8-2016

Platform-based Plug and Play of Automotive Safety Features - Challenges and Directions

Deepak Gangadharan
gdeepak@seas.upenn.edu

Jin Hyun Kim
University of Pennsylvania, jinhun@seas.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

BaekGyu Kim
Toyota Info Technology Center, bkim@us.toyota-itc.com

See next page for additional authors

Recommended Citation

Deepak Gangadharan, Jin Hyun Kim, Insup Lee, Oleg Sokolsky, BaekGyu Kim, and Shin'ichi Shiraishi, "Platform-based Plug and Play of Automotive Safety Features - Challenges and Directions", *The 22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2016)* , 76-84. August 2016.

The 22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2016), pp. 76--84. Daegu, South Korea, August 2016 (Invited paper).

Platform-based Plug and Play of Automotive Safety Features - Challenges and Directions

Abstract

Optional software-based features are increasingly becoming an important cost driver in automotive systems. These include features pertaining to active safety, infotainment, etc. Currently, these optional features are integrated into the vehicles at the factory during assembly. This severely restricts the flexibility of the customer to select and use features on-demand and therefore, the customer will either have to be satisfied with an available set of feature options or pre-order a car with the required features from the manufacturer resulting in considerable delay. In order to increase flexibility and reduce the delay, it is necessary to provide the option to configure the vehicle on-demand at the dealership or remotely.

In this paper, we present our vision and challenges involved in developing a platform infrastructure that allows on-demand deployment of automotive safety features and ensures their correct execution.

Disciplines

Computer Engineering | Computer Sciences

Comments

The 22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2016), pp. 76--84. Daegu, South Korea, August 2016 (Invited paper).

Author(s)

Deepak Gangadharan, Jin Hyun Kim, Insup Lee, Oleg Sokolsky, BaekGyu Kim, and Shin'ichi Shiraishi

Platform-based Plug and Play of Automotive Safety Features - Challenges and Directions

Deepak Gangadharan, Jin Hyun Kim, Insup Lee and Oleg Sokolsky
University of Pennsylvania
Email: {deepakg,jinhyun}@seas.upenn.edu, {lee,sokolsky}@cis.upenn.edu

BaekGyu Kim and Shinichi Shiraishi
Toyota Info Technology Center USA
Email: {bkim,sshiraishi}@us.toyota-itc.com

***Abstract*—Optional software-based features are increasingly becoming an important cost driver in automotive systems. These include features pertaining to active safety, infotainment, etc. Currently, these optional features are integrated into the vehicles at the factory during assembly. This severely restricts the flexibility of the customer to select and use features on-demand and therefore, the customer will either have to be satisfied with an available set of feature options or pre-order a car with the required features from the manufacturer resulting in considerable delay. In order to increase flexibility and reduce the delay, it is necessary to provide the option to configure the vehicle on-demand at the dealership or remotely.**

In this paper, we present our vision and challenges involved in developing a platform infrastructure that allows on-demand deployment of automotive safety features and ensures their correct execution.

I. Introduction

In the recent years, there has been a lot of innovation in optional automotive software features, especially the ones pertaining to active safety and infotainment. Car manufacturers are increasingly providing these optional software features in their vehicles that make driving safe and the journey enjoyable. However, all these features are currently integrated into the vehicle by the manufacturer. This leaves the customer with limited options to choose a set of software features that he/she requires. The customer has to either be content with the set of pre-configured features provided by the dealership or pre-order a car with the required set of features which will take significant amount of time to be delivered by the manufacturer. In the above context, it is desirable to improve the customer experience by providing more flexibility with less delay for deployment of a new required feature.

The possible alternative to improve the flexibility of deployment of a safety feature according to customer requirement are illustrated in Fig. 1. The scenario shown in Fig. 1(a) depicts the current process of ordering, manufacturing and delivering the vehicle, which has to go through the manufacturer leading to a significant delay. It is shown in Fig. 1(a) that the customer requires the Lane Keep Assist (LKA) feature, but has to be satisfied with a combination of LKA and Blind Spot Detection (BSD) features that is offered by the dealership. The more advanced alternative shifts the deployment of an on-demand safety feature from a cloud server, whereby the customer can request for a new feature remotely. The cloud server would then determine the feasibility of deploying this new feature considering the configuration of the vehicle platform. If the requested safety feature is found to be feasible for deployment, then the vehicle is configured with the new feature by downloading the feature code and requirements from the cloud. This scenario is shown in Fig. 1(b). This new alternative allows the customer to configure only the LKA feature when required. However, this alternative would require a support infrastructure that can enable a safe deployment of the on-demand feature such that the new feature does not interfere with the already running system adversely. In effect, the new alternative is plug and play (PnP) of automotive safety features.

In this paper, we present our vision of platform-based PnP of automotive safety features and discuss the areas that pose significant challenges in the realization of this new paradigm. The platform infrastructure realizing the new paradigm will be composed of hardware (sensors, actuators, electronic control units (ECUs) and communication networks) and software resources required for basic operations such as engine control, braking, steering, etc.
The main challenges we present in this paper are:

1) *developing a specification format, i.e., how can we specify features and the platform architecture taking into consideration the possibility of future additions of new system properties and resource types*
2) *determining safety of deploying on-demand features in terms of platform compatibility and timing schedulability*
3) *ensuring security constraint satisfaction*
4) *effiicent runtime management, i.e., providing adequate resource budgets at runtime*

Further, we also discuss possible solution approaches corresponding to the existing challenges.

The relevance of this PnP paradigm looks promising in the automotive industry as is observed in the car sharing business model. Several companies have launched different car sharing programs [1]. Leveraging the car sharing pro-

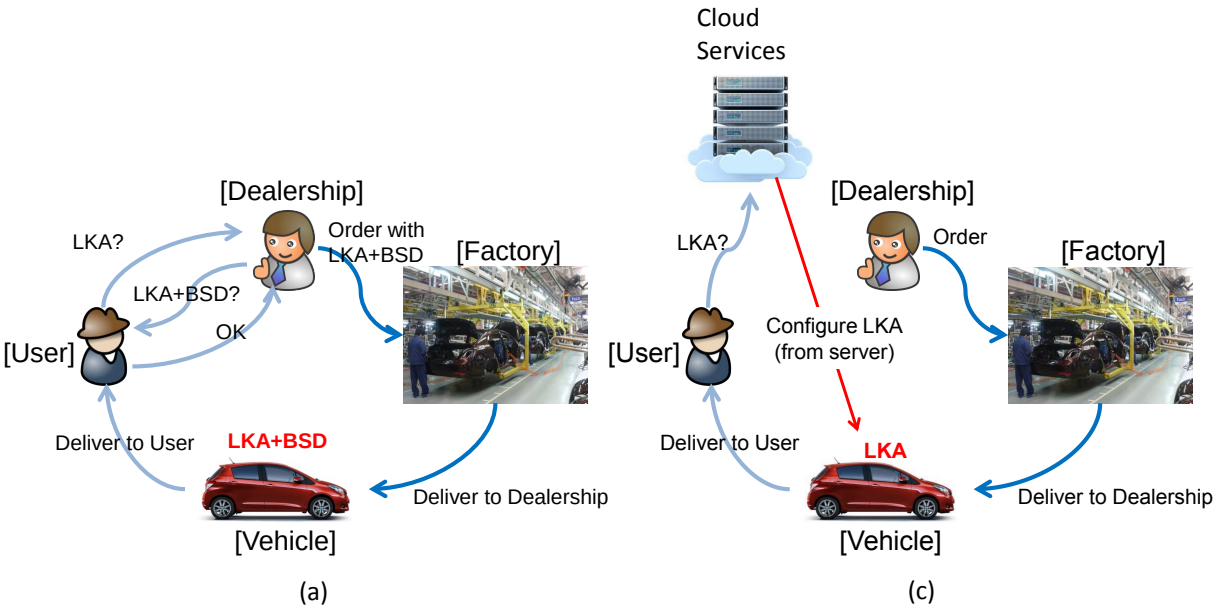

Figure 1: Illustration of Plug and Play in Automotives: (a) current ordering, manufacturing and delivering process, (b) on-demand deployment from the cloud server.

grams, the users can select a car brand and type of their choice and then may want to select more or less safety features depending on their driving skills and experience. The proposed platform-based infrastructure can play a role in allowing users to select the required safety features thereby providing a more personalized driving experience.

The rest of this paper is organized as follows. Section II presents the static (development time) and dynamic (run time) views of the platform architecture. Section III introduces the language needed to specify requirements of a feature that the platform needs to satisfy in order to safely deploy the feature. Section refsec:safetyassurance describes deployment-time checks performed by the platform to ensure that these requirements are satisfied. Section V introduces an approach to perform checks in a compositional manner. Section VI briefly introduces the problem of runtime resource management for dynamically installed features. Finally, Section VII concludes the paper.

II. Platform Architecture

In this section, we present the important building blocks that need to be developed in order to realize the paradigm of platform-based PnP of automotive safety features. The platform architecture can be described from two perspectives - the platform as seen by a feature and the internal platform perspective with concrete resources.

The high-level architecture of such an automotive platform from the perspective of a feature is shown in Fig. 2. Each on-demand feature sees virtual resources, i.e., abstract sensors, virtual ECUs, virtual network and abstract actuators. The feature gets access to these resources at runtime through a runtime service API interface. The on-demand feature is agnostic to the distributed resource manager and the deployment APIs.

The concrete platform architecture is shown in Fig. 3. This concrete platform view encompasses concrete resources clearly depicting what types and number of each resources exist on the platform. Given the concrete instance of the resources, an admission controller determines feasibility of deploying the feature on the platform which may already be running other safety features. The platform is also equipped with a distributed resource manager, which provides virtualization of the physical platform resources, such as ECUs, sensors and actuators. It also ensures that the sensor to task, task to task and task to actuator delays satisfy the timing requirements specified by the feature developer. A communication module/layer provides ability to retrieve an on-demand feature from the cloud server. A middleware layer will provide functionalities such as providing an interface to access the services of a feature, managing the installation, update and uninstallation of the feature on the platform, etc.

There are two possible phases of operation provided by the platform architecture to each feature - *deployment phase* and *runtime phase*. In the deployment phase, the requirements of a new feature are checked to analyze if it can be serviced with required resources along with already deployed features. Once a feature has been deployed, the runtime goal of the platform is to ensure that sufficient resources are provided to the feature at each and every instant when the feature executes on the platform.

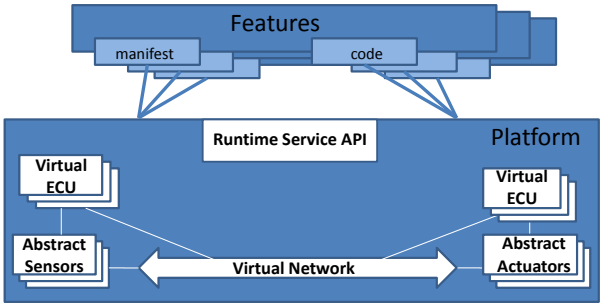

Figure 2: Platform Architecture with virtual resources as seen by an on-demand feature

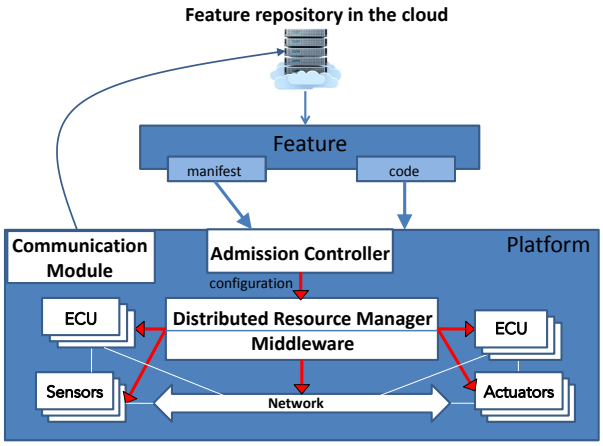

Figure 3: Platform Architecture Implementation with concrete resources

To deploy an on-demand feature, the deployment interface in the middleware would need the details of how many ECUs, sensors and actuators need to be allocated for the feature. In addition, it would also need information about what is the bandwidth allocated for the feature on each of the allocated resources. The deployment interface must also include functions to place a request to the cloud server, which will then initiate the admission controller to check the feasibility of the required feature and present the information regarding resource requirements back to the interface. The runtime service API provides an interface to execute the code of the on-demand feature with the required resource bandwidths at all times.

The feasibility of deploying a feature on the platform needs to be checked at the cloud server before the feature is actually deployed on the platform. The feasibility check may involve checks for timing schedulability, platform compatibility and verification of security requirements. The timing schedulability check will verify that the timing requirements in executing all the features are satisfied given the platform architecture, mapping of tasks to resources in the platform and the scheduling policies on the resources. The check for platform compatibility can include complex verifications to ensure that the required virtualized resources are available on the platform. This will require an extensible interface specification for the resources such that varying characteristics of different types of a resource can be specified using a general specification part and a resource specific specification part. We will discuss about this more while discussing the feature specification language. In addition, certain security requirements will be verified which ensures that there is no leakage of information from a security critical task/feature to a malicious task/feature. In the next section, we discuss the potential feature manifest specification, which will be required to test the feasibility of deploying the on-demand feature on the platform.

III. Feature Specification

In order to perform a feasibility check at the cloud server, a feature specification format/language is required that will specify the resource and timing requirements of the on-demand feature. This specification is in accordance with the platform view (shown in Fig. 2) that is visible to a feature. Each feature is assumed to be built from a set of tasks/runnables that communicate with each other and also access the sensors and actuators. In order to deploy the features, it is essential to first specify their requirements. The requirements invariably include the resource requirements, temporal constraints and security constraints that need to be adhered to.

The feature manifest specifies certain requirements and characteristics of the feature:

1) a set of tasks and information flow between these tasks
2) workload requirement for each task
3) resources accessed by each task (for e.g., the number and type of sensors and actuators accessed)
4) timing requirements for task execution
5) criticality level of the feature

Challenges: One of the important characteristics that the specification language must exhibit is *extensibility*. The specification language must be extensible in two dimensions. In PnP automotive systems, there can be a large number of system properties that need to be considered while ensuring a safe system. However, the list of properties keeps getting updated and new properties may need to be included later. For example, in-vehicle communication security was not considered earlier as a design parameter, while it is increasingly becoming an important design parameter. Therefore, the language must be flexible enough to accommodate new properties with simple additions to the specification.

The second dimension is that the specification language should also have the flexibility to specify different classes of the same resource. For example, a camera sensor can have some basic parameters, but there can be variations in the camera sensors based on the rate at which the device captures the image, the resolution of the image captured, etc.

Hence, it is essential that the specification language takes into consideration the basic parameters and the variation in parameters so that any new resource variation can be accommodated with ease.

Solution Approach: The feature specification language must be capable to describe the architecture of the platform, connections between the different resources and the timing parameters concerning the usage of the resources. It must also describe the feature details such as the constituent tasks and their dependencies along with the workload imposed by each task. These details can be captured by any architecture description language (ADL). A comprehensive description of the commonly used ADLs has been provided in a survey [2]. Although, many ADL frameworks are well developed and have necessary analysis tools, they have not been designed keeping in mind the required PnP nature of the system. Recently, a domain specific language based on Scala was developed for on-demand medical systems [3]. As there are some similar concerns for system safety between on-demand medical systems and PnP automotive platforms, we have used the specification language presented in [3] for feasibility analysis of platform-based PnP of automotive safety features. There are additional safety concerns and resource characteristics that need to be considered in the automotives, which are different from on-demand medical systems. Some of these aspects have been added to the language presented in [3] and we foresee that more will be added in the future. Our specification format for PnP of automotive safety features was extensively discussed in [4].

We are currently working on making the specification format extensible in the two dimensions mentioned earlier.

IV. Admission Control

Admission control is an important step in the correct deployment of a new feature in the platform. This is shown in the concrete platform view in Fig. 3. There are several reasons that may cause rejection of features from executing on the platform. Each platform instance has a different set of resources and may not necessarily have all the resources required by a feature. For example, some required sensors may not be available on a platform instance. It may also happen that the ECU does not have sufficient capacity to satisfy the workload requirement of the feature due to already deployed features. This is where admission control is important to prevent the incorrect deployment of an on-demand feature. In our context, admission control broadly consists of checking for safety and security assurance in order to admit a feature to run on the platform. Therefore, we first present a detailed description of what safety and security assurance are and the challenges posed by them. In the next section, we present a potential solution to check for safety assurance, which is based on compositional analysis.

A. Safety Assurance

This section discusses the challenges and issues concerning the assurance of the safety of PnP applications (apps) in terms of the resource virtualization. Prior to installation and execution of such apps, the automotive PnP apps must be assured to guarantee and contribute to the driving safety and vehicle security.

The PnP automotive application in this paper is a safety and security-concerned software system which performs a safety functionality (feature), e.g. LKA and BSD, that improves vehicle safety. The safety of vehicle systems in ISO 26262 [5] is classified by Safety Integrity Level (SIL).

The safety integrity of the automotive PnP apps also conforms to SIL requirements which requires a satisfiable integrity degree of safety analysis of apps installed on vehicles. The safety analysis time of automotive apps depends on a SIL that an app has to meet; the highest SIL demands rigorous and exhaustive analysis, such as formal analysis techniques, which demands relatively more time and analysis resources that lower SIL apps do. Thus, the first challenge to guarantee the safety of the automotive PnP apps is analysis time. Since automotive PnP apps need to be deployed within a relatively limited time since the customer requests any app to install, the analysis of the apps has more constraints upon its verification time and resources than the automotive apps installed at manufacturing time.

Second, distributed resources, e.g. sensors, actuators, motors, in the PnP environment are quite often shared to satisfy different purposes while the resources have been dedicated to a particular control. Hence, the arbitration of resources need to be more sophisticated and well-tuned than ever.

1) Platform Compatibility Analysis: Static availability constraints mandate that the checking algorithm verifies that the virtual resource specified in the manifest has a matching physical resource in the platform instance. It does not depend on the other features deployed in the system. This resource matching is termed as compatibility of the feature resource requirements with the available resources on the platform. The compatibility analysis can range from checks for simple resource matches to checks for more detailed resource matches.

Typically, a check for a simple resource match would involve a check for the particular resource type that the feature needs. For e.g., if a safety feature needs a ECU, checking for the availability of ECU on the platform is a simple check. However, a more detailed check would involve a check to find the exact resource if the requirements specify multiple other properties of the resource. For e.g., checking for a camera sensor that satisfies the data rate requirements, resolution and other properties is a more detailed check as compared to just checking for the availability of a camera.S

2) Resource Contention Analysis: There are several techniques from literature that may be exploited to perform timing analysis of the new feature with the already deployed

features. It is important here that the timing analysis must be done taking into consideration the safety criticality of the features. There are several admission control techniques for different scheduling algorithms in literature. These techniques are based on schedulability analysis concepts from the real-time scheduling domain. Several admission control algorithms have been developed for rate monotic (RM), deadline monotonic (DM) and earliest deadline first (EDF) scheduling strategy.

In the seminal paper by Liu and Layland [6], they proposed a utilization bound for a set of tasks scheduled by rate-monotonic scheduling. If the cumulative utilization of all the tasks along with the new task is below the proposed utilization bound, then the new task can safely be admitted into the system. Although this type of admission control is computationally simple, it only gives a sufficient but not necessary condition for schedulability. Therefore, the processor utilization achieved using this bound may be lower leading to under utilization of resources. On the other hand, there are works in literature that propose exact schedulability tests [7], [8], [9], which give necessary and sufficient conditions. However, they are not suitable for admission control due to the pseudo-polynomial complexity.

Several algorithms have been proposed to increase the accuracy while maintaining low complexity. One such algorithm was presented in [10], where a load test was proposed for DM tasks with deadlines less than or equal to periods. The proposed method computes an upper bound on the worst-case response times (WCRT) of tasks taking into consideration the accepted and the arriving task. If the upper bound on WCRT is always less than or equal to the respective deadlines, the new task and the accepted tasks are schedulable under DM and the new task can be admitted. Similar algorithmic improvements for admission control under EDF scheduling policy also has been conducted [11], [12], [13]. Admission control has also been achieved using real-time interfaces [14], [15], whereby the authors propose the formalism of component interfaces, which describe the required resource requirement in terms of demand curves. The admissibility is therefore checked by ensuring that the resource supply curve of the platform instance is sufficient to satisfy the component demand curve.

There hasn't been much work done on efficient admission control algorithms considering safety criticality levels of the incoming and already deployed features. One work [16] exploits the different demands of tasks along with EDF-VD scheduling approach, whereby deadlines of the higher criticality tasks are scaled according to their demand. One future research direction with regards to this area will be to devise new interface concepts to handle safety-critical scenarios in a mixed criticality setting.

B. Security

In todays vehicles, security features must include not just physical access and protection of confidential information, but also critical safety systems such as drive-by-wire braking and steering [17]. In other words, compromising the vehicle safety-critical units can result in the safety failure of the vehicle system. Compromising the security in vehicle systems is possible in two ways: vehicle network system and vehicle applications that potentially are connected to safety-critical vehicle control units.

The vehicle network system is either of vehicle-to-vehicle (V2V) or vehicle-to-cloud (V2C), and the attack to the safety-critical units is possible through those networking systems [18], [19], [20]. The another possibility to attack the safety-critical components is to use a PnP application that intends to compromise and breach the vulnerability of vehicle systems with malicious purposes.

The first challenge is to develop security threat models potential in the emerging automotive communication environment and applications so that appropriate security mechanisms and systems can be developed to address those threatening models.

The second challenge is to overcome the legacy networking resource limit. Currently, the most popularly used internal networking system is CAN (Controller Area Network bus). In order to protect safety-critical software units, such as Engine Control Unit, Transmission Control Units, etc, from malicious access through CAN channel shared with non-safety-critical software components, such as infotainment systems, one of popular ways to bring the security to CAN is the entity authentication between communication nodes using MAC (Message Authentication Code) [21], [22]. Since CAN is originally not designed to account for the security problem [23], some of data bits for data transmission need to be sacrificed for the authentication; The more secured channel needs the more data bits, i.e. the data bits of CAN used for data transmission should be scarified to guarantee the highly secured channel. For this reason, the acquisition of the resource for security protocols is critical to guarantee the security of PnP apps. The literature [21], [22] proposes a resource configuration method that guarantees a secure communication using MAC upon CAN data payload. However, it needs to take into account a dynamic security resource allocation for PnP app environment.

To overcome the bandwidth limitation of CAN, a new version of CAN, CAN Flexible Data rate (CAN FD), that supports up to 64 Bytes payload instead of 8 Byte is proposed by ISO 11898-7. However, it requires a new design of vehicle software communication and the update of the legacy software codes of the manufacturer.

The third challenge is that the security analysis integrity of automotive software has not been officially regulated by any regulation and standard organization. In principle, it is

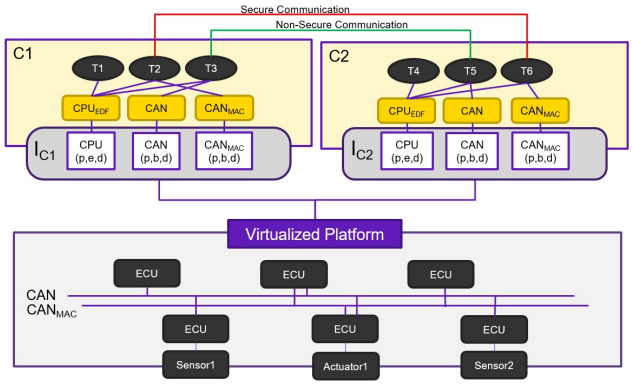

Figure 4: Analysis framework of platform-based design

logical that the security level of automotive software should depend on the safety integrity level because the safety of automotive systems is the most significant property and the security of an automotive software system needs to satisfy the corresponding safety integrity level required of the software system.

In short, the automotive PnP apps should challenge to satisfy not only a safety degree corresponding to a given SIL but also to be deployed within a relatively limited analysis time. Furthermore, it should challenge to overcome the limit of the resources that support security protocols in the current automotive platforms.

V. Compositional Analysis Approach

The platform-based approach views the safety from a resource constraint perspective. We assume that the safety is failed when execution resources, i.e. CPU, for apps are not sufficient.

The safety analysis of applications in this framework is performed in two steps: platform-independent analysis with virtual resources, and platform-specific analysis with concrete resources. The first analysis step is to figure out the *optimal (minimal)* resource requirements that enable an app to fulfill both functional and non-functional requirements. The purpose of this step is to quantify necessary resources of apps independently from platforms. This step is done by app developers. The second analysis step checks if a given platform providing actual resources supports the virtual resource requirements from apps. This second analysis step is a part of the admission control in the platform-based approach.

Figure 4 shows an example of an app. It consists of two components $(C1, C2)$ of which each is composed of 3 tasks. Each task has executable programs of the target functionality. In $C1$ component, every task is assigned to a CPU which is scheduled by EDF (Earliest Deadline First). $T2$ uses CAN_{MAC} for secure communication while $T3$ uses a normal CAN CAN. Each task requires a CPU to execute programs, a CAN (Controller Area Network) to support non-secure communication, a secured CAN supported by MAC (Message Authentication Code) for secure communications. If a task is periodic, it is characterized by real-time parameters, such as a period (p), an (worst-case) execution time (e), and a deadline (d). For the communication with tasks in other components, a task may require a CAN. Its requirement can be specified in a similar way with the CPU requirement using real-time parameters, such as a period (p), the (minimum) transmission bit (b), and a deadline (d). The specification of a CAN (p, b, d) means that the task requires b bits bandwidth every p time units with the deadline d. In order to make a secure communication, a task may require a secure channel, i.e. CAN supported by MAC, with the same real-time parameters as the CAN not supported by MAC.

A set of tasks is encapsulated by a component, which is responsible for controlling tasks with specific scheduling mechanisms for such shared resource as CPU and CAN. In principle, individual resource requirements of each task in a component is abstracted into a collective resource requirements specification. In Figure 4, the component $C1$ specifies its collective CPU requirements that satisfy its encompassing tasks, $T1$, $T2$, and $T3$, on a CPU resource specification. In addition, $C1$ exposes similar collective resource requirements for CAN and CANs supported by MAC. The CAN is used by $T2$, and the CAN supported by MAC for secure communication is used by $T3$. The component $C2$ exposes similar resource requirements. The specification of those collective resource requirements for CPU, CAN and CAN supported MAC is called component interface. For instance, I_{C1} and I_{C2} are component interfaces for $C1$ and $C2$.

Prior to analysis of apps without a given platform, A component interface that satisfies the functional and non-functional requirements of apps is firstly computed, and that is called *virtual configuration* in the platform-based framework. In order to satisfy functional and non-functional requirements, real-time parameters of each tasks of an app are computed in such a way that they require the minimal amount of resources. This computation accounts not only for task's real-time properties but also for scheduling (distribution) mechanisms for sharing resources.

Next, the compatibility of component interfaces is checked against a given specific platform to see if a new app can achieve its functional and non-functional requirements under platform constraints. In next section, we discuss an approach to the virtual configuration check and the compatibility check.

A. Compositional Framework

The compositional framework for hierarchical scheduling systems [24], [15], [25] presents the fundamentals and the underlying techniques for the computation of component interfaces and the compatibility check.

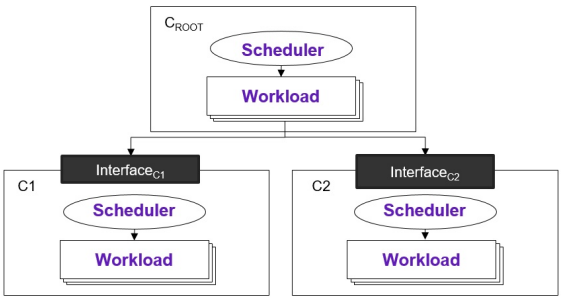

Figure 5: Compositional framework for hierarchical scheduling systems

In the compositional framework, resource requirements of real-time tasks are also introduced by an interface, and a supply pattern of resources available to tasks is represented by a resource model. The schedulability, one of compatibility checks, is verified by checking if an interface of a component is satisfied by a resource model, i.e. the quantity of resources demanded by every task in a component is always be provided by a resource model.

A component in a hierarchical scheduling systems as shown in Figure 5 consists of a set of workloads (tasks) and a scheduling mechanism, such as EDF (Earliest Deadline First) and RM (Rate Monotonic). The scheduling system is constructed in a hierarchical manner such that a workload of components may become a component which also consists in a set of workloads and a scheduling mechanism. Thus, either a workload of a component can be viewed as a task controlled by the component, or it can be viewed as a component including another set of workloads. Individual components, except the root component, and workloads are given in a requirements specification, called interface. For instance, a periodic interface (resource) model [24] specifies a periodic resource request, such as the interface (p, e) that requires the amount e of resources every p time units. Both interfaces of workload and component share the same interface such that the interface of child level components can be analyzed as one of workloads at the parent level component. In order to construct an optimal hierarchical scheduling system, the construction of a hierarchical scheduling system is done in a bottom-up way that the terminal workloads are grouped into a component and their optimal resource requirements are computed as the interface of the component, which is regarded as a workload at its parent level and grouped again into another component.

B. Challenges in Adopting Compositional Framework to Platform-based Approach

The compositional framework can be adopted by the platform-based approach for the analysis of PnP apps. As mentioned in Section V, the functionality of an app are composed of one or more components, and each component requires the resource requirements using an interface. The interfaces of an app would be provided to a concrete platform so that it is checked whether or not the interface can be supported by the platform.

However, the classical compositional framework presents some open issues. First, the interface of components of automotive PnP apps should account for multi-typed resources. The classical compositional framework accounts for a single type resource, e.g. single core or multi-core CPU. However, automotive PnP apps requires multi-typed resources, such as CPU, CAN, sensors and actuators.

Second, the timing modality of resource requests might not be regular. The classical compositional framework handles a regular resource request pattern, such as periodic resource model, where a task requests the same amount of resources every specific period. However, the PnP apps might have to consider a more dynamic and divergent resource request pattern of tasks, which is beyond the analysis scope of the classical analytic methods.

Third, the classical interface of the compositional framework presents inherent pessimism in sharing resources. The interface of components is a collective resource requirements specification abstracting all resource requirements of individual workload in a component. Hence, such an abstraction presents an inherent pessimism, i.e. the quantity of resources required by an interface is more than the workloads of a component actually require.

To address to the issues above, the platform-based safety analysis needs a more flexible and sophisticated verification technique. The next section discusses alternative methods that support rigorous analysis of complicated and divergent automotive PnP apps.

C. Model-based Compositional Approach to PnP App Analysis

Formal methods are analysis techniques recommended by ISO 26262 [5] to assure the highest safety integrity of automotive E/E software systems. Model checking in formal methods is an exhaustive analysis method that checks whether or not a given model of the system satisfies a given property by exploring all the states of a system model. The model checking is very attractive since it is so exhaustive that its analysis is 100% certain with respect to a given property even though it has the state-explosion issue. Moreover the analysis procedure does not need any human effort, i.e. it is fully automatic. For this reason, the international standards recommends model checking as a verification alternative for the highest safety integrity of automotive software system.

The model checking approach to compositional systems has been studied for many applications, such as [26], [27]. The idea behind the compositional model checking is as follows (Figure 6): Given two components $P1$ and $P2$, if the interface $A1$ of $P1$ is a projection of $P1$ to the component $P2$, the verification of the composition of $P1$ and $P2$ is

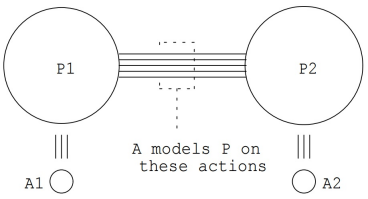

Figure 6: Interface rule [26]

possible by checking the composition of the interface $A1$ and $P2$. If $P1$ and $P2$ interact with each other, the interface $A1$ ($A2$) represents $P1$'s ($P2$'s) behavior projected to $P2$ ($P1$). This approach reduces the searching space of the states for model checker to explore and increases the chance to return a result within linear time. The compositional approach to real-time systems can be found from [28], [29], [30]. The approach to scheduling systems are founded from [31], [32]; The key idea behind them is to abstract a resource supply from the system as an interface model and check the schedulability of the workloads of a component against the interface model. For instance, in Figure 5, the interfaces $Interface_{C1}$ and $Interface_{C2}$ are resource requirement specifications demanded by child components consisting. Instead of modeling individuals of all components, each interface is modeled as a resource model that supplies resources according to its specification. In the parent level, the interface is checked in such a way that the workload that inherits real-time parameters from the interface is checked by the configuration of the parent level. In this way, individual components are compositionally checked with respect to the other components using the interface (resource) model.

The model checking is known to be flexible enough to be capable of handling sophisticated real-time settings which are divergent from classical real-time parameters of the classical frameworks. In particular, [33] presents a more flexible schedulability analysis framework, in which the scheduling component includes probabilistic tasks whose real-time parameters are dependent on various probabilistic distributions. [32] presents a resource model that is divergent from the classical regular resource supply patterns. Hence, the model checking can be an alternative that complements the flexibility of the classical analytic methods.

VI. Runtime Resource Management

Runtime resource management (RRM) is another key area that poses several challenges in realizing the paradigm of our PnP platform for automotive safety features. Once a feature is found to be admissible, the first primary goal of RRM should be to provide the features with adequate budgets on each of the resources they require no matter what is the amount of workload from other features. It is essential that the resource allocation at runtime is done within a bounded time and using as less resources as possible. Further, the resource allocation policy should be able to handle the dynamic scenario in a PnP platform, i.e., the installation and uninstallation of features.

There are several works in literature [34], [35], which look at the problem of runtime resource management in various types of system. Currently, we are evaluating these approaches so that the necessary aspects can be identified in runtime resource allocation, which need to be addressed for our PnP automotive platform.

VII. Conclusion

In this paper, we presented a vision of platform-based plug and play of on-demand safety features in automobiles. A prototype architecture of the envisioned platform was described briefly with functionalities of the constituent blocks. Further, we presented the key areas which pose challenges to realizing such a platform architecture. Specifically, the three main challenging areas discussed included specification approach for such a platform and features, approaches to ensure safety of the system in terms of platform compatibility and schedulability of features, and ensuring security.

Acknowledgment

This work, carried out at the University of Pennsylvania, has been supported in part by the NSF grant CNS-1329984 and a gift from Toyota ITC.

References

[1] A. Cornet, D. Mohr, F. Weig, B. Zerlin, and H. Arnt-Philipp, "Mobility of the future - opportunities for automotive oems," *Advanced Industries*, 2012.

[2] P. C. Clements, "A survey of architecture description languages," in *8th International Workshop on Software Specification and Design*. IEEE Computer Society, 1996, p. 16.

[3] A. L. King, L. Feng, O. Sokolsky, and I. Lee, "Assuring the safety of on-demand medical cyber-physical systems."

[4] D. Gangadharan, O. Sokolsky, I. Lee, B. Kim, C.-W. Lin, and S. Shiraishi, "Platform-based automotive safety features," in *SAE Technical Paper*, 2016.

[5] "ISO/DIS 26262-1 - Road vehicles Functional safety Part 1 Glossary," Geneva, Switzerland, Tech. Rep., Jul. 2009.

[6] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[7] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Real Time Systems Symposium*. IEEE, 1989, pp. 166–171.

[8] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993.

[9] E. Bini and G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1462–1473, 2004.

[10] A. Masrur, S. Chakraborty, and G. Färber, "Constant-time admission control for deadline monotonic tasks," in *Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 220–225.

[11] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, *Deadline scheduling for real-time systems: EDF and related algorithms*. Springer Science & Business Media, 2012, vol. 460.

[12] U. C. Devi, "An improved schedulability test for uniprocessor periodic task systems," in *Euromicro Conference on Real-Time Systems*. IEEE, 2003, pp. 23–30.

[13] A. Masrur, S. Chakraborty, and G. Färber, "Constant-time admission control for partitioned edf," in *Euromicro Conference on Real-Time Systems*. IEEE, 2010, pp. 34–43.

[14] E. Wandeler and L. Thiele, "Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling," in *International Conference on Embedded software*. ACM, 2005, pp. 80–89.

[15] I. Shin and I. Lee, "Compositional real-time scheduling framework," in *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*. IEEE, 2004, pp. 57–67.

[16] A. Masrur, D. Mueller, and M. Werner, "Bi-level deadline scaling for admission control in mixed-criticality systems," in *International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 2015.

[17] R. Soja, "Automotive security: From standards to implementation," *Freescale White Paper, Document Number: AUTOSECURITYWP REV*, vol. 1, 2014.

[18] A. Aijaz, B. Bochow, F. Dötzer, A. Festag, M. Gerlach, R. Kroh, and T. Leinmüller, "Attacks on inter vehicle communication systems-an analysis," *Proc. WIT*, pp. 189–194, 2006.

[19] J. Blum and A. Eskandarian, "The threat of intelligent collisions," *IT Professional*, vol. 6, no. 1, pp. 24–29, Jan 2004.

[20] M. Amoozadeh, A. Raghuramu, C. n. Chuah, D. Ghosal, H. M. Zhang, J. Rowe, and K. Levitt, "Security vulnerabilities of connected vehicle streams and their impact on cooperative driving," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 126–132, June 2015.

[21] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive can networks–practical examples and selected short-term countermeasures," in *Computer Safety, Reliability, and Security*. Springer, 2008, pp. 235–248.

[22] C.-W. Lin, B. Zheng, Q. Zhu, and A. Sangiovanni-Vincentelli, "Security-aware design methodology and optimization for automotive systems," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 21, no. 1, p. 18, 2015.

[23] C. W. Lin, Q. Zhu, C. Phung, and A. Sangiovanni-Vincentelli, "Security-aware mapping for can-based real-time distributed automotive systems," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2013, pp. 115–121.

[24] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *RTSS*. IEEE Computer Society, 2003, pp. 2–13.

[25] ——, "Compositional real-time scheduling framework with periodic model," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, p. 30, 2008.

[26] E. M. Clarke, D. E. Long, and K. L. McMillan, "Compositional model checking," in *Logic in Computer Science, 1989. LICS'89, Proceedings., Fourth Annual Symposium on*. IEEE, 1989, pp. 353–362.

[27] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems," *Information and computation*, vol. 111, no. 2, pp. 193–244, 1994.

[28] F. Laroussinie and K. G. Larsen, *Compositional model checking of real time systems*. Springer, 1995.

[29] H. Garavel, F. Lang, and R. Mateescu, "Compositional verification of asynchronous concurrent systems using cadp," *Acta Informatica*, vol. 52, no. 4-5, pp. 337–392, 2015.

[30] Y. Meller, O. Grumberg, and S. Shoham, "A framework for compositional verification of multi-valued systems via abstraction-refinement," *Information and Computation*, 2016.

[31] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou, "Hierarchical scheduling framework based on compositional analysis using uppaal," in *Formal Aspects of Component Software*. Springer, 2013, pp. 61–78.

[32] ——, "Widening the schedulability of hierarchical scheduling systems," in *Formal Aspects of Component Software*. Springer, 2014, pp. 209–227.

[33] A. Boudjadar, J. H. Kim, A. David, K. G. Larsen, M. Mikucionis, U. Nyman, A. Skou, I. Lee, and L. T. X. Phan, "Flexible framework for statistical schedulability analysis of probabilistic sporadic tasks," in *IEEE 18th International Symposium on Real-Time Distributed Computing, ISORC 2015, Auckland, New Zealand, 13-17 April, 2015*. IEEE Computer Society, 2015, pp. 74–83. [Online]. Available: http://dx.doi.org/10.1109/ISORC.2015.21

[34] G. C. Durelli, M. Pogliani, A. Miele, C. Plessl, H. Riebler, M. D. Santambrogio, G. Vaz, and C. Bolchini, "Runtime resource management in heterogeneous system architectures: The save approach," in *IEEE International Symposium on Parallel and Distributed Processing with Applications*. IEEE, 2014, pp. 142–149.

[35] J. M. Borrmann, F. Haxel, A. Viehl, O. Bringmann, and W. Rosenstiel, "Safe and efficient runtime resource management in heterogeneous systems for automated driving," in *18th IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2015, pp. 353–360.