99PC-315

A Distributed Control System Framework for Automotive Powertrain Control with OSEK Standard and CAN Network

Shoji Suzuki, Wataru Nagaura, Takaaki Imai, Satoru Kuragaki, Takanori Yokoyama
Hitachi Research Laboratory, Hitachi Ltd.

Kang G. Shin
Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan

Copyright © 1998 Society of Automotive Engineers, Inc.

ABSTRACT

This paper presents a distributed control system framework for next-generation automotive control systems, in which various control units are connected with CAN bus. The framework is a software platform that performs communication between control units and invocation of application programs. The framework includes necessary functions for data transmission to meet end-to-end timing constraints in distributed control systems. Application programmers don't have to write any communication procedure but focus on developing application programs with appropriate API. The framework is based on driving force management and also OSEK, which is a standard real-time operating system (OSEK-OS) and a communication protocol (CAN) for automotive control. We are now applying our prototype framework to an adaptive cruise control system in our experimental vehicle.

1. INTRODUCTION

In next-generation vehicle control systems, various control units inside a vehicle are connected to each other via a control network like CAN[7] and execute their functions cooperatively with others. A typical example conventional application program is shown in Figure 1. The figure shows an example ACC (Adaptive Cruise Control) which keeps the headway distance from the vehicle ahead. The ACC application program and the Throttle Valve Opening application program are embedded into the ACC unit and the PCM (Powertrain Control Module) unit, respectively. Each application program in Figure 1 is invoked periodically, and also sends or receives the Desired Driving Force between the two control units independently.

Conventionally, application programmers must provide all of the structure and flow of execution of application programs and make them call function libraries as necessary. All communication services are mixed and

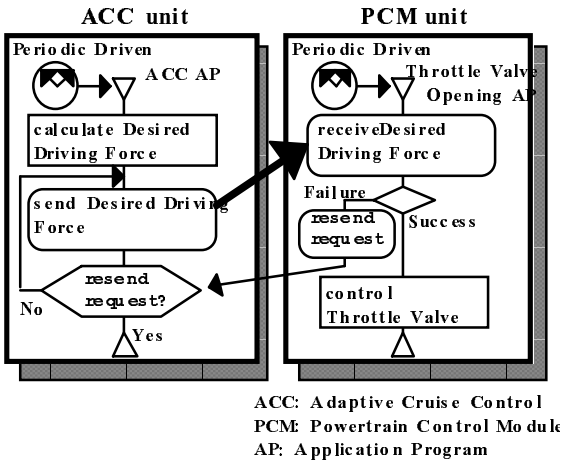


Figure 1: A conventional application program

embedded into the application programs independently on control units. This not only complicates the software but also requires exhaustive modification of application programs in most of control units when one wants to make a slight improvement/change, thus making it very difficult to ensure logical correctness and meet all end-to-end timing constraints in the whole distributed control system. Because of this complexity of software, it is the simplest and also easiest to guarantee all end-to-end timing constraints by invoking all application programs periodically.

In order to solve/alleviate the problems of conventional application programs, we introduce a new framework[1] for distributed control systems software. The framework allows (1) application programmers to focus only on developing application programs which is essential for controlling devices in automotive control systems, and (2) a system integrator to focus on the software structure of distributed control systems. It also reduces software-complexity and enables construction of application

programs while meeting all end-to-end timing constraints in the system.

In the next section, we describe the features of the framework and then present its implementation. The memory size and CPU load of the framework on our experimental vehicle is evaluated and conclusions are drawn.

2. THE FRAMEWORK

The framework provides the basic software infrastructure of distributed control systems and the flow of control, and invokes all application programs and communication services. Application programs are described according to the API (Application Program Interface) that the framework provides.

2.1 APPLICATION PROGRAM INVOCATION AND COMMUNICATION CONTROLS

The framework invokes application programs and communication services among them and provides an integrated control in distributed control systems. An example structure of the framework is shown in Figure 2. The example application is the ACC in Figure 1. In this example, the framework invokes the ACC unit periodically. Then it transfers the Desired Driving Force (DDF) which is calculated from the ACC application program on the ACC unit to the Throttle Valve Opening application program on the PCM unit. Finally, it invokes Throttle Valve Opening application program on completing the transfer of DDF.

The framework calls procedures in application programs, and reads and writes data for communication. The framework separates application programs from communication procedures; decreases software-complexity dramatically as compared to conventional application programs; and makes it easy to meet all end-to-end timing constraints in the system.

2.2 PERIODICAL AND EVENT DRIVEN CONTROLS

The framework provides the optimal invocation of application programs and communication services by a combination of periodical and event driven controls.

Periodic control: invokes programs and services periodically (e.g. interrupts by a timer). This control makes it easy to predict the timing behavior of execution. This is also the default in the framework.

Event-driven control: invokes programs and services upon occurrence of an event (e.g. interruption of a switch, receipt of a message via network,...). Though this control makes it difficult to predict the behavior of execution on time axis, it can execute application programs with much faster response than periodic control. This is generally applied to time-critical and hard real-time applications.

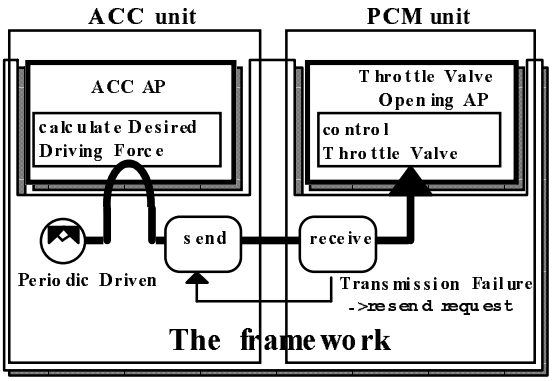


Figure 2: The framework

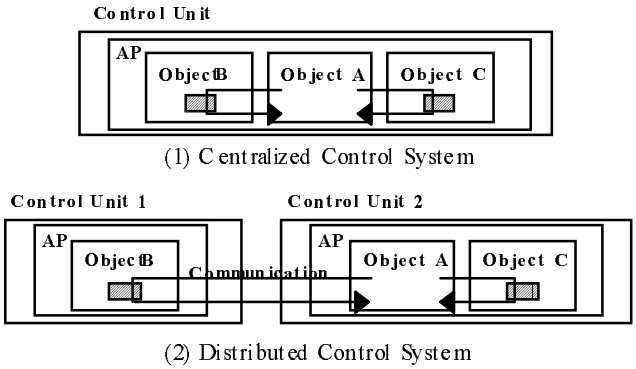


Figure 3: Location transparency

2.3 LOCATION-TRANSPARENT API IN DISTRIBUTED CONTROL SYSTEMS

The object-oriented programming is the default programming style for development of application programs in the framework. The example shown in Figure 3 shows the location-transparent API which the framework provides. By this API, an application program consisting of object A, B and C in Figure 3 (1) can be divided into several groups (object A and B, and object C) and allocated to several separate control units without any modification of codes in the application program (Figure 3 (2)).

2.4 INTRODUCTION OF STANDARD TECHNOLOGIES

In order to keep abreast of users' demand with open systems and also reduce time to market, we introduce OSEK/VDX[6] and CAN into our implementation on the framework as the software platform and the communication protocol for automotive control.

2.5 IMPLEMENTATION WITH LESS RESOURCE

In order to reduce the cost, we need to implement most of control units with 1 chip microprocessor of low price and also to embed all programs into internal memory of

the microprocessor. It is required to implement the framework with less resource (memory size and CPU load).

3. IMPLEMENTATION

In order to realize the facilities and to meet the requirements of the framework mentioned above, we implement the framework as follows.

3.1 SOFTWARE STRUCTURE

The software structure of the framework consists of the API software, the middleware, OSEK-COM, OSEK-NM, OSEK-OS and the CAN driver as shown in Figure 4. Most contemporary software for automotive control does not use OS, but the framework is implemented to support both systems with OS (OSEK-OS) and without OS.

The API software controls the invocation of objects in application programs and communication services among objects. The middleware contains message communication services to improve the efficiency of data transmission. OSEK-COM provides basic message communication services. The CAN driver is the interface between OSEK-COM and a Hitachi's CAN controller device, HCAN. OSEK-NM provides basic network management. OSEK-OS controls tasks which execute application programs and provides the mutual execution service among tasks.

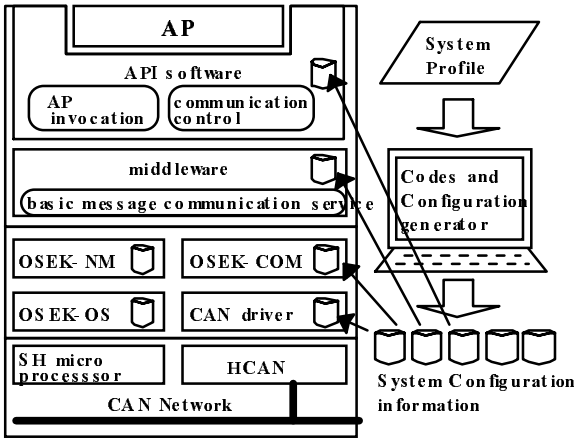


Figure 4: Software structure

3.2 API SOFTWARE

The API software invokes objects in application programs and controls communication services among objects and executes the integrated control of objects in distributed control systems. It also provides location-transparency of objects by making replicas of the original object.

The implementation of replica objects is shown in Figure 5.The replica of object B on the sender control unit1 is implemented at the API software on the receiver control unit 2. The object A on the unit 2 calls the replica object located at the API software on the same unit instead of original object B, making it possible to realize the location-transparency of objects. The copy of data from the original object to the replica object is provided by the data transmission facility below the middleware layer of the framework.

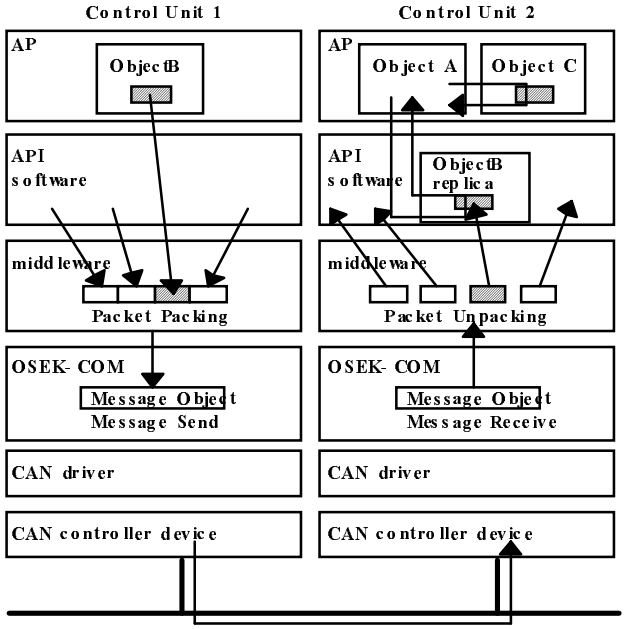


Figure 5: Location transparency and packet packing

3.3 MIDDLEWARE

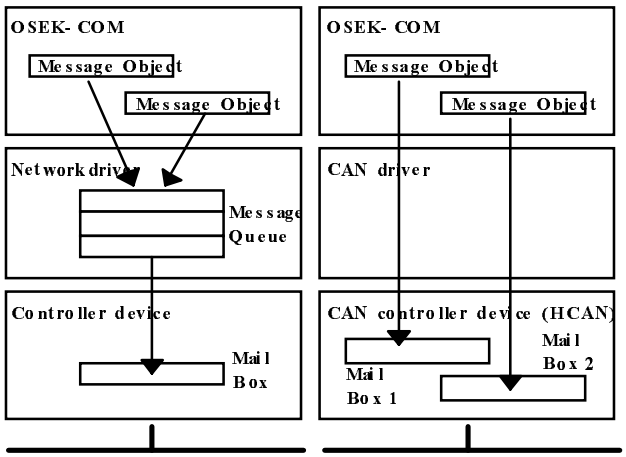
The middleware provides the packing and unpacking services of data into or from packets for data transmission in order to improve the efficiency of data transmission. As shown in Figure 5, the API software reads several small data of objects in application programs and pack them into one packet with this packing service, which copies these several packed data into a message object of OSEK-COM and sends into other control unit(s). Meanwhile, the API software extracts several packed data from the receipt message object with this unpacking service, then copies the unpacked data into the replica objects in application programs in the sender control unit 1. The middleware also provides the priority-based message communication service of data transmission with the message ID (arbitration field) of CAN packet protocol, which makes the guarantee of real-time control easier.

3.4 OSEK-COM AND CAN DRIVER

As CAN controller devices for communication, we adopt Hitachi's CAN controller device, HCAN with the internal

packet arbitration service. HCAN has 16 send/receive mail boxes and enables priority-based transmission control of packets on the device, even in the case when the packet of lower priority on the device is pending for transmission because of traffic congestion on the network. The CAN driver software is the interface between OSEK-COM and a HCAN, while OSEK-COM software provides the message communication services to the middleware. We implement OSEK-COM and CAN driver on HCAN with the internal arbitration service of controller device.

The conventional implementation of communication driver software is shown in Figure 6 (1). The message objects on OSEK-COM are copied into the FIFO message queue and the order of message transmission to the network is fixed as the order of message queue. Our implementation of CAN driver is shown in Figure 6 (2). The CAN driver does not have message queue but each message object on OSEK-COM is written directly onto the corresponding message box on HCAN by OSEK-COM. Then HCAN sends the message with the highest priority on message boxes at first. This implementation decreases not only CPU load but also the consumption of memory, because the CAN driver does not require the message queue.



(1) Conventional Implementation (2) Our Implementation

Figure 6: Implementation of OSEK-COM and CAN Driver

3.5 EMERALDS-OSEK

We have developed EMEALDS-OSEK[4] with the University of Michigan, based on their EMERALDS microkernel[5]. The original EMERALDS is designed for small-memory embedded applications. EMERALDS is not only very small (its code size is just 13 KBytes even in the full-spec version) but also has low overhead kernel services. EMERALDS-OSEK provides full-feature of OSEK interface constructed with the minimal set of functions provided by the original EMERALDS.

3.6 CODES AND CONFIGURATION GENERATOR

Each component in the software structure of the framework contains the configuration information of the system as shown in Figure 5, and executes its processing control based on the information. Only the codes of the API software as well as its configuration information depends much on application programs, while other software of the framework is static and only its configuration information is modified according to the system configuration. The codes of the API software and the configuration information of all software of the framework will be generated automatically by a tool, which requires system profile information as input (this feature is now under development).

3.7 PROTOTYPE FRAMEWORK

We are now applying our prototype framework to the ACC system in our experimental vehicle to show the improvement both on software productivity and also on ease in guaranteeing end-to-end timing constraints in the distributed automotive control. The overview of prototype framework on the ACC control is shown in Figure 7. The distributed automotive control system is constructed on the radar unit, the ACC unit, and the PCM unit. The framework provides the flow of control, that is, when and how the sequences of control - not only calling each application program but also transmitting the data - are determined and invoked by the framework.

In case of Figure 7, the sequence of control is invoked by a 50 msec. timer on the radar unit, and then calls the application program on the unit to measure both the headway distance and the relative speed between the vehicles with a milli-wave radar device, and send the data to the ACC unit via the CAN network. In the ACC unit, the sequence of control is invoked by the data receipt event and invokes the application program to calculate the desired speed of the vehicle with the measured data on the radar unit, then invokes the another application program to calculate the desired driving force (DDF), and send the calculated DDF to the PCM unit via the network. In the PCM unit, the sequence of control is also invoked by the calculated DDF information receipt event and invokes the application program of driving force manager which controls opening the throttle valve as well as the fuel injection, the transmission ratio and the braking force. This makes the response time of throttle valve opening control about 50 + α msec., while α is the execution time of the sequence on the framework (at most, several msec.).

In conventional application programming style, however, it is difficult to meet all end-to-end timing constraints in the system and each application program is generally executed periodically. In case that the period of invoking application programs on each control unit is 50 msec., the worst-case response time of throttle valve opening control is about 150 + β msec., while β is the execution time of the sequence (at most, several msec.).

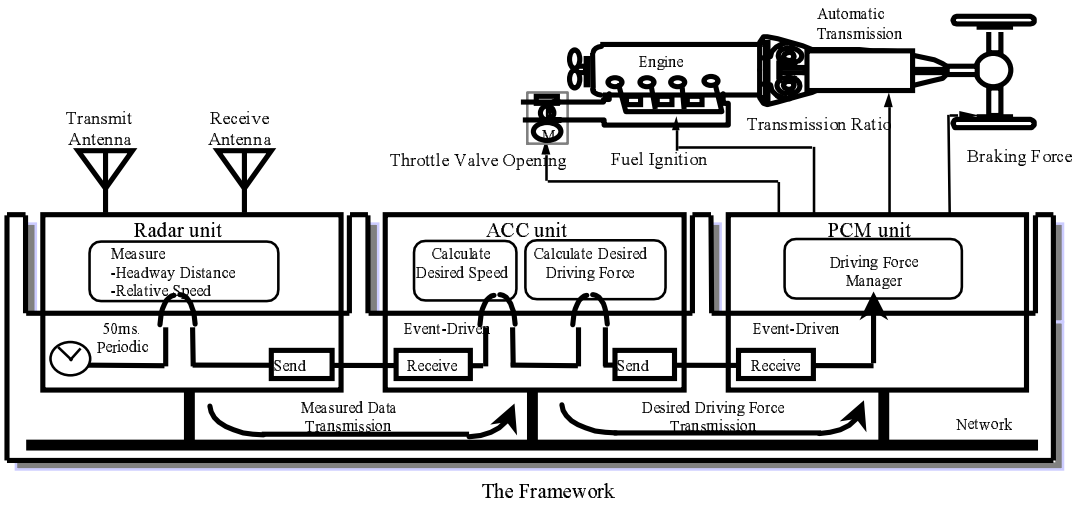


Figure 7: Prototype framework on ACC Control

4. PERFORMANCE EVALUATION

We have developed most of primitive functions on the API software and the middleware, and completed the development of OSEK-COM, and CAN driver software. Here, we show the memory size and CPU load of the framework. The evaluation is done for SH2 32bit RISC microprocessors (20MHz) on which the application program is running, HCAN microcontroller, and the CAN network (1Mbps), which are the hardware of the ACC system mentioned in Section 3.7.

4.1 MEMORY SIZE

The code sizes of the framework are as follows: the middleware is about 1.5 KBytes of ROM, OSEK-COM is about 2 KBytes, CAN driver is about 2 KBytes and OSEK-OS (EMERALDS-OSEK) is about 5 KBytes. The code size of API software depends on the application programs and is proportional to their code sizes. But most of execution of API software is just calling application programs or the services of the middleware and its code size is rather smaller than application programs. Here, we don't consider the code size of API software.

The total code size of the framework (except API software) is about 10 KBytes of ROM with OSEK-OS or about 5 KBytes without OSEK-OS. The code size with OSEK-OS is twice as large as without OSEK-OS. However, the internal ROM size of recent microprocessor for automotive control increases larger and several dozen of KBytes to 1 hundred KBytes or more of internal ROM becomes more general. So, 10 Kbyte-code size of the framework with OSEK-OS is enough small to be implemented into the application for automotive control.

4.2 CPU LOAD

On our experimental ACC unit, the required throughput of data transmission via 1Mbps CAN network is 56 Kbit/sec., and CPU load on the unit is about 7 % with OSEK-OS or about 4 % without OSEK-OS. In both cases, the load is enough small to execute application programs on it.

5. CONCLUSION

We presented a distributed control system framework for next-generation automotive control systems, in which various control units are connected with a CAN bus and execute their missions cooperatively. The framework is a software platform that performs both communications between control units and invocation of application programs. The framework also includes necessary functions for data□transmission. These features of framework are effective to improve software productivity as well as to meet real-time constraints on distributed control systems.

Future work includes completion the development of prototype framework, the tool generating the codes of the API software and the configuration information of all software of the framework automatically, and implementing OSEK-NM in the framework.

6. ACKNOWLEDEGMENTS

The authors would like to thank Khawar Zuberi and Padmanabhan Pillai for their EMERALDS-OSEK development and cooperation of the performance evaluation on this paper.

REFERENCES

[1] *Leveraging Object-Oriented Frameworks*, IBM Java Education - White Papers, Tutorials, and Articles - (available from http://www.ibm.com/java/education/papers-oo.html)

[2] W. Nagaura, T. Imai, S. Suzuki and T. Yokoyama, "Development of a Distributed System Framework for Automotive Controllers (1) - Goal and Features -," *in Proc. IEICEJ Society*, Oct. 1998 (written in Japanese)

[3] T. Imai, W. Nagaura, S. Suzuki and T. Yokoyama, , "Development of a Distributed System Framework for Automotive Controllers (2) - Functions and Implementation -," *in Proc. IEICEJ Society*, Oct. 1998 (written in Japanese)

[4] K. M. Zuberi, P. Pillai, K. G. Shin, T. Imai, W. Nagaura and S. Suzuki, "EMERALDS-OSEK: A Small Real-Time Operating System for Automotive Control and Monitoring," *in Proc. SAE International Congress & Exhibition*, Mar. 1999

[5] K. M. Zuberi and K. G. Shin, "EMERALDS: A microkernel for embedded real-time systems," *in Proc. RTAS*, pp.241-249, June 1996

[6] *OSEK/VDX Version 2.0*, OSEK Group, 1997.

[7] *Road vehicles - Interchange of digital information - Controller area network (CAN) for high-speed communication, ISO 11898*, 1st edition, 1993.