Demo: Design of a Virtualized Smart Car Platform

Roberto Morabito, Riccardo Petrolo, Valeria Loscrì, Nathalie Mitton

Demo: Design of a Virtualized Smart Car Platform

Roberto Morabito
Aalto University - Ericsson Research, Finland
roberto.morabito@ericsson.com

Riccardo Petrolo
Inria Lille - Nord Europe
riccardo.petrolo@inria.fr

Valeria Loscrì
Inria Lille - Nord Europe
valeria.loscri@inria.fr

Nathalie Mitton
Inria Lille - Nord Europe
nathalie.mitton@inria.fr

Abstract

In this demonstration, we present a system that combines the use of Single-Board Computer with the lightweight characteristics of container virtualization technologies for the deployment of a Smart Car platform. Our approach allows the definition of an architecture that exploits the flexibility of containers in terms of dynamic service allocation even on embedded systems. The whole is combined with the design of an inner orchestrator that acts as manager for the scheduling of different virtualized instances according to specific levels of application priorities. We practically show how this integrated environment can facilitate the development of functional and versatile car On Board Units, even on top of constrained Single-Board Computer with several applications instantiated.

1 Introduction and Motivation

Modern vehicles are equipped with electronic systems that are becoming very sophisticated day by day. The functionalities provided by these platforms are several and vary from engine control, predictive diagnostics, driving assistance, and infotainment. However, with the increasing number of functionalities, computing and communication resources, that have to be handled by the On Board Unit (OBU), is constantly growing [2]. OBUs are embedded systems with limited hardware resources and a critical software design that makes also a simple update procedure a not trivial operation. These constraints are combined with the typical software lifetime, which is much shorter than the lifetime of mechanical and technological components.

Bearing all these points in mind, we have identified in the use of lightweight virtualization technologies a suitable mechanism, which allows to design an OBU that can satisfy several requirements in terms of efficient software and hardware resources management.

In particular, container-based virtualization technologies -such as Docker[1]- can bring the advantage of fast allocation and flexibility in managing different services. This is reflected in an easier OBU programmability -also for what concerns updating process, deployment of new software, and management of parallel processes- and in a clever way to deal with the limited OBU hardware resources. This last aspect represents the main novelty respect previous works like Carberry[2], in which no applications allocation mechanism was considered in order to deal with the limited computational resources of Single-Board Computer.

2 System Design

The architecture of our prototype is depicted in Figure 1. The platform has been designed to meet specific software and hardware requirements. Below, we discuss our implementation choices from the hardware, software, and architecture point of view.

Hardware and Software setup. We use the Single-Board Computer Raspberry Pi (RPi) as hardware platform for the deployment of our system. Two key aspects have led us to the choice of the RPi platform. First, the possibility to easily integrate the platform in the vehicle without impacting the car design, thanks to the small RPi size. Second, the RPi capability on efficiently managing virtualized applications by means of container technologies. From this point of view, in [3], authors demonstrated how the introduction of lightweight virtualization in devices with low computational resources introduces a negligible impact in terms of performance overhead. The OBD-II standard [1] interface is directly connected to the RPi and it is used to read data coming from the vehicle, allowing to monitor the current operating status of a vehicle, and to identify malfunctioning in the vehicle itself. OBD interface provides two types of data: real-time vehicle data and several Diagnostic Trouble Codes (DTCs). From the software perspective, as base Operating System, we use the image provided by Hypriot that is running Raspbian Jessie with Linux kernel 4.4.10. The Hypriot image provide a lightweight environment optimised for the execution of Docker container technologies, by also offering dedicated tools for container orchestration.

Architecture description. At the application level, our architecture entails three main components: *(i)* a set of virtu-

[1] https://www.docker.com
[2] http://www.carberry.it

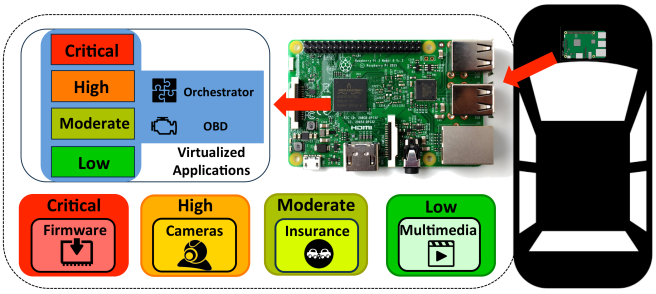

Figure 1. Overview of the system components: the RPi 3 supports the software allocation of virtualised instances, which are in turn characterised by different priority levels defined according to specific requirements.

alized applications tagged with different priority levels; *(ii)* an *OBD* container in charge of receiving and handling data from the vehicle; and *(iii)* the Orchestrator that has the task of monitoring the resources used by the entire system and by each virtualized application. We defined four application/service types: *(i)* **Critical** priority applications are characterized by the highest level of priority (e.g., firmware update/restore applications, or demanding control data); *(ii)* **High** priority applications, which include e.g., driver assistance, camera data; *(iii)* **Moderate** priority applications consist e.g., of tools provided by auto insurance companies, which offer reduced premiums if OBD-II vehicle data loggers are installed; *(iv)* **Low** priority applications include Entertainment/Multimedia contents streamed by an internal and/or external device. In defining the priorities, we refer to the application requirements defined by authors in [2]. A key role is played by the orchestrator, which instantiates the execution of a virtualized application over another in line with the different priority levels. Furthermore, it deals with the limited computation capabilities of the RPi, by opportunistically scheduling the running applications also according to the available hardware resources.

3 Application Scenario

We analyze the case of an anomalous car gas emission detected by the OBD. We suppose that with this particular vehicle issue, the platform has to react with the instantiation of an application characterised by *High* priority. We also assume that, prior to the anomaly detection, two applications characterized by lower priority level, e.g. *Moderate* and *Low*, are simultaneously running in the platform. According to our design, the platform has to guarantee the execution of a specific application that can somehow help on solving and/or tracing the vehicle behaviour during the critical status. This will be possible thanks to the dynamic allocation of a container dedicated to this purpose. In practice, as soon as the orchestrator -which constantly receives data from the OBD- detects the anomaly, it executes the activation of another virtualized application dedicated to the execution of a particular task (Fig. 2). Referring to this example, a database is initialized in order to record all the parameters associated to the evaporative (EVAP) emission system. Moreover, considering that the execution of this task has an higher priority compared to the applications that were already running, the concurrent instances with lower priority will be paused to make available

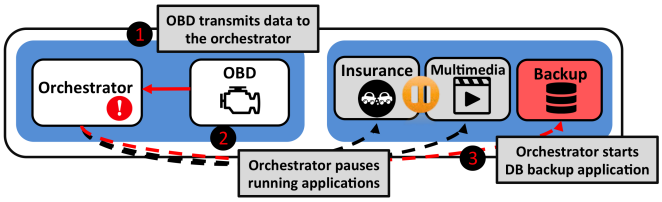

Figure 2. Example of application scenario. Lower priority applications are paused once that the orchestrator detects a critical status in the vehicle. An application with backup functionalities is launched in order to record specific vehicle values involved in the anomaly.

all the hardware resources to the application marked by *High* priority. This example shows how the versatility given by the use of containers enlarges the potentialities of the OBU platform in terms of backup functionality, and capacity of running-dedicated applications at given times, and efficient hardware resources management. Another important aspect coming from the example is the flexibilty of our architecture in the RPi harware performance optimization. Indeed, the definition of different priorities, together with the dynamic management of containers, is also useful to schedule which application takes priority when the hardware resources are kept busy -can not be neglected that the number of applications that can simultaneously run on top of the RPi has an upper-bound.

4 Demo

The demo will show the details of our proposal, by demonstrating all the architecture concepts defined in the previous sections. We show in practice how the orchestrator can cleverly manage the virtualized running instances. In particular, we refer to specific application scenarios in which the orchestrator has to deal with potential vehicle issues and it has to adequately respond in order to ensure a fast and dedicated counter-response to that specific issue. At the same time, the orchestrator will efficiently deal with the platform performance by providing all the necessary resources to favourite the higher-priority applications.

5 Conclusions

We have designed and implemented an embedded all-in-one device for Smart Car applications. We have shown how emerging virtualization technologies can be employed to build a flexible and versatile OBU platform and demonstrated that our architecture can be efficiently embedded in credit card-sized Single-Board Computers, such as Raspberry Pi.

6 Acknowledgments

This work is partially supported by CPER DATA, the FP7 VITAL, and by the FP7 Marie Curie METRICS.

7 References

[1] ISO. 15765-2004 Road Vehicles - Diagnostics on Controller Area Networks (CAN).

[2] H.-T. Lim, L. Völker, and D. Herrscher. Challenges in a future ip/ethernet-based in-car network for real-time applications. In *Proc. of DAC - 48th Int. ACM Design Automation Conference*, 2011.

[3] R. Morabito. A performance evaluation of container technologies on internet of things devices. In *Proc. of IEEE Conf. on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016.