A Simulink to UML Transformation Tool for Embedded Control Software Design

Tatsuya Kamiyama, Takahiro Soeda, Myungryun Yoo, and Takanori Yokoyama

Abstract— **The paper presents a tool to transform a Simulink model to a UML model. The embedded control software design process consists of the control logic design phase and the software design phase. MATLAB/Simulink is widely used to build controller models in control logic design. On the other hand, UML is generally used to build software models, which are modified or composed with other UML models during software design. To shift from the control logic design phase to the software design phase smoothly, we have developed a transformation tool to transform a Simulink model to a UML model. The tool generates class diagrams and object diagrams of UML. Each class or object corresponds to a subsystem block of the Simulink model, which represents a part of control logic. So a part of the control logic can be represented and reused as a class. We have applied the transformation tool to a number of Simulink models and have found it useful for embedded control software design.**

Index Terms—**Embedded control system, model-based design, UML, simulink.**

I. Introduction

The embedded control software development process consists of the control logic design phase and the software design phase. In the control logic design phase, model-based design with a CAD/CAE tool such as MATLAB/Simulink[1] has become popular. In model-based design, a controller model is designed with block diagrams and verified by simulation.

In the software design phase, UML is widely used to represent a software model. We build a software model to implement the controller model, taking account of not only functional properties but also non-functional properties. An embedded control system is a hard real-time system with timing constraints and the control software is usually executed in a preemptive multi-task environment. So we have to design the software to meet timing constraints and to correctly execute the control logic in the preemptive multi-task environment. UML provides a number of diagrams that is useful for both functional design and non-functional design.

Research on aspect-oriented modeling to separate non-functional properties from functional properties has been done. Non-functional properties are represented as aspects and woven into the program with functional properties. Wehrmeister et al. have discussed model level aspects for non-functional requirements of distributed embedded real-time systems[2]. We have presented a design method with aspect patterns for embedded control systems[3]. To apply UML-based aspect-oriented design to embedded control systems, a Simulink model built in control logic design should be transformed to a UML model.

Research on model-driven software development such as OMG's MDA (Model-Driven Architecture)[4] has been also done. UML is generally used to represent a software model in model-driven software development. In MDA, a PIM (Platform Independent Model) is transformed to a PSM (Platform Specific Model) and both PIM and PSM are represented in UML. A Simulink model is a kind of PIM because a Simulink model represents implementation-independent control logic. To apply model-driven software development techniques and transformation tools to embedded control software design, Simulink models should be transformed to UML models.

A control system consists of various software modules. Simulink models are suitable to represent control logics such as feedback control and feedforward control. On the other hand, UML is suitable for some software modules such as application modules with procedural algorithms, input and output modules and network communication modules. To integrate a controller model with software models, the controller model should be translated to a UML model before the integration because UML is suitable for software design.

Ramos-Hernandez et al. have presented a tool that transforms a Simulink model to a UML model[5][6]. The tool generates interface classes corresponding to each blocks of the Simulink model. A dependency is generated corresponding to a line that connects blocks. An interface class *Demux* is inserted if a junction of lines exists. Müller-Glaser et al. have presented a method to transform a Simulink model to a UML model, in which each object of the generated UML model corresponds each element of the Simulink model[7][8][9]. Blocks, lines and junctions are represented as objects in the UML model. However, UML models generated by their tools are not suitable for software design, in which software models are modified to meet non-functional requirements. The generated UML models represent just the structures of the block diagrams of the Simulink models, not represent the control logics.

The goal of the research is to develop a tool that transform a Simulink model to a UML model that represents the control logic of the Simulink model. To achieve the goal, we define rules to transform a Simulink model to a UML model and

Manuscript received April 10, 2012; revised May 5, 2012. This work is supported in part by KAKENHI (20500037).

T. Kamiyama and T. Soeda were with Graduate School of Engineering, Tokyo City University, 1-28-1, Tamazutsumi, Setagaya-ku, Tokyo 158-8557 Japan. He is now with A&D Company, Limited, 1-243, Asahi, Kitamoto-shi, Saitama-ken 364-8585 Japan.

M. Yoo and T. Yokoyama are with Department of Computer Science, Tokyo City University, 1-28-1, Tamazutsumi, Setagaya-ku, Tokyo 158-8557 Japan (e-mail: {yoo, yokoyama}@cs.tcu.ac.jp).

develop a transformation tool based on the rules.

The tool generates class diagrams and object diagrams of UML. Each class or object corresponds to a subsystem block of the Simulink model, which represents a part of control logic. So a part of the control logic can be represented and reused as a class. A generated class diagram and a generated object diagram are functional models that represent functional properties. We can modify the functional model to build an implementation model with non-functional properties during software design.

The rest of the paper is organized as follows. Section II describes a control software development process with model transformation. Section III describes correspondence between a Simulink model and a UML model. Section IV describes a model transformation tool we have developed. Section V shows examples of model transformation with the tool. Finally, Section VI concludes the paper.

II. Control Software Development Process

Fig. 1 shows the embedded control software development flow that consists of the control logic design phase, the software design phase and the programming phase. In the control logic design phase, we build a Simulink model that represents control logic.

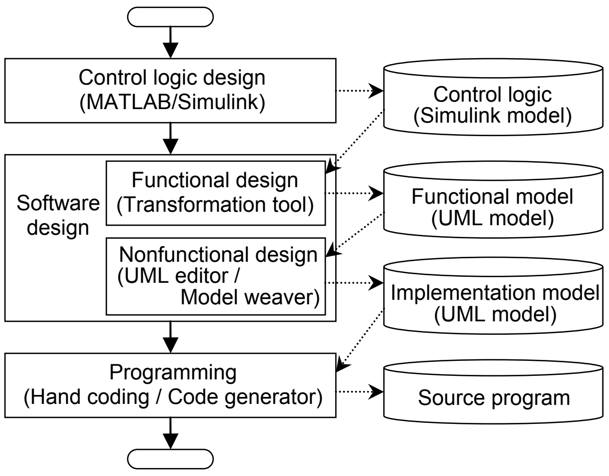


Fig. 1. Development flow of embedded control software.

Then, in the software design phase, we build a software model in UML. Software design can be divided into functional design and non-functional design. We transform a Simulink model into a UML model in functional design. We call the UML model the functional model because the model represents implementation-independent control logic without non-functional properties. We present a tool for the transformation in this paper.

In non-functional design, we build an implementation model in UML taking account of non-functional properties. For example, we design the task structure, scheduling policy, task priorities to meet timing constraints. We also add mechanisms such as synchronization, mutual exclusion and inter-task communication to the model so that the software correctly executes in the preemptive multi-task environment.

We have presented an aspect-oriented non-functional design method, in which non-functional properties for embedded control software are represented as aspect patterns[3]. For example, mechanisms for triggering methods (time-triggered or event-triggered)[10], synchronizations and inter-task communications are defined as aspect patterns. We select aspect patterns for the software and weave them into the functional model with a model weaver. Then we get an implementation model as an output of the model weaver.

Finally, in the programming phase, we write source program to implement the software model. The source program may be automatically generated from the software model[11] or the controller model[12].

III. Simulink Model and UML Model

A. Simulink Model

We define rules to build a Simulink model and to transform the Simulink model to a UML model. A Simulink model is a kind of block diagram. We have presented a method to transform a block diagram to a class diagram of UML [13], [14]. We define the transformation rules of the tool based on the method.

A Simulink model must be built according to the rules before the transformation. A Simulink model of a control system usually consists of a plant model and a controller model. The target of the transformation is the controller model.

A controller model can be layered. Fig. 2 shows an example of a layered Simulink model. Subsystem blocks are used to build a layered Simulink model. This example has two subsystem blocks at the higher layer. The details of the calculation algorithm of each subsystem block are described at the lower layer.

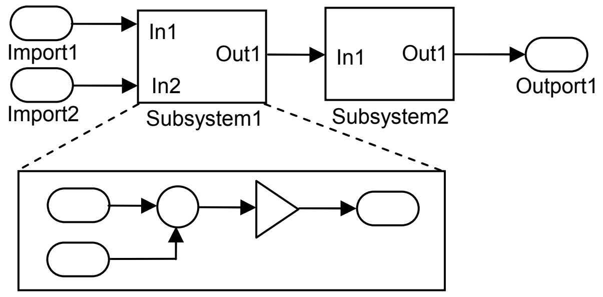


Fig. 2. Layered Simulink model.

To make subsystem blocks of the higher layer be reusable, data presented at the higher layer should be reasonable physical values such as input values, output values, observed status variables of the system, and target values of the system [12]. These data are rarely deleted or added even if the detailed control logic is modified. The modification is done just at the lower layer and the higher layer model can be reused without modification.

The target of the transformation is the higher layer model, which consists of subsystem blocks, inport blocks and outport blocks. An inport block is used to represent an inputs and an outport block is used to represent an output. The transformation tool deals with a Simulink model that consists of the three kinds of blocks.

Data presented at the higher layer must be named. Fig. 3 shows the method to declare data names in a Simulink model. The name of an inport block must be the name of the data from the inport block. The name of an outport block inside of a subsystem block must be the name of the data from the subsystem block. The data name is presented in the subsystem block. A data type can be declared as a name of a line corresponding to the data.

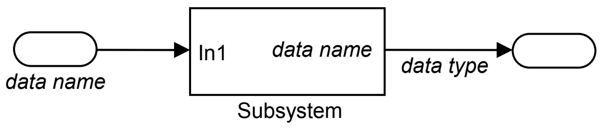


Fig. 3. Naming of data.

Fig. 4 shows an example Simulink model, which is a higher layer model of a part of automotive control logic. The Simulink model inputs *engine revolution*, *engine status* and *accelerator opening*, and outputs *throttle opening*. The model consists of three inport blocks for *engine revolution*, *engine status* and *accelerator opening*, two subsystem blocks to calculate *torque* and *throttle opening*, and an outport block for *throttle opening*. The data type of *torque* and *throttle opening* is *uint16* (16 bit unsigned integer). The details of the calculation of *torque* and *throttle opening* are described in the lower layer models. The calculations are periodically executed in the control period.

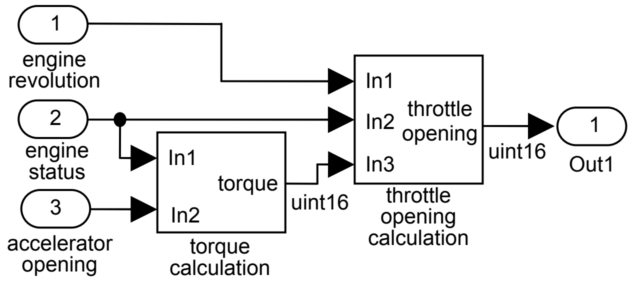


Fig. 4. An example Simulink model.

B. UML Model

The output of the transformation tool is a functional model represented in UML: class diagrams and object diagrams. A class diagram is generally used to represent the structure of object-oriented software. An object diagram is useful for the embedded control system with static structure. Most objects are statically created at the initialization process, not dynamically created, in embedded control systems with limited resources such as automotive control systems. So we think not only class diagrams but also object diagrams are useful for embedded control software design.

The model transformation method is based on the method we have presented[13][14]. The method identifies objects referring to the data flow of the block diagram that represents control logic. The data presented at the higher layer model are candidates for objects.

Fig. 5 shows the base abstract class named *ValueObject* for functional models. The class has an attribute named *value* for storing the data and a method named *update* for updating (calculating and storing) the value. If values stored in other objects are required to calculate its own value, the required values are obtained by calling methods *get* of the relevant objects. Concrete classes of functional models are subclasses of the base abstract class.

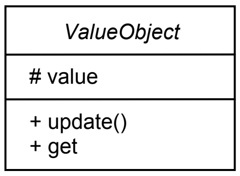


Fig. 5. Base abstract class of data object.

Fig. 6 shows the class diagram of the functional model corresponding to the Simulink model shown by Figure 4. The class diagram consists of five classes: *EngineRevolution*, *EngineStatus*, *AcceleratorOpening*, *Torque* and *ThrottleOpening*. The association named *cons* means that the following class consumes the value of the preceding class. For example, method *update* of class *Torque* gets the value of *EngineStatus* and the value of *AcceleratorOpening*, calculates its own value, and stores the calculated value in attribute *torque*. Method *update* of class *Torque* and method *update* of class *ThrottleOpening* are called periodically in the control period.

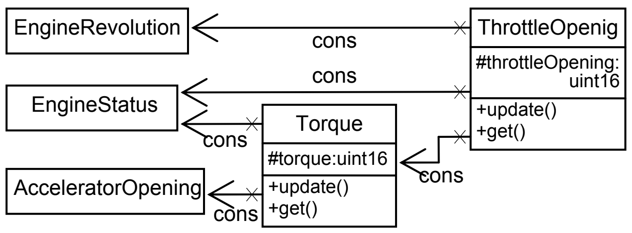


Fig. 6. An example class diagram.

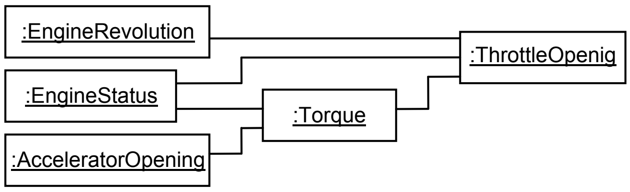


Fig. 7. An example object diagram.

Fig. 7 shows the object diagram of the functional model corresponding to the Simulink model shown by Figure 4. The object diagram consists of five objects of the classes shown in Figure 6. Just one object of each class exists because each block of the Simulink model is just one instance of the block.

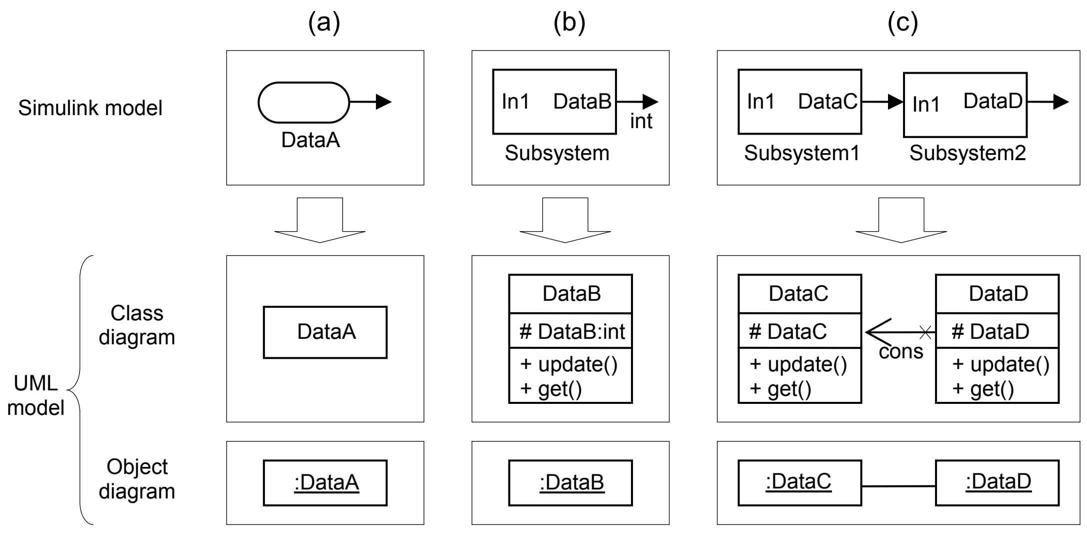


Fig. 8. Transformation rules from Simulink model

The objects are connected with links that correspond to the lines of the Simulink model shown by Figure 4 and to the associations of the class diagram shown by Figure 6.

IV. Simulink to UML Transformation Tool

A. Transformation Rules

Fig. 8 shows rules to transform elements of a higher layer Simulink model to elements of a UML model. The transformation is done as follows according to the rules.

- Classes and objects are generated referring to the data of the Simulink model.
- Associations and links are generated referring to the lines of the Simulink model.

A class and an object correspond to a data in a Simulink model. A data presented in a higher layer Simulink model is an output of an inport block or a subsystem block. Column (a) of Figure 8 shows the rule to transform a data from an inport block to a class of the class diagram and an object of the object diagram. A class with just the name is generated referring to the data. The name of the generated class is the name of the data (*DataA* in this case). An object of the object diagram is also generated referring to the data.

Column (b) of Figure 8 shows the rule to transform a data from a subsystem block to a class of the class diagram and an object of the object diagram. A class with the name, attributes and methods is generated referring to the data. The name of the generated class is the name of the data (*DataB* in this case). The name of the attribute is also the name of the data. If the data type (*int* in this case) is declared in the Simulink model, the data type is added to the attribute of the class. The generated class has method *update* and method *get*. An object of the object diagram is also generated referring to the data.

Column (c) of Figure 8 shows the rule to transform a line between subsystem blocks (*Subsystem1* and *Subsystem2* in this case) to an association between classes (*DataC* and *DataD* in this case) of the class diagram and a link of the object diagram. The preceding block with a line can be an inport block. The association *cons* is generated referring to the line. the link is also generated referring to the line.

B. Transformation Rules

Figure 9 shows the processing of the transformation tool. The transformation tool inputs a mdl file, which is a file to store information on a Simulink model. The tool analyzes the input file and extracts information needed for the transformation. Then the tool transforms the information to elements of UML according to the transformation rules described in Section IV.*A*. Finally, the tool generates XMI files of a class diagram and an object diagram. XMI is a standard file format of UML[15].

V. Experimental Evaluation

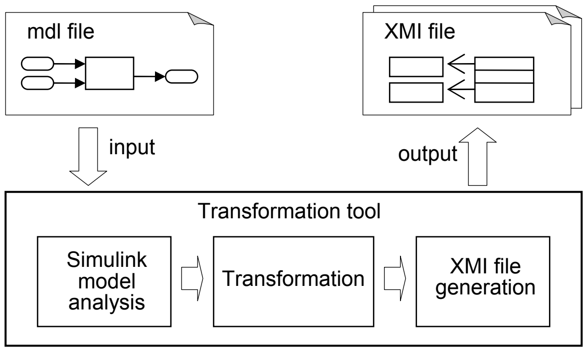


Fig. 9. Transformation tool.

We have applied the transformation tool to a number of Simulink models such as a fuel injections system, a hybrid electric vehicle system and a stepping motor control system provided by the MathWorks, Inc.[1]. The original Simulink models do not conform to the layering rules described in section III.*A*, so we modified the original models into layered models that conform to the layering rules before the transformation.

We show the case of a hybrid electric vehicle system. The example hybrid electric vehicle is a series-parallel hybrid electric vehicle that consists of a gasoline engine and an electric motor. Figure 10 shows the Simulink model, the generated class diagram and the generated object diagram.

The Simulink models used in the experiments represent just control logics. They are built by control engineers without considering implementation. After layering the original models, the transformation tool successfully transforms the layered Simulink models to class diagrams and object diagrams. So we think the transformation tool can be applied to embedded control software design.

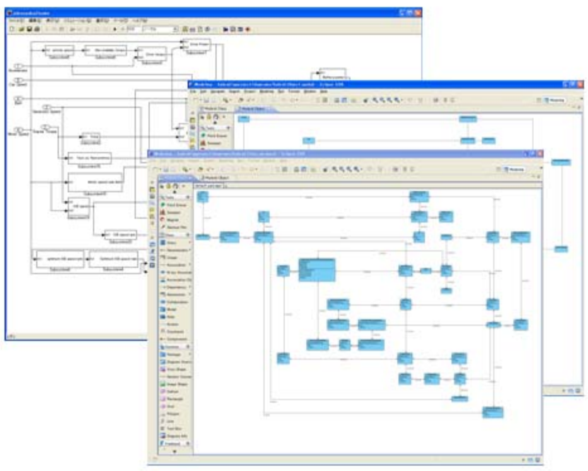


Fig. 10. Models of a hybrid electric vehicle

VI. Conclusion

We have developed a tool that transforms a Simulink model to a UML model. The tool generates class diagrams and object diagrams. Each class or object corresponds to a subsystem block in a Simulink model that represents a part of control logic. We have also applied the tool to a number of Simulink models and successfully have got the corresponding UML models.

To apply the transformation tool, we have to build a layered Simulink model, the higher layer of which consists of just subsystem blocks, inport blocks and outport blocks. To make the transformation more efficient, we are going to add a function to support the layering of the Simulink model. We are also going to extend the tool to generate behavioral UML diagrams such as sequence diagrams.

References

[1] The Math Works Inc, http://www.mathworks.com/.

[2] M. A. Wehrmeister, E. Freitas, C. E. Pereira, and F. R. Wagner, "An Aspect-Oriented Approach for Dealing with Non-Functional Requirements in a Model-Driven Development of Distributed Embedded Real-Time Systems," in *Proc. 10th IEEE Int. Symposium on Object and Component-Oriented Real-Time Distributed Computing,* pp. 428-432, 2007.

[3] T. Soeda, Y. Yanagidate, and T. Yokoyama, "Embedded Control Software Design with Aspect Patterns," in *Proc. Int. Conf. on Advanced Software Engineering and Its Applications*, pp. 34-41,2009.

[4] *MDA Guide,* version 1.0.1, Object Management Group, 2003.

[5] D. N. Ramos-Hernandez, P. J. Fleming, and J. M. Bass, "A Novel Object-Oriented Environment for Distributed Process Control Systems," *Control Engineering Practice,* vol. 13, pp. 213-230, 2005.

[6] D. N. Ramos-Hernandez, I. Zubuzarreta, P. J. Freming, S. Bennett, and J. M. Bass, "Towards a Control Software Design Environment Using a Meta-Modeling Technique," in *Proc. 15th IFAC World Congress,* pp. 255-260, 2002.

[7] K. D. Müller-Glaser, G. Frick, E. Sax, and M. Kühl, "Multiparadigm Modeling in Embedded Systems Design," *IEEE Trans. Control Systems Technology,* vol. 12, no. 2, pp. 279-292, 2004.

[8] M. Kühl, C. Reichmann, I. Prötel, and K. D. Müller-Glaser, "From Object-Oriented Modeling to Code Generation for Rapid Prototyping of Electronic Systems," in *Proc. 13th IEEE Int. Workshop on Rapid System Prototyping,* pp. 108-114, 2002.

[9] M. Kühl, B. Spitzer, and K. D. Müller-Glaser, "Universal Object-Oriented Modeling for Rapid Prototyping of Embedded Electronic Systems," in *Proc. 12th IEEE Int. Workshop on Rapid System Prototyping,* pp. 149-154, 2001.

[10] H. Kopetz, "Should Responsive Systems be Event-Triggered or Time-Triggered," *IEICE Trans. Information and Systems,* vol. E76-D, no. 11, pp. 1325-1332, 1993.

[11] F. Narisawa, H. Naya, and T. Yokoyama, "A Code Generator with Application-Oriented Size Optimization for Object-Oriented Embedded Control Software," in *Object-Oriented Technology: ECOOP'98 Workshop Reader,* Springer LNCS-1543, pp. 507-510, 1998.

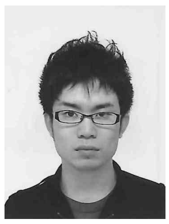
[12] K. Yoshimura, T. Miyazaki, T. Yokoyama, T. Irie, and S. Fujimoto, "A Development Method for Object-Oriented Automotive Control Software Embedded with Automatically Generated Program from Controller Models," in *Proc. 2004 SAE World Congress,* 2004, 2004-01-0709.

[13] T. Yokoyama, H. Naya, F. Narisawa, S. Kuragaki, W. Nagaura, T. Imai, and S. Suzuki, "A Development Method of Time-Triggered Object-Oriented Software for Embedded Control Systems," *Systems and Computers in Japan,* vol. 34, no. 2, pp. 338-349, 2003.

[14] T. Yokoyama, "An Aspect-Oriented Development Method for Embedded Control Systems with Time-Triggered and Event-Triggered Processing," in *Proc. 11th IEEE Real-Time and Embedded Technology and Application Symposium,* 2005, pp. 302-311.

[15] *XML Metadata Interchange Specification,* version 2.0.1, Object Management Group, 2005.

Tatsuya Kamiyama was born in 1987. He received his B.E. degree and M.E. degree from Tokyo City University in 2010 and 2012 respectively. He joined A&D Company, Limited in 2012.



Takahiro Soeda received his B.E. degree and M.E. degree from Tokyo City University in 2009 and 2012 respectively.

Myungryun Yoo received B.E from Andong National University, Korea in 1994, M.S from Pohang Unversity of Science & Technology, Korea in 1996 and Ph.D. degree from YeoungNam University, Korea in 2002. She joined Andong Information Technical Junior College as Assit. Professor in 1996. She received Ph.D. degree from Graduate School of Information, Production & Systems, Waseda University, Japan in 2006. She joined Tokyo City University as Associate Professor in 2006. Her research field is Real-Time System, Scheduling, Mutimedia System, etc.

Takanori Yokoyama received his B.E. degree, M.E. degree, and Dr. degree in information science from Tohoku University in 1981, 1983, and 2002 respectively. He joined Hitachi, Ltd. in 1983. He joined Musashi Institute of Technology in 2004. He is now a professor of Tokyo City University. His research interest includes embedded systems, distributed systems and software engineering. He is a member of IEEE, ACM, IPSJ and IEICE.