Toward Integrated Virtual Execution Platform for Large-scale Distributed Embedded Systems

Yukikazu Nakamoto†, Issei Abe†, Tatsunori Osaki†, Hiroyuki Terada†, and Yu Moriyama‡

†Graduate School of Applied Informatics, University of Hyogo
1-3-3, Higashi-Kawasaki-cho, Chuou-ku, Kobe 650-0044, Japan
‡FUJITSU TEN LIMITED
2-28,Gosho-dori,1-chome,Hyogo-ku,Kobe 652-8510,Japan
nakamoto@ai.u-hyogo.ac.jp

Abstract. The size and complexity of large-scale distributed embedded systems such as automotive and process control have increased recently. Sophisticated systems that are safe and environment friendly in the distributed systems require numerous types of sensor data, which are collected from various devices and sent to computers through networks. In order to develop the large-scale distributed embedded systems with high productivity and quality, a virtual execution environment platform is required. This platform integrates numerous CPU simulators and various device simulators through the network and provides network-wide simulation functionalities in the distributed system. In this paper, we present the requirements and initial system designs of a virtual execution environments platform for the development of the large-scale distributed embedded system software.

1 Introduction

The size and complexity of large-scale distributed embedded systems such as automotive and avionics systems have increased recently. The large-scale distributed embedded systems consist of numerous CPUs and devices which are controlled and accessed by the CPUs. We call CPUs and devices networked components. Such device includes physical objects and various hardware. We explain the situation with the automotive system. Approximately one hundred electronic control units (ECU), which are controllers for sensing data and actuating vehicle components, are used shown in Fig 1. The total program size of these systems is said to be more than seven million lines. Sophisticated automotive systems that are safe and environment friendly, such as radar cruise control with an all-speed tracking function and a pre-crash safety system, requires numerous types of sensor data, which are collected from various vehicle components and sent to computers through vehicle networks. For example, an adaptive cruise control system (ACC) requires several dozens of input parameters such as brake pressures and throttle angle (Eg. [5]). Further, information

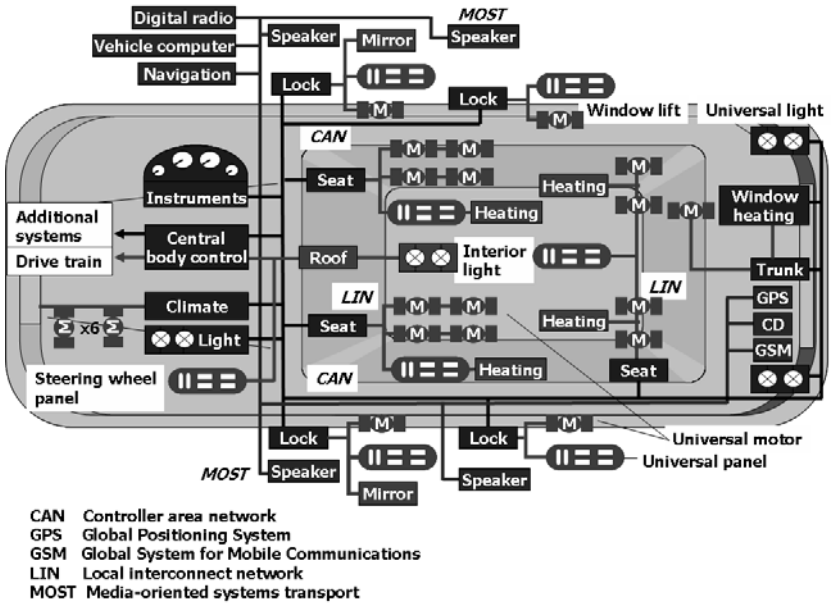

Fig. 1. Network in automotive [9]

system such as navigation systems and vehicle control functionalities will be integrated in the next-generation automotive system (Eg. [4]).

Meanwhile, the market pressure reduces the development period of large-scale distributed embedded systems as automotives is typical. As a consequence of the shorter development period, various types of simulators have been used to validate embedded systems. Simulators for the automotive system development are not only ECU (CPU) simulators but also vehicle component simulators that are controlled by ECUs, such as engines and brakes. To test ECU programs in the absence of actual vehicle components, a hardware-in-the-loop simulation (HILS) is used. The HILS is a real-time simulator and its inputs and outputs are performed at the same rate as those of actual vehicle components [6].

In order to develop the large-scale distributed embedded systems such as the next-generation automotive system with high productivity and quality, a virtual execution environment platform (VEEP) is required. The VEEP aims at validating of the large-scale distributed embedded system software.

In this paper, we address the requirements and initial system designs based on the requirements for a virtual execution environment for the large-scale distributed systems.

2 Requirements

We propose a VEEP for the large-scale distributed embedded systems on the basis of the software trends in the systems, as mentioned above. In order to perform

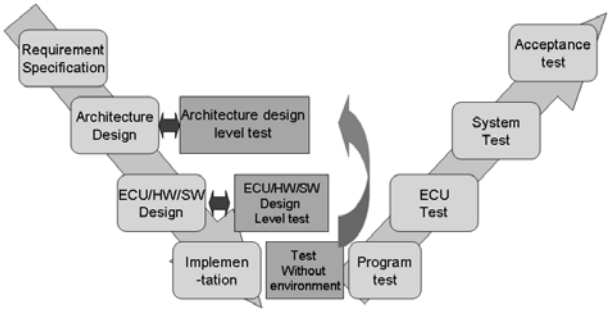

Fig. 2. Design level test using virtual environments

the validating of CPU programs in the distributed embedded system networks, various simulators such as CPU simulators and device simulators should be integrated into the network and provides network-wide simulation functionalities in the large-scale distributed embedded systems. A device simulator is one that gives data detected by sensors to CPU and is controlled by the CPU based on the data. The VEEP implements the integration with the communication middleware, and the CPU simulators and device simulators are connected to the communication middleware.

Next, we summarize the requirements for the VEEP. We identify them from a viewpoint of validating networked components through the whole of the large-scale distributed systems such as automotive and avionics systems.

R1: Open standardized interface
Various simulators such as CPU simulators and device simulator are connected to the communication middleware. For the connection to be established to the communication middleware easily, the interface of the communication middleware should be open and standardized.

R2: Faster CPU simulation speed
The simulation speed of a cycle-accurate simulator, which is used in hardware-software co-design, is slow for debugging and testing programs in a CPU. The execution speed of simulators is expected to be faster and more flexible. For example, when testing for error injection, a slower simulation speed is desirable. Meanwhile, it is expected to be as fast as in an actual hardware in order to implement a man-machine functionalities or the hybrid simulator, as given below.
Meanwhile, the time model in the simulators will be quite simple. CPU and device simulators are synchronized with fine-grain time intervals needed for the validation in our application areas.

R3: Enabling design level validation
In the test phase, a bug in the design phase is detected and fixing this bug is expensive. To avoid such situation and to reduce the test time, validating the system design and software design at a higher level is very effective when using the device simulators in the VEEP, as shown in Fig 2. In the design

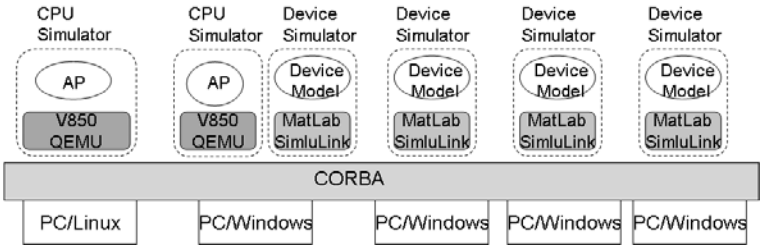

Fig. 3. Integrated Virtual Execution Platform for Large-scale Distributed Embedded Systems

level, dynamic properties of the programs are evaluated such as controllability, stability, and their timeliness.

At the design level of the automotive system, AutoSar[1] has proposed an automotive software architecture and development process to increase the software productivity. In AutoSar architecture, automotive software is designed as a software component independent of the automotive infrastructure such as ECUs and networks, and the software is delivered from component vendors to component users. In order to make the software independent of the infrastructure, AutoSar defines a virtual function bus (VFB) [1]. If we use AutoSar software components on the top of the VFB along with the device simulators in the VEEP, it becomes very effective for validating the ECU software design.

R4: Enabling hybrid simulation

If an actual device such as an ECU or a hardware simulator of a device is used, it connects to the communication middleware of the VEEP instead of the corresponding software simulator. Thereby the embedded software can be validated in the execution environment that is closer to the actual execution environment.

3 Initial system designs of VEEP

The proposed architecture of the VEEP is shown in Fig. 3. The VEEP consists of a communication middleware, CPU simulators, and device simulators. We present the initial system designs of each subsystem of VEEP.

Communication middleware: We select Common Object Request Broker Architecture (CORBA) as the communication middleware as it provides solutions for **R1**. CPU and device simulators are executed as CORBA objects and communicate using the CORBA communication mechanism. We evaluated ACE/TAO

[1] http://www.autosar.org

[2] and MICO [3], and select MICO for the present evaluation. Since periodical communication with small-size data are done within hundreds μs in the large-scale distributed embedded systems, functionalities of MICO are considered to be sufficient for the purpose.

As an alternative, a framework to integrate an ECU simulator and mechanical simulators is proposed in [7]. In the framework, the mechanical simulator is implemented by MATLAB/Simulink [4] and a named pipe is currently used for commutation between the ECU simulator and the mechanical simulator. However, the named pipe is not scalable to integrate a huge number of the CPU and device simulators. Further the framework does not provide the synchronization methods by utilizing the clock services that are present in its own simulator [13].

We must consider the synchronization between CPU simulators and device simulators. We have two solutions: a centralized control and a decentralized control. In the central control, a certain time service software provides the synchronization information. For example, [11] defines the standardized time service interfaces present in CORBA. [3] describes yet another time service architecture in CORBA. In this architecture, the Synchronization Scheduler distributes a clock event to Synchronized Clerk, which exists in a node and delivers a clock event to a CORBA object in the node. The jitter time in the CORBA object side is limited to 100 μs and its average value is approximately 50 μs. This is implemented by the COBRA Event Service [12]. In the decentralized control, each node on the CORBA manages the time and delivers the synchronization information to the objects in the node. The TMO object provides an implementation method for that [8]. We think to develop a light-weight communication middleware which is optimized for minimum required CORBA APIs.

CPU simulator: We utilize QEMU [2] for developing CPU simulators. The QEMU itself is a virtual execution environment generator. The virtual execution environment, which is generated by QEMU, translates the target machine codes to host machine codes, and executes them. The translation is in two steps: firstly, a target machine code is translated to a number of intermediate codes and the intermediate code is compiled to host machine codes. The benefits of using QEMU in a CPU simulator in an embedded system support a variety of CPUs and QEMU provides many libraries for peripheral simulation of the target machine. A QEMU user can easily implement the functionalities when the target programs read or write with the byte access and word access.

We have developed the NEC V850 simulator that modifies the QEMU [10], which is used for various type of embedded systems. The total program size of the modified and appended C source is approximately 3,000 lines, while the total size of the QEMU is 233,000 lines. Moreover, we have modified the QEMU so that translated codes can communicate with the CORBA's communication middleware.

[2] http://www.cs.wustl.edu/ schmidt/TAO.html
[3] http://www.mico.org/
[4] MATLAB/Simulink are registered trademark of MathWotks, Inc.

Device simulator: In order to implement a wide variety of device simulators, we use MATLAB/Simulink to simulate a physical object since they are widely used for modeling control systems and simulating models. A user develops a model the behavior of devices. MATLAB/Simulink with a device model is connected to the CORBA as a CORBA object.

4 Conclusions

In this paper, we present the requirements and initial system designs of a virtual execution environments platform for the development of next-generation large-scale distributed embedded systems. At present, we are in a nascent stage of the development. In the future, we intend developing the VEEP, especially to solve issues related to synchronization.

References

1. AUTOSAR. *Technical Overview*, June 2006.
2. Y. Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proc. USENIX 2005 Annual Technical Conference*, pages 41–46, Apr. 2005.
3. I. Calvo, L. Almeida, and A. Noguero. A Novel Synchronous Scheduling Service for CORBA-RT Applications. In *Proc. 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pages 181–188, May 2007.
4. N. U. Center for Embedded Computing System. Operating System for in-vehicle multimedia systems. http://www.nces.is.nagoya-u.ac.jp/project/e-index.html.
5. D. Han, K. Yi, and S. Yi. Evaluation of Integrated ACC(Adaptive Cruise Control)/CA(Collision Avoidance) on a Virtual Test Track. In *Proc. 2006 SICE-ICASE International Joint Conference*, pages 2127–2132, Oct. 2006.
6. R. Isermann, J. Schaffnita, and S. Sinsel. Hardware-in-the-loop simulation for the design and testing of engine control systems. *Control Engineering Practice*, 7(5):643–653, May 1999.
7. M. Ishikawa, D. McCune, G. Saikalis, and S. Oho. CPU Model-Based Hardware/Software Co-design, Co-simulation and Analysis Technology for Real-Time Embedded Control Systems. In *Proc. 13th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 3–11, Apr. 2007.
8. K.H. Kim and J.Q. Liu and H. Miyazaki and E.H. Shokri. TMOES: A CORBA Service Middleware Enabling High-Level Real-Time Object Programming. In *Proc. 5th International Symposium on Autonomous Decentralized Systems*, pages 327–335, Mar. 2001.
9. G. Leen and D. Heffernan. Expanding automotive electronic systems. *IEEE Computer*, 35(1):88–93, Jan. 2002.
10. NEC. *V850 FAMILY 32-bit Single-Chip Microcontroller Architecure User Manual*, 1994.
11. OMG. *Time Service Specification, Version 1.1*, 2002.
12. OMG. *Event Service Specification, Version 1.2*, 2004.
13. OMG. *Real-time CORBA Specification, version 1.2*, 2005.