$$\forall \hat{s} \in S \backslash \{s\} : used(b', \hat{s}) = used(b, \hat{s}) \backslash \{o' \in O | o' \subseteq o \wedge \hat{s} <_{o'} s\} \tag{L2.1}$$

$$\Phi(b', s) = \Phi(b, s) \cup o \tag{L2.2}$$

Proof for Lemma 2

We first prove (L2.1): Let $s \neq s'$ and state v satisfies EAP. In function del_{so} (9.1.4 - 8) all display areas o' which are a subset of o are removed from the sets of received display areas of all applications depending on s according to o'. We follow $\forall \hat{s} \in S \backslash \{s\} : \Lambda(b', \hat{s}) = \Lambda(b, \hat{s}) \backslash \bigcup \{o' \in O | o' \subseteq o \wedge \hat{s} <_{o'} s\}$. This means, (4.1) is violated and the display area o' is no longer in the set of used display areas of the according applications. Hence, we can directly conclude statement (L2.1).

We proof (L2.2): Due to (R1.2), we know $(s', o) \in granted(b, s)$ which leads to $o \subseteq \Lambda(b, s)$. Hence, (4.1) is satisfied. With (9.1.7) we conclude $\Gamma(b', s) = \Gamma(b, s) \backslash o$. Next, we show $o \cap \Gamma(b', s) = \emptyset$, which means (4.2) is satisfied:

$$\begin{aligned} o \cap \Gamma(b', s) &= o \cap (\Gamma(b, s) \backslash o) && (7.4) \\ &= (o \cap \Gamma(b, s)) \backslash (o \cap o) && \text{(Distr. law)} \\ &= (o \cap \Gamma(b, s)) \backslash o \\ &= \emptyset && (\text{since } o \cap \Gamma(b, s) \subseteq o) \end{aligned}$$

We conclude statement (L2.2). □

Lemma 1 and 2 say that a transition applying an add_{so} or a del_{so} operation do not modify the united set of used display areas of all applications.

Next, we define the Lemma EAP which states that a transition from a state which satisfies EAP will always end in a state which also satisfies EAP.

Lemma EAP: Let $v_0 \in V$, $\Psi(v_0)$ be a system and $(v, r, v') \in V \times R \times V$ with $(v, r, v') \gg \Psi(v_0)$. $v = (b, dr)$ satisfies EAP $\Rightarrow v' = (b', dr')$ satisfies EAP.

Proof for Lemma EAP

According to Def. 9, the request r is in RO or in RD. The case $r \in RD$ is trivial, since $b' = b$ due to $v = v'$ (Def. 9 Rule 2). Hence, changes in dr are not relevant in Lemma EAP. In case $r = (ra, s, s', o) \in RO$, we have to consider the following three subcases:

(R1.1): Let $ra = append$ and $cond_1$ be fulfilled. It follows $trans(v, r) = (b', dr) \in V$ with $b' = add_{so}(b, s, s', o)$. The set of permissions and used display areas of all applications beside s and s' do not change in v' (9.1.3). Hence, $\forall \hat{s} \in S \backslash \{s, s'\} : \big(used(b', \hat{s}) = used(b, \hat{s})\big)$ due to $b'(\hat{s}) = b(\hat{s})$. This means we only have to prove EAP in v' for s and s'. Therefore we show the following

statements:

$$\Phi(b', s) \cap \Phi(b', s') = \emptyset \quad \text{(i)}$$
$$\forall \hat{s} \in S \backslash \{s, s'\} : \Phi(b', s) \cap \Phi(b', \hat{s}) = \emptyset \quad \text{(ii)}$$
$$\forall \hat{s} \in S \backslash \{s, s'\} : \Phi(b', s') \cap \Phi(b', \hat{s}) = \emptyset \quad \text{(iii)}$$

We prove the statements (i), (ii) and (iii) by using Lemma 1 and 2.

(i): $\Phi(b', s) \cap \Phi(b', s')$

$$\begin{aligned}
&= (\Phi(b, s) \backslash o) \cap \Phi(b', s') && \text{(L1.1)}\\
&= (\Phi(b, s) \backslash o) \cap (\Phi(b, s') \cup o) && \text{(L1.2)}\\
&= ((\Phi(b, s) \backslash o) \cap \Phi(b, s')) \cup ((\Phi(b, s) \cap o) \backslash (o \cap o)) && \text{(Distr. law)}\\
&= ((\Phi(b, s) \backslash o) \cap \Phi(b, s')) \cup (o \backslash o) && \text{(R1.1)}\\
&= ((\Phi(b, s) \backslash o) \cap \Phi(b, s'))\\
&= (\Phi(b, s) \cap \Phi(b, s')) \backslash (o \cap \Phi(b, s')) && \text{(Distr. law)}\\
&= \emptyset \backslash (o \cap \Phi(b, s')) && \text{(EAP)}\\
&= \emptyset
\end{aligned}$$

(ii): Let $s'' \in S \backslash \{s, s'\}$ be arbitrary. Then the following is valid:

$$\begin{aligned}
\Phi(b', s) \cap \Phi(b', s'') &= (\Phi(b, s) \backslash o) \cap \Phi(b', s'') && \text{(L1.1)}\\
&= (\Phi(b, s) \backslash o) \cap \Phi(b, s'') && \text{(9.1.3)}\\
&= (\Phi(b, s) \backslash \Phi(b, s'')) \backslash (o \cap \Phi(b, s'')) && \text{(Distr. law)}\\
&= \emptyset \backslash (o \cap \Phi(b, s'')) && \text{(EAP)}\\
&= \emptyset
\end{aligned}$$

(iii): Let $s'' \in S \backslash \{s, s'\}$ be arbitrary. Then the following is valid:

$$\begin{aligned}
\Phi(b', s') \cap \Phi(b', s'') &= (\Phi(b, s') \backslash o) \cap \Phi(b', s'') && \text{(L1.2)}\\
&= (\Phi(b, s') \backslash o) \cap \Phi(b, s'') && \text{(9.1.3)}\\
&= (\Phi(b, s') \backslash \Phi(b, s'')) \cup (o \cap \Phi(b, s'')) && \text{(Distr. law)}\\
&= \emptyset \cup (o \cap \Phi(b, s'')) && \text{(EAP)}\\
&= \emptyset
\end{aligned}$$

(R1.2): Let $ra = discard$ and $cond_2$ be satisfied. Hence, $trans(v, r) = (b', dr) \in V$ with $b' = del_{so}(b, s', s, o)$. In this case, we have to consider those applications which are in relation according to a display area $o' \subseteq o$ (9.1.8). We know with (L2.1) that the following is satisfied: $\forall \hat{s} \in S \backslash \{s\} : \Phi(b', \hat{s}) \subseteq \Phi(b, \hat{s})$. Since state v is satisfying EAP we only have to prove EAP for the applications s and s' in state v'. The proof for this case is similar to the first case. This means, we have to prove the following:

$$\Phi(b', s) \cap \Phi(b', s') = \emptyset \quad \text{(i)}$$
$$\forall \hat{s} \in S \backslash \{s, s'\} : \Phi(b', s) \cap \Phi(b', \hat{s}) = \emptyset \quad \text{(ii)}$$
$$\forall \hat{s} \in S \backslash \{s, s'\} : \Phi(b', s') \cap \Phi(b', \hat{s}) = \emptyset \quad \text{(iii)}$$

The proofs for (i), (ii) and (iii) are similar to the case (R1.1).

(i): $\Phi(b', s') \cap \Phi(b', s)$

$= (\Phi(b, s')\backslash o) \cap \Phi(b', s)$ (L2.1)

$= (\Phi(b, s')\backslash o) \cap (\Phi(b, s) \cup o)$ (L2.2)

$= ((\Phi(b, s')\backslash o) \cap \Phi(b, s)) \cup ((\Phi(b, s') \cap o)\backslash(o \cap o))$ (Distr. law)

$= ((\Phi(b, s')\backslash o) \cap \Phi(b, s)) \cup ((\Phi(b, s') \cap o)\backslash o)$

$= ((\Phi(b, s')\backslash o) \cap \Phi(b, s))$ $((\Phi(b, s') \cap o) \subseteq o)$

$= (\Phi(b, s') \cap \Phi(b, s))\backslash(\Phi(b, s) \cap o)$ (Distr. law)

$= \emptyset\backslash(\Phi(b, s) \cap o)$ (EAP)

$= \emptyset$

(ii): Let $\hat{s} \in S\backslash\{s, s'\}$ be arbitrary then the following is valid:

$\Phi(b', s) \cap \Phi(b', \hat{s}) = (\Phi(b, s)\backslash o) \cap \Phi(b', \hat{s})$ (L2.2)

$= (\Phi(b, s)\backslash o) \cap \Phi(b, \hat{s})$ (9.1.8)

$= (\Phi(b, s)\backslash \Phi(b, \hat{s}))\backslash(o \cap \Phi(b, \hat{s}))$ (Distr. law)

$= \emptyset\backslash(o \cap \Phi(b, \hat{s}))$ (EAP)

$= \emptyset$

(iii): Let $\hat{s} \in S\backslash\{s, s'\}$ be arbitrary then the following is valid:

$\Phi(b', s') \cap \Phi(b', \hat{s}) = (\Phi(b, s')\backslash o) \cap \Phi(b', \hat{s})$ (L1.2)

$= (\Phi(b, s')\backslash o) \cap \Phi(b, \hat{s})$ (9.1.8)

$= (\Phi(b, s')\backslash \Phi(b, \hat{s})) \cup (o \cap \Phi(b, \hat{s}))$ (Distr. law)

$= \emptyset \cup (o \cap \Phi(b, \hat{s}))$ (EAP)

$= \emptyset$

(otherwise): Since v satisfies EAP and $v' = v$ then v' also satisfies EAP. □

sThe following Lemma CP states that a transition from a state which satisfies CP and EAP will always end in a state which also satisfies CP.

Lemma CP: Let $v_0 \in V$, $\Psi(v_0)$ be a system with $(v, r, v') \in V \times R \times V$ and $(v, r, v') \gg \Psi(v_0)$: $v = (b, dr)$ satisfies CP and EAP $\Rightarrow$ $v' = (b', dr')$ satisfies CP.

Proof for Lemma CP: Let $\Omega_{used}(b) = AO$. We show $\Omega_{used}(b') = AO$. The request r is either in RD or in RO (Def. 9). The case $r \in RD$ is trivial, since $b' = b$ due to $v = v'$ (Def. 9 Rule 2), and dr does not affect Lemma CP. Let $r = (ra, s, s', o) \in RO$.

With $\{o \in O | \exists s \in S : o \in used(b, s)\} \stackrel{!}{=} \{o \in O | \exists s \in S : o \in used(b', s)\}$ (*)

we conclude:

$$\begin{aligned} AO &= \Omega_{used}(b) \\ &= \bigcup\{o \in O | \exists s \in S : o \in used(b,s)\} \\ &= \bigcup\{o \in O | \exists s \in S : o \in used(b',s)\} \\ &= \Omega_{used}(b') \end{aligned}$$

Hence, we only need to prove the proposition (*). Therefore, we show that the union of the sets of used display areas does not change if a transition is applied. Three cases have to be considered.

(R1.1): Let $ra = append$ and $cond_1$ be fulfilled. It follows $trans(v,r) = (b', dr)$ with $b' = add_{so}(b, s, s', o)$. Therefore, the following is valid: $\forall \hat{s} \in S \backslash \{s, s'\} : \Phi(b, \hat{s}) = \Phi(b', \hat{s})$. The sets of used display areas are trivially equal. Hence, we need to prove the following: $\Phi(b,s) \cup \Phi(b,s') = \Phi(b',s) \cup \Phi(b',s')$.

$$\begin{aligned} \Phi(b',s) \cup \Phi(b',s') &= (\Phi(b,s) \backslash o) \cup \Phi(b',s') && \text{(L1.1)} \\ &= (\Phi(b,s) \backslash o) \cup (\Phi(b,s') \cup o) && \text{(L2.1)} \\ &= ((\Phi(b,s) \backslash o) \cup (\Phi(b,s')) \cup ((\Phi(b,s) \backslash o) \cup o) && \text{(Distr. law)} \\ &= ((\Phi(b,s) \backslash o) \cup (\Phi(b,s')) \cup \Phi(b,s) \\ &= \Phi(b,s) \cup ((\Phi(b,s) \backslash o) \cup \Phi(b,s')) && \text{(Com. law)} \\ &= (\Phi(b,s) \cup (\Phi(b,s) \backslash o) \cup \Phi(b,s')) && \text{(Ass. law)} \\ &= \Phi(b,s) \cup \Phi(b,s') \end{aligned}$$

Hence, the union of the sets of used display areas is in b and b' equal.

(R1.2): Let $ra = discard$ and $cond_2$ be fulfilled.

$$\forall \hat{s} \in S : \Phi(b,s) \cup \Phi(b,\hat{s}) = \Phi(b',s) \cup \Phi(b',\hat{s}) \qquad (**)$$

Let $s'' \in \hat{S}$ be arbitrary and $o'' := \cup\{o' \in O | o' \subseteq o \wedge s'' <_{o'} s\}$.

We conclude: $\Phi(b', s'') \cup \Phi(b', s)$

$$\begin{aligned} &= (\Phi(b,s'') \backslash o'') \cup \Phi(b',s) && \text{(L2.1)} \\ &= (\Phi(b,s'') \backslash o'') \cup (\Phi(b,s) \cup o) && \text{(L2.2)} \\ &= ((\Phi(b,s'') \backslash o'') \cup \Phi(b,s)) \cup ((\Phi(b,s'') \backslash o'') \cup o) && \text{(Distr. law)} \\ &= ((\Phi(b,s'') \backslash o'') \cup \Phi(b,s)) \cup ((\Phi(b,s'') \cup o) \backslash (o'' \cup o)) && \text{(Distr. law)} \\ &= ((\Phi(b,s'') \backslash o'') \cup \Phi(b,s)) \cup \Phi(b,s'') && o'' \subseteq o \\ &= \Phi(b,s'') \cup ((\Phi(b,s'') \backslash o'') \cup \Phi(b,s')) && \text{(Com. law)} \\ &= (\Phi(b,s'') \cup (\Phi(b,s'') \backslash o'')) \cup \Phi(b,s) && \text{(Ass. law)} \\ &= \Phi(b,s'') \cup \Phi(b,s) \end{aligned}$$

(otherwise): Since v satisfies CP and $v' = v$ then v' also satisfies CP. □

The Lemma DP states that a transition from a state which satisfies DP will always end in a state which also satisfies DP.

Lemma DP: Let $\Psi(v_0)$ be a system and $v_0 \in V$ which satisfies DP. Let $(v,r,v') \in V \times R \times V$ with $(v,r,v') \gg \Psi(v_0)$: v satiesfies DP $\Rightarrow$ v' satiesfies

DP.

Proof for Lemma DP: We prove this lemma using a proof by contradiction. Let $s, s' \in S; o \in O$, $s \neq s'$. We assume state $v = (b, dr)$ satisfies DP but state $v' = (b', dr')$ does not satisfies DP.
Therefore, we consider the two possible cases in which DP can be violated.
(1) Permissions are granted between applications which are not in a delegation relation (R1.1)
(2) The delegation relation between applications was removed but they still have permissions granted between each other (R1.2).
Hence, we have to consider (R1.1) and (R1.2).

We first show (1): Let w.l.o.g. the application s be granting a permission for display area o. Since in state v' DP is not satified the following is valid: $\Gamma(b', s) \cap \Lambda(b', s') \neq \emptyset$ and $s \notin dr(s')$ or $s' \notin dr(s)$. Due to (R1.1) we know that the transition $trans(v, r) = v'$ with rule $b' = add_{so}(b, s, s', o)$ will not be applied since $cond_1$ is not satified. Hence, $b = b'$ is valid. This means, that the following have to be valid: $\emptyset \neq \Gamma(b', s) \cap \Lambda(b', s') = \Gamma(b, s) \cap \Lambda(b, s')$. But $s \notin dr(s')$ or $s' \notin dr(s)$ does not satisfy DP in state v which contradicts our assumption.

We show (2): Let $i, j \in I^n$ and $s, s', s_1, ...s_j, ...s_i \in S$ with $s <_o s_1 <_o ... <_o s_j <_o ... <_o s_{i+1} <_o s'$. Let w.l.o.g. application s_j remove the delegation relation to application s_{j+1} from its set. State v' does not satiesfies CP and the following is valid: $\Gamma(b, s) \cap \Lambda(b, s') \neq \emptyset$ and $s_{j+1} \notin dr'(s_j)$. Due to (R2.2) we know that the transition $trans(v, r) = v'$ with rule $dr' = del_{dr}(dr, s, s')$ will not be applied since $cond_4$ is not satified. But state v does not satiesfies DP due to $\Gamma(b, s) \cap \Lambda(b, s') \neq \emptyset$ and $s_{j+1} \notin dr'(s_j) = dr(s_j)$ which contradicts our assumption. □

Finally, we proof Proposition 1, 2 and 3 by complete induction.
Let $(x_r, x_v) \in \Psi(v_{start})$ with $x_r = (r_0, ..., r_{n-1}) \in R^{I^{n-1}}$, $x_v = (v_0, ..., v_n) \in V^{I^n}$. We define $v_0 = v_{start}$ as a initial state and generates the states in x_v by using our transition: $\forall i \in I^n \backslash \{0\} : v_i = trans(v_{i-1}, r_{i-1})$. The state v_0 satisfies CP, EAP and DP. Let $\forall i \in I^n \backslash \{0\} : (v_{i-1}, r_{i-1}, v_i) \gg \Psi(v_0)$ according to Def. 12.3.
Base: v_0 satisfies CP, EAP and DP. With $v_1 = trans(v_0, r_0)$ we conclude v_1 satisfies CP, EAP and DP according to Lemma EAP, CP and DP.
Induction hypothesis: v_i satisfies CP, EAP and DP.
Induction step: Let v_i satisfy CP, EAP and DP. From the Lemmas CP, EAP and DP follows with $v_{i+1} = trans(v_i, r_i)$ satisfies CP, EAP and DP. □

5 Implementation

We have created a proof-of-concept implementation which demonstrates the feasibility to implement our access control system. The system architecture of our Linux-based implementation is depicted in Fig. 6. We deployed our implementation in the cockpit demonstrator depicted in Fig. 7. The demonstrator

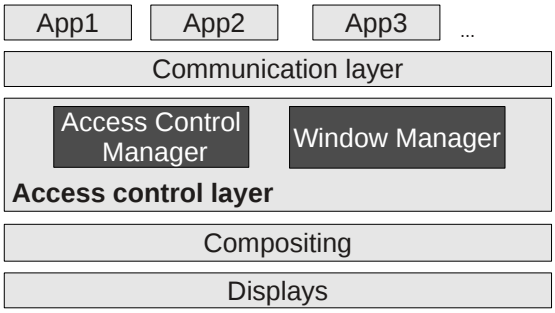

Fig. 6. Implemented architecture

uses two automotive 12" displays each with a resolution of 1440 × 540 pixels which are connected to an embedded i.MX6 platform from Freescale. We also connected the steering wheel buttons and the central control knob which are used to control the applications.

Next, we describe the components of our demonstrator. We use different typical IC and HU **applications**, e.g., speedometer, check engine indicator, phone, and navigation software. We implemented a client API for access control management and window management that can be used by the applications to interact with the **Access Control Manager (ACM)** and the **Window Manager (WM)**. The applications do not directly communicate with each other for permission exchange and do not know all applications currently displaying windows. Requests and responses for permission exchange are complete managed by the ACM as a mediator. Each application has an unique id and optionally an application class (e.g., an application class for indicators).

Fig. 7. Cockpit demonstrator

The **Communication Layer** provides session-based fifo communication between applications, ACM and WM. The ACM is the access control unit that per-

forms access decisions in the access control layer. Each application is connected to the ACM and can send requests which the ACM forwards to the application specified in the request. If the receiving application wants to grant a permission to the requesting application, it sends it to the ACM. The ACM checks the validity, updates its permission mapping tables and notifies the client. A permission is only valid if it can be derived from a root permission by a chain of grants. The root permission covers the whole display surface and is initialized by the ACM at startup of the system. If an applications want to grant a permission in response to a request the ACM first checks the validity of the permissions. This means, permissions will only be forwarded if the granting application has the according permissions. If the permissions are valid the ACM stores them locally and transmits the permissions to the intended application. Hence, the ACM has always a consistent view of all granted permissions and can ensure consistency by preventing invalid permission exchanges. Each display area is defined by the position and size. A permission depends on another permission if the rectangular area is completely included in the rectangular area of the other permission. Each application has a set of XML files which contain the application IDs or application class IDs of the applications they want to be in a delegation relation with.

The WM is responsible for creating, destroying and positioning of windows. Applications that want to create or move a window send a request to the WM. The WM checks by calling the ACM if the request matches existing permissions of the according application and initiates the creating or moving of the window by interacting then performs respective API calls to the compositing layer. Otherwise the WM rejects the request. Windows will be deleted on behalf of the according application but also if the necessary permission for the windows is revoked. Each time the WM applies changes to windows it updates the screen by initiating in the compositing layer the respective API call for the affected windows.

Our proof-of-concept implementation of the **compositing layer** uses X11 with Mesa 8.01 for compositing. The WM is the only process that can access X11. for creating, destroying, and mapping of windows. The X-server creates a window on WM request and sends back the windowID which the WM forwards to the applications. Applications send configuration messages (e.g., eglCreateWindowSurface for mapping a 3D surface to the window) to the WM instead of X11. Since the X-server lacks suitable security mechanisms the interception of the communication by the WM is necessary to prevent security issues like resizing of windows by any application [8].

Implemented Scenarios

Next, we describe two scenarios in our proof-of-concept implementation where an application uses—depending on the current state—either a display area on the IC display, the HU display, or no display area.

In the first scenario we describe the granting and revoking of permissions to display a media application. The required request-response calls are depicted in

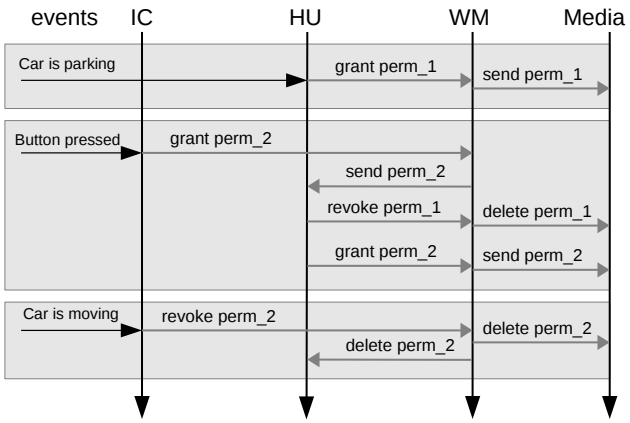

Fig. 8. Sequence of operations for scenario 1

Fig. 8. If the car reaches parking position, the HU application detects that state change and grants the HU display area (message "grant perm_1") to the (yet hidden) media application which plays a video after receiving the permission from the WM (message "send perm_1"). In order to playback videos on the IC display, we implemented a button at the IC which switches the assigned display area between IC and HU display. If the user presses this button, the IC grants a permissions for the IC display to the HU (message "grant perm_2"). The HU passes this permissions to the media application (message "grant perm_2") and revokes the previously granted permission (message "revoke perm_1"). The WM deletes the previously granted permission (message "delete perm_1"), sends the new permission (message "send perm_2") and updates the location of the media player window, which then will be visible on the IC display according to the permission. As soon as the car starts moving again, the IC will revoke the granted permission from the IC (message "revoke perm_2"), which automatically revokes the IC permission from the media player making the video disappear immediately (message "delete perm_2").

In the second scenario we describe the pro-active granting of a permission to display a warning. In particular, we assume the car is using adaptive cruise control system that uses radar to monitor the distance to the next object (e.g., another vehicle) in front of the car. Initially, the application *Trip Computer* requests access to the IC display (message "request access"). After receiving the request forwarded by the WM the IC grants a permission to the Trip Computer (message "grant perm_1"). Next the Trip Computer receivs the permission from the WM (message "send perm_1") and displays information about average fuel consumption or traveled distance. This state persists until the adaptive cruise control system detects that the distance to the vehicle in front is falling below the safety threshold. Thus, the event "Emergency brake" is immediately signaled to the IC and the application *Warning and Information Manager (WIM)* gets a

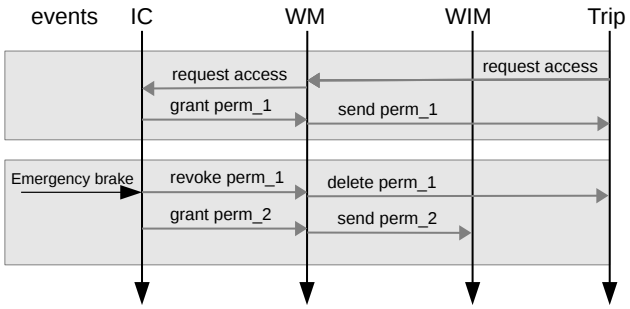

Fig. 9. Sequence of operations for scenario 2

permission pro-actively granted by the IC (message "grant perm_2"), i.e., without a request initiated by the WIM. Therefore, the IC automatically revokes the permission for the according display area (message "revoke perm_1") which currently is used by the application Trip Computer and therefore will be deleted by the WM (message "delete perm_1"). The WIM immediately displays the warning "Please brake" as soon as it receives the permission from the WM (message "send perm_2"). This event-driven assignment of permissions helps to reduce the latency for granting permissions.

6 Related Work

So far, there exists no fine-grained access control for displays or graphics resources. Feske et al. provide a concept for overlay management [10] of application windows executed in different virtual machines and a minimized secure graphical user interface called Nitpicker [11] with focus on low-level mechanisms to address security issues caused by spyware or Trojan horses. Hansen proposes a display system called Blink [15] which allows multiplexing of graphical content from different virtual machines onto a single GPU in a safe manner. However, both related works neglect any restriction in window management and consider the user as the supervisor for window placement. Similarly, Epstein et al. address security issues in the X-server [8] and propose mechanisms [7] to prevent them.

Protection of shared resources by using access control has been researched almost since the beginnings of operating systems [20]. Although later work (e.g., [22]) is based on hierarchical permissions, related work does not support priority-based access control using hierarchical granting and revoking of permissions. Birget et al. [5] unifies a user and a resource hierarchy based on access relations into a single one which simplifies access control management but this technique is only applicable when the system changes slowly.

Our model is a state-based system similar to the Bell and LaPadula model (BLP) [4]. BLP also defines a state machine for enforcing access control and uses an access control matrix for restricting access to data in order to provide confidentiality of information. However, BLP does neither prevent concurrent

access nor allow flexible granting of permission by subjects. Related work based on the BLP is provided by [21] which extended the BLP for access control in hierarchical organizations but only targets on the information flow and not on permission granting with exclusive access.

7 Summary and Future Work

In this paper, we presented an access control model which can be used for safety-critical automotive HMI systems. Our model supports hierarchical granting of display permissions and allows applications to be dynamically added and removed during runtime without modifying the access control layer. We proofed the correctness of our model and showed that it fulfills all the requirements we consider to be relevant for safe automotive HMI systems. Finally, we described our proof-of-concept implementation showing its feasibility in an automotive cockpit demonstrator. In future work we want to extend our access control model for context handling and constraints. Additionally, we want to improve our implementation by using native hardware accelerated window compositing without the X-server. Furthermore we want to evaluate the overhead and the timing paths of our implementation.

Acknowledgment This paper has been supported in part by the Automotive, Railway and Avionics Multicore Systems (ARAMiS) project under the research grant of the German Federal Ministry of Research and Education (BMBF), project number 01IS11035R.

Bibliography

[1] Mercedes-Benz integration of iPhone App in A-Class. `http://www.iphone-ticker.de/mercedes-benz-iphone-integration-a-klasse-30952/` (2013)
[2] New Version of QNX CAR Platform... `http://www.qnx.com/news/pr_5602_1.html` (Jun 2013)
[3] AAM: Statement of Principles, Criteria and Verification Procedures on Driver Interactions with Advanced In-Vehicle Information and Communication Systems. Alliance of Automotive Manufacturers (July 2006)
[4] Bell, D.E., Lapadula, L.J.: Secure Computer System: Unified Exposition and MULTICS Interpretation. Tech. Rep. ESD-TR-75-306, The MITRE Corporation (1976)
[5] Birget, J.C., Zou, X., Noubir, G., Ramamurthy, B.: Hierarchy-based access control in distributed environments. In: In IEEE International Conference on Communications (ICC01). vol. 1, pp. 229 –233
[6] Ebert, C., Jones, C.: Embedded software: Facts, figures, and future. Computer 42(4), 42–52 (April 2009)

[7] Epstein, J., McHugh, J., Pascale, R., Orman, H., Benson, G., Martin, C., Marmor-Squires, A., Danner, B., Branstad, M.: A prototype b3 trusted x window system. In: Proceedings of the 7th Annual Computer Security Applications Conference. pp. 44–55 (Dec 1991)
[8] Epstein, J., Picciotto, J.: Trusting X: Issues in building Trusted X window systems – or – what's not trusted about X. In: Proc. of the 14th National Computer Security Conference. vol. 1. National Institute of Standards and Technology, National Computer Security Center (Oct 1991)
[9] 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)
[10] Feske, N., Helmuth, C.: Overlay window management: User interaction with multiple security domains (2004)
[11] Feske, N., Helmuth, C.: A Nitpicker's guide to a minimal-complexity secure GUI. In: Proceedings of the 21st Computer Security Applications Conference. pp. 85–94 (Dec 2005)
[12] Ford: Software development kit (sdk). `https://developer.ford.com/develop/openxc/` (Feb 2013)
[13] Gansel, S., Schnitzer, S., Dürr, F., Rothermel, K., Maihöfer, C.: Towards Virtualization Concepts for Novel Automotive HMI Systems. In: Proceedings of IESS. IFIP LNCS, Springer Berlin Heidelberg (2013)
[14] Gansel, S., Schnitzer, S., Gilbeau-Hammoud, A., Friesen, V., Dürr, F., Rothermel, K., Maihöfer, C.: An Access Control Concept for Novel Automotive HMI Systems. Tech. Report 2013/02, University of Stuttgart, IPVS, Germany (July 2013), `http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2013-02`
[15] Hansen, J.G.: Blink: Advanced Display Multiplexing for Virtualized Applications. In: Proceedings of the 17th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV). pp. 15–20 (2007)
[16] ISO 26262: Road vehicles – Functional Safety. ISO, Geneva, Switzerland (Nov 2011)
[17] JAMA: Guideline for In-vehicle Display Systems – Version 3.0. Japan Automobile Manufacturers Association (Aug 2004)
[18] Janker, H.: Straßenverkehrsrecht: StVG, StVO, StVZO, Fahrzeug-ZulassungsVO, Fahrerlaubnis-VO, Verkehrszeichen, Bußgeldkatalog. C.H. Beck (2011)
[19] Krüger, I.H., Nelson, E.C., Prasad, K.V.: Service-Based Software Development for Automotive Applications. In: Proceedings of the CONVERGENCE 2004. Convergence Transportation Electronics Association (Jan 2004)
[20] Lampson, B.W.: Protection. SIGOPS Oper. Syst. Rev. 8(1), 18–24 (Jan 1974)
[21] Wang, J., Zhou, L., Tan, C.: A blp-based model for hierarchical orgnizations. In: Proceedings of the 2009 Second International Workshop on Computer Science and Engineering - Volume 01. pp. 456–459. IWCSE '09, IEEE Computer SocietyI, Washington, DC, USA (2009)

[22] Wilde, E., Nabholz, N.: Access control for shared resources. In: Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-1 (CIMCA-IAWTIC'06) - Volume 01. pp. 256–250. CIMCA '05, IEEE Computer Society, Washington, DC, USA (2005)