[redacted]

1. INTRODUCTION

Development of an advanced driver assistant system(ADAS) for the smart and connected cars has become one of the hottest issues both in the ICT and automotive industries. There are many efforts to link between many mobile applications and in-vehicle ADAS through connecting internet service for smart cars. Whereas, the OS(operating system) is a troublesome concern in a system of smart and connected cars because the OS provides a basic platform specifying an environment of software programs. Recently, many operating systems targeting infotainment system of smart cars are released from mobile OS providers as well as traditional automotive OEMs. OnStar and Mylink[1] are an example of the earliest smart car infotainment system, which provides subscription-based communications, in-vehicle security, hands free calling, navigation, and remote diagnostics systems. Microsoft releases Windows Embedded Automotive[2], which gives an extensible software platform for automakers and third parties to deliver connected vehicle applications. Similarly, Apple's Car Play [3] is the iOS platform which provides direct access to device functionality, control, and usage. The Open Automotive Alliance[4] is an alliance among automotive and ICT companies aggregated at using Android OS in the smart vehicles. Moreover, there have been various activities to link smartphones and in-vehicle infotainment system by synchronizing the data in real-time. However, one of the biggest hurdles in the current approach is a compatibility problem might be occurred when the OS of in-vehicle system and user preferred or owned mobile phone is different. Because in-vehicle system must be optimized and proved to satisfy functional safety requirement such as ISO26262 functional safety standard strongly enforced from the automotive industry, it is not easy to link between in-vehicle systems with application software developed by 3rd parties using different OS or platform. Besides, the smart-car owner may not want to change her/his smartphone for solving the compatibility problem between the phone and in-vehicle systems. Various researches have been done or in progress to improve accessibilities and flexibilities such as user-adaptive and context-aware approach[5]. However, these methods can only personalize within one operating system. In this paper, we propose a personalized multiple operating system based on a virtualization technique, where the driver can select her/his preference by using the smart-key. A detailed concept of the proposed method and software architecture will be presented in the section 2 and section 3, respectively. Proposed prototype with experimental results are introduced in section 4, and conclusion summarizes the proposal in section 5.

2. CONCEPT OF THE PROPOSED SYSTEM

The concept of the virtual desktop is introduced to the in-vehicle system of connected cars using the virtual machines (VM) technology.

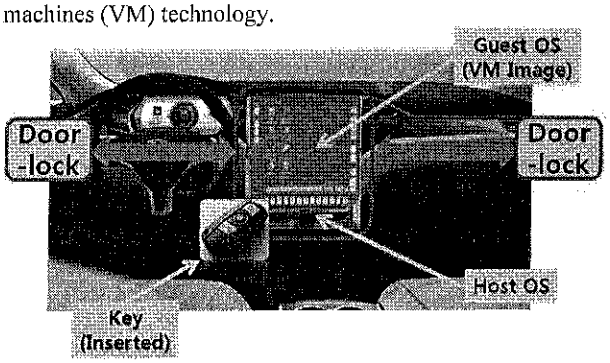

Fig. 1 Basic Concept of the proposed system.

In order to provide personalization of the infortainment system of smart car, the smart-key is used as a storage medium of driver's preference information for personal authentication as well as an identification device in the proposed virtualization technique. An image file stored in the key contains information about the preferred guest OS of the driver. The OS can be fully customizable, by writing the image file into a storage device in the key. When the driver attempts to open the door or operate the vehicle by starting an engine, the system validates whether a current driver is

registered. If the vehicle is accessed by an authorized person, the door-lock and turn-on switch operates in a normal mode. Otherwise, the accessing or maneuvering the vehicle is so prohibited that incoming inputs are unauthorized attempts.

Once the driver unlocks the door and operates the vehicle by inserting the key, the vehicle turns an engine on if the key is authorized. Along with ignition procedure, it finds an image about the guest OS used in a virtual machine. Once an image file is found, the guest infotainment OS is launched onto the top of the host OS.

3. ARCHITECTURE OF THE PROPOSED SYSTEM

As shown in Figure 2, our software architecture is stacked on HAL layer. Then, the certification process, named Certificate, is integrated onto the HAL.

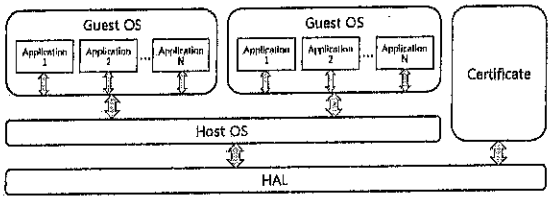

Fig. 2.Platform Architecture of the proposed system.

Host OS is the root operating system in the hierarchy. A virtual machine manager is installed and executed in this layer. Finally the Guest OS's stack onto the top of the platform. Applications run in a selected Guest OS, which is a virtualized OS. Our software architecture is focused to maintain reliability in the Certificate layer, since the door-lock and ignition must be operated in a second and always be ready-to-work. A personalized operating system is developed based on the virtual machine. However, a validation module runs in an independent area because the vehicles' locking system must be reliable as well as its validation and lock/unlock processes must be done instantly. This module communicates with the host OS, and the host OS broadcasts to the guest OS. In the virtual machine area, the guest OS runs upon the host OS. The guest OS can be any image files since the vehicle owner may use different operating systems. However, the host OS is unique while this is a factory-dependent produced. Based on the host OS, we install our virtual machine application, and runs the guest OS based on the virtualization software. In virtualization technique, its bottleneck always comes when applications running on the guest OS attempt to access the hardware. To resolve this, we use the pipe to link between the guest and host OS's hardware access. The hardware interface must be interconnected from the guest OS to an actual hardware. Then, the host OS creates pipes to forward the signals to access the hardware from the guest OS when the guest OS is launched. Using this approach, we could successfully redirect target signals from an application on the virtualized layer to the HAL.

When the vehicle detects request by either pulling or touching the handle, an acknowledge signal is transmitted from the vehicle to the key. If the key is valid and registered, it responses back to the vehicle with an encrypted message. Once the vehicle receives the data, it decrypts the message what the key generates. If the smart key system is integrated as a part of the ECU, there is no way to communicate with an infotainment system. For this reason, we need to store information on the key not only in ECU but also in an infotainment system. It isn't required to know the whole information of the smart key, but the least information is needed for validation with customization.

4. MULTIPLE OS PARALALLIZATION METHODS

An operating system integrated into the vehicle mainly considers the reliability and safety. We use graceful degradation with recovery of duplicated Guest OS to guarantee reliability. To provide graceful degradation, it is mandatory to isolate each application into their own containers. If they are not isolated, fatal error from one application may affect to other applications as well as the kernel which is critical to the system. Thus, docker is adapted to guarantee the complete isolation between applications on the same layer. Once the critical runtime error is occurred, the system first attempts to drop the container which an error is generated. In most cases, quitting a container with issues is sufficient. However, the system equipped in the vehicle frequently accesses the hardware and therefore memory corruptions in the kernel as well as HAL may exist. In this case, the failure caused by an application can influence on the bottom layer, which is the Host OS. Once the system is corrupted, an entire system comes to stop working. Therefore, we need to force quit the services associated into an application. Because reliability and consistency are the most important factors among the vehicles, the degraded services with closed application container maintains the minimal operation of the system. While the Guest OS is in the degraded mode, the duplicated backup OS becomes primary Guest OS with information exchange between the backup Guest OS and the degraded OS. To achieve their information exchange, primary OS requests to the degraded one with the UID of the destination OS and the request time. When the host system receives requests from the primary OS, the message is delivered to the final destination OS based on UID recorded in the header. Then, the destination OS updates application data using the messages received

from the host. Once synchronization processes are achieved successfully, the host OS switches the primary OS to the duplicated OS with removal of the degraded OS container. As the result, the duplicated OS which is running background becomes the primary with synchronization of applications' data, as illustrated in Figure 3. Using the proposed method, reliability and consistency can be guaranteed if the host and background OS is working properly.

As mentioned above, we use more than 2 operating systems to guarantee data consistency. However, these OS' data must be identical in real-time. There are traditional methods to share the data which is called "shared memory." However, this method is not efficient in the proposed system since "collision" increases as the number of OS increases. Thus, we adapt a messaging system to synchronize the state and date of these OS as described in Figure 4.

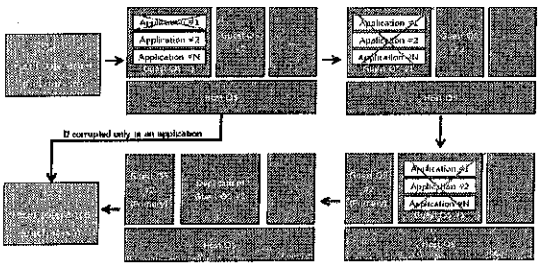

Fig. 3 Flows to guarantee reliability in Guest OS.

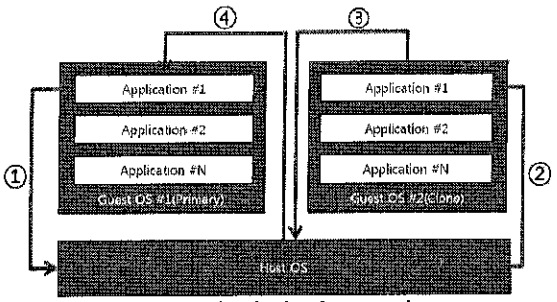

Fig. 4 Flows to maintain the data consistency

To provide this, the manager and broadcaster are nested in the host OS. If 'A' Guest OS wants to make a synchronization with the B, then A dispatches the data to the messaging manager in the host OS. Then, the manager receives the request and dispatches the message to the receiver in B. When B receives the message with the data, it updates the memory. To define the protocol between the host OS and Guest OS, an XML format is used(not shown in the paper). Because the data in the message contain the physical memory, it is critical if the message is stolen or sniffed. Therefore, these must be a security manager with data encryption and decryption using a certification.

5. EXPERIMENTAL PROTOTYPE

As shown in Figure 5 we present a prototype of a personalized in-vehicle infotainment OS with virtualization using the personalized smart key. Since android OS is one of the major candidates in selection of automobile infotainment system, we have chosen the android tablet which integrates NFC capabilities in it. At this point, we show that the guest OS, which is shown in the windows OS, is successfully executed upon the android host OS.

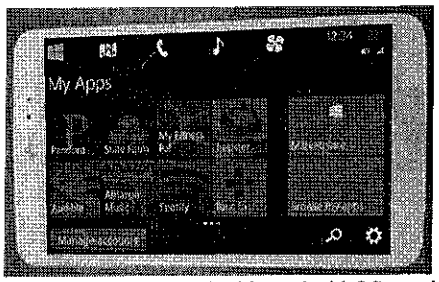

Fig. 5 A prototype configured with Android OS as a host OS, and MS Window as a guest OS.

6. SUMMARY AND FURTHER WORKS

We have proposed and presented a personalized smart-key initiated in-vehicle multiple operating system with virtualization technique. The proposed infotainment system can provide a fully customizable operating system with personalized environment. We believe that the proposed multiple OS may influence the future direction of infotainment as well as the in-vehicle ADAS of smart cars. The driver can select her/his preferred OS among the guest operating systems running on the host OS. Further works to complete the key-initiated in-vehicle system are required, a loading speed the guest OS needs to be significantly improved since the booting process must finished before the vehicle starts to move..

ACKNOWLEDGMENT

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the "IT Consilience Creative Program" (NIPA-2014-H0201-14-1002) supervised by the IITP

REFERENCES

[1] Chevrolet MyLink, www.chevrolet.com/mylink-vehicle-technology.html"
[2] microsoft.com/windowsembedded/en-us/auto.aspx
[3] www.apple.com/kr/ios/carplay
[4] OAA, www.openautoalliance.net
[5] S. R. Garzon, "Intelligent In-Car-Infotainment Systems: A Contextual Personalized Approach," The 8th Int. Conf. Intelligent Environments, pp. 315-318, Jun. 2012