An Access Control Concept for Novel Automotive HMI Systems

Simon Gansel* Stephan Schnitzer† Ahmad Gilbeau-Hammoud† Viktor Friesen*
Frank Dürr† Kurt Rothermel† Christian Maihöfer*

*System Architecture and Platforms Department
Mercedes-Benz Cars Division, Daimler AG
Sindelfingen, Germany
Email: firstname.lastname <at> daimler.com

†Institute of Parallel and Distributed Systems
University of Stuttgart
Stuttgart, Germany
Email: lastname <at> ipvs.uni-stuttgart.de

ABSTRACT

The relevance of graphical functions in vehicular applications has increased significantly during the few last years. Modern cars are equipped with multiple displays used by different applications such as speedometer or navigation system. 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 applications as the dashboard 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 prove the validity of this model, and present a proof-of-concept implementation to demonstrate the feasibility of our concept.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access controls; H.5.2 [**User Interfaces**]: Windowing systems

Keywords

Access Control; State-based Model; Automotive; Windows

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 headrests 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. Additionally, head-up displays are used for displaying navigation instructions or assistance messages on the windshield.

Moreover, as demonstrated by advanced use cases already implemented in concept cars [1], 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 full-screen videos on the IC display while the car is not moving. Moreover, 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 [12, 3] and running in isolated execution environments—e.g., an Android partition within QNX [3] or on the mobile phone of the user [2]—can be granted access to these displays.

Although these use 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 function has to be assessed and suitable methods must be implemented to minimize the risks caused by malfunctions, c.f., ISO 26262 [15]. Moreover, automotive design guidelines (e.g., [9]) require that safety-critical content must be displayed in defined display areas and while driving potentially distracting content (like video playback) must not be displayed to the driver. Additionally, country-specific laws must be fulfilled, e.g., as regulated by German law (StVZO §57 [17]), in a moving car the speedometer must be visible.

Since display sharing results in scenarios where safety-critical applications, share the same display with uncritical applications, concepts for the safe sharing of displays between applications are required. Specifically, display areas have to be isolated such that it is guaranteed that the output of different applications does not interfere. For instance, the brake warning light may indicate a potentially severe hydraulic problem in the brake 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

SACMAT'14, June 25–27, 2014, London, Ontario, Canada.
Copyright 2014 ACM 978-1-4503-2939-2/14/06 ...$15.00.
http://dx.doi.org/10.1145/2613087.2613104.

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 is to provide isolation for window placement by defining 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 dynamic display sharing is required.

In this paper, we present a novel access control model for sharing graphical displays to offer flexibility without compromising safety. Basically, our model grants applications dynamic permissions to draw to certain display areas. To support the decentralized software development process involving OEMs and subcontractors and possibly third-party developers, permissions are managed based on a delegation hierarchy such that applications can pass permissions to subcomponents, e.g., third-party or subcontractor 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 for this model. 3. A proof-of-concept implementation to show the feasibility of this approach.

The rest of this paper is structured as follows. In Sec. 2 we present our system model and requirements. Sec. 3 defines our access control model and presents properties for the model with the proof in Sec. 4. We present our implementation in Sec. 5, and related work in Sec. 6. We conclude with a summary and an outlook on future work in Sec. 7.

2. SYSTEM MODEL & REQUIREMENTS

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.

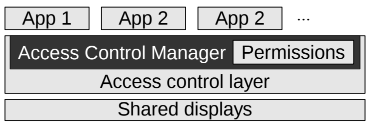


Figure 1: System model

Applications communicate with an 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). 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. Applications can be deployed or removed dynamically during runtime. Applications provided by the OEM or third-party developers are stored on a backend server infrastructure (cf. Fig. 2, [20]). To deploy or update an application in the car, the vehicle establishes a secure connection to the backend server to download binaries, a list of permissions, and certificates from the server.

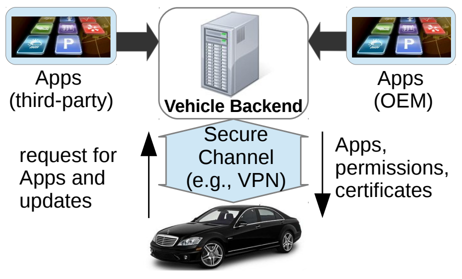

Apps (third-party)
Vehicle Backend
Apps (OEM)
request for Apps and updates
Secure Channel (e.g., VPN)
Apps, permissions, certificates

Figure 2: App deployment

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 Manager.* Before an application can access a display area, it needs a permission from the Access Control Manager that centrally manages all granted permissions.

2.2 Requirements

In the following, we present our 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 revocable 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. 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.

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: To facilitate the software development process, the OEM may pass usage permissions for display areas to software development companies or even individual developers, which again can 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. For instance, as depicted in Fig. 3, the OEM passes different usage permissions to Company 1. Company 1 decides to pass a subset of its permissions to

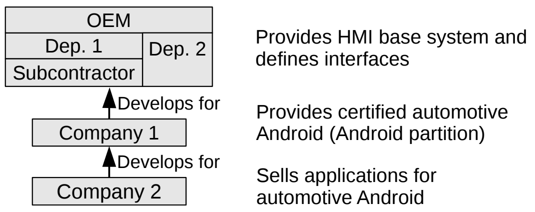

OEM
Dep. 1
Dep. 2
Subcontractor
Develops for
Company 1
Develops for
Company 2
Provides HMI base system and defines interfaces
Provides certified automotive Android (Android partition)
Sells applications for automotive Android

Figure 3: Software development hierarchy scenario

Company 2. This example of permissions requires a *delegation relation* between the parties for exchanging permissions. More precisely, we assume a *decentralized development process* as it is commonly applied in today's car industry, e.g., by service-based software development [18]. This process involves different departments of the OEM, subcontractors, as well as third-party developers like application developers for a (future) app store for vehicular apps. In the scenario depicted in Fig. 3, 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 provides an Android partition—running isolated from the OEM applications, e.g. within QNX [3]—with an automotive app menu which is compatible and certified for the OEM's HMI system. This app menu uses display areas dedicated by the OEM to display Android applications. Company 2 sells automotive Android applications that enhance the features of the car. For instance, an application which display relevant information about service stations nearby the current position of the car.

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.

3. ACCESS CONTROL MODEL

We now present the first main contribution of this paper: an access control model for automotive HMI systems. We first give an overview of the model, before we formally define the model and verify its properties.

3.1 Overview

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* whose pixels are modified by the application. We use *permissions* to define that a certain application is granted access to a certain display area. Since permissions can be granted and revoked dynamically, we need a formal model that can express this dynamic behavior. To this end, we introduce *states* that model the mapping of permissions to applications at a certain point in time. *Transitions* between states are triggered, whenever a permission is granted or revoked, i.e., whenever the mapping of permissions to applications changes. The state transitions over time can be modeled as a *graph*, where states correspond to vertices and transitions to edges.
We introduce the notion of a *safe state*. A safe state obeys all requirements of Sec. 2.2. A system is safe, if it starts in a safe state and every transition that occurs over time leads to a safe state. In order to ensure that only safe states can occur, we define which transitions are allowed. As stated

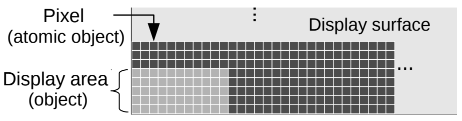


Figure 4: Entities: Pixels and display areas

in our requirements, our system should support granting of permissions where an application passes on a permission for a display area to another application. In order for a transition to be valid, an application can only pass a permission for a display area if it owns the permission for this area.

To further control the transfer of permissions, we introduce a *delegation relation* between applications. This relation is defined based on the decentralized development process where the OEM might delegate the development of a certain application component to a contractor, contractors might delegate parts of the application to subcontractors, etc. In order to show the GUI of a component, the actual developer can receive the permission to draw into a certain display area if he is in a delegation relation with the grantor. If a permission has been granted, the grantor cannot access the display area anymore. However, he can *revoke* a granted permission to get access to the corresponding display area again. Based on the rules to transfer permissions, we can then prove the correctness of our model by complete induction showing that the initial state is valid and each transition from a valid state leads again to a valid state.

Next, we present formal definitions of our model and an outline of the proof.

3.2 Subjects and Objects

A display area is defined as a set of pixels as depicted in Fig. 4. The smallest possible area consists of a single pixel, i.e., an *atomic object*. The complete display area is called the *display surface* and consists of all pixels.

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* $n \geq 1$.

3.3 States

Each *state* of our model is represented by the currently valid *permissions* and *delegation relations.*

Definition 3. *A permission grants an application access to a certain display area. Formally,* $P = \mathcal{P}(S \times O)$ *represents the set of all possible combinations 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 an application has* received *from other applications and permissions it has* granted *to other applications.*

Let $b \in B, s \in S$. Set $received(b, s) := \{r | (r, g) \in b(s)\}$ denotes the set of permissions that application s has received. And set $granted(b, s) := \{g | (r, g) \in b(s)\}$ denotes the set of permissions application s has granted. Accordingly, $(s, o) \in received(b, s')$ indicates that application s' has received the permission to access display area o from

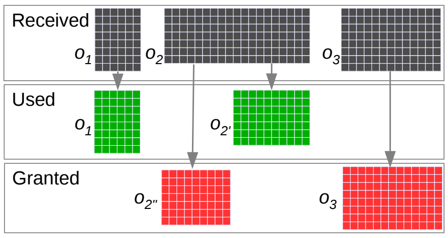


Figure 5: Example for a set of received, granted, and used display areas of an application

application s. $(s, o) \in granted(b, s')$ indicates that application s' has granted the permission to access display area o to application s.

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. A display area is in the set of used display areas of an application if the display area is in the set of received display areas and it must not be granted to other applications. Applications can only set the graphical content of display areas which are part of their set of used display areas.

Definition 4. *$used : B \times S \to \mathcal{P}(O)$ 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 $o \in used(b, s) \Leftrightarrow$*

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

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

In Fig. 5 we depict an example of the sets of received, granted, and used display areas. The application received the display areas o_1, o_2, and o_3 and decided to grant o_3 and part of o_2 to another application. Therefore, the display areas $o_{2''}$ and o_3 are in the set of granted display areas. Hence, o_1 and $o_{2'}$ are in the set of used display areas and the application can set their graphical content.

We say an application s' that received a display area o from another application s *depends on* that application since it can revoke display area o at any time. If display area o will be granted by s' to another application s'', then s'' also depends on application s since revoking of display area o by s will recursively revoke o from application s''. Fig. 6 depicts the hierarchical dependencies between applications. Application s_1 granted part of its display area o_1 to the applications s_2 and s_3 keeping only a small display area in the upper left corner to set its graphical content. The applications s_4 and s_5 received each a part of the display area s_2 received from s_1. Similarly, s_6 received part of the display area which s_3 received from s_1. Hence, the applications s_4, s_5, and s_6 indirectly depend on s_1 since revoking of the display areas granted to s_2 and s_3 would also revoke the display areas of these applications.

We introduce the operator $<_o$ that denotes if an application depends on another applications directly or indirectly according to a display area. More precisely, $s <_o s'$ means, application 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. The formal definition is in Sec. A, Def. 10.

For instance, in Fig. 6 application s_3 depends on s_1 according to display area o_3, i.e., $s_3 <_{o_3} s_1$. Hence, application s_6 depends on s_3 and s_1 according to o_6, i.e., $s_6 <_{o_6} s_3$ and

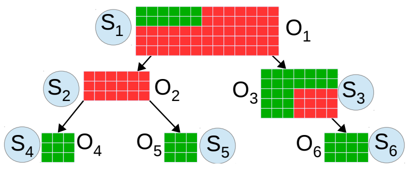


Figure 6: Example for permissions dependencies

$s_6 <_{o_6} s_1$. Let $s, s' \in S$. We define $s \neq_o s' \Leftrightarrow \nexists o \in O : s <_o s' \lor s' <_o s$. That is, application s and s' do not have a dependency due to any display area $o \in O$. For instance, in Fig. 6 the applications s_2 and s_3 do not have any display areas granted directly or indirectly, i.e., $s_2 \neq_o s_3$.

Applications will only grant permissions to or receive permissions from applications they are in a delegation relation with. Each application can be in a delegation relation with one or more other applications. To prevent granting of permissions from non-safety-critical applications to safety-critical applications a delegation relation is only established if a mutual agreement between the applications exists.

Definition 5. *We map to each application the set of applications it allows to be in a delegation relation with. This mapping is performed by a function $dr \in DR = \{dr : S \to \mathcal{P}(S)\}$. Applications s and $\tilde{s}$ are in a delegation relation with each other 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 relations between applications restrict the propagation of permissions. To support dynamic deployment of applications during runtime, applications are allowed to dynamically declare with which applications they want to be in a delegation relation.

After we introduced permissions and delegation relations, we formally define states.

Definition 6. *A* state *consists of active permissions and delegation relations between applications. Formally, $v \in V$ is a state in $V = B \times DR$.*

3.4 Properties of States

States have properties that represent the requirements of Sec. 2.2. First, we define Ω_{used} as the union set of all pixels in the sets of used display areas in b. Ω_{used} represents all display areas whose graphical content is set by applications.

Definition 7. *$\Omega_{used} : B \to \mathcal{P}(O)$ is a function which returns a set of all used areas according to $b \in B$. We define*

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

Additionally, 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.

Next, we define three *properties* that determine the consistency of our model and correspond to Req. 3 and Req. 4 introduced in Sec. 2.2. A state is called *safe*, if, and only if, it satisfies all three properties.

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 $\Leftrightarrow$ $\forall s, s' \in S : s \neq s' \Rightarrow \Phi(b, s) \cap \Phi(b, s') = \emptyset$.

This means, the sets of used display areas of all applications are intersection-free. Therefore, competing access to display areas is precluded and Req. 3.1 fulfilled.

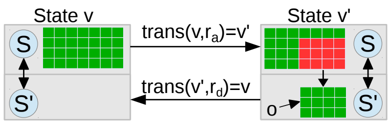


Figure 7: Example of a transition from state v to v'

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 must be in $\Omega_{used}(b)$ which represents all pixels of all used display areas. That is, for each pixel there exists an application that sets the pixels color, which fulfills Req. 3.2.

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

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

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

$$\begin{aligned} &\forall i \in \{0, ..., n\} \subset \mathbb{N}_0 : s_i <_o s_{i+1} \wedge \\ &\qquad s_i \in dr(s_{i+1}) \wedge s_{i+1} \in dr(s_i) \end{aligned} \tag{DP.3}$$

(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). Since permissions are only granted if a delegation relation exists, Req. 4 is fulfilled.

3.5 Transitions

Transitions between states are triggered by application requests. In more detail, four requests might trigger state changes: Requests for granting or revoking permissions, and requests for adding or deleting an application to or from a delegation relation. Next, we define these four requests.

Definition 8. *A transition is triggered by a request to add or delete permissions or delegation relations. The operation mode is determined by $RA = \{append, discard\}$. To alter permissions the set of requests RO is used. A request $r \in RO$ consists of operation mode, grantor, grantee and the display area; formally, $RO = RA \times S \times S \times O$.*
To alter delegation relations the set of requests RD is used. A request $r \in RD$ consists of operation mode and the two applications of which the first wants to establish or withdraw a delegation relation with the second; formally, $RD = RA \times S \times S$. The set of all requests is the set $R = RO \cup RD$.

Next, we formally define transitions between states.

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

As described in Sec. 3.4, states must fulfill all three properties to be safe states. A transition changes from one state to another. Transition rules ensure that the new state is also a safe state. Therefore, we say a sequence of transitions from safe state to safe state is called *safe state sequence*.

Since applications can request to trigger a transition to add or delete permissions or delegation relations, we need four rules to restrict transitions for the two operation modes on permissions and delegation relations.

In the following, we define Rule 1.1 and Rule 1.2 to describe how permissions can be changed, and Rule 2.1 and Rule 2.2 to describe how delegation relations can be changed.

Rule 1.1 If application s' wants to have a permission for display area o from application s, this is expressed by the request $r_a \in RO$ with $r_a = (append, s, s', o)$. If application s decides to grant a permission, $trans(v, r_a)$ is executed as depicted in Fig. 7. In detail, $trans$ calls a function $add_{so}(b, s, s', o)$ (cf. Def. 11 in Sec. A), which adds o to the set of granted permissions of s, and adds it to the set of received permissions of s' if the following condition for R1.1 is satisfied. Condition $cond_1$ is satisfied, if

(1) request r is in RO and operation mode ra is *append*.
(2) different s and s' have a delegation relation.
(3) the application s' does not receive a permission for a display area which is part of a display area granted by s', thus, preventing cyclic grants.
(4) display area o is in the set of used display areas of s.

Formally, we define

$$\begin{aligned} cond_1 = &r \in RO \wedge ra = append \wedge && (1)\\ &s \neq s' \wedge s' \in dr(s) \wedge s \in d(s') \wedge && (2)\\ &\exists \hat{o} \in O : \hat{o} \in used(b, s) \wedge o \subseteq \hat{o} \wedge && (3)\\ &\nexists (\tilde{s}, \tilde{o}) \in granted(b, s') : o \subseteq \tilde{o} && (4) \end{aligned}$$

Rule 1.2 An application s can revoke a permission for display area o from another application s'. This is expressed by the request $r_d \in RO$ with $r = (discard, s, s', o)$ as depicted in Fig. 7. In this case, $trans$ calls a function $del_{so}(b, s, s', o)$ (cf. Def. 12 in Sec. A), which removes o from the set of granted permissions of s and removes it from the set of received permissions of s' with $trans(v', r_d) = v$ if the following condition for R1.2 is satisfied. Condition $cond_2$ is satisfied if (1) request r is in RO, operation mode ra is *discard*, and (2) revoking of display area o was previously granted by application s to s' and $s \neq s'$. Formally, we define

$$\begin{aligned} &cond_2 = r \in RO \wedge ra = discard \quad \wedge && (1)\\ &s \neq s' \wedge (s', o) \in granted(b, s) \wedge (s, o) \in received(b, s') && (2) \end{aligned}$$

If s' has granted permissions that contain part of o, the function del_{so} will recursively revoke all these permissions. Therefore, del_{so} removes all permissions with display areas that are part of o from the sets of granted and received permission if the applications depend on s. Furthermore, the permission which 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.

Rule 2.1 The set of delegation relations DR can 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 function $add_{dr}(dr, s, s')$ (cf. Def. 13 in Sec. A), which adds the new relation $s \rightarrow s'$ to dr if the request r is in RD, the operation mode ra is *append*, and $s \neq s'$, i.e., no self-referencing delegation relation can be created. Formally, Condition $cond_3 = r \in RD \wedge ra = append \wedge s \neq s'$.

Granting of permissions is restricted to applications which are in a delegation relation. For instance, in Fig. 6 application S_1 has to be in a delegation relation with application S_2 before permissions can be granted between them.

Rule 2.2 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 $r = (discard, s, s')$. In this case, *trans* calls function $del_{dr}(dr, s, s')$ (cf. Def. 14 in Sec. A), which removes the delegation relation $s \to s'$ from dr if the following condition is satisfied. Condition $cond_4$ is satisfied if (1) request r is in RD, operation mode ra is *discard*, and (2) application s revoked or returned all permissions granted between s and s' beforehand and $s \neq s'$. Formally, we say

$$cond_4 = r \in RD \wedge ra = discard \quad \wedge \tag{1}$$

$$s \neq s' \wedge \forall o \in O : s \neq_o s' \tag{2}$$

Next, we formally define *trans* using the four 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

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

If none of the conditions of Rule 1.1, Rule 1.2, Rule 2.1, and Rule 2.2 are fulfilled, then the state v does not change.

4. SYSTEM VERIFICATION

In this section, we verify our model against the requirements in Sec. 2.2. Req. 1 (Dynamic permissions) is directly given by the granting and revoking of permissions by applications. The hierarchical dependencies between applications represent priorities according to Req. 2. The three properties Exclusive Access, Completeness, and Delegation (cf. Sec. 2.2) correspond to Req. 3.1 (exclusive access), Req. 3.2 (completeness), and Req. 4 (delegation), respectively.

While Req. 1 and Req. 2 are given by design of the model, Req. 3 and 4 are properties which we prove using complete induction. In this section we describe the main steps of our proof using Lemmas whose proofs can be found in Sec. B.

First, we define a *system* that consists of *sequences of states and requests*. We use this system to define propositions which we prove by using complete induction over the states. Finally we prove that a system is *safe* if the initial state fulfills the properties EAP, CP, and DP.

4.1 System

We prove the correctness of our system using complete induction over a sequence of transitions. Therefore, we define a *system* which represents all possible sequences of transitions between states reachable from a given initial state. First, we denote a sequence of $n-1$ requests as $x_r := (r_0, ..., r_{n-1})$ and a sequence of n states as $x_v := (v_0, ..., v_n)$ with $n \in \mathbb{N}_0$. A system $\Psi(v_0)$ consists of all possible transitions between states using requests, starting with initial state v_0. That is, in such a system each sequence of states x_v triggers $n-1$ transitions traversing the states x_v, denoted as $(x_r, x_v) \in \Psi(v_0)$. Furthermore, we denote $(v_x, r_x, v_y) \gg \Psi(v_0)$ if a sequence of states and requests beginning from state v_0 exists in the system, and contains the state v_x, where, using request r_x a transition to v_y is performed. The formal definition can be found in Sec. A, Def. 15 to 18.

A system is *safe* if all states in that system are safe states. That is, a system consists only of states that satisfy the properties EAP, CP, and DP.

4.2 Propositions

Since the states and transitions of our model consist of mathematical formulations we can define propositions that correspond with our properties defined in Sec. 3.4. We define three propositions that correspond to the properties and help us to prove 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 $\Leftrightarrow \forall (v, r, v') \in V \times R \times V : (v, r, v') \gg \Psi(v_0) \Rightarrow v, v'$ satisfy EAP

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

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

Proposition 1 says that all sequences in a system $\Psi(v_0)$ satisfy EAP if, and only if, for all states v' which can be directly generated from any state v with one request, implies that states v and v' also satisfy EAP. Proposition 2 and Proposition 3 are similar to the Proposition 1 but they target CP, and DP. By proving these three propositions we can conclude that every system $\Psi(v_0)$ is a safe system if state v_0 satisfies our properties in Sec. 3.

4.3 Proof by Complete Induction

To prove the correctness of Proposition 1, Proposition 2, and Proposition 3, we define a lemma for each of them. Additionally, 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 prove the propositions. We present the formal proofs of Lemma 1, Lemma 2, Lemma EAP in Sec. B. Due to space restrictions the proofs of Lemma CP and Lemma DP are published in [13].

We first define Lemma 1, which states that after a transition using (R1.1) with $add_{so}(b, s, s', o)$ the display area o moved from the used display areas of application s to 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 $b' = add_{so}(b, s, s', o)$.

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

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

The following Lemma 2 says that after a transition using rule (R1.2) with del_{so} each display area o' (being a subset of the display area o) moves from the set of used display areas of application s' to that of application s.

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 $v = (b, dr)$ satisfy EAP and $ra = discard$ and $cond_2 \Rightarrow trans(v, r) = (b', dr) \in V$ with $b' = del_{so}(b, s, s', o)$:

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

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

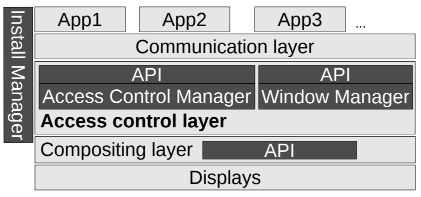


Figure 8: Implemented architecture

Lemma 1 and 2 say that a transition with an add_{so} or a del_{so} do not modify the union set of used display areas of all applications. The proofs are in Sec. B.

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. The proof is in Sec. B.

The 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.

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

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

Finally, we prove Proposition 1, 2, and 3 by complete induction. Without loss of generality, we assume initial state $v_0 \in V$ with $v_0 = (\{(s_{root}, (\{(s_{root}, AO)\}, \emptyset))\}, \emptyset)$. That is, v_0 maps the permission to access the whole display area to the root application s_{root}. Since the permission has not been granted by another application, we set both, grantor and grantee, to s_{root} and no delegation relation exists.

Base: v_0 satisfies EAP and CP, since only s_{root} has a permission, and has access to all pixels. DP is satisfied, since $\nexists s, s' \in S : s \neq s' \wedge s <_o s'$ (cf., DP.1).

Induction hypothesis: v_i satisfies EAP, CP, and DP—with $(x_r, x_v) \in \Psi(v_0)$ and v_i a state of sequence x_v.

Induction step: Let v_i satisfy EAP, CP, and DP. From the Lemmas EAP, CP, and DP follows $v_{i+1} = trans(v_i, r_i)$ satisfies EAP, CP, and DP. $\square$

We follow that $\Psi(v_0)$ is a safe system, i.e., our rules do not violate EAP, CP, or DP.

5. IMPLEMENTATION

We implemented a proof-of-concept prototype which demonstrates the feasibility to implement our access control system. First, we introduce our system architecture.

5.1 Implemented Architecture

The system architecture of our Linux-based implementation is depicted in Fig. 8. To demonstrate the functionality of our access control system we used typical Instrument Cluster (IC) and Head Unit (HU) *Applications*. The *Communication Layer* provides session-based FIFO communication between applications, *Access Control Manager (ACM)*, and *Window Manager (WM)*. The access control layer contains the two components ACM and WM. The ACM is the access control unit that performs access decisions in the access control layer. The WM is responsible for creating, destroying, and positioning of windows. Applications that want to create, modify, or move a window send a request to the WM. We implemented a client API for access control management and window management that can be used by the applications to interact with the ACM and the WM. We implemented the *Compositing Layer* that provides an API which allows for resizing and mapping of windows. Each time the WM applies changes to windows it updates the screen by initiating the respective API call to the compositing layer. The implemented compositing layer uses the driver API of the Image Processing Unit (IPU) provided by the i.MX6 board for bit-blit operations in framebuffers. The *Install Manager* is responsible for the deployment of applications, XML-based permissions, and delegation relations.

We deployed our implementation in the cockpit demonstrator depicted in Fig. 9. As HCI devices the demonstrator uses two automotive 12" displays each with a resolution of 1440 × 540 pixels, which are connected to an embedded i.MX6 platform from Freescale Semiconductor. We also connected the steering wheel buttons and the central control knob, which are used to control the applications.

Next, we describe the main system components in detail.

5.2 Applications

To demonstrate automotive scenarios, we use 15 applications like speedometer, tachometer, check engine indicator, phone, and navigation software. In addition, we use two Linux applications that represent an Android menu and an Android application. Each application has a unique id called App_{id} and an application class $AppClass_{id}$ (e.g., an application class for indicators or video playing applications). Applications receive notifications about event changes, e.g., if the car starts moving. The deployment of an application includes the deployment of an application certificate to the ACM which is required to verify the authenticity of the application. The certification authority is either the OEM or a trusted third-party company, e.g., hosting an automotive Android app store. A certificate contains information about the application, i.e., the App_{id}, $AppClass_{id}$, company, public key, certification authority, and issue date (cf. X.509 [16]). The ACM stores the certificate of each application. An application can request a connection to the ACM via the Communication Layer. This requires the authentication of the application against the ACM which can be done by using a certificate-based authentication technique like described in ISO/IEC 9594-8 [16]. Each application has a set of XML files which contain the App_{id} and the $AppClass_{id}$ of the applications they want to be in a delegation relation with. In addition, the XML files can contain restrictions according to the display area an application shall get. For instance, the IC shall only grant a small display area to the requesting phone application which wants to display information like phone number in case of an incoming phone call. We determined the hierarchical dependency of applications in two steps. First, we assigned each application to one of the three classes *IC*, *HU*, and *Android*. Class *IC* contains IC applications which are normally safety-critical like the indicators and therefore have the highest priority. Class *HU* contains OEM applications like navigation. Class *Android* contains third-party applications which have the smallest

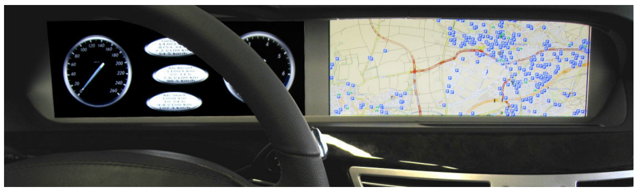


Figure 9: Cockpit demonstrator

priority. As second step, we determined the hierarchical dependency within a class according to given requirements (e.g., automotive ISO standards [9], legal restrictions [17]) or designed the dependencies between the applications according to HMI usability. In order to deploy an application, it is passed to the Install Manager. The deployment of an application consists of the application binaries, the XML files, and updates to XML files of already existing applications (e.g., to establish a delegation relation to an existing application). To prevent malicious modifications of the XML files, these are digitally signed, cf. [16].

5.3 Access Control Manager

According to our access control concept, permissions are centrally managed by the ACM. Each application is connected to the ACM and can send requests which the ACM forwards to the application specified in the request or which is responsible for the requested display area. If the receiving application grants a permission to the requesting application, it sends it to the ACM. The ACM denies all requests in case the application is not authenticated or the receiving and requesting applications are not in a delegation relation. Otherwise, the ACM checks if the granted permission is valid and does not violate existing permissions. Then it updates its permission mapping tables and notifies the client. All permissions are only valid if they 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. This initial state is safe, since it fulfills the properties in Sec. 3.4, cf. Sec. 4. The ACM has always a consistent view of all granted permissions and can ensure consistency by preventing invalid permission exchanges.

In the following we describe the four access control API functions which correspond to the rules of our model.

Grant a permission (R1.1) Applications can grant permissions to other applications by using the request $GrantPermissions(DisplayArea\ \mathrm{o}, App_{id}\ \mathrm{id})$. The requesting application specifies the display area o for which the application with id shall get a permission.

Revoke a permission (R1.2) For revoking a permission, $RevokePermissions(DisplayArea\ \mathrm{o}, App_{id}\ \mathrm{id})$ is used. The requesting application specifies the display area o for which the permission shall be revoked from id.

Create a delegation relation (R2.1) To create a delegation relation to another application id, an application uses the request $CreateDelegationRel(App_{id}\ \mathrm{id})$ and is implicitly pending while waiting for the confirmation from the ACM. The ACM stores id in a table of the requesting application. If application id did not request a delegation relation, yet, the ACM notifies it with $DelegationRelPending(App_{id}\ \mathrm{id})$. A delegation relation is only established if both applications have requested the delegation relation. The ACM sends $ConfirmDelegationRel(App_{id}\ \mathrm{id})$, where id is the application a delegation relation is established with.

Delete a delegation relation (R2.2) An application requests the deletion of a delegation relation by calling $DeleteDelegationRel(App_{id}\ \mathrm{id})$. If there exists any granted permission between those two applications the ACM denies the request. Otherwise it deletes the entry in the according table and notifies both applications.

5.4 Window Manager

The WM is the only process that can access the compositing layer. Thus, the WM checks, by calling the ACM, if the a request for creating a window by an application matches existing permissions of the requesting application and initiates the creating or moving of the window by performing respective API calls to the compositing layer. Next, we describe the three API functions for interaction with the WM.

Create a window Applications need a window mapped to the screen to display graphical content. To this end, after receiving a permission an application can issue the request $CreateWindow(Window\ \mathrm{w})$ to map a window to the display area it previously received a permission for. The parameter w specifies the size and the position of the window. The WM sends the request $Verify(Window\ \mathrm{w}, App_{id}\ \mathrm{id})$ to the ACM, which verifies that a permission of application id matches window w and responds with $ResponseVerify(Ack\ \mathrm{ack})$. If an appropriate permission exists the WM sends an acknowledgment and the window id in response with $ResponseCreateWindow(Ack\ \mathrm{ack}, Window_{id}\ \mathrm{id})$.

Modify a window Furthermore, permission changes can require applications to modify windows using the request $ModifyWindow(Window\ \mathrm{w})$ with new window parameters w. The WM also verifies if a permission exists by sending a request $Verify(Window\ \mathrm{w}, App_{id}\ \mathrm{id})$ to the ACM which replies $ResponseVerify(Ack\ \mathrm{ack})$. Finally, the WM confirms the modify request with $ResponseModifyWindow(Ack\ \mathrm{ack})$.

Delete a window Applications can delete their windows by using the request $DeleteWindow(Window\ \mathrm{w})$. In case the necessary permission to display a window is revoked, the ACM notifies the application by sending $DeletePermission(DisplayArea\ \mathrm{o})$ and it sends the WM a notification about the permission change by using $NotifyPermChanged(DisplayArea\ \mathrm{o}, App_{id}\ \mathrm{id})$. Then, the WM starts a timeout. If the application does not have any further permissions or does not modify the window to switch to another permission by using the request $ModifyWindow(Window\ \mathrm{w})$, the window will be deleted by the WM after the timeout expires. The WM notifies the application about the deleted window by sending $WindowDeleted(Window\ \mathrm{w})$.

Implemented Scenarios

Next, we describe two implemented scenarios where an application uses—depending on the current state—either a display area on the IC display or the HU display.

In the first scenario, we demonstrate granting and revoking of permissions to display an application *Media*. The required request-response calls are depicted in Fig. 10. If the car reaches parking position ❶, the (yet hidden) application *Media* gets notified and, in order to display the last presented video in full-screen on the HU display, requests

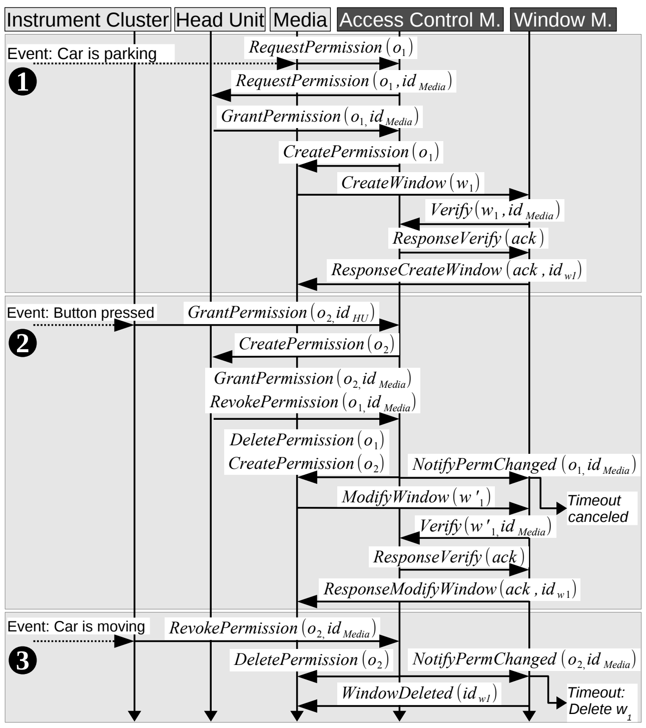
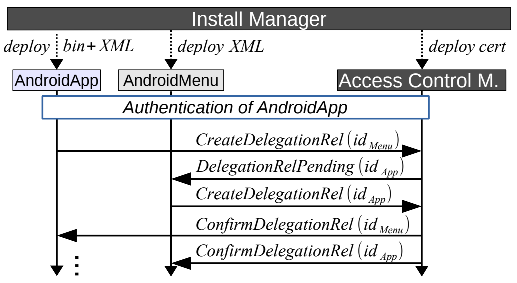


Figure 10: Sequence of operations for scenario 1

a permission for display area o_1 ($RequestPermission(o_1)$) from the ACM. $RequestPermission(o_1, id_{Media})$ is then forwarded to the HU application which grants a permission to $Media$ ($GrantPermission(o_1, id_{Media})$) since the car is not in motion. The ACM checks if the HU application has a valid permission which covers o_1 and creates a permission for $Media$ by using $CreatePermission(o_1)$. Then, $Media$ sends $CreateWindow(w_1)$ to the WM to create a window. With $Verify(w_1, id_{Media})$ the WM lets the ACM check if w_1 matches a permission of $Media$. Then, the WM confirms the creation of the window and sends back the window id id_{w_1}. Now, $Media$ has a valid window and can present the video in full-screen on the HU display.

Then, the driver decides to shift $Media$ to the IC display and to watch the video in full-screen. This requires the modification of the window position and therefore the granting of a new permission for the IC display. To perform the shift, the driver presses a button ❷ which triggers the IC to grant a permission ($GrantPermission(o_2, id_{HU})$) for its display area to the HU since no car related content like speedometer needs to be displayed while the car is not moving. Then, the ACM creates a permission with $CreatePermission(o_2)$ for the HU. The HU replaces the permission previously granted to $Media$ by a new one which covers the display area of the IC using the calls $GrantPermission(o_2, id_{Media})$ and $RevokePermission(o_1, id_{Media})$. The ACM in response creates a new permission and deletes the old one by using $CreatePermission(o_2)$ and $DeletePermission(o_1)$.

Deleting a permission requires a notification of the WM ($NotifyPermChanged(o_1, id_{Media})$) since the window w_1 has no valid permission anymore. Therefore the WM hides the window and starts a timeout after which the window will be deleted if still no appropriate permission exists. Since



Install Manager
deploy
bin+XML
deploy XML
deploy cert
AndroidApp
AndroidMenu
Access Control M.
Authentication of AndroidApp
CreateDelegationRel (id Menu)
DelegationRelPending (id App)
CreateDelegationRel (id App)
ConfirmDelegationRel (id Menu)
ConfirmDelegationRel (id App)

Figure 11: Sequence of operations for scenario 2

$Media$ got a new permission for o_2, it modifies its window id_{w_1} by sending the request $ModifyWindow(w_1')$ to the WM. In response the WM stops the timeout and lets the ACM check with $Verify(w_1', id_{Media})$ if the w_1' is covered by a permission of $Media$. Finally, the WM confirms the modification with $ResponseModifyWindow(ack, id_{w_1})$. Now, the video is presented in full-screen on the IC display.

After watching part of the video, the driver starts the engine and accelerates the car. Therefore, $Media$ is no longer allowed to present the video. The IC receives a notification about that state change ❸. Since $Media$ belongs to an application class known to display videos, the IC calls $RevokePermission(o_2, id_{Media})$ to revoke the current permission. In addition, the IC would now grant the permissions to IC applications like speedometer, tachometer, and indicators, which is not depicted in Fig. 10. The ACM deletes the permission with $DeletePermission(o_2)$ and sends $NotifyPermChanged(o_2, id_{Media})$ to the WM. Finally, the WM deletes the window id_{w_1} after the timeout.

In the second scenario we focus on the deployment of the new application *AndroidApp*. As described in Sec. 2 we assume Company 1 provides an Android partition for the OEM infotainment system and implemented an application *AndroidMenu* designed for usage in vehicles. Company 2 implemented an application *AndroidApp* which shall be deployed on a vehicle. Company 2 negotiates with Company 1 about a permission for displaying the *AndroidApp* which shall be selectable in *AndroidMenu*. Company 1 defines an XML file which contains the information about the *AndroidApp* and the display area that shall be granted to it. Company 2 also defines an XML file which contains the information about *AndroidMenu* and creates a certificate for *AndroidApp*. The XML files, the binary of *AndroidApp*, and its certificate are uploaded on the vehicle backend server and get deployed on the vehicle by the *Install Manager* on request of the user. As soon as *AndroidApp* is loaded it sends a request for authentication to the ACM. The ACM can verify the authenticity by using the certificate of *AndroidApp* and applying an authentication procedure as described in ISO/IEC 9594-8 [16]. Then the *AndroidApp* requests a delegation relation to *AndroidMenu* by sending $CreateDelegationRel(id_{Menu})$. The ACM sends $DelegationRelPending(id_{App})$ since the *AndroidMenu* did not send a delegation request, yet. As soon as *AndroidMenu* receives the XML file, it also requests the creation of a delegation relation to *AndroidApp*. Then, the ACM calls $ConfirmDelegationRel$ with the parameters id_{App} and id_{Menu} to confirm the delegation relation. Now, *AndroidMenu* can grant permissions to *AndroidApp*.

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 [14] which allows multiplexing of graphical content from different virtual machines safely onto a single GPU. However, both works neglect restrictions in window management. Similarly, Epstein et al. address security issues like weak authentication, unlimited sharing of X resources, between applications or overlapping windows in X11 [8] and propose mechanisms [7] to prevent them. However, they do not enforce permission based display access restrictions.

Protection of shared resources by using access control has been researched almost since the beginnings of operating systems [19]. 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] unify 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.

Bell and LaPadula [4] defined a model for secure information sharing and information flow control. The model is defined as a state-based system for enforcing access control and uses an access control matrix for restricting access to data in order to provide confidentiality of information. Restrictions are enforced for access of users or applications to files or resources concering the sensitivity level and security level. Therefore, their model does not guarantee exclusive access on resources. Additionally, it does not support decentralized permission management and consequently cannot be used for a decentralized development process. Related work [21]—based on the Bell and LaPadula model—extends the model for hierarchical organizations and distinguishes between read access and write access. They therefore target the flow of information, but do not support 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 proved the correctness of our model and showed that it fulfills all 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 model for context handling and constraints. Additionally, we want to improve our implementation by using a virtualized Android partition which is running applications using our API. Furthermore, we want to analyze overhead and performance of our implementation.

8. ACKNOWLEDGMENT

This paper was supported by the ARAMiS project of the German BMBF with funding ID 01IS11035.

9. REFERENCES

[1] Mercedes-Benz F125. http://www.motorauthority.com /news/1070046_mercedes-benz-ponders-the-future-of-in-car-tech, 2014.

[2] Mercedes-Benz integration of iPhone App in A-Class. http://www.iphone-ticker.de/mercedes-benz-iphone-integration-a-klasse-30952/, 2014.

[3] New Version of QNX CAR Platform. `http://www.qnx.com/news/pr_5602_1.html`, 2013.

[4] D. E. Bell and L. J. LaPadula. Secure Computer System: Unified Exposition and MULTICS Interpretation. Tech. Report ESD-TR-75-306, 1976.

[5] J.-C. Birget, X. Zou, G. Noubir, and B. Ramamurthy. Hierarchy-based access control in distributed environments. In *Communications, 2001. ICC 2001.*

[6] C. Ebert and C. Jones. Embedded software: Facts, figures, and future. *Computer*, 42(4):42–52, April 2009.

[7] J. Epstein et al. A prototype B3 trusted X Window System. In *Proceedings of the 7th Annual Computer Security Applications Conference*, Dec. 1991.

[8] J. Epstein and J. Picciotto. 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*, volume 1, Oct. 1991.

[9] ESOP. *On safe and efficient in-vehicle information and communication systems.* Commission of the European Communities, 2008.

[10] N. Feske and C. Helmuth. Overlay window management: User interaction with multiple security domains, 2004.

[11] N. Feske and C. Helmuth. A Nitpicker's guide to a minimal-complexity secure GUI. In *Proceedings of the 21st ACSAC*, pages 85–94, Dec. 2005.

[12] Ford. Software development kit (SDK). https://developer.ford.com/develop/openxc/, 2013.

[13] S. Gansel et al. An Access Control Concept for Novel Automotive HMI Systems. Tech Report 2013/02, University of Stuttgart, IPVS, Germany, 2013.

[14] J. G. Hansen. Blink: Advanced Display Multiplexing for Virtualized Applications. In *Proceedings of the 17th NOSSDAV*, 2007.

[15] ISO 26262. *Road vehicles – Functional Safety.* ISO, Geneva, Switzerland, Nov. 2011.

[16] ISO/IEC 9594-8. *The Directory: Public-key and attribute certificate frameworks.* Switzerland, 2008.

[17] H. Janker. *Straßenverkehrsrecht: StVG, StVO, StVZO, Fahrzeug-ZVO, Fahrerlaubnis-VO, Verkehrszeichen, Bußgeldkatalog.* C.H. Beck, 2011.

[18] I. Krüger et al. Service-Based Software Development for Automotive Applications. In *Proceedings of the CONVERGENCE 2004. CTEA*, Jan. 2004.

[19] B. W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1):18–24, Jan. 1974.

[20] M. Stümpfle and H. Kohler. Die Konnektivität als Kernmerkmal von Premium-Fahrzeugen. Xpert.press, pages 125–144. Springer Berlin Heidelberg, 2013.

[21] J. Wang, L. Zhou, and C. Tan. A BLP-Based Model for Hierarchical Organizations. In *Proc. of the 2009 Second IWCSE*, pages 456–459, 2009.

[22] E. Wilde and N. Nabholz. Access Control for Shared Resources. In *Pro. of the CIMCA-IAWTIC'06*, 2005.

APPENDIX

A. FORMAL DEFINITIONS

Definition 10. *We define the transitive operator $<_o$ that denotes 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 $s <_o s' \Leftrightarrow$

$$\exists s_1, ..., s_n \in S; \exists o_1, ..., o_{n-1} \in O : \tag{10.1}$$
$$s_1 = s' \wedge s_n = s \wedge o \subseteq o_{n-1} \wedge \tag{10.2}$$
$$\forall i : 1 \leq i < n : (s_i, o_i) \in received(b, s_{i+1}) \wedge \tag{10.3}$$
$$\forall i : 1 \leq i < n-2 : o_{i+1} \subseteq o_i \tag{10.4}$$

Definition 11. *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 $b' = add_{so}(b, s, s', o) \Leftrightarrow$*

$$b'(s) = (received(b, s), granted(b, s) \cup \{(s', o)\}) \wedge \tag{11.1}$$
$$b'(s') = (received(b, s') \cup \{(s, o)\}, granted(b, s')) \wedge \tag{11.2}$$
$$[\forall s_3 \in S \backslash \{s', s\} : b'(s_3) = b(s_3)] \tag{11.3}$$

Definition 12. *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*

$$b' = del_{so}(b, s, s', o) \Leftrightarrow [\forall \hat{s} \in S : \hat{s} <_o s \Rightarrow \tag{12.1}$$
$$received(b', \hat{s}) = received(b, \hat{s}) \backslash \{(\tilde{s}, o') \in S \times O | o' \subseteq o \wedge \hat{s} <_{o'} \hat{s}\} \wedge \tag{12.2}$$
$$granted(b', \hat{s}) = granted(b, \hat{s}) \backslash \{(\tilde{s}, o') \in S \times O | o' \subseteq o \wedge \tilde{s} <_{o'} \hat{s}\}] \wedge \tag{12.3}$$
$$b'(s) = (received(b, s), granted(b, s) \backslash \{(s', o)\}) \wedge \tag{12.4}$$
$$\forall s'' \in S \backslash \{s', s\}; \forall o' \in O : o' \subseteq o \wedge s'' \neq_{o'} s \vee s <_o s'' \Rightarrow b'(s'') = b(s'') \tag{12.5}$$

Definition 13. *Function $add_{dr} : DR \times S \times S \to DR$ is defined as follows. Let $dr, dr' \in DR; s, s' \in S$. We define $dr' = add_{dr}(dr, s, s') \Leftrightarrow dr'(s) = dr(s) \cup \{s'\} \wedge \forall \hat{s} \in S : \hat{s} \neq s \Rightarrow dr'(\hat{s}) = dr(\hat{s})$*

Definition 14. *Function $del_{dr} : DR \times S \times S \to DR$ is defined as follows. Let $dr, dr' \in DR; s, s' \in S$. We define $dr' = del_{dr}(dr, s, s') \Leftrightarrow dr'(s) = dr(s) \backslash \{s'\} \wedge \forall \hat{s} \in S : \hat{s} \neq s \Rightarrow dr'(\hat{s}) = dr(\hat{s})$*

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

Definition 15. *Operator $\succ$ denotes whether an element is part of a sequence of states. The set of sequences of states is a set of n-tuples defined as $X^{I^n} = \{(x_0, ..., x_i, ..., x_n) | x_i \in X \wedge i \in I^n \wedge x_i = f(i)$ with $f : I^n \to X\}$.*
$(x_0, x_1, ..., x_n) \in X^{I^n}$ 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. 15, we next define the sequences of requests, the sequences of states and a distinct mapping between these sequences by using transitions.

Definition 16. *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*
$\forall i \in I^{n-1} : v_{i+1} = trans(v_i, r_i)$.

Definition 17. *A system generated by $v_0 \in V$ is stated as $\Psi(v_0) \subset R^{I^n} \times V^{I^n}$. Let $x_r = (r_0, ..., r_{n-1}) \in R^{I^{n-1}}$, $x_v = (v_0, ..., v_n) \in V^{I^n}$. We define $(x_r, x_v) \in \Psi(v_0) \Leftrightarrow \forall i \in I^n \backslash \{0\} : v_i = trans(v_{i-1}, r_{i-1})$. Let $(v, r, v') \in V \times R \times V$, $v_0 \in V$. We define $(v, r, v') \gg \Psi(v_0) \Leftrightarrow$*

$$\exists x_r \in R^{I^{n-1}}, \exists x_v \in V^{I^n}, \exists i \in I^n \backslash \{0\} : \tag{17.1}$$
$$(x_r, x_v) \in \Psi(v_0) \wedge v_i \succ x_v \wedge v_{i+1} \succ x_v \wedge r_i \succ x_r \wedge \tag{17.2}$$
$$(v, r, v') = (v_{i-1}, r_{i-1}, v_i). \tag{17.3}$$

(v, r, v') is part of a system if a sequence of requests and states (17.1) exists of which v, r and v' are part of (17.2), and, a transition from state v to v' by request r (17.3) exists.

Definition 18. *$v \in V$ 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 $\Psi(v_0) \subset V^{I^n} \times R^{I^n}$ with $x_r \in R^{I^{n-1}}$ and $x_v = (v_0, ..., v_n) \in V^{I^n}$ is a safe system $\Leftrightarrow \forall (x_r, x_v) \in \Psi(v_0) : x_v$ is a safe sequence.*

B. PROOFS

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:

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

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

(i): Due to (11.1), we know $(s', o) \in granted(b', s)$ after a transition to state v'. Since $o \subseteq \hat{o}$, it follows $\hat{o} \cap o = o \neq \emptyset$. Thus, the condition (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:

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

(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 (11.1) $\Lambda(b, s) = \Lambda(b', s)$. 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 $v = (b, tr)$ we conclude $\hat{o} \cap \Gamma(b, s) = \emptyset$. With (11.1) we conclude $\Gamma(b', s) = \Gamma(b, s) \cup o$ and therefore (b) is true in state v'.

$$\begin{aligned}(\hat{o} \backslash o) \cap \Gamma(b', s) &= (\hat{o} \backslash o) \cap (\Gamma(b, s) \cup o) && (6.1)\\ &= ((\hat{o} \backslash o) \cap (\Gamma(b, s))) \cup \\ &\quad ((\hat{o} \backslash o) \cap o) && \text{(Distr. law)}\\ &= \emptyset \cup ((\hat{o} \backslash o) \cap o) = \emptyset && (\hat{o} \cap \Gamma(b, s) = \emptyset)\end{aligned}$$

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:

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

(i): We directly follow $received(b', s') = received(b, s') \cup \{o\}$ due to (11.2).

(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 $\Gamma(b, s') \cap o \neq \emptyset$. Then we 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 *Def.* 11.2 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 (11.3).

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

Proof for Lemma 2

We first prove (L2.1): Let $s \neq s'$ and state v satisfies EAP. In Def. 12 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' are no longer in the set of used display areas of the according applications. Hence, we can directly conclude statement (L2.1).

We prove (L2.2): Due to (R1.2), we know that $(s', o) \in granted(b, s)$ which leads to $o \subseteq \Lambda(b, s)$. Hence, (4.1) is satisfied. With (12.1) 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:

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

Hence, we conclude statement (L2.2). □

Proof for Lemma EAP: 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 changes of dr do not affect Lemma EAP. In case $r = (ra, s, s', o) \in RO$, there are 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' (11.3). Hence, $\forall \hat{s} \in S \backslash \{s, s'\} : \big(used(b', \hat{s}) = used(b, \hat{s})\big)$ due to $b'(\hat{s}) = b(\hat{s})$.

Since v satisfies EAP, we conclude $\forall \hat{s}, \tilde{s} \in S \backslash \{s, s'\} : \hat{s} \neq \tilde{s} \Rightarrow \Phi(b', \hat{s}) \cap \Phi(b', \tilde{s}) = \emptyset$. This means we only have to prove EAP in v' for s and s'. To this end, we show the following statements:

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

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

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

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

(ii): Let $s'' \in S \backslash \{s, s'\}$ be arbitrary, then follows:

$$\begin{aligned}
&\Phi(b', s) \cap \Phi(b', s'')\\
&= (\Phi(b, s) \backslash o) \cap \Phi(b', s'') && \text{(L1.1)}\\
&= (\Phi(b, s) \backslash o) \cap \Phi(b, s'') && (11.3)\\
&= (\Phi(b, s) \backslash \Phi(b, s'')) \backslash (o \cap \Phi(b, s'')) && \text{(Dist. law)}\\
&= \emptyset \backslash (o \cap \Phi(b, s'')) = \emptyset && \text{(EAP)}
\end{aligned}$$

(iii): Let $s'' \in S \backslash \{s, s'\}$ be arbitrary, then follows:

$$\begin{aligned}
&\Phi(b', s') \cap \Phi(b', s'')\\
&= (\Phi(b, s') \backslash o) \cap \Phi(b', s'') && \text{(L1.2)}\\
&= (\Phi(b, s') \backslash o) \cap \Phi(b, s'') &&\\
&= (\Phi(b, s') \backslash \Phi(b, s'')) \cup (o \cap \Phi(b, s'')) && (11.3)\\
&= \emptyset \cup (o \cap \Phi(b, s'')) = \emptyset && \text{(Dist. law, EAP)}
\end{aligned}$$

(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 $o' \subseteq o$ (12.5). We know with (L2.1) that $\forall \hat{s} \in S \backslash \{s\} : \Phi(b', \hat{s}) \subseteq \Phi(b, \hat{s})$. Since state v satisfies EAP we only have to prove EAP for applications s and s' in state v', namely:

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

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

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

(ii): Let $\hat{s} \in S \backslash \{s, s'\}$ be arbitrary then follows:

$$\begin{aligned}
&\Phi(b', s) \cap \Phi(b', \hat{s})\\
&= (\Phi(b, s) \backslash o) \cap \Phi(b', \hat{s}) && \text{(L2.2)}\\
&= (\Phi(b, s) \backslash o) \cap \Phi(b, \hat{s}) && (12.2)\\
&= (\Phi(b, s) \backslash \Phi(b, \hat{s})) \backslash (o \cap \Phi(b, \hat{s})) && \text{(Distr. law)}\\
&= \emptyset \backslash (o \cap \Phi(b, \hat{s})) = \emptyset && \text{(EAP)}
\end{aligned}$$

(iii): Let $\hat{s} \in S \backslash \{s, s'\}$ be arbitrary then follows:

$$\begin{aligned}
&\Phi(b', s') \cap \Phi(b', \hat{s})\\
&= (\Phi(b, s') \backslash o) \cap \Phi(b', \hat{s}) && \text{(L1.2)}\\
&= (\Phi(b, s') \backslash o) \cap \Phi(b, \hat{s}) && (12.1)\\
&= (\Phi(b, s') \backslash \Phi(b, \hat{s})) \cup (o \cap \Phi(b, \hat{s})) && \text{(Distr. law)}\\
&= \emptyset \cup (o \cap \Phi(b, \hat{s})) = \emptyset && \text{(EAP)}
\end{aligned}$$

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