Cyber-physical MBD for Multi-physics Automotive Systems

Sujit S. Phatak
DJ McCune
George Saikalis, Ph.D.
Yasuo Sugure

OVERVIEW: The need for CPSs derives from the complexity of modern automotive embedded systems, which can contain more than 100 ECUs, a large ROM capacity, and a large amount of software code. Another issue is that traditional design methodologies find it difficult to cope when fundamental hardware problems are identified late in the development process, during the validation phase. The MBD approach is one possible solution to these issues and it lays the foundation for CPSs.

INTRODUCTION

THE defining feature of cyber-physical systems (CPSs) is their close integration between physical processes and systems on one hand and computational systems on the other[1]. CPSs integrate the dynamics of physical processes with those of the software and network, providing abstractions for modeling and design as well as analysis techniques suitable for integrated systems. In contrast to traditional embedded systems, where the emphasis is more on operation as a standalone device, CPSs emphasize the network of interacting systems. Similarly, while traditional embedded systems focus on the computational components, CPSs primarily focus on the interfaces between the computational elements of the system. Their scope of application covers all sensor-based control systems, embedded systems, and autonomous systems. These cover a very wide range, from device-based systems such as automotive systems, entertainment and home appliances to integration systems such as social infrastructure, energy, freight and transportation, aeronautical and space applications, and healthcare, and also technical platforms such as manufacturing systems.

The mechatronic control systems that are typically implemented in automotive applications, such as engine control, transmission control, throttle control, and braking, typically involve multiple complex physical systems with dedicated embedded controllers that communicate with each other via a vehicle network, such as Controller Area Network (CAN) or FlexRay. Model-based design (MBD) is adopted as a way of making the design process for these complex system more efficient[2]. The system design stage integrates models of physical system behavior (also called "plant models") with controller models to produce an abstracted system implementation. The controller models can be implemented either at the algorithm level, using popular tools such as MATLAB*1/Simulink*1, or they can be implemented at a lower abstraction level using virtual central processing unit (CPU) modeling techniques.

Virtual CPU modeling[2] involves development of a software model of the microcontroller hardware itself. This microcontroller model can then be integrated with the behavioral models of the plant (physical system) so that realistic system performance measurement and validation can be performed. This approach allows concurrent development of the plant models and control software applications, and also their validation, including the realtime operating system (RTOS) and device drivers.

Sometimes the system may consist of multiple plant models (physical systems) representing different components of the mechatronic system that need to be implemented in different domains, and that need to be connected to a controller (computational system) via some interface. Together, these elements form a mechatronic CPS.

This article reviews the basic approach to implementing a CPS adopted at the Automotive Products Research Laboratory of Hitachi America, Ltd., and describes a CPS case study in the form of a gasoline fuel pump that was implemented as a multi-domain co-simulation platform[3].

NEED FOR CPS IN AUTOMOTIVE APPLICATIONS

Challenges for Automotive Embedded Systems

The automotive embedded systems used in different parts of modern vehicles (including the powertrain, transmission, vehicle dynamics, and infotainment) can be very complex, consisting of more than 100 electronic control units (ECUs). These ECUs are microcontroller-based embedded systems.

*1 MATLAB and Simulink are registered trademarks of The MathWorks, Inc.

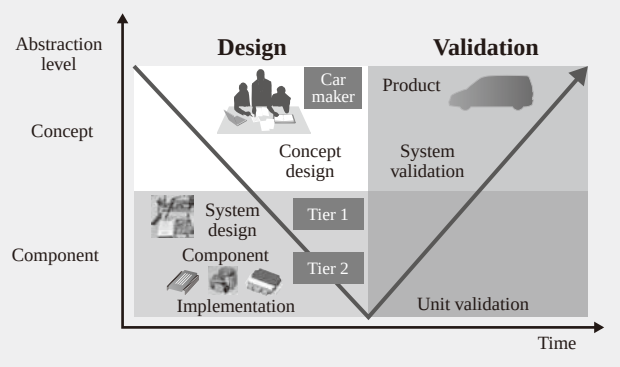

Fig. 1—Traditional V-cycle Development Process.
The typical development cycle for automotive electronic control units (ECUs), including the role of the respective contributors.

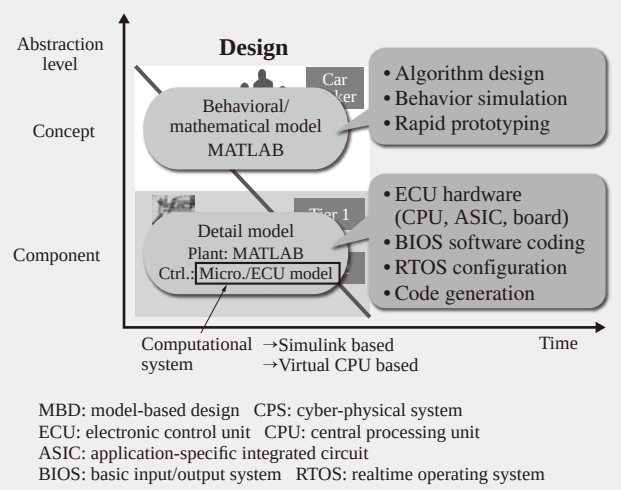

Fig. 2—V-cycle Incorporating MBD and CPS Approach.
The role of MBD and the CPS approach in V-cycle development, including the different activities involved.

Typically, each sub-component of the physical system has a dedicated ECU. Automotive network protocols such as CAN and FlexRay are used to link these ECUs in a network.

Modern automotive applications often require more than 4 Mbyte of read-only memory (ROM) (program memory), which stores a dedicated application program. The size of software code is also increasing exponentially. It is estimated to grow more than 1,000 times in the next 20 years. In contrast, the capacity of the microcontroller hardware (computational system) typically grows by only 20 times over 10 years. To cope with this gap between increasing code size and hardware capacity, and to improve cycle performance and reliability, it is anticipated that multi-CPU or multi-core architectures will be required in the near future.

Inefficiency in Traditional Design Methodology

Traditionally, automotive embedded controller development has often followed the V-cycle shown in Fig. 1.

The V-cycle is broadly divided into two phases, the design phase and the validation phase. The vertical axis shows the level of abstraction while the horizontal axis represents time. The design phase begins with the concept design at a high level of abstraction. This takes place at the car maker or original equipment manufacturer (OEM). Typically, the process then proceeds through progressively lower levels of abstraction, to system design at a tier-one supplier and then to the actual components and hardware supplied by a tier-two supplier. This is the implementation phase, and is followed by the validation phase. This begins with unit validation, which happens at the same level of abstraction as the design implementation. Next is system validation, which represents a higher level of abstraction, with the final product as the end result.

A problem with this traditional V-cycle is that, if a fundamental hardware problem is not detected until the validation phase, going back to the design phase to identify its cause is not always time- and cost-effective. It is important to identify any inherent hardware problems early in the design phase, before investing in hardware development. Also important for business expansion and innovation is the research, development, and testing of future advanced system architectures (hardware and software).

MBD is one possible solution to these problems. The MBD approach is also called “model in the loop simulation” (MILS). It lays the foundation for CPS development. Fig. 2 shows the V-cycle when MBD and CPS are incorporated.

The figure shows how use of MBD concentrates activity in the design phase. This includes algorithm design, behavior simulation, and rapid prototyping, which are carried out during concept design. Initially, signal flow or conserved system simulators like MATLAB/Simulink or Synopsys[*2] Saber[*2] are used to develop behavioral or mathematical models.

This is followed by component simulation at a lower abstraction level. The main development activities here are ECU hardware specification, basic input/output system (BIOS) software coding, RTOS configuration, and autocode generation. In terms of MBD, this phase involves developing detailed models

*2 Synopsys is a registered trademark of Synopsys, Inc. and Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

of both the physical system (plant) and the ECUs and control system (computational system).

The computational system can be built in Simulink or using a virtual CPU. This gives two different configurations for the CPS. Typically, when using MBD, the next step after developing a Simulink-based computational system is to perform autocode-generation and implement the generated software on a real hardware ECU. This is called "processor in the loop simulation" (PILS). The work described here, however, adopts a new approach whereby a simulation of the processor (CPU) is also included in the model. This approach could be called "virtual PILS," although the term "virtual CPU modeling" is used here for consistency. The following section gives a detailed description of CPS using both the Simulink and virtual CPU methods.

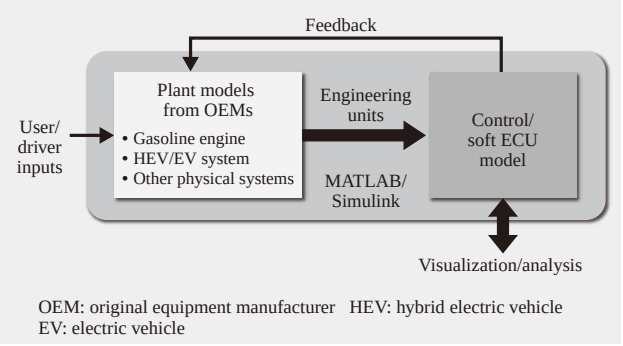

Fig. 3—Simulink-based CPS.
This simplified block diagram shows the CPS simulation model.

CPS FOR AUTOMOTIVE APPLICATIONS

Simulink-based CPS

Fig. 3 shows a simplified block diagram of a Simulink-based CPS.

The system consists of two main parts: the plant model (physical system) and the control / soft ECU model (computational system). The plant models model the physical systems of the vehicle, such as the engine for a gasoline car or the electric motor of a hybrid electric vehicle (HEV) or electric vehicle (EV). The control models implement the logic used to control the plant models in such a way that they achieve the desired functionality. These models also provide visualization of the control parameters.

Simulink is used for simulation, and communication between the plant and the control models is in terms of physical quantities or parameters represented in engineering units (Pascals for pressure or Amperes for current). The system uses feedback control whereby the user/driver inputs are applied to the plant model together with the actuator inputs determined by the control algorithm running on the control model. An important point to note here is that the link between the plant and control models is handled within the simulation system via a direct connection between the models.

In the case when different models running on different simulators are used, a co-simulation needs to be implemented to provide the connection between the simulators. Co-simulation can be achieved by a direct connection between the models if their respective simulators support such a function, or it can be achieved using a co-simulation bus[4]. The physical location of models may or may not be on the same personal computer (PC) in the case of a co-simulation bus implementation. Co-simulation also requires that the simulators support its use, or that it can be implemented using a high level programming language such as C/C++.

Use of a co-simulation bus offers several other benefits. (1) It enables a multi-domain/multi-physics system simulation where models stay in their native domain but work together. (2) The multi-domain interaction yields a robust design methodology. (3) The co-simulation bus also provides a framework for implementing the interface between the computational and physical systems of the CPS.

Virtual-CPU-based CPS

Fig. 4 shows a simplified block diagram of a virtual-CPU-based CPS.

The system architecture for a CPS based on a virtual CPU is an extension of the Simulink-based CPS architecture. In this case, the control model is implemented using a virtual CPU simulator, which can emulate a microcontroller model or virtual CPU model for the specific microcontroller specified by the user. Microcontroller models are available for the main automotive semiconductor suppliers, including Freescale Semiconductor, Inc., Renesas Electronics Corporation, and Infineon Technologies AG.

These models emulate the microcontroller hardware and can execute the same object code as the actual hardware. This simulator also gives access to internal information about the microcontroller hardware, such as the program counter, instruction cycles, interrupts, and software states.

As in a real microcontroller, these models process analog or digital voltage signals and cannot work with engineering units directly. Hence an additional sensor model (sensor models in Fig. 4) needs to be

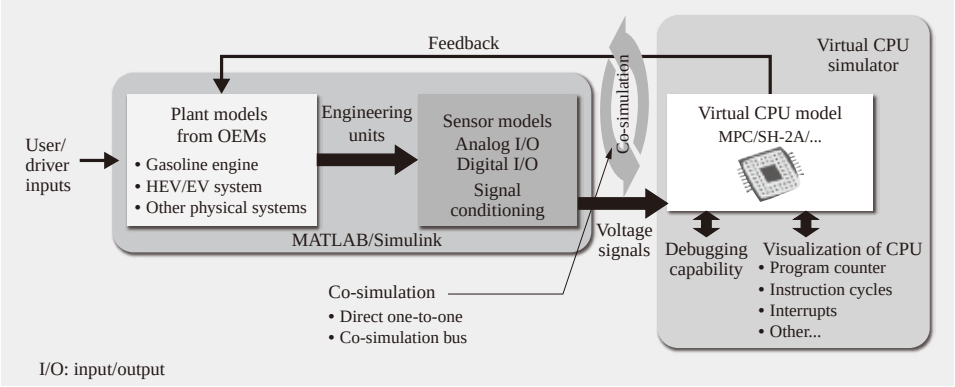

Fig. 4—Virtual-CPU-based CPS. The simplified block diagram of the CPS shows how it now includes a virtual CPU model instead of the traditional Simulink-based control model.

implemented to act as an interface and perform the required signal conditioning between the plant model and the virtual CPU model. In other words, this block handles analog and digital input and output (I/O) and converts engineering units into voltage signals, which can be processed by the virtual CPU model. The rest of the system architecture remains the same, including the feedback loop and the user/driver inputs.

Another important point to note is the use of co-simulation. The plant models and sensor models are typically implemented using MATLAB/Simulink while the virtual CPU is implemented using a dedicated virtual CPU simulator as described above. Therefore, a co-simulation is required between these simulators and can be achieved either via a direct one-to-one connection or by using the co-simulation bus. While a direct one-to-one connection is available for systems that run entirely within Simulink, this is not supported for some other simulators, in which case the co-simulation bus approach must be used. This is described in detail in the case study in the next section.

AUTOMOTIVE CPS CASE STUDY: GASOLINE FUEL PUMP SIMULATION

A gasoline fuel pump simulation model developed by Hitachi was used to investigate CPS implementation by comparing the results obtained using the Simulink-based and virtual-CPU-based approaches respectively.

Simulink-based CPS for Gasoline Fuel Pump

A physical system model (plant model) was implemented using a co-simulation bus (see Fig. 5).

The physical system of the gasoline fuel pump consists of three parts: a driver circuit simulated using an electromechanical simulator, and separate pump and inlet valve models, which are simulated using the hydraulics simulator. Using a co-simulation bus, the driver circuit was implemented in the electromechanical simulator running on one PC and the pump and inlet valve models of the actual pump were implemented in a hydraulic simulator running on a second PC.

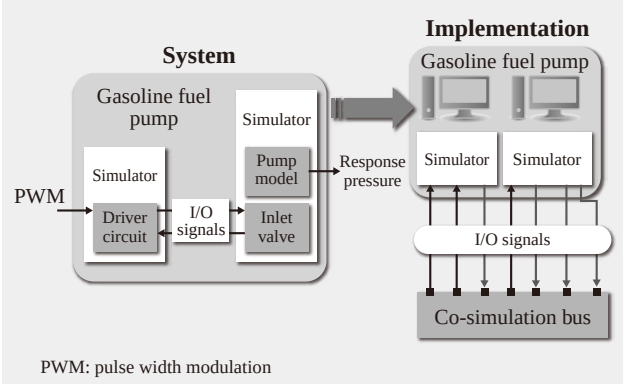

Fig. 5—Gasoline Fuel Pump Physical System. This block diagram of the gasoline fuel pump system and its simulation model represents a multi-PC implementation with a co-simulation bus.

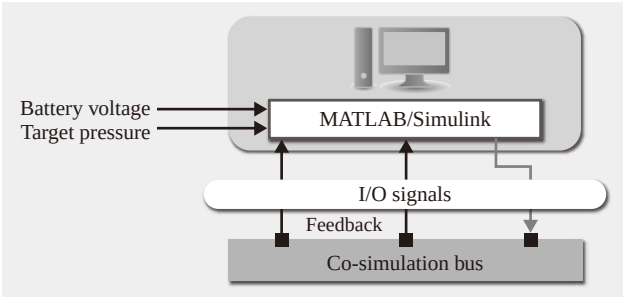

Fig. 6—Gasoline Fuel Pump Computational System. The simulation model of the gasoline fuel pump control system interfaces to a co-simulation bus.

The computational system (control model) was implemented using Simulink (see Fig. 6). Fig. 7 shows the complete integrated Simulink-based CPS for the gasoline fuel pump.

Proportional-integral (PI) control was used to generate the controller output based on the difference between the target pressure and the pressure feedback

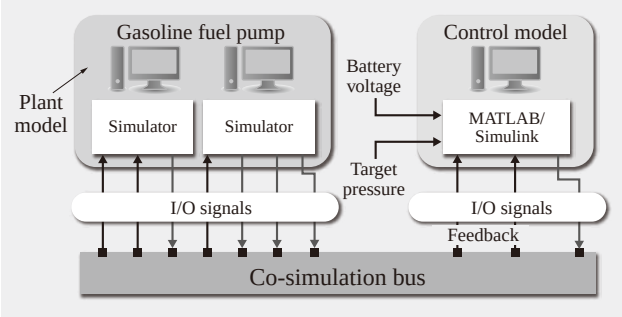

Fig. 7—Simulink-based CPS for Gasoline Fuel Pump. This Simulink-based CPS uses a co-simulation bus.

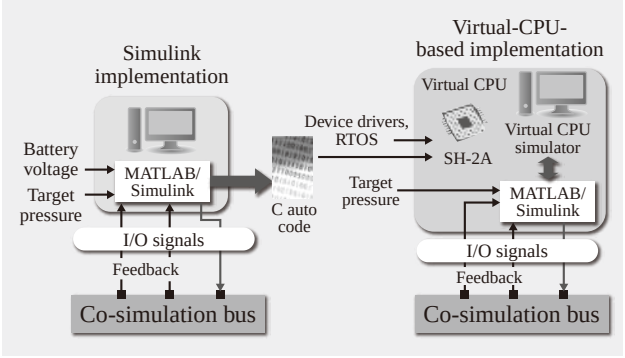

Fig. 8—Auto-code Generation for SH-2A CPU. The Simulink-based control model was converted into C code, compiled with the device drivers and RTOS, and executed on the Renesas Electronics Corporation's SH-2A CPU.

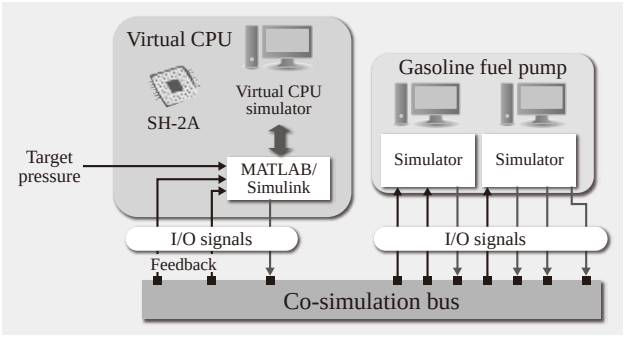

Fig. 9—Virtual CPU-based CPS for Gasoline Fuel Pump. The virtual SH-2A control model integrates with the plant model of the gasoline fuel pump via a co-simulation bus.

at a constant battery voltage input level. Since the final goal was to implement this control algorithm on hardware (virtual in this case), and as triggering the solenoid in response to the controller output costs energy, pulse width modulation (PWM) duty-cycle control was added to the original control. While the solenoid had to be fully triggered to open the inlet valve, less force was required for holding it open and therefore the current through the solenoid could be reduced. The percentage reduction in this solenoid current was determined by the PWM duty factor. Based on several tests and on experience, data were gathered on the relationship between the duty factor for various start angles and the revolutions per minute (RPM) and battery voltage. From these were produced a look-up table that could be used to obtain the duty-cycle value[1]. The PWM based control also enabled the use of the virtual CPU based approach for implementing this CPS.

Virtual-CPU-based CPS for Gasoline Fuel Pump

The first requirement for the virtual-CPU-based CPS was to obtain the object code for the software control algorithm used in the Simulink CPS. This was achieved using auto-code generation to generate the application layer of the software from the Simulink control model. The virtual CPU used was a Renesas Electronics Corporation's SH-2A*3 microcontroller. Other development included hand coding the device driver software for the SH-2A and configuring the RTOS to be used for this application (see Fig. 8).

The virtual CPU simulation itself combined two simulators: the virtual CPU simulator and MATLAB/Simulink. Accordingly, the virtual CPU simulation used a direct one-to-one co-simulation between these two simulators. In terms of the overall system, this co-simulation bus approach involved using the MATLAB/Simulink portion of the virtual CPU as an interface to the co-simulation bus since the virtual CPU simulator did not directly support the co-simulation bus. Fig. 9 shows the complete system using the virtual-CPU-based CPS for the gasoline fuel pump.

This system enabled the virtual tuning of the PI control algorithm, which will help accelerate overall pump controller development in the future.

Correlation between Simulink-based CPS and Virtual-CPU-based CPS

After adopting both the Simulink-based and virtual-CPU-based approaches for CPS implementation, the next step was to observe the relationship between the behaviors of the pump simulation using these two approaches under identical simulation conditions and configuration (see Fig. 10).

The comparison shows a very good correlation between the Simulink-based and virtual-CPU-based approaches for the gasoline fuel pump control system under identical simulation configurations (same target pressure at different engine RPMs).

*3 SH-2A is a trademark or registered trademark of Renesas Electronics Corporation.

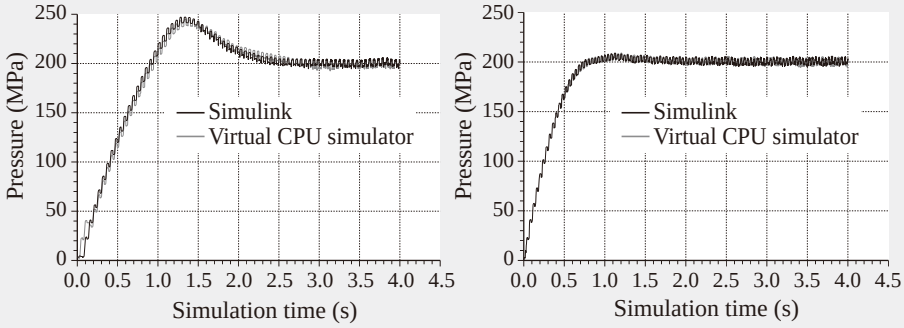

Fig. 10—Correlation between Simulink-based and Virtual-CPU-based CPS for Gasoline Fuel Pump.
This comparison shows the proportional integral (PI) control profiles for the gasoline fuel pump for two target pressure settings.

FURTHER ACHIEVEMENTS USING CPS

After the successful implementation of the CPS for a gasoline fuel pump control system at the Automotive Products Research Laboratory of Hitachi America, Ltd., a decision was made to experiment with the flexibility offered by the co-simulation bus approach for a multi-location co-simulation. The trade-off when a multi-location CPS implementation was tested was that the analysis took longer due to the slower data transfer time. The problem in this case was slower speed due to latency on the Ethernet-based Transmission Control Protocol/Internet Protocol (TCP/IP) network. At a sampling rate of 100 µs, for example, the co-simulation across offices in Japan and the USA was three times slower than the equivalent co-simulation run in the USA (see Fig. 11).

As shown in the figure, the gasoline fuel pump physical model was implemented on two PCs, one at the Japan office and one at the USA office, while the computational system model was implemented on a PC at the USA office. Apart from the problem with the data transfer time, this demonstration showed that a co-simulation could span two offices, one in Japan and the other in the USA, and proved that it was possible to achieve a flexible CPS implementation across offices in different countries.

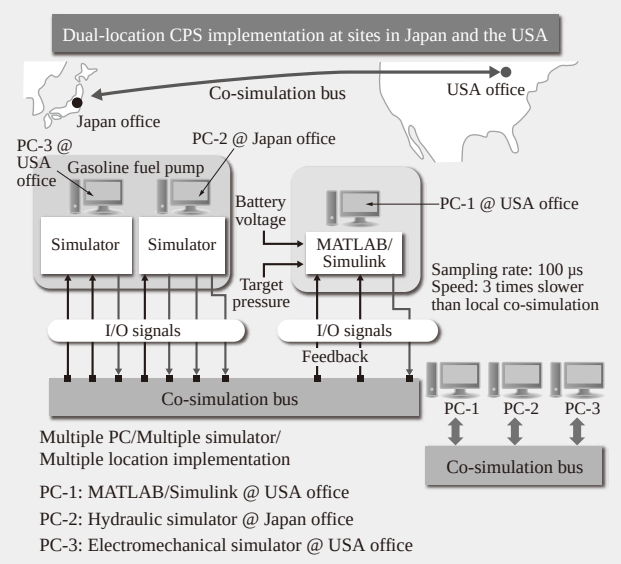

Fig. 11—International Multi-location CPS.
This distributed implementation of the gasoline fuel pump control operated a co-simulation bus between two Hitachi offices in USA and Japan.

CONCLUSIONS

This article has reviewed the basic approach to implementing a CPS adopted at the Automotive Products Research Laboratory of Hitachi America, Ltd., and described a CPS case study in the form of a gasoline fuel pump that was implemented as a multi-domain co-simulation platform.

An automotive CPS involving a gasoline fuel pump was implemented to enable virtual tuning of the PI control algorithm and accelerate the overall development process. The system behavior was first simulated using Simulink and later using a virtual-CPU-based approach. Both approaches achieved a good correlation. An international multi-location CPS implementation was also tested and validated to prove the flexible multi-location implementation of these systems as a multi-physics robust design methodology.

REFERENCES

(1) S. Phatak et al., "Cyber Physical System: A Virtual CPU Based Mechatronic Simulation," IFAC (Sep. 2010).
(2) G. Saikalis et al., "Virtual Embedded Mechatronics System," SAE Technical Paper # 2006-01-0861.
(3) J. Sinnamon et al., "Co-simulation Analysis of Transient Response and Control of Engines with Variable Valvetrains," SAE Technical Paper, 2007-01-1283.
(4) ChiasTek, 2009, http://www.chiastek.com

ABOUT THE AUTHORS

Sujit S. Phatak

Joined Hitachi America, Ltd. in 2007, and is a Researcher at the Research & Development Division, Automotive Products Research Laboratory. He is currently involved in model based development for embedded systems and virtual prototyping systems. Mr. Phatak is a member of the IEEE and the Society of Automotive Engineers (SAE).

George Saikalis, Ph.D.

Joined Hitachi America, Ltd. in 1990, and now works at the Research & Development Division, Automotive Products Research Laboratory. He is currently Vice President engaged in many aspects of automotive R&D. Dr. Saikalis is a member of the IEEE and SAE.

DJ McCune

Joined Hitachi America, Ltd. in 2003, and now works at the Automotive Products Research Laboratory. He is currently engaged in modeling and simulation of cyber physical systems. Mr. McCune is a member of the SAE.

Yasuo Sugure

Joined Hitachi, Ltd. in 1999, and now works at the Central Research Laboratory. He is currently engaged in virtual prototyping systems using microcontroller models for embedded control systems. Mr. Sugure is a member of the SAE and Institute of Electronics, Information and Communication Engineers (IEICE).