Model-Driven Requirements Validation for Automotive Embedded Software using UML

Kwan-Hyung Lee, Pan-Gi Min, Ju-Hyung Cho and Dong-Jin Lim
Department of Electronic and System Engineering
Hanyang University, Ansan
Gyeonggi-do 426-791 Korea
saicocoisa@hanyang.ac.kr

Abstract— Nowadays, A lot of effects are making a change in how the automotive field develops software. Some of these effects include: growing complexity of the embedded software; increasing amount of software in the vehicle; limited resources of development period, manpower, money; and the emergency of software-driven safety-critical systems. However, since all works including the communication about design, data, constraint, performance between supplier and original equipment manufacturer (OEM) are only document-driven, such document-driven development (DDD) process has some intrinsic problems. Obscure specification may generate errors and misunderstandings and so on. In this paper, the model-driven software development process is realized and that for a body control module (BCM) is demonstrated to verify and validate automotive embedded software requirements for solving these problems without any additional cost and equipments.

Keywords-component; BCM (Body Control Module); Model-driven; UML(Unified Modeling Language); Requirement validation

I. INTRODUCTION

Automotive systems are recently encountering with a great variety of challenges and development process of those is characterized by rising comfort and technical requirements within a competitive time-to-market. The complexity of these systems is determined by increasing functionality of the single as well as the steady growth of interactions and interconnections between various systems. On the other hand systems have to be developed tightly by using fewer financial resources and less manpower for a lot of product lines, models and variants.

On the automotive electronic systems, comfort and safety-critical requirements such as remote keyless entry (RKE) system, electric power steering (EPS) system and so on lead to an increasing number of embedded systems with various and many software solutions using several distributed electronic control units (ECU) [1]. Consequently, the period, the human resources and the costs for software testing and calibration increase considerably with system complexity, thus necessitating the adjustment of new development process. Model-driven development has been widely considered by both suppliers and OEMs because it results in a reduced development period as well as costs and provides the opportunity to verify and validate system model in an earlier phase of development.

In this paper, the model-driven requirement validation process is presented for automotive embedded software. As a case study of the realization, availability, and reliability of the process, we analyze and design the body control module (BCM) for evaluating the previously mentioned process. In validating the embedded software for BCM, we have used the tool (called Rational Rhapsody) supporting unified modeling language (UML) diagrams and automatic code generator.

The remainder of this paper is structured as follows: section II will discuss model-driven versus document-based development [9], section III and IV will describe briefly UML fundamentals and BCM. Section V will represent our proposed development process with the case study, and section VI will conclude the paper.

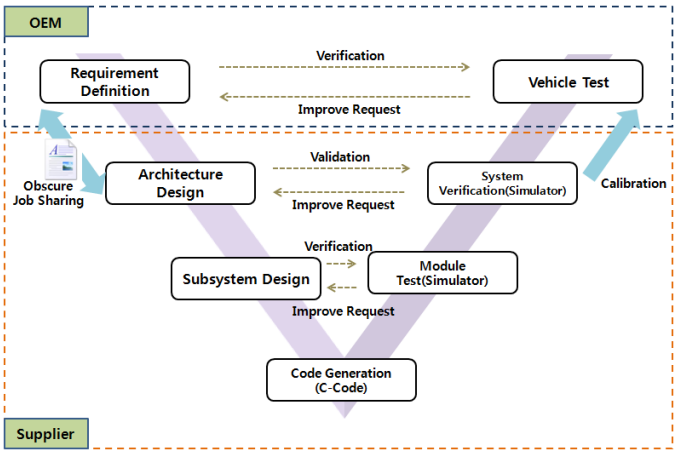

Figure 1. Document-based development process

II. MODEL-DRIVEN VS DOCUMENT-DRIVEN

A. *Document-based development*

In the document-driven development (DDD) process as shown Figure 1, the OEM submits software and hardware specification including the costs requirement and technical information to the supplier. By the supplier, the whole system

is completed depending on these requirements and then provided to the OEM [2]. Unfortunately, such the DDD process has some intrinsic weaknesses. The first one is the communication. Since all works including the communication about design, data, constraint, performance between the supplier and the OEM are only document-driven, the obscure specification may generate errors and misunderstandings and so on. The next one is the fault detection. Actually, it is very difficult to detect bug or error in the beginning stages of development even though a number of fault is inclined to occur early in the DDD process. In general, almost all faults are detected in the last phase of development. Consequently, this delays improving requirements and spends additional time and costs in correcting faults.

In the DDD process, the role of the supplier during software development has grown more serious since the supplier has to provide completed solutions and results according to the OEM's added or modified requirements. In addition, if the supplier finds the errors or bugs of the software, they have to enlighten OEM about the problems by using the method and process based on requirements.

B. *Model-driven development*

In model-driven development shown in the Figure 2, the development focuses on a system model including the requirements capture, design, implementation and test.

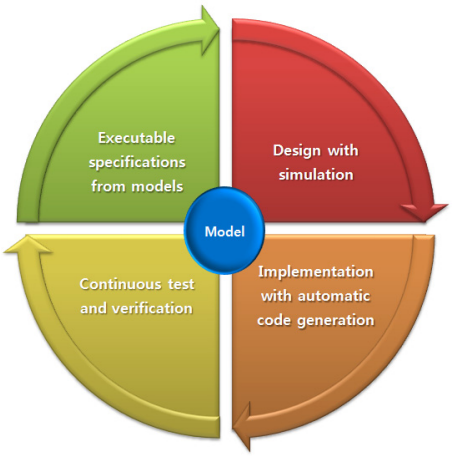

Figure 2. Model-driven development

This system model is the core of an executable specification which utilized and described in designing the model. In addition, inputs, outputs, limitations, development environment and links to the requirements are also included in the executable specification [3]. The executable specification aims at communicating the design goal clearly and allowing feasibility and compatibility analysis of the requirements by using simulations.

With model-driven development (using UML), engineers improve efficiency by [4]:

- Sharing common design environment across among project teams and companies(supplier and OEM)
- Tracing links between designs and requirements
- Combining design with testing together to continuously detect and correct errors
- Improving algorithms by using multiple domain simulation
- Automatic (embedded) code generation
- Developing and reusing test suites
- Automatic report generation

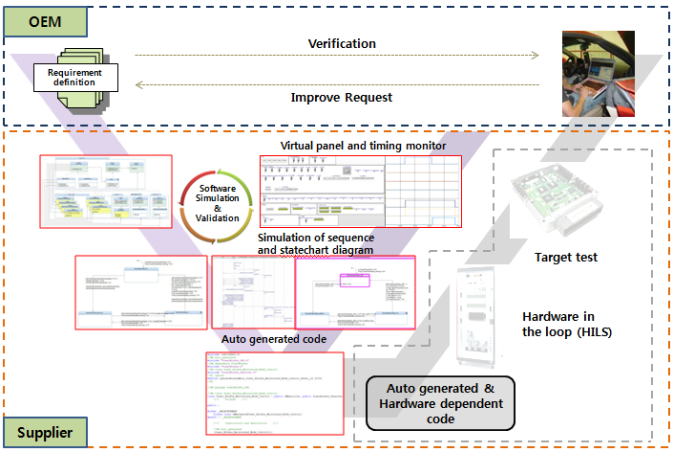

Figure 3. Model-driven development process

To complete with the modified software development process, it is essential to include the entire information about the system within the model that would conventionally be added in the later phase of the V design process. (Compare Figure 1 with Figure 3) Model-driven development requires additional human resources and information early in the modeling phase while efforts can be reduced in the next phases. The main strong point of this development process is that it is feasible to test, evaluate and generate the documentation at an early phase in the design process as shown in the lavender part of the modified V design process in Figure 3. Instead of waiting for the completed hardware components (target and equipment), the executable software already allows developers to verify and validate their model, as well as generate executable code. The entire process is thus sped up significantly and time-to-market is reduced relatively.

III. UML FUNDAMENTALS

This section provides an overview of the UML diagrams used in this paper. The goal is to provide subscribers with an idea of what is feasible with UML in developing the model-driven automotive embedded software. The subscriber is referred to the standard UML and some of the many UML books for a thorough treatment of the subject.

A. *Class diagram*

The class diagram is the most essential of all the UML diagrams. A class diagram depicts the elementary software factors that compose a system and the static relationships among them. A class stands for a collection of things that include a common set of data (called attribute) and behavior (called method) [5].

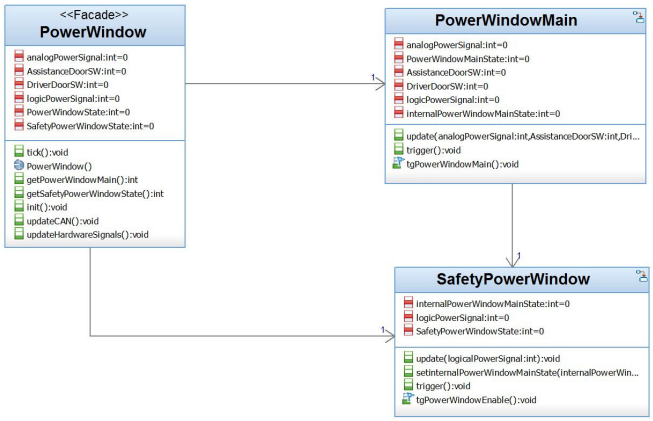

Figure 4. Class diagram

For example, in Figure 4 the class "PowerWindow" represents the general set of all power windows. Its attributes might include details such as "analogPowerSignal", "AssistanceDoorSW", "DriveDoorSW", "logicWindowState" and so on. Its methods might include "tick()", "init()", "update()" and so on. This figure illustrates the relationship (called direct association) between the class "PowerWindow" and other classes.

B. *Sequence diagram*

Sequence diagram are employed to describe the sequence of method calls made by objects of each class. A specific class may be in charge of supplying certain functionality, but sometimes it should assign sections of that functionality to other classes in the system, in the similar way that supervisors in a group assign some tasks to inferiors. Objects of a class call the methods of other objects to activate their behavior. Sequence diagrams trace and record progressively, from top to bottom, the method calls needed to invoke a piece of functionality [5]. This figure illustrates the initialization sequence of objects related to power window.

Figure 5. Sequence diagram

C. *Statechart diagram*

Statechart diagrams are used to express the internal state transitions of UML fundamentals such as a class, subsystem or the entire system. These are very helpful in depicting the operation of event-driven real-time embedded systems. The statechart in UML 2.0 has been improved to easily express hierarchical state machines, reusable states, composite states for parallel processing and high-level transitions. In addition, the statechart can be used to highlight transitions and associated input/output signals that would help developers model and debug ECU such as BCM [5]. The statechart shown in Figure 6 describes the behaviors of "PowerWindow" class shown in Figure 4.

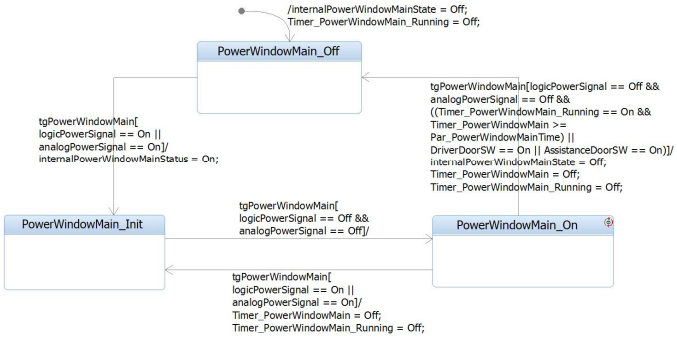

Figure 6. Statechart diagram

D. *Timing diagram*

Timing diagram is employed to illustrate the influences of method calls on objects as time progresses. In addition, the timing constraints and sequence of method calls can be observed by using the timing diagram [5].

IV. BODY CONTROL MODULE

The body control module (BCM) covers a wide range of comfort(power window, power mirror, rain sensing wipers and so on) security(remote access control, power lock and so on), lighting(exterior lighting, interior lighting and so on) and access technologies inside and outside the passenger cabin [6], [7]. Figure 7 shows the BCM block diagram.

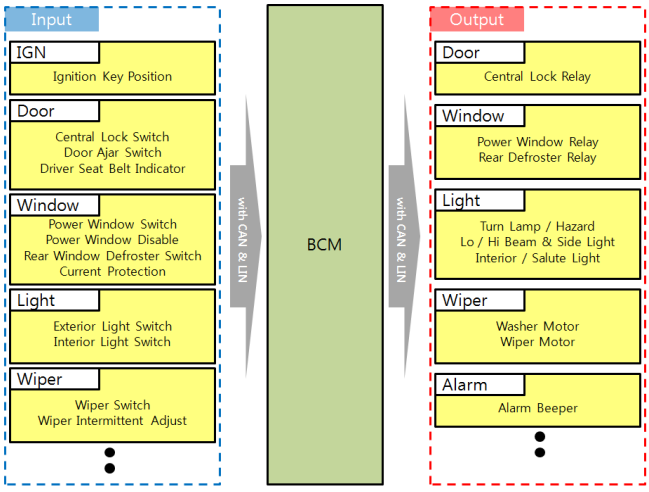

Figure 7. BCM block diagram

V. MODEL-DRIVEN SOFTWARE DEVELOPMENT PROCESS

A. *Software requirement definition*

The requirements of OEM are analyzed to collect all the software requirements. For the model-driven software development process, the following perspectives: the functional behavior description, design and implementation constraint, performance requirements are included in software requirements.

B. *Software architecture & implementation design*

The modules of BCM are composed of its functions that are represented as class and those are described by class diagram. The behavior of functions is described by statechart diagram. To enable application software to be independent of the elements related to time, the observer pattern is used. In addition, the factory pattern is used to reused the design and add the new functions easily as the software requirements of BCM vary. Figure 8 shows the architecture of BCM with observer and factory pattern.

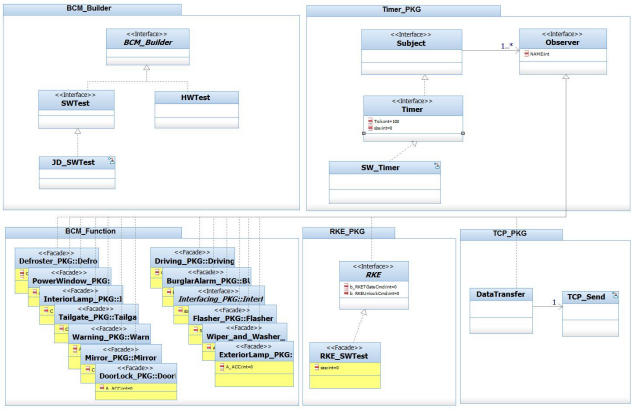

Figure 8. Software architecture design

C. *Software testing*

For software testing, simulation (called animation in Rational Rhapsody which is model-driven development environment supporting UML diagrams and code generation) is used to observe the execution of behaviors and associated definitions in the model. Model designed in this paper is simulated by executing the code which is generated from model automatically, with instrumentation, for classes, operations, and associations. Once the simulation of the model is started, simulated diagrams, which let developers observe the model as it is running and perform design-level debugging. Developers can step through the model, set and clear breakpoints, inject events, and generate an output trace.

It is good practice to test the model incrementally using model execution, which developers have practiced in earlier stage. The pieces of model can be simulated as it developed. This gives developers the opportunity to determine whether the model meets the requirements and find defects early on. Then the entire model can be tested. In this way, the model is iteratively built, and then entire model validation is performed with each iteration. For example, in the simulated statechart shown in Figure 9, "PowerWindowMain_off" state becomes inactive (olive), and "PowerWindowMain_Power_On" state become active (magenta) when the condition of olive transition is true.

Panel diagram (not standard UML) and timing monitor software (just monitoring) provide developers with a number of graphical control elements that they can use to create a graphical user interface(GUI) to monitor and regulate an application. Each control element is bound to be a model element (attribute, event, state). During simulation, the simulated panel diagram and timing monitor software can be used to monitor (read) and regulate (change values, send event) the application (BCM embedded software). Figure 10, 11 illustrates the panel diagram and timing monitor software of the entire BCM model.

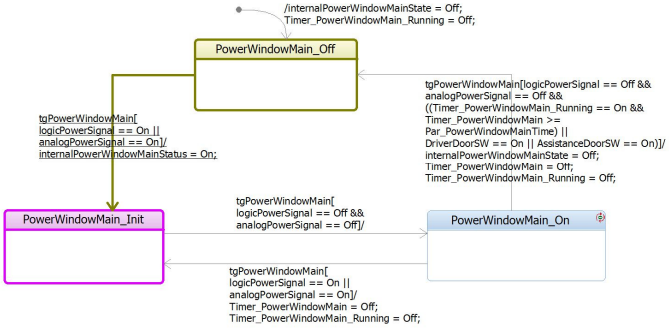

Figure 9. Simulation of statechart diagram

Figure 10. Panel diagram

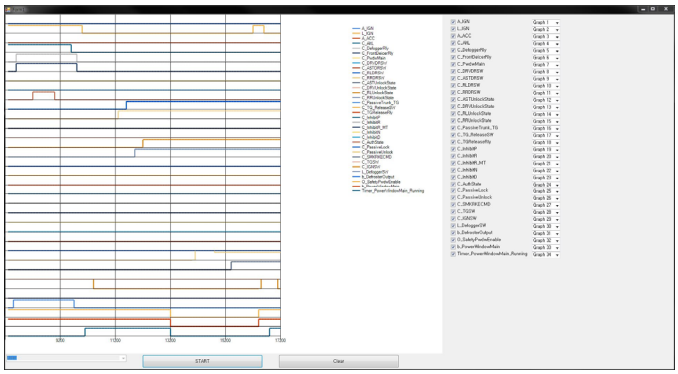

Figure 11. Timing monitor software

VI. CONCLUSION

The model-driven software development process using UML and executable model offers many advantages against the conventional document-based software development process, especially concerning error correction, verification, and validation. It is being driven by shorter product development cycles, the increased complexity of the software, the expectation of product quality and reduced cost by using computer-aided design methodologies earlier in the development cycle.

The model-driven approach is increasingly being used in the automotive software development. However, it is used only partly in the OEMs. It is even rarer that suppliers develop automotive embedded software using this process. This is because the model-driven approach requires suitable software model as well as simulation tool or program. And, it also requires so much additional cost and manpower.

In this paper, automotive embedded software have been demonstrated through the model-driven development process without any additional cost and equipments such as data acquisition software and hardware, high-performance computer and real-time OS and so on for suppliers. In doing so, we could significantly cut down the time, cost and manpower of the development and especially, reduce the efforts to validate software requirements in side of suppliers.

ACKNOWLEDGMENT

This work was supported by the GRRC program of Ansan city and Shinchang Inc.. [(GRRC Hanyang 201101400020004), Development of ubiquitous sensor network software using model-driven development method].

REFERENCES

[1] A. Wagener, H. Kabza, C. Koerner, and P. Seger, “Model-based Driventrain Development and Rapid Prototyping for a Hybrid Electric Car,” Automotive and Transport Technology congress, 2001.

[2] Won Hyun Oh, Jung Hun Shin, Woo Sub Kim, Hyoung Geun Kwon and Sang Gil Lee, “Model Based Development Process (MBDP) for the Embedded System in Vehicle,” SAE Technical Paper Series 2008-01-0748, Detroit, 2008.

[3] Peter J. Schubert, Lev Vitkin, and Frank Winters, “Executable Specs: What Makes One, and How are They Used?” SAE Technical Paper Series 2006-01-1357, Detroit 2006.

[4] Paul F. Smith, Sameer M.j Prabhu and Jonathon Friedman, “Best Practices for Establishinga Model-Based Design Culture,” SAE Technical Paper Series 2007-01-0777, Detroit 2007.

[5] OMG hompage, http://www.uml.org.

[6] Freescale Semiconductor hompage, http://www.freescale.com/files/training_pdf/WBNR_LA_AUTO_BCM_SPANISH.pdf.

[7] Jungduk Son, Insik Lee, “Research on software verification method for Body Control Module,” KSAE05-F0196.

[8] Jianrui Wang and Akhil Kumar, “A framework for Document-Driven Workflow Systems,” Lecture Notes in Computer Science, 2005, Volume 3469/2005, 285-301