A Model Transformation Environment for Embedded Control Software Design with Simulink Models and UML Models

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

***Abstract*—The paper presents a model transformation environment to transform a Simulink model to a UML model. The embedded control software development process consists of the control logic design phase and the software design phase. MATLAB/Simulink is widely used to build a controller model in the control logic design phase. On the other hand, UML is widely used in the software design phase. To shift from the control logic design phase to the software design phase smoothly, we have developed a model transformation tool to transform a Simulink model to a UML model. The UML model generated by the transformation tool consists of classes that encapsulate data and calculation methods of the data. To improve the reusability of the classes, the Simulink model should be well-layered. We have also developed a layering support tool for efficient layering of the Simulink model. We have applied the model transformation environment to a number of Simulink models and found it useful for embedded control software design.**

***Index Terms*—embedded software, model-based design, software tools, control systems, real-time systems.**

I. Introduction

The embedded control software development process can be divided into the control logic design phase and the software design phase. In the control logic design phase, control engineers design control logic, just considering functional properties. In the software design phase, software engineers design the software structure and behavior to implement the control logic, considering not only functional properties but also nonfunctional properties.

Model-based design has become popular in embedded control software design, especially in the automotive control domain. In model-based design, a CAD/CAE tool such as MATLAB/Simulink[1] is used to design control logic. A controller model is designed with block diagrams and verified by simulation, and source code can be generated from the controller model. However, such CAD/CAE tools are not sufficient for software design. Sangiovanni-Vincentelli and Di Natale pointed out the shortcomings of the tools: lack of separation between the functional and architecture model, lack of support for defining the task and resource model, lack of modeling for analysis and backannotation of scheduling-related delays and lack of sufficient semantics preservation[2]. CAD/CAE tools such as MATLAB/Simulink should be used for just control logic design, not for software design. Software modeling languages such as UML should be used for software design.

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 those models, Simulink models should be transformed to UML models before the integration because UML is suitable for software design. UML provides a number of kinds of diagrams, which are useful for not only functional design but also nonfunctional design.

Ramos-Hernandez et al. have presented a tool that transforms a Simulink model to a UML model[3][4]. The tool generates classes corresponding to each blocks of the Simulink model. A dependency is generated corresponding to a line that connects blocks. 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[5][6]. Blocks, lines and junctions are represented as objects in the UML model. Sjöstedt et al. have presented a tool that transforms a Simulink model to a UML model[7]. The tool generates composite structure diagrams as structural models and activity diagrams as behavior models. However, classes of UML models generated by those tools may not be reusable because each class represents just an element of the original Simulink models. To improve the reusability of the software, a UML model should be structured based on the object-oriented concept.

The goal of the research is to develop a model transformation environment, which transforms a Simulink model to a reusable UML model. To achieve the goal, we define rules to transform a Simulink model to a UML model based on the design method for the time-triggered object-oriented software[8][9][10]. A control systems designed by the design method consists of objects that represent reasonable physical quantities in the control logic, for example, input values, output values, observed values, estimated values and desired values. We develop a model transformation tool based on the defined rules. The tool generates UML structural models and behavioral models: class diagrams, object diagrams and sequence diagrams. Each class of the generated UML model corresponds to a reasonable physical quantity in the Simulink model. The method of the class corresponds to the subsystem block in the layered Simulink model. So a subsystem block calculating a physical quantity can be reused as a class. We also develop a layering support tool for efficient layering of

Manuscript received December 14, 2011; revised January 10, 2012.

M. Tamura, T. Kamiyama and T. Soeda are with Graduate School of Engineering, Tokyo City University, 1-28-1, Tamazutsumi, Setagaya-ku, Tokyo 158-8557 Japan (e-mail: {g1181524, g1081516, g0981517}@tcu.ac.jp).

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).

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

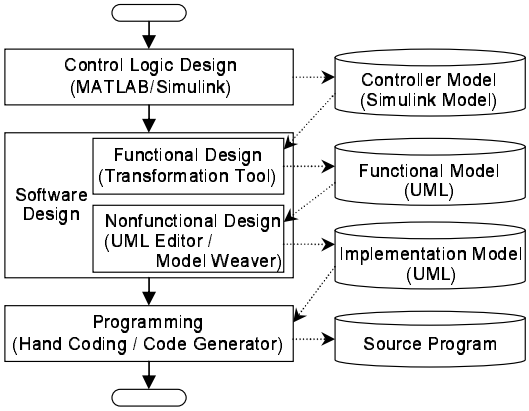

Fig. 1. Development Flow of Embedded Control Software

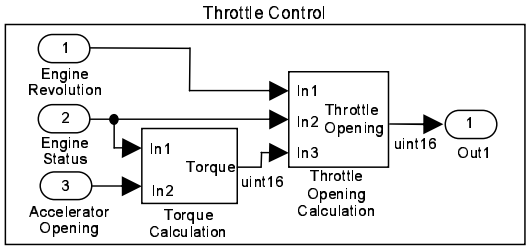

Fig. 2. Example Simulink Model

a Simulink model.

The rest of the paper is organized as follows. Section II describes the control software development process with model transformation. Section III describes details of the model transformation environment and shows model transformation examples. Section IV describes the experiments of the model transformation environment. Finally, Section V concludes the paper.

II. Control Software Development Process

A. Software Development Flow

Figure 1 shows the embedded control software development flow, which 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 a control system. A Simulink model of a control system usually consists of a plant model and a controller model. The controller model represents control logic. Figure 2 shows an example Simulink model, which is a throttle control part of an automotive control system. 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. Figure 2 shows the higher layer model of the layered Simulink model. 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.

In the software design phase, we build a software model in UML to implement the controller model. Software design can be divided into functional design and nonfunctional design. In functional design, we transform a Simulink model into a UML model. We call the UML model the functional model because the model represents implementation-independent control functionalities. A functional model may be integrated with other models built in UML. The details of functional design are described in Section II-B. In nonfunctional design, we build an implementation model taking account of nonfunctional properties. The details of nonfunctional design are described in Section II-C.

Finally, in the programming phase, we write source program to implement the implementation model. The source program may be automatically generated from the software model[11] or the controller model[10].

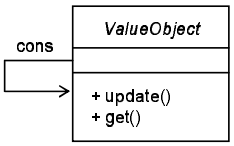

Fig. 3. Base Abstract Class of Value Object

B. Functional Design

We transform a Simulink model into a functional model represented in UML with a model transformation environment, details of which are described in Section III. Our model transformation method is based on the design method of the time-triggered object-oriented software[8][9][10]. A control system designed by the design method consists of objects that correspond to data in the block diagram. The design method identifies objects referring to the data flow of the block diagram representing control logic. The important data representing reasonable physical quantities, such as input values, output values, observed values, estimated values and desired values, are candidates for objects, because those values are rarely deleted or added even if the detailed control logic is modified[10].

The object representing data is called the value object. The value object encapsulates the data and the calculation method of the value of the data. Figure 3 shows the base abstract class named *ValueObject*. The class has method *update* that calculates its own value and stores the value in an attribute of the class. 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 value objects are subclasses of the base abstract class.

Figure 4 shows the class diagram of the functional model corresponding to the Simulink model shown by Figure 2. The class diagram consists of six classes: *Engine revolution*, *EngineStatus*, *AcceleratorOpening*, *Torque*, *ThrottleOpening* and *ThrottleControl*. *ThrottleControl* is a whole object, which corresponds to the whole Simulink model shown by Figure 2. 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 *exec* of *ThrottleControl*, which is periodically executed in the control period, calls method *update* of class *Torque* and method *update* of class *ThrottleOpening*.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

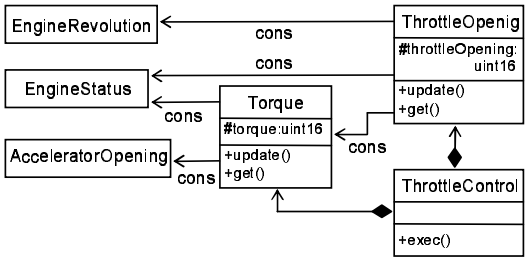

Fig. 4. Example Class Diagram of Functional Model

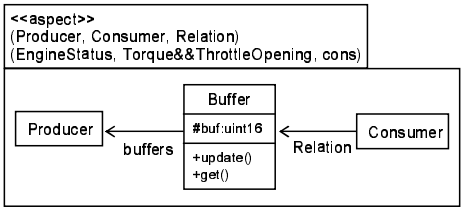

Fig. 5. Aspect Pattern of Buffering

C. Nonfunctional Design

An embedded control system is a hard real-time system with timing constraints. We design the task structure, scheduling policy, task priorities to meet timing constraints in nonfunctional design. We may 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. Aspect-oriented programming[12] has been applied to separate non-functional properties from functional properties. Model level aspects for non-functional requirements have also been presented[13][14].

Our nonfunctional design is based on the aspect-oriented design method we have already presented[15][16]. We have also presented aspect patterns for nonfunctional properties of embedded control software and developed a model weaver to weave the aspect patterns into the functional model. For example, mechanisms for triggering methods (time-triggered or event-triggered[17]), synchronizations and inter-task communications are defined as aspect patterns. We select aspect patterns and weave them into the functional model with the model weaver to get the implementation model.

We consider the case in which the calculation of the values of *EngineRevolution*, *EngineStatus* and *AcceleratorOpening* and the calculation of the values of *TargetTorque* and *ThrottleOpening* are executed by different periodic task. If the priority of the former task is higher than the priority of the latter task, the latter task may be preempted by the former task. So a mechanism of mutual exclusion or inter-task communication is needed for data integrity. Here, we use buffering mechanism, which is one of wait-free inter-task communications. Figure 5 shows the class diagram of the aspect pattern of buffering, which connects a producer object and a consumer object. Class *Buffer* has attribute *buf* to store the value, method *update* to get the value from *Producer* and store the value in *buf*, and method *get* for *Consumer* to get the value stored in *buf*. The class diagram of the aspect pattern is enclosed by a package with stereotype <<*aspect*>>, which represents that the enclosed diagram is an aspect.

The binding expression is written under <<*aspect*>>.

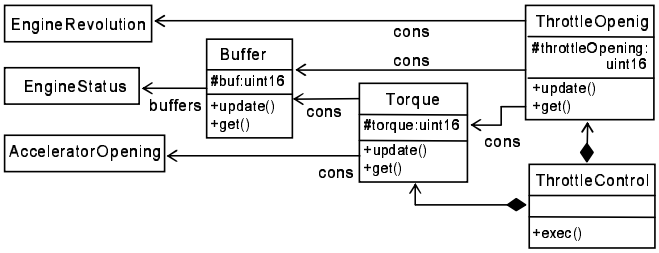

Fig. 6. Example Class Diagram of Implementation Model

Crosscutting elements of the aspect pattern are represented as variables (variable elements), which is bound with the actual elements of the base model. In this case, *Producer*, *Consumer* and *Relation* are variables. The binding expression means that variable *Producer* is bound with *EngineStatus*, variable *Consumer* is bound with *TargetTorque* and *ThrottleOpening*, and variable *Relation* is bound with *cons*. We put the class diagram shown by Figure 4 and the aspect shown by Figure 5 into the input of the model weaver, and we get the woven class diagram shown by Figure 6.

III. Model Transformation Environment

A. Layering Support Tool

The target of the transformation to a UML model is the higher layer model of the layered Simulink model, which consists of subsystem blocks, inport blocks and outport blocks. As described in Section II-B, a value object corresponds to a data in the higher layer model and method *update* of the value object corresponds to the subsystem block that calculates the value of the data. To make a class reusable, the Simulink model should be well-layered before transformation so that subsystem blocks calculating the important data such as reasonable physical quantities are presented at the higher layer.

We have developed a layering support tool to select important data in a Simulink model and layer the Simulink model. Figure 7 illustrates the layering work with the tool. The layering support tool analyzes an input Simulink model, and shows all data of the Simulink model on the window of the tool. Each data of the Simulink model is shown by a row of the table on the window. Column *System* means the subsystem or the whole model in which the data is presented, column *Src Block* means the source block of the data, column *Dst Block* means the destination block of the data, and *Data Name* means the name of the data if the data has a name. For example, the third row of the table in Figure 7 shows the data from *Sum1* to *Subsystem1* with no name in *Controller1* (the whole model).

We can select the data to be presented in the higher layer by checking column *Upper Layer*. In this case, the data named *Data1* from *Subsystem1* to *Out1* in *Controller1* and the data from *Gain1* to *Sum1* in *Subsystem1* are checked. If the checked data has no name, we have to attach the name to the data. In this example, name *DataB* is attached to the data from *Gain1* to *Sum1* in *Subsystem1*. Then, the tool generates a layered Simulink model in which the just the checked data are presented in the higher layer. In this example, there are two subsystem blocks in the higher layer of the generated Simulink model. One subsystem block outputs *DataB* and another subsystem block outputs *DataA*.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

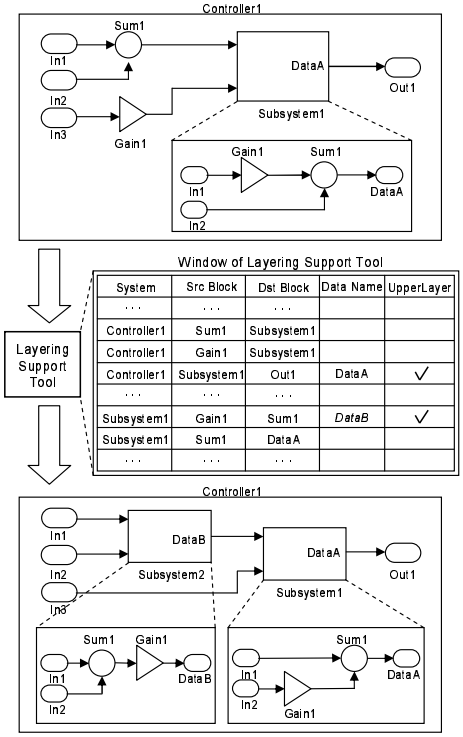

Fig. 7. Layering of Simulink Model

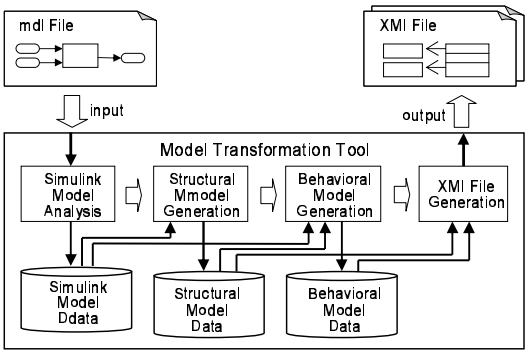

Fig. 8. Model Transformation Tool

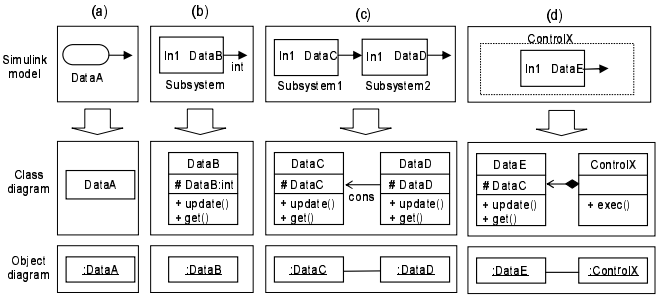

Fig. 9. Transformation Rules for Structural Model

B. Model Transformation Tool

We have developed a model transformation tool to transform a layered Simulink model to a UML model. We developed the first version of the tool to generate class diagrams and object diagrams as the UML structural model[18]. A class diagram is generally used to represent the structure of object-oriented software. An object diagram is also useful for the embedded control system, in which most objects are statically created at the initialization process, not dynamically created. Then we have extended the model transformation tool to generate sequence diagrams as the UML behavioral model. A Sequence diagram is used to represent interactions between objects in time sequence.

Figure 8 shows the internal processing of the model transformation tool. The tool inputs a mdl file, which is a file to store the information on a Simulink model. Then the tool analyzes the mdl file and extracts Simulink model data needed for transformation. The tool generates structural model data referring to the Simulink model data. The tool also generates behavioral model data referring to the Simulink model data and the structural model data. Finally, the tool translates the structural model data and the behavioral model data into XMI files. XMI is a standard file format of UML[19]. The details of structural model generation and behavioral model generation are described in Section III-C and Section III-D.

C. Structural Model Generation

Figure 9 shows rules to transform elements of the Simulink model to elements of the class diagram and the object diagram. Column (a) of Figure 9 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 9 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. 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 9 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. A link of the object diagram is also generated referring to the line.

Column (d) of Figure 9 shows the rule to generate a whole object and composition. Composition means whole-part relationship. A whole object is generated corresponding to the whole Simulink model (*ControlX* in this case). Composition between a whole object (*ControlX* in this case) and a value object (*DataE* in this case) is generated.

Figure 4 shows the generated class diagram from the Simulink model shown by Figure 2. Figure 10 shows the generated object diagram from the same Simulink model. The object diagram consists of six objects of the classes shown in Figure 4. Just one object of each class exists because each subsystem block of the Simulink model is just one instance of the subsystem block. The objects are connected with links that correspond to the lines of the

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

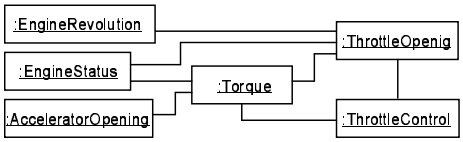

Fig. 10. Example Object Diagram of Functional Model

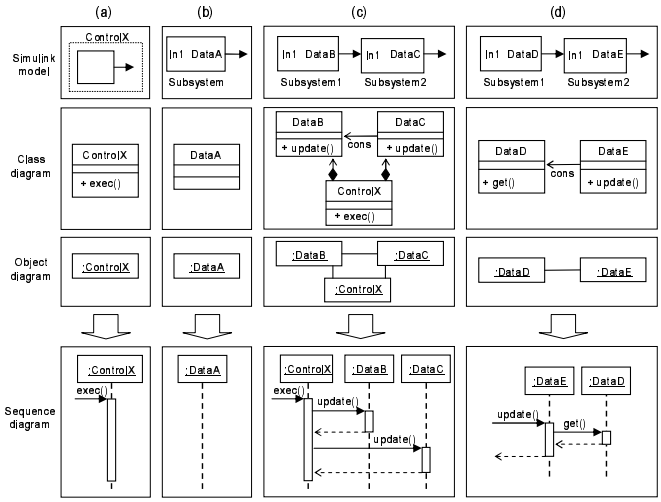

Fig. 11. Transformation Rules for Behavioral Model

Simulink model shown by Figure 2 and to the associations of the class diagram shown by Figure 4.

D. Behavioral Model Generation

Figure 11 shows rules to transform elements of the Simulink model to elements of the sequence diagram. The sequence diagram is generated referring to not only the Simulink model but also the generated class diagram and the generated object diagram.

Column (a) of Figure 11 shows the rule to generate a life-line of a whole object (*ControlX* in this case) and execution activated by message *exec*. Column (b) of Figure 11 shows the rule to generate a value object the lifeline.

Column (c) of Figure 11 shows the rule to generate *update* message sequence and execution activated by the message sequence. The whole object (*ControlX* in this case) calls methods *update* of value objects (*DataB* and *DataC* in this case). The order of the message sequence is determined according to the (partial) order of the data flow of the Simulink model.

Column (d) of Figure 11 shows the rule to generate *get* message and execution activated by the message. If an object consumes (gets) the value of another object, method *get* of the latter object is called by method *update* of the former object.

Figure 12 shows the generated sequence diagram from the Simulink model shown by Figure 2. The sequence diagram shows that method *exec* of object *ThrottleControl* calls method *update* of object *Torque* and method *update* of object *ThrottleOpening* sequentially. Method *update* of object *Torque* calls methods *get* of *EngineStatus* and *AcceleratorOpening* to get those values. Method *exec* of *ThrottleControl* is executed periodically in the control period.

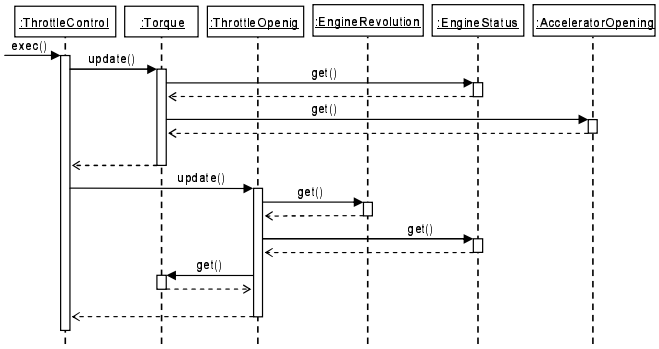

Fig. 12. Example Sequence Diagram of Functional Model

TABLE I
MODELS USED IN EXPERIMENTS

Target System	Number of Blocks		
	Subsystem	Inport	Outport
Fuel Injections	15	4	1
Hybrid Electric Vehicle	30	6	5
Stepping Motor Control	8	1	4

IV. EXPERIMENTS

We have applied the model transformation environment to a number of Simulink models such as a fuel injections system, a hybrid electric vehicle system and a stepping motor control system, which are provided by the MathWorks, Inc.[1].

At first, we made the original Simulink models layered with the layering support tool. Table I shows the number of blocks of the layered Simulink models used in the experiments. The column *Subsystem* shows the number of subsystem blocks, the column *Inport* shows the number of inport blocks and the column *Outport* shows the number of outport blocks. Then we transformed the layered Simulink models to class diagrams, object diagrams and sequence diagrams using the model transformation tool.

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 13 shows the higher layer of the layered Simulink model of the hybrid electric vehicle system. Figure 14 shows the generated structural model: the class diagram and the object diagram. The class diagram consists of thirty-seven classes. Figure 15 shows the generated behavioral model: the sequence diagram.

The original 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, object diagrams and sequence models. So we think the transformation tool can be applied to embedded control software design.

V. CONCLUSION

We have developed a model transformation environment: a Simulink model layering support tool and a Simulink to UML model transformation tool. The model transformation tool generates class diagrams, object diagrams and sequence

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

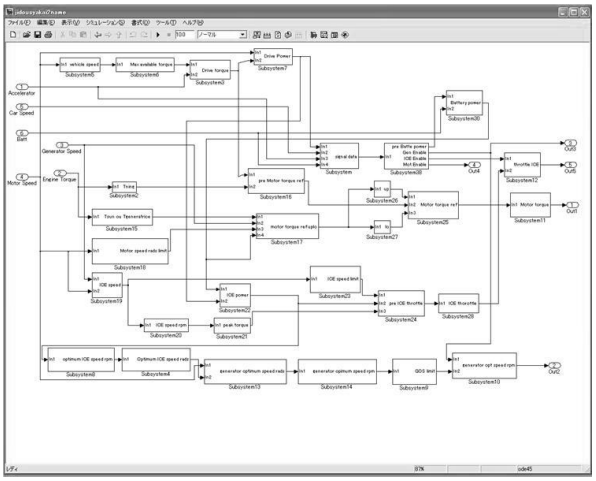

Fig. 13. Simlink Model of Hybrid Electric Vehicle System

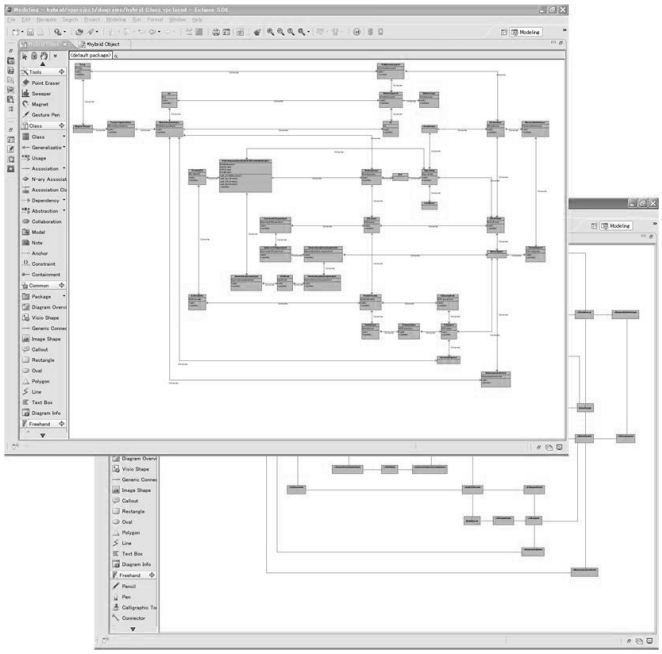

Fig. 14. Class Diagram and Object Diagram of Hybrid Electric Vehicle System

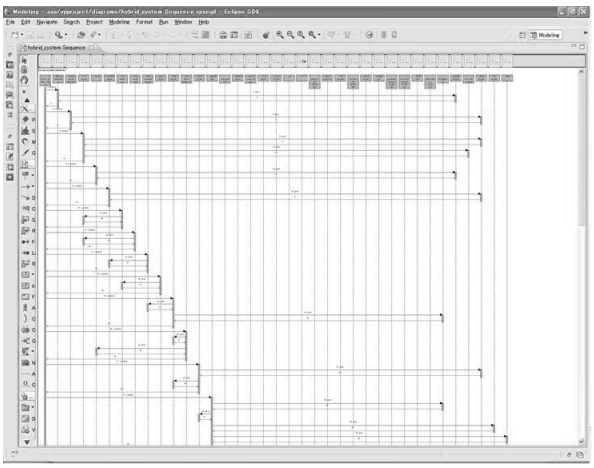

Fig. 15. Sequence Diagram of Hybrid Electric Vehicle System

diagram. Each class of a generated UML model corresponds to a data in the Simulink model and the method of the class corresponds to the subsystem block that calculates the value of the data. We have also applied the tool to a number of Simulink models and found it useful for embedded control software design.

We are going to extend the model transformation tool to generate a state machine diagram to make software design more efficient and to deal with Simulink models with Stateflow charts.

REFERENCES

[1] The MathWorks Inc., http://www.mathworks.com/.
[2] Sangiovanni-Vincentelli, A. and Di Natale, M., Embedded System Design for Automotive Applications, *IEEE Computer*, Vol.40, No.10, pp.42–51, 2007.
[3] Ramos-Hernandez, D. N., Fleming, P. J., Bennett, S., Hope, S., Bass, J. M. and Baxter, M.J., Process Control Systems Integration Using Object Oriented Technology, *Proceeding of Technology of Object-Oriented Languages and Systems TOOLS 38*, pp.148–158, 2001.
[4] Ramos-Hernandez, D. N., Fleming, P. J. and Bass, J. M., A Novel Object-Oriented Environment for Distributed Process Control Systems, *Control Engineering Practice*, vol.13, Issue 2, pp.213–230, 2005.
[5] Kühl, M., Spitzer, B. and Müller-Glaser, K. D., Universal Object-Oriented Modeling for Rapid Prototyping of Embedded Electronic Systems, *Proceedings of the 12th IEEE International Workshop on Rapid System Prototyping*, pp.149–154, 2001.
[6] Müller-Glaser, K. D., Frick, G., Sax E. and Kühl, M., Multiparadigm Modeling in Embedded Systems Design, *IEEE Transactions on Control Systems Technology*, Vol.12, No.2, pp.279–292, 2004.
[7] Sjöstedt, C.-J., Shi, J., Törngren, M., Servat, D., Chen, D., Ahlsten, V. and Lönn, H., Mapping Simulink to UML in the design of embedded systems: Investigating scenarios and structural and behavioral mapping, *OMER 4 Post Workshop Proceedings*, 2008.
[8] Yokoyama, T., Naya, H., Narisawa, F., Kuragaki, S., Nagaura, W., Imai, T. and Suzuki, S., 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.
[9] Yokoyama, T., An Aspect-Oriented Development Method for Embedded Control Systems with Time-Triggered and Event-Triggered Processing, *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Application Symposium*, pp.302–311, 2005.
[10] Yoshimura, K., Miyazaki, T., Yokoyama, T., Irie, T. and Fujimoto, S., A Development Method for Object-Oriented Automotive Control Software Embedded with Automatically Generated Program from Controller Models, *2004 SAE World Congress*, 2004-01-0709, 2004.
[11] Narisawa, F., Naya, H. and Yokoyama, T., A Code Generator with Application-Oriented Size Optimization for Object-Oriented Embedded Control Software, *Object-Oriented Technology: ECOOP'98 Workshop Reader*, Springer LNCS-1543, pp.507–510, 1998.
[12] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. Loingtier, J. M. and Irwin, J., Aspect-Oriented Programming, *Proceedings of 11th European Conference on Object-Oriented Programming*, pp.220–242, 1997.
[13] Wehrmeister, M. A., Freitas, E., Pereira, C. E. and Wagner, F. R., An Aspect-Oriented Approach for Dealing with Non-Functional Requirements in a Model-Driven Development of Distributed Embedded Real-Time Systems, *Proceedings of 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pp.428–432, 2007.
[14] Driver, C., Reilly, S., Linehan, E., Cahill, V. and Clarke, S., Managing Embedded Systems with Aspect-Oriented Model-Driven Engineering, *ACM Transactions on Embedded Computing Systems*, Vol.10, No.2, pp.21:1–26, 2010.
[15] Soeda, T., Yanagidate, Y. and Yokoyama T., Embedded Control Software Design with Aspect Patterns, *Proceedings of International Conference on Advanced Software Engineering and Its Applications 2009*, pp.34–41, 2009.
[16] Soeda, T., Yanagidate, Y. and Yokoyama T., Embedded Control Software Design with Aspect Patterns, *Journal of the Chinese Institute of Engineers*, vol.34, Issue 2, pp.213–225, 2011.
[17] Kopetz, H., Should Responsive Systems be Event-Triggered or Time-Triggered?, *IEICE Transaction on Information & Systems*, Vol.E76-D, No.11, pp.1325–1332, 1993.
[18] Kamiyama, T., Soeda, T., Yoo, M. and Yokoyama, T., A Simulink to UML Transformation Tool for Embedded Control Software Design, *Proceedings of 2010 International Conference on Computer and Software Modeling*, pp.93–97, 2010.
[19] Object Management Group, *XML Metadata Interchange Specification*, Version 2.0.1, 2005.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)