PAPER *Special Section on Software Engineering for Embedded Systems*

Extraction of Transformation Rules from UML diagrams to SpecC

Tetsuro KATAYAMA[†a)], ***Member***

SUMMARY Embedded systems are used in broad fields. They are one of the indispensable and fundamental technologies in a highly informative society in recent years. As embedded systems are large-scale and complicated, it is prosperous to design and develop a system LSI (Large Scale Integration). The structure of the system LSI has been increasing complexity every year. The degree of improvement of its design productivity has not caught up with the degree of its complexity by conventional methods or techniques. Hence, an idea for the design of a system LSI which has the flow of describing specifications of a system in UML (Unified Modeling Language) and then designing the system in a system level language has already proposed. It is important to establish how to convert from UML to a system level language in specification description or design with the idea.

This paper proposes, extracts and verifies transformation rules from UML to SpecC which is one of system level languages. SpecC code has been generated actually from elements in diagrams in UML based on the rules. As an example to verify the rules, "headlights control system of a car" is adopted. SpecC code has been generated actually from elements in diagrams in UML based on the rules. It has been confirmed that the example is executed correctly in simulations. By using the transformation rules proposed in this paper, specification and implementation of a system can be connected seamlessly. Hence, it can improve the design productivity of a system LSI and the productivity of embedded systems.

key words: *Transformation rules, UML (Unified Modeling Language), SpecC, System level languages, VisualSpec, Embedded systems, System LSI (Large Scale Integration)*

1. Introduction

By development of microprocessor technology, applicable fields of computers built into home electronics, measurement apparatus, and so on, i.e., embedded systems are expanded. Moreover, as apparatus or devices for control or management are more highly efficient and complex, it is remarkable that embedded systems are large-scaled and complicated. Furthermore, digitization of apparatus is also increasing the importance of embedded systems. The present embedded systems are used in broad fields, such as a personal digital assistant, an information appliance, and apparatus in car. They are one of the indispensable and fundamental technologies in a highly informative society in recent years.

As embedded systems are large-scale and complicated, it is prosperous to design and develop a system LSI (Large Scale Integration) which realizes all functions of a system with one chip. The structure of the system LSI has been increasing its complexity every year. The degree of improvement of its design productivity has not caught up with the degree of its complexity by conventional methods or techniques[1]. The following methods can be considered to solve the situation.

- Description of specifications in UML (Unified Modeling Language)[2]
- Design by system level languages

These two methods can be used independently. Hence, an idea for the design of a system LSI which has the flow of describing specifications of a system in UML and then designing the system in a system level language has already proposed[3]. By establishing this idea, it becomes possible to realize shortening a design period and improving a design quality. Moreover, since the object-oriented view of UML can be used, the system LSI which is powerful against specification change and is able to reuse of design data can be constructed. It is important to establish how to convert from UML to a system level language in specification description or design with the idea.

This paper proposes, extracts and verifies transformation rules from UML to a system level language. In this paper, SpecC[4] which is advanced spread or promotion activities mainly in Japan is adopted as a system level language. VisualSpec[5] of InterDesign Technologies, Inc. is used as a design tool or editor for SpecC. Here, the transformation rules proposed in this paper express the rule for conversion from elements in some diagrams in UML to the elements needed when codes in SpecC are described.

Section 2 explains briefly object-oriented modeling language UML, system level language SpecC, and design tool VisualSpec which this paper uses. Section 3 shows the transformation rules extracted in this paper. Section 4 verifies the rules expressed in Section 3. SpecC code is generated actually from elements in diagrams in UML based on the rules and the generated code is verified in simulations. As an example, "headlights control system of a car"[6] is adopted. Section 5 describes discussion and evaluation.

Manuscript received September 24, 2004.
Manuscript revised January 5, 2005.
Final manuscript received February 4, 2005.

[†]The author is with the Faculty of Engineering, University of Miyazaki, Miyazaki-shi, 889-2192 Japan.
a) E-mail: kat@cs.miyazaki-u.ac.jp

2. Preparation of Research

2.1 UML

UML (Unified Modeling Language) is a visual language[2]. It helps to understand the structure and the dynamic behavior or action for the business or the various systems. Introducing UML can reduce a backtracking such as missing recognition of the specifications. This is why communication gap between a user and a developer, or developers can dissolve and user requirements can become accurate. Moreover, object-oriented design by UML promotes modularization effectively. It is possible that a maintenance cost is reduced.

UML has been proposed by Grady Booch, James Rumbaugh, and Ivar Jacobson, who the methodology of an object-oriented analysis and design was devised in the separate organization, to OMG (Object Management Group)[7] as a unification methodology, and agreed officially with UML1.0 in 1997. After that, the standard of UML is expanded, it is UML1.5 in Mar., 2003, and it is said that UML2.0 is released soon. This paper uses the specification of UML1.5 as the official "Available UML Specification" at present.

The following nine kinds of diagrams exist in UML, and each is handled respectively corresponding to the use.

- class diagram
- object diagram
- usecase diagram
- statechart diagram
- sequence diagram
- activity diagram
- collaboration diagram
- component diagram
- deployment diagram

UML can show various aspects of the system by using nine diagrams.

2.2 SpecC

SpecC[4] is one of system level languages. Features of the system level languages are feasibility, modularity, and completeness. The modularity means that function and connection can be divided completely. The completeness means that all of structural hierarchy, behavior, parallelism, synchronization, exception-handling timing, and state transition which are common concepts in any embedded systems can be treated. Unlike the existing programming languages, a system level language has the feature which can realize operation of hardware or software with real-time restrictions. On the other hand, it is also possible to use as a hardware description language or mere programming language.

The system level languages have four advantages as follows:

- It is possible to solve the trouble resulting from specification being described by natural languages.
- It is executable. (A simulation is possible)
- It is possible to use hardware software co-design or co-verification.
- It is possible to unify one description language from specification to implementation.

Some system level languages which have the above advantages are proposed such as "SystemC"[8], "SpecC"[4], "Cynlib"[9], and so on.

However, since all are the steps where a proposal and spread or promotion activities still started just, the following points are mentioned as faults.

- At present, it cannot judge which language becomes a de facto standard.
- At present, each language is not clear about the ability of its description.

In this paper, "SpecC"[4] is adopted as a system level language. It is one of the system level languages for describing or designing systems was born by the research and development over many years at the University of California, Irvine. It is advanced spread or promotion activities mainly in Japan at present. SpecC adds concepts, such as parallelism, synchronization and communication, and timing, and the syntax to implement the concepts to ANSI C.

Main elements (features) of SpecC are described as follows. A functional block is important in considering the specification of a system. The overall function of a system can be checked by clarifying the interface between functional blocks. The interface is declared as an event. That is, a functional block is operated through the event in using the functional block from the exterior. A behavior can express the interior of a functional block in more detail. It causes state transition by three kinds of transition types: FSM (Finite State Machine) event, condition, and completion. A channel can be used for communication between two or more behaviors. It can hide a detailed communications protocol in itself. Moreover, since it can separate behavior and communication completely, the maintenance to specification change becomes easy.

2.3 VisualSpec

It becomes possible to shorten remarkably the period from specification to design by describing specifications of a system in UML and then designing the system in a system level language. Here, it is an important part that the conversion from UML to a system level language. Hence, this paper extracts the transformation rules from elements in some diagrams in UML to elements in system level language SpecC.

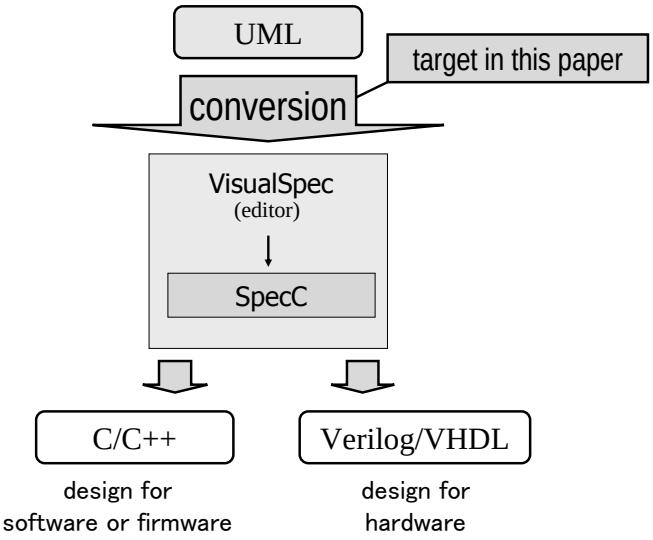


Fig. 1 Outline of the flow from specification to design in this paper

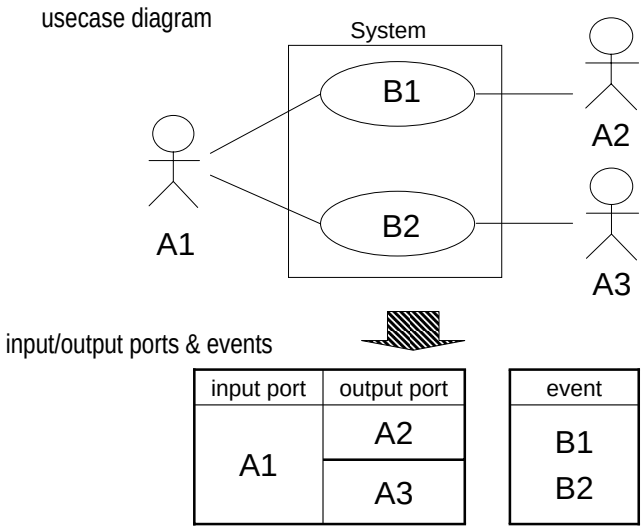


Fig. 2 Conversion from usecase diagram to input/output ports and events

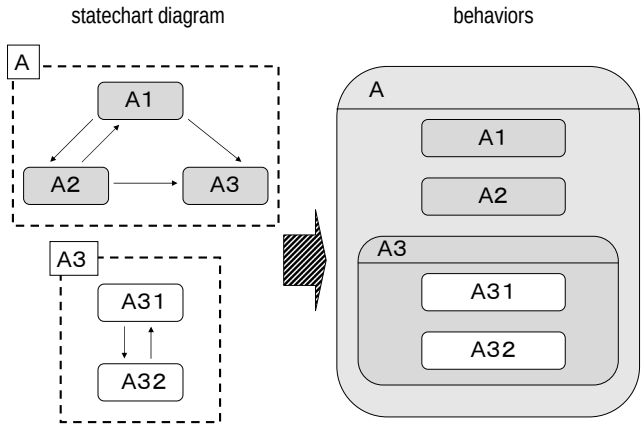


Fig. 3 Conversion from statechart diagram to behaviors and their hierarchy

This paper uses VisualSpec[5] of InterDesign Technologies, Inc. as a design tool or editor for SpecC to realize the conversion. VisualSpec offers the design environment which consists in "specification editor" to input and edit a SpecC program and "specification debugger" to compile, execute, and debug the program. It can design a system simply by visual input and edit.

Figure 1 shows an outline of the flow from specification to design in this paper.

3. Transformation Rules from UML to SpecC

First, in a system design with SpecC, the system can know the demand from a user through an input port, and output the result through an output port. What can express external specification best in UML is a usecase diagram. Actors in a usecase diagram show roles of user in system and usecases show functions in system. Hence, there is some possibility of defining input/output ports by referring to actors in a usecase diagram. By being conscious of such a mapping in writing usecase diagrams, actors can be corresponded to input/output ports. In the usecase diagrams described under the idea as above, events can be declared by referring to usecases in the usecase diagrams because it can be considered that usecases are demand elements from a user or are an only communication means opened to the public to the exterior and a functional block is operated through an event when the functional block is used from the exterior in a system.

Figure 2 shows an example of conversion from usecase diagram to input/output ports and events. An input port from actor A1, output ports from actor A2, A3, and events from usecase B1, B2 can be defined, respectively.

Next, in order to be able to overlook the function of the whole over a system, it is necessary to extract functional blocks and to clear interfaces between each of blocks. Interfaces can be declared ones called an event from usecase diagram as above. Functional blocks can be defined through class diagram, which can describe the static structure of a system by connecting two or more classes with relation, or be extracted by referring to lanes in activity diagram, which express a sequence of the processing assigned to a specific object or subsystem. Moreover, the relation of the functional blocks can be defined by referring to the relation among classes in class diagram, or the relation among lanes or action states in activity diagram.

And then, behaviors, which express the details of functional blocks, can be defined by referring to usecases in usecase diagram or operations, which present the processing in a class, in class diagram. Behaviors can also be defined by referring to statechart or activity diagram, which can express the dynamic side of a sys-

Table 1 Transformation rules from UML to SpecC

generated elements	reference diagrams & elements	
input/output port	usecase diagram	actor
event	usecase diagram	usecase
relation of an input/output port and an event	usecase diagram	relation of an actor and a usecase
functional block	class diagram	class
	activity diagram	lane
relation among functional blocks	class diagram	relation among classes
	activity diagram	relation among action states or among lanes
behavior	usecase diagram	usecase
	class diagram	operation
	statechart diagram	state
	activity diagram	action state
hierarchy of behaviors	statechart diagram	relation of statechart diagrams (if multiple statechart diagrams exist)
parallel execution	statechart diagram	state transition
transition or flow of behaviors	activity diagram	transition of action states
	statechart diagram	state transition
condition for transition	statechart diagram	event or action
	activity diagram	conditional judgment
variables	class diagram	attribute
channel	class diagram	stereotype, operation, or relation of classes (if they communicate through dependency, events, and so on)
synchronization	class diagram	operation or relation of classes (if they communicate through events)
	activity diagram	action states before or after a synchronous bar (if activity diagram expresses the synchronizations)

tem. Moreover, if statechart diagram exists to states of another statechart diagram, referring to the relation of those statechart diagrams can define the hierarchy of behavior. A behavior transits to another behavior under some conditions. Since this is almost the same as the role of statechart diagrams, referring to state transition in the statechart diagram can define transition of the behavior. In that case, referring to events or conditions in the statechart diagram can define FSM event, condition, or completion which are the transition types of the state of behavior.

Figure 3 shows an example of conversion from statechart diagram to behaviors and their hierarchy. One statechart diagram expresses state transitions of an object A and the other expresses state transitions of a state A3 which is corresponded to one state in the object A. A hierarchy of behaviors which consists of a parent behavior A, its children behaviors A1, A2, A3, and A3's children behaviors A31, A32 can be defined from these two statechart diagrams.

Finally, variables can be defined by referring to types and arguments described as attributes in class diagram. Cannels can be defined by referring to the class name, stereotype, or operations in classes if they communicate each other by the relation of the classes such as dependency, events, and so on in class diagram. Synchronizations between behaviors which execute concurrently can be defined by referring to class diagram as similarly. Or action states before and after a synchronous bar can define synchronizations if activity diagram expresses the synchronizations.

The above can be made into the transformation rules from UML to SpecC in this paper. Table 1 shows the summarized transformation rules. In order to create the element in SpecC on the left-hand side of the table, it means referring to the elements in diagrams of UML on the right-hand side.

4. Verification of the Transformation Rules

In order to confirm the validity of the transformation rules extracted in Section 3, SpecC code is generated actually from elements in diagrams in UML based on the rules and the generated code is verified in simulations. As an example, "headlights control system of a car"[6] is adopted.

4.1 Application of the Transformation Rules

First, the elements to generate SpecC code in VisualSpec is extracted from usecase, class, statechart, and activity diagrams in UML, respectively, based on the transformation rules. And then, the SpecC code is generated by inputting the elements in VisualSpec.

Figure 4 shows a usecase diagram in the headlights control system of a car. Input/output ports from the actors in the diagram can be defined. And their values and contents from the usecases can be defined. Table 2, 3 show the input and output ports, respectively.

Moreover, from the usecases, events can be declared. Figure 5 shows them in VisualSpec.

Figure 6, 7, and 8 are a class diagram, a statechart

Table 2 Lists of input ports

Signal	Port	Value
headlights switch	Pi0	0: headlights OFF 1: parking lamps only ON 2: headlights & parking lamps ON
main switch	Pi2	0: ON 1: OFF
fog lamps switch	Pi3	0: ON 1: OFF
high / low beam change	Pi4	0: high beam ON 1: low beam ON
opening / closing doors	Pi5	0: OPEN 1: CLOSE

Table 3 Lists of output ports

Signal	Port	Value
high beam	Po0	0: high beam ON 1: high beam OFF
low beam	Po1	0: low beam ON 1: low beam OFF
fog lamps	Po2	0: fog lamps ON 1: fog lamps OFF
parking lamps	Po3	0: parking lamps ON 1: parking lamps OFF

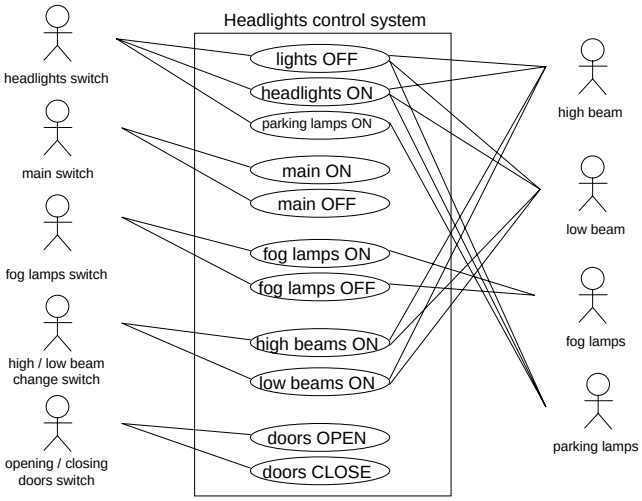


Fig. 4 Usecase diagram of the headlights control system of a car

diagram, and activity diagram of the headlights control system of a car, respectively. From the class and activity diagrams, functional blocks can be defined because the overall structure and flow of the system can be understood from the diagrams. From the statechart diagram, behaviors of the headlights control and hierarchy of the behaviors can be defined. Figure 9 shows the registration of behaviors in VisualSpec based on the above.

Furthermore, the state transitions of the behaviors of a headlights component, headlights ON, a low beam, and automatic headlights OFF can be defined from the statechart diagram. Figure 10 shows the definition of

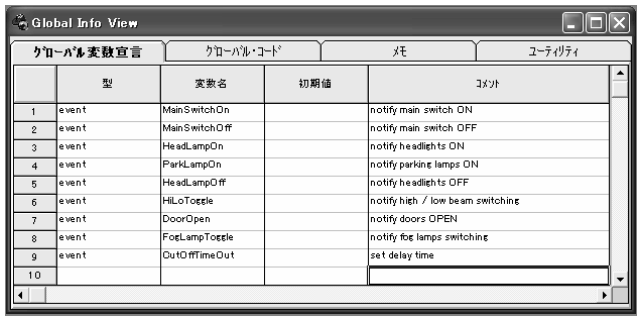


Fig. 5 Declaration of events (from VisualSpec)

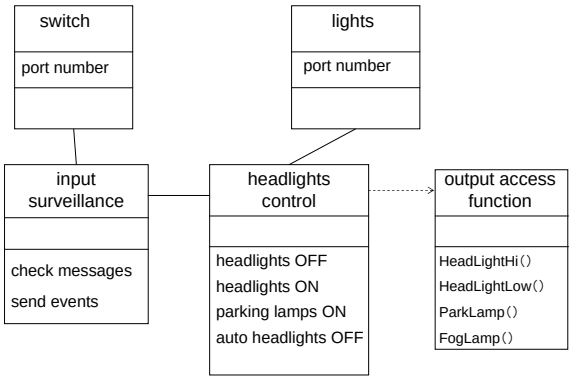


Fig. 6 Class diagram of the headlights control system of a car

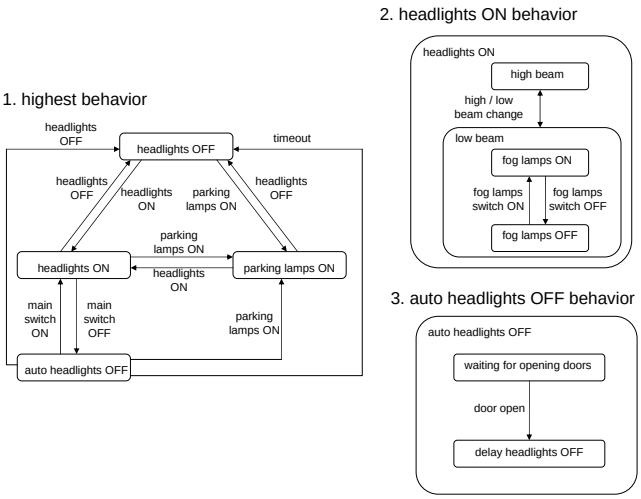


Fig. 7 Statechart diagram of the headlights control system of a car

the state transition of the headlights component. The other state transitions can be defined, similarly.

4.2 Generation of the SpecC code and simulation execution

The specification has been created for the headlights control system of a car based on the transformation rules in VisualSpec. Here, the codes for the input

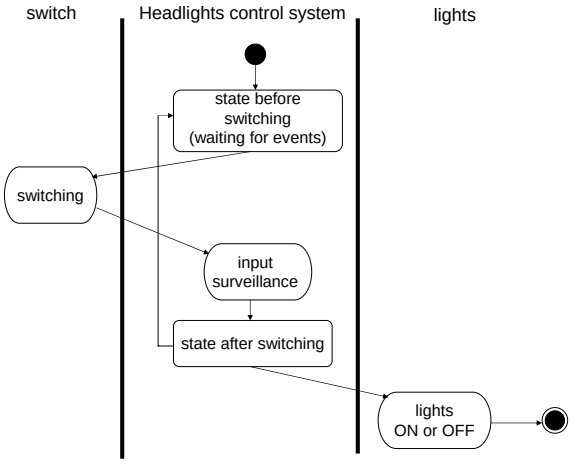


Fig. 8 Activity diagram of headlights control of a car

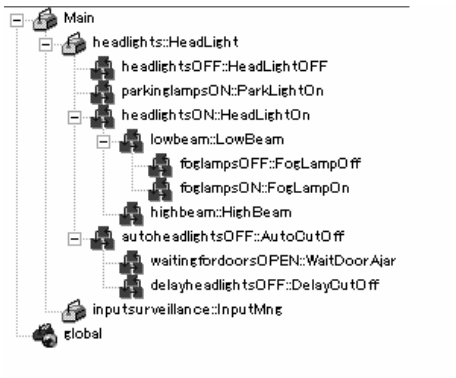


Fig. 9 Registration of behaviors (from VisualSpec)

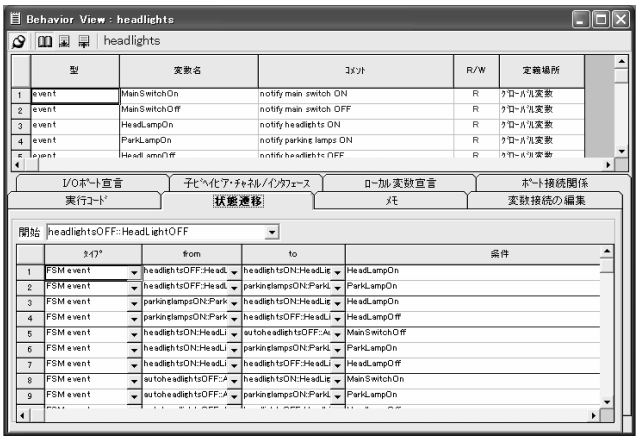


Fig. 10 Definition of the state transition of the headlights component (from VisualSpec)

surveillance, its behavior, and output access which are needed to verify the rules is described independently. The code (735 lines) in SpecC has been automatically generated by VisualSpec. Figure 11 shows a part of the codes.

It is checked that "headlights control system of a car" is executed correctly from the execution result of simulations based on the generated code in SpecC. Figure 12 shows a snapshot in the simulations. As mentioned above, the validity of the transformation rules

```
//
// HeadLightCtrl.sc
// Created by VisualSpec 2002 Version1.02
//
#include <sim.h>
#include <vscom.h>
#include <stdio.h>
#include <sim.h>
#include <vscom.h>
#include <string.h>
#define ON 1
#define OFF 0
void OUT_SetHeadLampHi (char bOn)
{
   switch (bOn) {
      case OFF:
      // Po0=0 ;
      // Po1=0 ;
        VS_SendMessage ( "Po0" , "0") ;
        VS_SendMessage ( "Po1" , "0") ;
      break;
      case ON :
      // Po0=1 ;
      // Po1=1 ;
        VS_SendMessage ( "Po0" , "1") ;
        VS_SendMessage ( "Po1" , "1") ;
      break ;
    }
}
                    •
                    •
                    •
behavior _BEH_6__HeadLightOn_MAIN(void)
{
note BehaviorName =
    "Headlights_HeadLightOn_MAIN";
_BEH_4_ HeadLightOn();
VSPEC_FSM_TRAP_WAIT_FSM_TRAP_WAIT( );
void main( void ) {
   fsm {
      HeadLightOn: {
      }
      _FSM_TRAP_WAIT : {
      }
   }
}
};
                    •
                    •
                    •
```

Fig. 11 A part of the SpecC code generated by VisualSpec

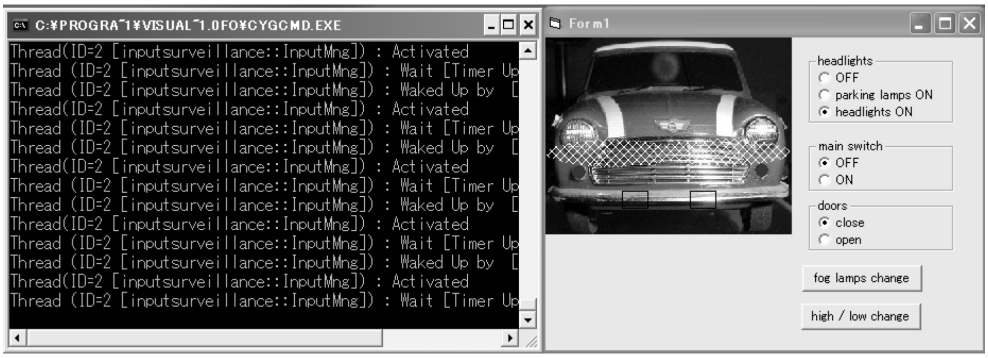


Fig. 12 A snapshot in the simulations

extracted in this paper has been confirmed.

5. Discussion

This paper extracts the transformation rules to SpecC from diagrams in UML to shorten design period and improve a design quality of a system LSI. SpecC code has been generated actually from elements in the example diagrams in UML based on the transformation rules. The generated code has been verified in simulations. The validity of the transformation rules extracted in this paper has been confirmed.

In case the requirements specification of a system is determined, it is becoming a de facto standard to describe the specification in UML. Flows from UML to software are being established. Actually, popular UML modeling tools exists (e.g. [10], [11]) and many methods or techniques about MDA(Model Driven Architecture) have been proposed (e.g. [12], [13]). On the other hand, fewer studies on flows from UML to hardware have been documented. An idea for the design of a system LSI which has the flow of describing specifications of a system in UML and then designing the system in a system level language has already proposed[3]. In order to apply UML to embedded systems, some reserches have expanded UML (e.g [14]–[18]). They introduce their specific metamodels, scripting languages, or extended profile over a concept of UML. The transformation rules extracted in this paper from only the specification of UML. Anyone who learns only UML are available and understandable for the rules easily.

By using the transformation rules, specification and implementation of a system can be connected seamlessly. Hence, it can improve the design productivity of a system LSI and the productivity of embedded systems, which are large-scale and complicated, increases.

The transformation rules extracted in this paper have been verified by applying the example. However, elements in SpecC which cannot be extracted as transformation rules such as pipeline execution, exception handling, timing, and so on exist at present. More rules must be extracted to improve productivity of embedded systems more. It is considered to use the other diagrams which are not used in this paper.

Although the transformation rules have been extracted and elements in SpecC have been inputted into VisualSpec in this paper, this work has been done manually. The tool automatically conversed into inputs of VisualSpec from diagrams in UML based on the transformation rules must be implemented.

This paper uses the specification of UML1.5 as the official "Available UML Specification" at present. It is said that the specification of UML2.0 is fixed soon[2]. Hence, it is expected that the transformation rules are corresponded to UML2.0 as soon as it is released formally.

6. Conclusion

In this paper, the transformation rules to SpecC from diagrams in UML are extracted to shorten design period and improve a design quality of a system LSI. The transformation rules proposed in this paper express the rules for conversion from elements in some diagrams in UML to the elements needed in describing codes in SpecC. SpecC which is advanced spread or promotion activities mainly in Japan is one of system level languages. VisualSpec of InterDesign Technologies, Inc. is used as a design tool or editor for SpecC. As an example to verify the rules, "headlights control system of a car" is adopted. SpecC code has been generated actually from elements in diagrams in UML based on the rules. The example has been executed correctly in simulations. As mentioned above, the validity of the transformation rules extracted in this paper has been confirmed.

By using the transformation rules proposed in this paper, specification and implementation of a system can be connected seamlessly. Hence, it can improve the design productivity of a system LSI and the productivity of embedded systems, which are becoming larger-scale and more complicated.

Future issues are as follows:

- More extraction of transformation rules
  In this paper, the transformation rules have been

extracted from four diagrams in UML: usecase, class, statechart, and activity diagrams. The transformation rules extracted in this paper have been verified by applying to the example. However, elements in SpecC which cannot be extracted as transformation rules such as pipeline execution, exception handling, timing, and so on exist at present. By extracting more rules through considering other diagrams also, productivity of embedded systems is improved more.

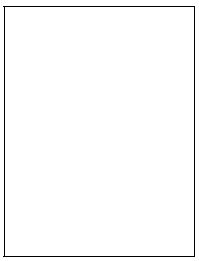
- Implementation of an automatic conversion tool
  Although the transformation rules have been extracted and elements in SpecC have been inputted into VisualSpec in this paper, this work has been done manually. By implementing a tool automatically conversed into inputs of VisualSpec from diagrams in UML based on the transformation rules, it aims at conversion for a shorter time.

Acknowledgments

I would like to thank Yoshifumi Koyashiki for implementing some codes to execute simulations, and thank Shin'ya Sasaki for making some figures from VisualSpec.

References

[1] Daniel D. Gajiski, Jianwen Zhu, Rainer Domer, Andreas Gerstlauer, and Shuqing Zhao: "SpecC: Specification Language and Methodology," Kluwer Academic (2000).
[2] Unified Modeling Language: http://www.uml.org/
[3] Koji Asari: "The modeling technique for the system using UML," Proc. 4th Summer Workshop on Embedded System Technologies (SWEST4), pp.3-4 (2002) (in Japanese).
[4] SpecC Technology Open Consortium: http://www.specc.gr.jp/eng/
[5] InterDesign Technologies, Inc. : "VisualSpec" http://www.interdesigntech.co.jp/english/visualspec.htm
[6] "Co-design embedded software and LSI in C language – Aim system architect –," Design Wave Magazine, CQ Publishing, No.32, pp.27-111 (2000) (in Japanese).
[7] Object Management Group: http://www.omg.org/
[8] Open SystemC Initiative: http://www.systemc.org/
[9] Forte Design Systems: http://www.forteds.com/
[10] IBM Co. : "Rose," http://www-136.ibm.com/developerworks/rational/products/rose/
[11] Embarcadero Technologies, Inc.: "Describe," http://www.embarcadero.com/products/describe/
[12] David S. Frankel: "Model Driven Architecture: Applying Mda to Enterprise Computing," John Wiley & Sons Inc. (2003).
[13] Jeff Gray, Jing Zhang, Yuehua Lin, Hui Wu, Suman Roychoudhury, Rajesh Sudarsan, Aniruddha Gokhale, Sandeep Neema, Feng Shi, and Ted Bapty: "Model-Driven Program Transformation of a Large Avionics Framework," Proc. 3rd Generative Programming and Component Engineering (GPCE 2004), Springer-Verlag LNCS 3286, pp.361-378 (2004).
[14] Bran Selic: "Using UML for Modeling Complex Real-Time Systems," Proc. ACM SIGPLAN Works. on Languages, Compilers, and Tools for Embedded Systems (LCTES'98), Springer-Verlag LNCS 1474, pp.250-260 (1998).
[15] Jorge L Diaz-Herrera, Jasmin Chadha, and Neil Pittsley: "Aspect-Oriented UML Modeling for Developing Embedded Systems Product Lines," Works. on Aspect-Oriented Modeling with UML Int'l Conf. on Aspect-Oriented Software (2002).
[16] Pierre Boulet, Jean-Luc Dekeyser, Cédric Dumoulin, and Philippe Marquet: "MDA for SoC Design, Intensive Signal Processing Experiment," Forum on specification & Design Languages (FDL'03), http://www.lifl.fr/west/publi/DBDM03f.pdf (2003).
[17] Qiang Zhu, Ryosuke Oishi, Takashi Hasegawa, and Tsuneo Nakata: "System-on-Chip Validation Using UML and CWL," Proc. 2nd IEEE/ACM/IFIP Int'l Conf. on Hardware/Software Codesign and System Synthesis, pp.92-97 (2004).
[18] CATS Co. Ltd.: "Xmodelink," http://www.zipc.com/english/product/xmodelink/



Tetsuro Katayama received the Ph.D degree in engineering from Kyushu University, Fukuoka, Japan in 1996. From 1996 to 2000 he has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. Since 2000 he has been an Associate Professor at the Department of Computer Science and Systems Engineering, Faculty of Engineering, University of Miyazaki, Japan. His research interests include software testing, system softwares, and embedded systems. He is a member of the IPSJ and JSSST.