An Android-based Automotive Middleware Architecture for Plug-and-Play of Applications

Jong-Woon Yoo, Youngwoo Lee, Daesun Kim, Kyoungsoo Park
Department of Electrical Engineering
KAIST
Daejeon, Korea
jwyoo@core.kaist.ac.kr, ywlee@ndsl.kaist.edu, sundae21@kaist.ac.kr, kyoungsoo@ee.kaist.ac.kr

Abstract— **Modern vehicles are getting increasingly smarter and evolving to become new mobile computing platforms. As the software platformization of vehicles advances, the demands for reconfigurable cars, installing and upgrading vehicle software as plug-and-play have also grown. However, the existing automotive software platforms are not designed for dynamic reconfiguration and plug-and-play. This paper addresses the plug-and-play challenge in the recent automotive systems. Drawing from recent advancements in the mobile phone operating systems, we propose an Android-based software architecture which supports the play-and-play in automotive systems. In our architecture, an application can be downloaded from cloud services and executed on ECUs in a distributed fashion. Before installing an application, our system performs schedulability analysis to check real-time performance requirements for recent vehicles. We also present a model-based development tool for designing a workflow of automotive applications for our system. We implement a prototype and demonstrate the feasibility of our approach.**

Keywords-automotive; plug-and-play; middleware; workflow; software platform

I. INTRODUCTION

Modern vehicles are getting increasingly smarter and evolving to become a mobile computing platform. Several tens of Electronic Control Units (ECUs) are already embedded into today's vehicles, forming a distributed network to control various functional components of a vehicle, from operating a windshield wiper to programming automatic parking. In Consumer Electronics Show (CES) in 2012 [1], General Motors report their plan that they will programmatically export the real-time position data of a vehicle through the OnStar service, which detect and avoid car crashes by sharing the position data through cloud computing services. Recent trend shows that more programmable functionalities are expected to be available in the vehicles in the near future.

As the software platformization of vehicles advances, the demands for reconfigurable cars, upgrading software versions, or installing new applications as plug-and-play have also grown. However, the existing automotive software platforms in use are not designed for such dynamic reconfiguration and plug-and-play. AUTOSAR [2] is a de-facto software architecture standard for automotive E/E (Electrics/Electronics) systems as it allows reducing the application development efforts by separating the application layer from the underlying infrastructure, increasing the reusability of existing functional modules. In AUTOSAR, however, the system software like an operating system and runtime environment of an ECU is statically built into the applications that run on the ECU. This necessitates the rebuild of the entire software stack each time we upgrade or install a new application.

This paper addresses the plug-and-play challenge in the current automotive systems. We propose an Android-based automotive software architecture which supports the play-and-play of applications. In our approach, an in-vehicle ECU network consists of one master ECU and several slave ECUs. We adopt Android for the master ECU to enable developers to use well-defined development environment for writing vehicle applications running on the master ECU. The master ECU provides the access to external networks, allowing users to download new applications from cloud computing services. The master ECU collects automotive data such as velocity and gauge level by communicating with slave ECUs. The Android OS running on the master includes a library containing APIs for accessing those automotive data, allowing automotive applications, such as a dash board, can be implemented and run on the master ECU. To support plug-and-play, we add a specialized module, called Automotive Manager (AM), to the Android application framework. When a new application is installed, the AM analyzes real-time schedulability of the newly-installed component, and identifies whether the new component will meet the real-time constraints required by other applications or not. The AM then decomposes the application into several tasks and distributes them to slave ECUs that run an real-time operating system. We have built a runtime environment and application manager on the slave's real-time OS for plug-and-play support. We have developed a model-based application development tool to help developers easily write new applications for our environment. We demonstrate the feasibility of our approach by implementing a prototype using a small four-wheel motor car with embedded boards and plug-and-playing a steering application on that.

II. RELATED WORK

There have been several approaches that apply smart phone software platforms, such as Android, to vehicles that provide the car drivers with more flexible environments for infotainment [3], telematics [4], or Human-Vehicle Interaction (HVI) [5]. These approaches allow the drivers to download and

install vehicle applications from cloud computing services. One unique feature that distinguishes the vehicle applications from today's smartphone applications is that the vehicle application can access driving or vehicle status information, e.g., speed, rpm, or gas gauge level, through the software platform. In [3], for example, Android was modified to have so-called Automotive Manager that provides a programming interface between the vehicle applications and system software. However, dynamic reconfiguration and support for plug-and-play have not been available in these approaches. Tasks like upgrading the automatic parking algorithm or gas utilization policy would require ECU-wide plug-and-play.

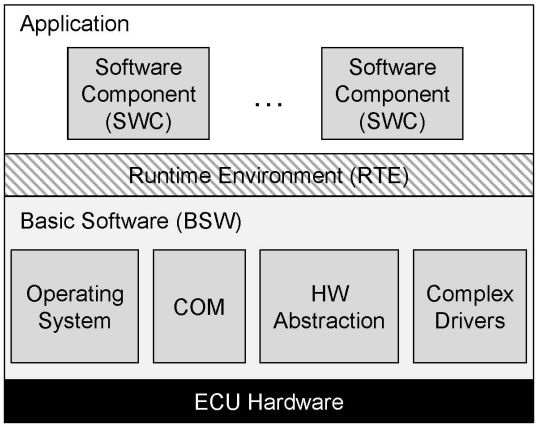


Figure 1. The layered architecture of AUTOSAR

For ECU software platforms, many major automobile manufacturers including BMW, Bosch, DaimerChrysler, and Volkswagen have been working together to develop and improve AUTOSAR[1] [2], an open software architecture standard for automotive E/E (Electrics/Electronics) systems since 2002. AUTOSAR provides a layered software architecture consisting of the hardware, basic software (BSW), runtime environment (RTE), and application layers, as shown in Figure 1. In AUTOSAR, an application is composed of one or more software components (SWCs), atomic functional blocks encapsulating the operation of the application. The SWCs of an application are interconnected and can be distributed over several ECUs. AUTOSAR abstracts the hardware layer such that application developers focus on the core logic of the program without dealing with the hardware details, such as ECU types, the physical location of ECU in the internal car network, and mapping information of SWCs to ECUs. Instead, the developers can assume that all SWCs are connected to a common logical bus, so-called virtual functional bus (VFB), and need to consider only the interconnection among the SWCs. The RTE in each ECU collaboratively implements the VFB so that the designed SWC connections can be realized on an actual in-vehicle infrastructure. The mapping of SWCs to ECUs and the generation of RTE and BSW are automatically done by the AUTOSAR development tool with the provided application design and system descriptions, as shown in Figure 2. This separation between the application and infrastructure plays an important role in increasing the reusability of SWCs, and thus, reducing the development costs and efforts.

[1]AUTomotive Open System ARchitecture

AUTOSAR, however, makes it inherently hard to accommodate the increasing demands for reconfiguring their vehicles, e.g., upgrading the automatic parking system algorithm or gas utilization policy.

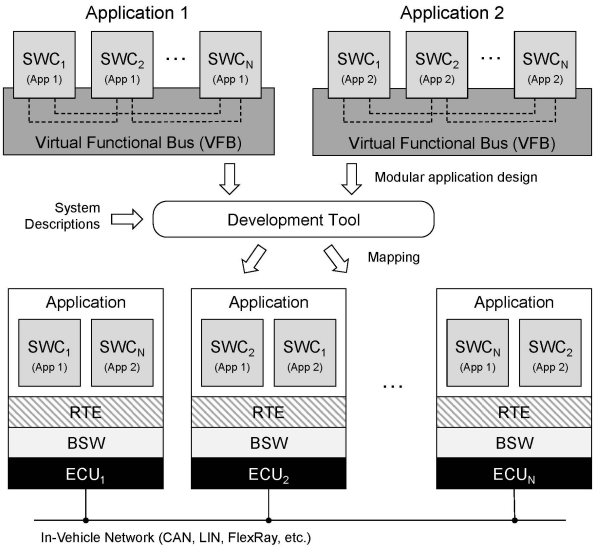


Figure 2. Development process of AUTOSAR

This is because, for each ECU, the entire software stack including from the BSW to application layer is built *statically* into a single executable. Furthermore, the RTE does not include a flexible routing function and SWCs use RTE APIs to directly communicate with other SWCs, whose names are formed by the combination of the API call's functionality, e.g., call or send, that defines the API root name and the service access point which the API operates. This means that we need to change the interconnection among SWCs or add a new SWC requires RTEs of not only corresponding ECU but also of other ECUs having dependency to be rebuilt entirely.

Many researchers have tackled the issue of ECU software reconfiguration, especially for enhancing fault tolerance [6], [7], [8], [9], [10], but, few works have focused on ECU-wide plug-and-play, which we address in our study.

III. THE PROPOSED ARCHITECTURE

This section describes the overall plug-and-play procedure and the proposed automotive middleware architecture in detail. Figure 3 illustrates the entire plug-and-play scenario from application development to ECU mapping. We have developed a model-based development tool, called *Workflow Designer* (WFD), which allows developers to easily configure applications via GUI. For each application, WFD generates an XML file describing the application's features, such as types and interconnections of the exploited software components (SWCs), real-time constraints, hardware resource requirements, and other information needed to install the application to vehicles ①. Similar to AUTOSAR, an SWC is an atomic functional module but is implemented with flexible communication interfaces for plug-and-play support. The details of WD and SWC will be given in Section Ⅲ-A.

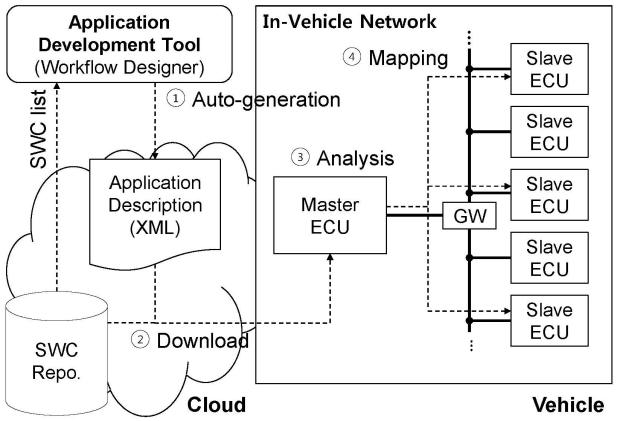


Figure 3. Plug-and-play procedure

Users download applications through cloud services and the master ECU running Android is responsible for application download. It downloads and analyzes the description file (XML) for a desired application ②. The analysis done by the master ECU includes real-time schedulability test, SWC dependency check, resource requirements check, and so on ③. If the analysis decides that the application is compatible to the vehicle, the master selectively downloads the SWCs which are not currently installed in the vehicle. Section Ⅲ-B presents the master ECU and its operations in detail. Finally, the downloaded SWCs are distributed to proper slave ECUs according to the requirements specified in the application description file ④. This process is explained in Section Ⅲ-C.

A. Workflow Designer

Several tools for the development of the AUTOSAR systems are already available. For example, SystemDesk[2] can be used to model the system architecture like the interconnections among SWCs. The behavior of each SWC can be described using Matlab with the Simulink extension[3]. However, because AUTOSAR does not support plug-and-play a new development tool that provides application developers with a convenient GUI development environment for authoring plug-and-play applications is needed. We have developed a web-based application design tool, *Workflow Designer* (WFD), which can be used to model an application structure with GUI-style interactions, e.g., click and drag-and-drop.

Through WFD can we define workflow, which designates the working processes of SWCs running on ECUs. Figure 4 shows a screen shot of WFD. On top of the screen buttons for creating, opening, and saving a workflow are there, and on the bottom left, there is a *SWC list*, which has names of the available SWCs. Double-clicking one of those names of SWCs would lead to a new icon for the selected SWC on *design panel*, which is on the bottom right of the WFD screen.

On design panel, we can organize the workflow by moving or linking some SWC icons, and setting attributes of each SWC. A link from SWC 1 to SWC 2 means that SWC 1 will send its output data to SWC 2, so it is shown in the form of a directed link. In order for SWCs to transmit data among them, the sender and the receiver should use the same data format and type. Therefore, when we select one SWC and try to link it to the other one, the design panel asks us first which communication port to use, where the port designates the transmitting data format and type at once. After choosing a port, design panel shows whether the target SWC under the mouse cursor can handle the data coming through the designated port. If it can, the link will be established. If not, the link will be cancelled. We can also modify some attributes of a SWC by right-clicking the SWC icon. After finishing working on the workflow, and clicking save button, we can finally get an XML file that has all the information on the workflow.

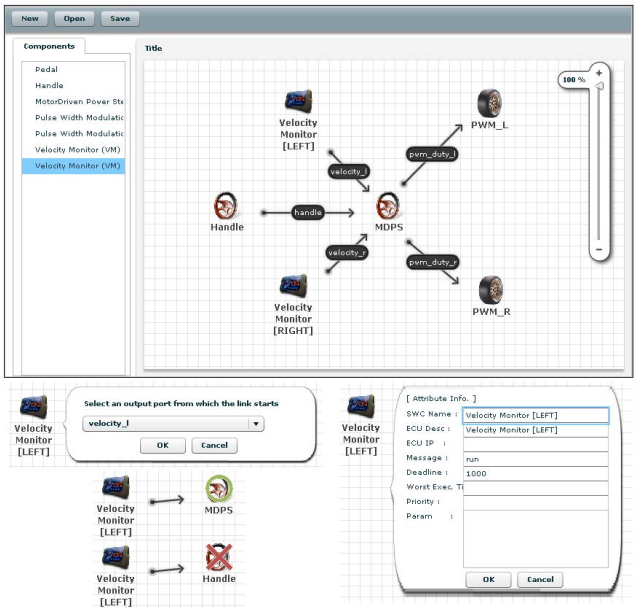


Figure 4. Workflow Designer: A model-based application development tool for our automotive system.

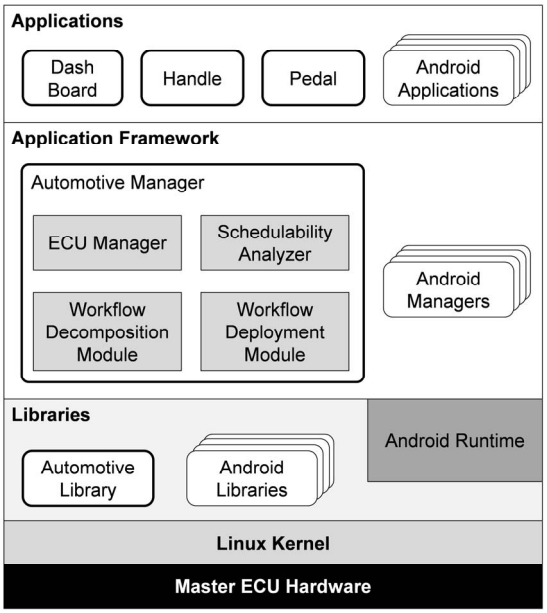
The generated workflow is used by the master ECU to properly deploy SWCs to slave ECUs when an application is installed. SWCs in AUTOSAR communicate via ports provided by the runtime environment (RTE), which include the information of the source/destination pair, message type, priority and so on. However, because RTE is statically built in AUTOSAR, neither relocating an installed SWC from an ECU to another nor installing a new SWC is possible, limiting plug-and-play. In order to address this issue, we propose a software architecture for slave ECUs, which features plug-and-play manager and RTE with routing functionality. Sections Ⅲ-B and Ⅲ-C explain how the generated workflow is used to support plug-and-play by master and slave ECUs, respectively.

B. Master ECU

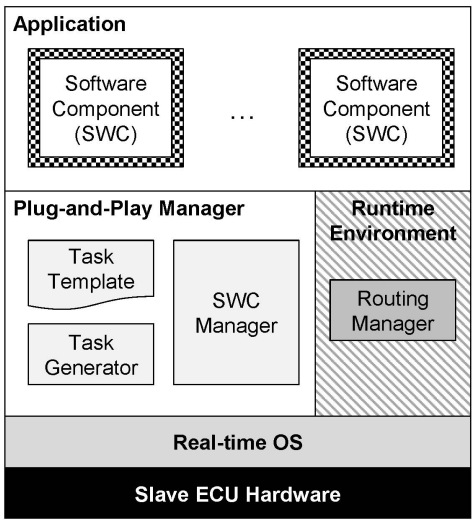
Figure 5(a) shows the overall architecture of master ECU. The master ECU architecture consists of an OS kernel based on Android, with libraries, application frameworks, and applications as the Android architecture. Because an automotive application is installed when the automobile is not operating, the master ECU operating system can be monolithic. There are several reasons for adopting Android to master ECU architecture. First, Android is an open source operating system.

[2]http://www.dspace.de
[3]http://www.mathworks.com/products/simulink

Therefore, we can easily add automotive associated libraries, managers, and applications. Second, Android supports multi languages including XML. In our case, the automotive application is designed in an XML file, thus, supporting XML is an important requirement for master ECU. Third, Android includes powerful and user-friendly APIs and applications.



(a) Master ECU



(b) Slave ECU

Figure 5. The proposed automotive middleware architecture

Master ECU manages installation, deletion, and modification of the automotive application. We implement Automotive Manager on Application Framework of Android to manage the entire automotive applications. Automotive Manager consists of ECU Manager, Schedulability Analyzer, Workflow Decomposition Module, and Workflow Deployment Module. Master ECU must contain all information about automotive applications and ECUs in the automobile for the arrangement of newly-installed applications. Moreover, master ECU should check whether an automobile could operate safely after new application is installed.

Automotive systems have strong real-time requirements that automotive applications must meet. At present, all OSEK-compliant OS provide the schedulability analysis to check whether all jobs can execute within their deadline [11]. Therefore, we provide the module for schedulability analysis, which is Schedulability analyzer. Schedulability analyzer needs three parameters: worst-case execution time, deadline, priority. Among three parameters, schedulability analyzer performs the modified response time test [12], which checks whether whole tasks can meet their deadline in priority assignment. If analyzer guaranteed the application satisfying real-time operation, automotive application could be installed.

The equation of modified response time test calculates the worst-case response time for every task. If we focus on *i*-th task the worst-case response time of the task is as follows:

$$W_i = C_i + \sum_{j:p_j<p_i} C_j < D_i, \quad (1)$$

where W_i, C_i, D_i, and p_i indicate the worst-case response time, the worst-cast execution time, the deadline, and the priority of the *i*-th task. Schedulability analyzer checks that the worst-case response time is smaller than the deadline for all tasks. The worst-case response time of the *i*-th task can be obtained by adding up the worst-case execution time of tasks that have higher priority of the *i*-th task. If the result of schedulability analysis passes, every task is guaranteed to meet the real-time requirement. Our schedulability analyzer has complexity of $O(n^2)$.

In summary of Automotive Manager, ECU Manager contains all ECU lists and SWC lists for each allocated ECU. Schedulability Analyzer provides deployment possibility test. If the test passes, an automotive application can be installed on the automobile. Thus, Workflow Decomposition Module parses and divides automotive applications to SWC for deployment. Finally, Workflow Deployment Module deploys divided SWCs to each arranged ECUs.

Afterward, Workflow Deployment Module distributes task generation message to target ECUs. A task is a set of SWCs that will be processed by a slave ECU. Task generation message contains ID of task, target destination address, priority, and deadline. ID of task is needed because of the collision of same components when several different automotive applications need the same set of SWCs in an ECU.

Android application is already widely used in many fields via Android Libraries. We implement applications in master ECU for monitoring or controlling the automobile. Moreover, we add Automotive Library in master ECU. These Libraries enable to get or set the velocity of the automobile through the network. We discussed detailed usages of the applications in section 4.

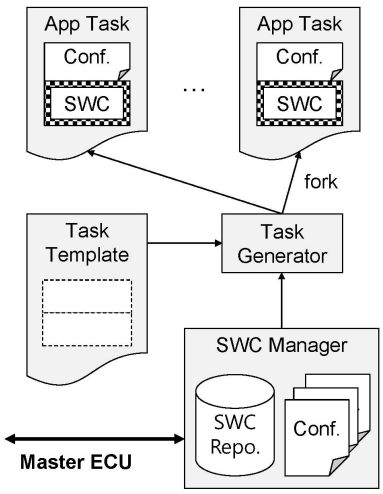


Figure 6. Workflow for the MDPS system

C. *Slave ECU*

The software architecture for a slave ECU is shown in Figure 5(b). It consists of an real-time operating system, plug-and-play manager (PnPM), runtime environment (RTE), and application layers. The real-time OS is responsible for hardware abstraction, communication, and scheduling and running given tasks in real-time. Similar to AUTOSAR, an application consists of several SWCs, which can be distributed over several ECUs to run. The runtime environment (RTE) provides a communication abstraction to SWCs so that they can talk to each other, regardless of their locations in the in-vehicle network. Unlike AUTOSAR, our RTE contains routing functionality to support plug-and-play. The routing information is managed by PnPM which communicates with the master ECU to download SWCs supposed to run on the same ECU. When a new application is being installed, the mapping between the application's SWCs and ECUs is given to PnPMs by the master ECU. Then, each PnPM analyzes the mapping and renews the RTE's routing information.

In AUTOSAR, a SWC can communicate only with pre-determined tasks as its destination address and port are not reconfigurable, limiting plug-and-play and reuse of installed SWCs. To address this issue, we use *task template*, an unified task structure for running software components with reconfigurable settings. Figure 6 illustrates the process of creating a new application task for a SWC. SWC manager stores SWC object files and their configuration files. A configuration file describes communication information, such as destination address and port for the SWC. It is transferred from the master ECU and can be changed when plug-and-play. A SWC and the corresponding configuration are loaded by task generator which creates a new application task for the SWC using the unified task template and the configuration.

IV. CASE STUDY: MDPS SYSTEM

In order to show that our middleware supports application-level plug-and-play, we port the middleware to experimental boards. We implement an example application which is like Motor Driven Power Steering (MDPS) or Electric Power Steering (EPS) system [13]. MDPS system receives two inputs: handle torque and vehicle velocity. Then MDPS control unit calculates and steers the vehicle by controlling steering motor. However, we assume that each wheel has its own motor like in-wheel motor vehicle [14] and the vehicle steers with differential speed of each side motors.

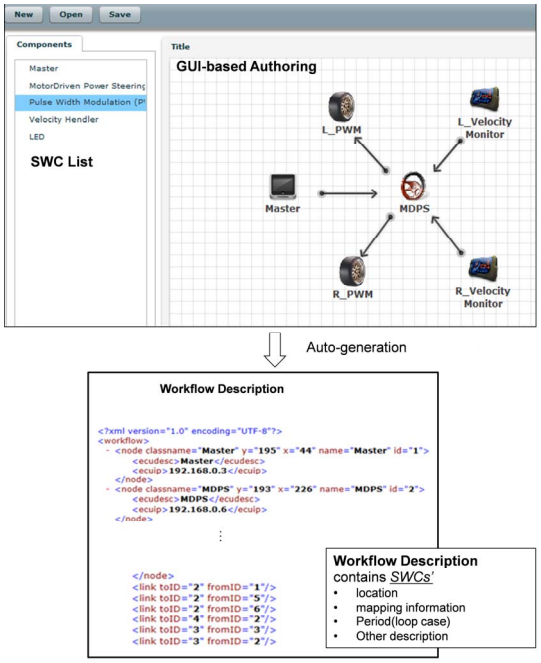


Figure 7. Workflow for the MDPS system

A dual core ARM board (Tegra 250) is used as master ECU and ported modified Android. ARM7 boards (AT91SAM7X-EX) are used as slave ECUs and ported modified FreeRTOS. A master ECU and three slave ECUs are placed on small vehicle platform (Figure 7). Two slave ECUs are connected and control each DC Motor. Another slave ECU steers vehicle platform. The boards are connected IP-based network by router. Modified Android uses TCP-IP protocol with wired Ethernet and Modified FreeRTOS uses uIP protocol [15] with wired Ethernet. An IP-based network is not prevalent for in-car communication networks since it cannot support real-time communication. However, IP-based network is becoming new in-car network because IP is a serious alternative technology with many advantages and benefits. [16] Moreover, we focus on applying plug-and-play concept to automobile, not showing real operation of automobile.

We provide a scenario with installation of MDPS application. Before MDPS application is installed, only a Driving application which can accelerate or decelerate the vehicle is installed. MDPS application is deployed to three slave ECUs. A slave ECU which steers vehicle receives vehicle velocity from other two motor connected ECUs periodically. When user sends vehicle control signal to the slave ECU, it calculates velocities of each side and sends velocity set signal to two other slave ECUs. Our objective is that installs MDPS application to three slave ECUs in run time.

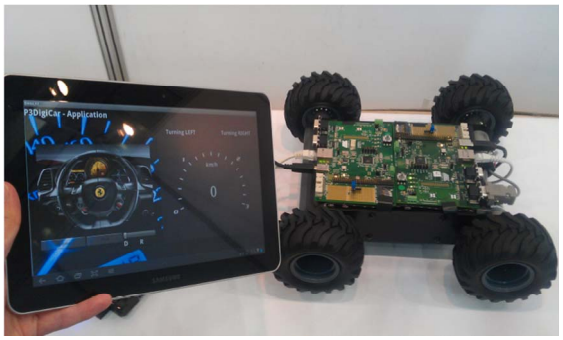


Figure 8. Prototype implementation

Instead of using real handle or accelerator, we implement android application to control the vehicle. We implement three applications: Dash Board, Handle, and Pedal. The applications use automotive library to control and monitor the vehicle. We make a MDPS application with our web-based tool. Master ECU downloads the MDPS application which is made XML language. Master ECU parses MDPS application with schedulability analyzer with other application which is installed in advance. After schedulability analysis, master ECU deploy the SWC to slave ECUs and slave ECUs install own SWC. We get the result that new application can operate correctly. In other words, we provide application plug-and-play concept to automobile.

V. Conclusion and Future Work

This paper presented a software architecture for plug-and-play support in automotive systems. The proposed in-vehicle ECU network consists of a master ECU and slave ECUs. The adopted Android for the master ECU allows developers to benefit from rich Android application development environment and our automotive library for writing vehicle applications running on the master ECU. For general applications which will run on several ECUs in a distributed manner, we devised a model-based development tool which generates XML files describing the application workflow. The master and slave ECUs collaboratively support plug-and-play using the generated XML files. We demonstrated the feasibility of our work with a MDPS case study.

We plan to make our plug-and-play architecture compatible to AUTOSAR. However, several parts of AUTOSAR, especially the runtime environment (RTE), should be redesigned to support plug-and-play, e.g., the RTE should include routing function because it should be aware of the location of software components which can be dynamically installed or removed. New RTE should be designed carefully as it plays a critical role in the system reliability, safety, and real-time performance. Moreover, ECU resources are quite limited, necessitating optimization in resource usage. We plan to evaluate the performance overhead in terms of real-time responsiveness, code complexity, and resource (typically, memory) usage caused by adding plug-and-play feature to AUTOSAR. This will lead us to enhance the presented architecture.

The SWC-ECU mapping significantly affects the total performance. Even though in this paper, we assumed that the mapping is given by application developers, it will not be optimal in many cases because the set of ECUs and already-installed applications will be different for each vehicle. We plan to develop an optimal mapping algorithm for our plug-and-play system.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012-0000979).

References

[1] International Consumer Electronics Show (CES), http://www.cesweb.org, 2012.

[2] AUTOSAR, http://www.autosar.org.

[3] G. Macario, M. Torchiano, and M. Violante, “An In-Vehicle Infotainment Software Architecture Based on Google Android,” in IEEE International Symposium on Industrial Embedded Systems (SIES’09), July 2009, pp. 257–260.

[4] Y.-H. Cheng, W.-K. Kuo, and S.-L. Su, “An Android System Design and Implementation for Telematics Services,” in IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS), vol. 2, Oct. 2010, pp. 206–210.

[5] J. Choi, H. S. Park, Y. Hwang, and K.-H. Kim, “Exhibition Speaker: Driver-Oriented Intelligent Human-Vehicle Interaction System,” in International Conference on Intelligent Systems, Modelling and Simulation. IEEE Computer Society, 2012, pp. 14–16.

[6] R. Anthony, A. Rettberg, D. Chen, I. Jahnich, G. de Boer, and C. Ekelin,“Towards a Dynamically Reconfigurable Automotive Control System Architecture,” in mbedded System Design: Topics, Techniques and Trends, ser. IFIP Advances in Information and Communication Technology. Springer Boston, 2007, vol. 231, pp. 71–84.

[7] H.-M. Pham, S. Pillement, and D. Demigny, “Reconfigurable ECU communications in Autosar Environment,” in International Conference on Intelligent Transport Systems Telecommunications (ITST), Oct. 2009, pp. 581–585.

[8] W. Trumler, M. Helbig, A. Pietzowski, B. Satzger, and T. Ungerer, “Self-configuration and Self-healing in AUTOSAR,” in 14th Asia Pacific Automotive Engineering Conference (APAC-14), 2007.

[9] H. Seebach, F. Nafz, J. Holtmann, J. Meyer, M. Tichy, W. Reif, and W. Sch?er, “Designing self-healing in automotive systems,” in Autonomic and Trusted Computing, ser. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2010, vol. 6407, pp. 47–61.

[10] B. Becker, H. Giese, S. Neumann, M. Schenck, and A. Treffer, “Model-Based Extension of AUTOSAR for Architectural Online Reconfiguration,”in Proceedings of the 2009 International Conference on Models in Software Engineering, ser. MODELS’09. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 83–97.

[11] RT-Druid, “RT-Druid: A tool for architecture-level design of embedded systems” White Paper, Evidence S.r.l. 2004.

[12] J. Lehoczky, L. Sha, and Y. Ding, “The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior,” in Real Time Systems Symposium, Dec. 1989, pp. 166–171.

[13] Y. Shimizu and T. Kawai, “Development of electric power steering,” in SAE Transactions, no. 910014.

[14] S. ichiro Sakai, H. Sado, and Y. Hori, “Motion control in an electric vehicle with four independently driven in-wheel motors,” in IEEE/ASME Transactions on Mechatronics, vol. 4, Mar. 1999, pp. 9–16.

[15] Adam Dunkels, http://www.contiki-os.org.

[16] R. Steffen, R. Bogenberger, J. Hillebrand, W. Hintermaier, M. Rahmani, and A. Winckler, “Design and realization of an ip-based in-car network architecture,” in International Symposium on Vehicular Computing Systems (ISVCS), July 2008.