COMMUNICATION MODEL EXPLORATION IN ASPECT-ORIENTED EXECUTABLE UML

Tatsuya Hoshino, Akira Teruya, Eiichiro Iwata
Masahito Sugai, Noriko Matsumoto, Norihiko Yoshida
Department of Information and Computer Sciences
Saitama University
255 Shimo-Ohkubo, Saitama 338-8570, JAPAN

ABSTRACT

Embedded systems have recently been working on networks; consequently, communication design has been getting complex. Although system-level design is a design methodology of embedded systems, the methodology is inefficient in communication design, and it is difficult to explore the suitable communication modules for a system. Therefore the preceding studies presented how communication exploration can be done in distributed embedded system design. On the other hand, embedded systems have also been getting large and complex, system-level design is demanded more efficient. For design efficiency, it is important to enhance design abstractions, and to formalize the refinement procedure of system-level design. We proposed to apply executable UML and aspect-oriented techniques to system-level design. However, these approaches are only applied to single embedded systems, not applied to communication design of distributed embedded systems. Hence this paper proposes to apply aspect-oriented executable UML to communication model exploration, to improve the design efficiency of distributed embedded systems.

KEYWORDS

Distributed embedded systems, Exploration of communication models, Executable UML, Aspect-oriented design

1. INTRODUCTION

Embedded systems have recently been decentralizing and working on networks due to the development of information technologies, and they are called distributed embedded systems. In distributed embedded systems, communication design is more important and complex than in single embedded systems, because of the characteristic of communicating with other embedded systems. According to the spread of distributed embedded systems, communication design is presently getting difficult.

Embedded system design methodologies include system-level design [Gajski, 2000; Gerstlauer, 2002] in which a design process is divided into some steps, and in which methodology, designers proceed with design processes by refining system models in a stepwise manner. However, system-level design has an issue that communication exploration is not separated from architecture exploration; it is difficult to explore an appropriate combination of modules among several possibilities. In order to address this issue and improve the efficiency of communication design, the preceding studies [Kobayashi, 2007; Kinoshima, 2007] presented how communication exploration can be done, and they described the communication models in SpecC which is one of system-level languages. Except for these studies, there has been none yet on communication exploration in distributed embedded system design.

On the other hand, embedded systems have also been getting larger and more complex; system-level design is demanded more efficient. For design efficiency, it is important and effective to enhance reusability of properties by design abstraction, and also to automate the refinement work by formalizing the procedure of system-level design as well.

As the approach to the former, Kimura et al. [Kimura, 2008] proposed system-level design in executable UML [Mellor, 2002]. Executable UML, which is based on Model-Driven Architecture (MDA) in software engineering, is a profile of UML with model execution environments. System-level design based on UML leads to design abstraction of properties, and also increases their reusability. As the approach to the latter,

which is to automate the refinement work, Teruya et al. [Teruya, 2008] made a proposal to formalize the stepwise refinement procedures with refactoring [Fowler, 1999] and aspect-oriented design [Chiba, 2005]. However, both approaches are only applied to single embedded systems, not applied to design of distributed embedded systems.

Hence we presents exploration of communication models in aspect-oriented executable UML [Teruya, 2008] for efficient design of distributed embedded systems, and then aims to formalize the refinement of the communication models by using aspect-oriented techniques for automation of the refinement works. This paper not only proposes to improve the design efficiency of distributed embedded systems, but also indicates the possible application of executable UML and aspect-oriented design for embedded system design as well.

Section 2 and 3 summarize the essentials of executable UML and aspect-oriented techniques as background knowledge of this paper. Section 4 proposes exploration of communication models. Section 5 shows our research approach and some experiments in aspect-oriented executable UML. Section 6 mentions some concluding remarks.

2. EXECUTABLE UML (xUML)

Executable UML (xUML) is a profile of UML 2.0 [OMG, 2000], and is a modeling language composed of UML diagrams, an action language and a model compiler. The action language, which conforms to the UML Action Semantics [OMG, 2001], is used for a description of model behaviors.

xUML tools have execution environments; and xUML models can be executed and verified at any steps. Designers use class diagrams for static structures, and state chart diagrams for dynamic structures. Each state in the statechart diagram has a procedure described in the action language. In this research, we use iUML [Kennedy Carter Ltd] as an xUML tool.

3. ASPECT-ORIENTED PROGRAMMING

Aspect-oriented programming (AOP) is a programming paradigm that separates crosscutting concerns from other concerns. Here, a concern means a set of behaviors (e.g., procedures, classes, methods) in programs, software, and systems. A crosscutting concern straddles multiple concerns, therefore we cannot handle it singly: for example, logging, caching, exception handling, etc. While object-oriented programming is difficult to modularize crosscutting concerns, AOP can modularize it as *aspect*.

AOP mechanism is based on Join Point Model (JPM) which is composed of *joinpoints*, *advices* and *pointcuts*. In source codes, we have some places (candidates) where we can insert aspects, such as method calls or assign statements. JPM calls these candidates joinpoints. An aspect consists of advices and pointcuts: an advice is program codes to be inserted, and a pointcut specifies a place to insert, which is selected from joinpoints. The compiler (called *Weaver* in AOP) integrates source codes and aspects by inserting advices to joinpoints depending on pointcuts. We can get improved program codes without tinkering with source codes. Hence, AOP increases modularity, maintainability and reusability of programs.

4. EXPLORATION OF COMMUNICATION MODELS

4.1 Classification of Communication Models

There are two major communication models in embedded system networks: event-triggered and time-triggered.

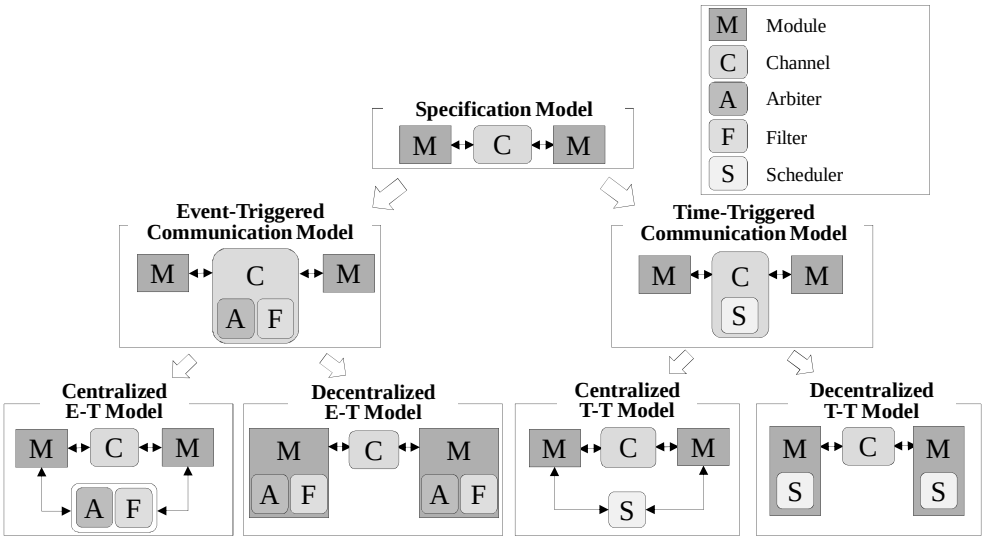

Figure 1. Stepwise exploration of communication models

In the event-triggered model, communication timing depends on its sender module, and the sender can send a data arbitrarily according to an occurrence of events. Since there is a possibility of data collisions, this model needs an arbitration structure. In the time-triggered model, on the other hand, communication timing is statically scheduled, and the sender depends on a schedule cycle. The scheduling structure controls the communication timing on the whole network. This model is suitable for systems with hard real-time communication, because data collisions never happen.

Each of the above, from an architectural point of view, can be classified into a centralized model and decentralized model. In the centralized event-triggered/time-triggered model, a single arbiter/scheduler controls the communication between sender and receiver modules. In the decentralized event-triggered/time-triggered model, each module has the function of arbitration/scheduling, and they control the communication themselves. In terms of design, the centralized model is with lower cost and easier to implement, because of its simple architecture. The decentralized model, in contrast, is faster and more robust than the centralized models, although the design is more complex.

For example, the above categories are applicable to some protocols for in-vehicle networks as follows: CAN (Controller Area Network) is a decentralized event-triggered model, LIN (Local Interconnect Network) is a centralized time-triggered model and FlexRay is a decentralized time-triggered model.

4.2 Stepwise Exploration of Communication Models

As mentioned above, in development of distributed embedded systems, communication design has recently been getting complex. Consequently, the exploration of a communication model suitable for a system is more difficult than before. In order to solve this issue, the preceding studies [Kobayashi, 2007; Kinoshima, 2007] investigated how communication exploration can be done in distributed embedded system design, and they presented the process as shown in Figure 1.

At the beginning, there is a specification model as the most abstract model in communication exploration. In the specification model, sender and receiver modules communicate through an abstract channel. At this step, the specification model leaves concrete protocols out of consideration. Secondly, the specification model is refined into an event-triggered or a time-triggered model. Through this refinement, the event-triggered channel has the function of arbitration and filtering, or the time-triggered channel has the function of scheduling. Next these two communication models, which are event-triggered and time-triggered models, are refined into centralized or decentralized models. In the centralized event-triggered/time-triggered model, an arbiter and filter, or a scheduler is independent from the channel; and they control communication as a single module. In the decentralized event-triggered/time-triggered model, the function of arbitration and filtering, or scheduling is embedded into the all sender and receiver modules.

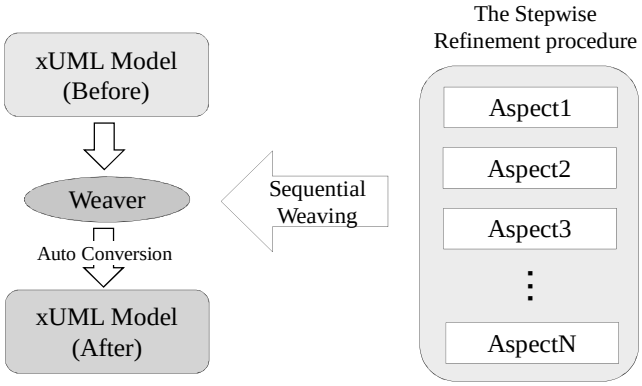

Figure 2. Model conversion using aspects

Owing to the stepwise refinement as presented above, a designer can efficiently designs and choice a model appropriate for embedded systems. Also, the designer verifies a model in a stepwise manner, and thereby it can prevent that design mistakes remain in downstream design. In addition, thanks to making system components at each step, the stepwise refinement accelerates reusability of modules.

5. EXPLORATION OF COMMUNICATION MODELS IN ASPECT-ORIENTED EXECUTABLE UML

Kinoshima et al. [Kinoshima, 2009] have already described the communication models mentioned above in xUML. By use of xUML to describe models, designers can make more abstract design properties without dependence on any system-level languages. This approach thus enhances reusability of modules, and improves design efficiency.

However, the procedure of the stepwise refinement of communication models is unformulated and undefined; consequently, it is difficult to automate the refinement.

Hence we propose to formalize the refinement procedures of communication models in aspect-oriented executable UML. The below presents the overview of aspect-oriented executable UML and our approach.

5.1 Aspect-Oriented Executable UML

Aspect-oriented executable UML, which we have suggested in [Iwata, 2008; Teruya, 2008], is an application of aspect-oriented techniques to xUML. In aspect-oriented xUML, a model structure can be automatically transformed by weaving an aspect of a model transformation. It is also possible to weave multiple aspects into an xUML model (Figure 2).

In this research, we consider the model refinement procedures as model transformations, which are formalized in aspects. By weaving the aspects, namely the refinement procedures, into the xUML model using the aspect weaver, the model transformation works can be automated.

This paper leaves the details of the implementation of the weaver and the aspect description format in aspect-oriented xUML to [Iwata, 2008; Teruya, 2008].

5.2 Aspect Descriptions and Experiment of Applying Aspects to xUML Models

In order to formalize the refinement procedures of communication models, we investigated what aspects are needed, and described as aspects. Here, this paper takes the refinement to event-triggered and time-triggered model for instance.

Figure 3 shows the class diagram of the specification model in xUML before an aspect weaving. We described the refinement procedure from the specification model to the event-triggered model using 48 aspects. For example, Figure 4 represents an outline of the aspect descriptions of the refinement from the specification model to the event-triggered model.

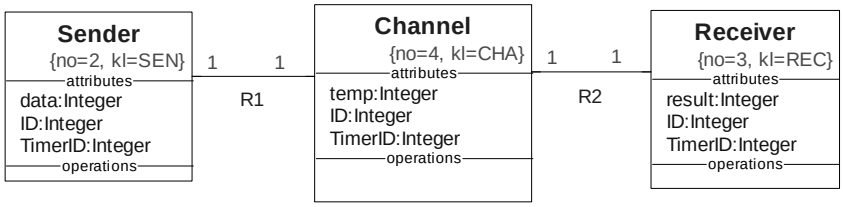

Figure 3. Specification model (before aspect weaving)

1. The addition of an arbiter class
   - apsect1: Defining class *Arbiter*
   - aspect2: Adding attribute *chechuse* (type: Boolean) to class *Arbiter*
   - aspect3: Adding attribute *ID* (type:Integer) to class *Arbiter*
   - …
   - aspect17: Adding the action definition to state *Arbiter_Check*
2. The addition of a filter class
   - aspect18: Defining class *Filter*
   - aspect19: Adding attribute *ID* (type: Integer) to class *Filter*
   - aspect20: Adding state *Filter_Init* to class *Filter*
   - …
   - aspect28: Adding the action definition to state *Filter_Check*
3. The addition of the function of arbitration and filtering to class *Channel*
   - aspect29: Change the action definition of state *Send_Wait* of class *Sender*
   - aspect30: Adding attribute *temp_id* (type:Integer) to *Channel*
   - aspect31 : Adding state *Channel_Arbiter_Check* to class *Channel*
   - …
   - aspect46: Change the action definition of state *Channel_End* of class *Channel*
4. The adjustment of associations
   - aspect47: Adding association R4 between class *Channel* and class *Arbiter*
   - aspect48: Adding association R5 between class *Channel* and class *Filter*

Figure 4. Aspect descriptions of the refinement from the specification model to the event-triggered model

We applied the aspect descriptions to the specification model (Figure 3) using the weaver. As the result, the specification model was automatically transformed into the event-triggered model (which is the refined model) shown in Figure 5. Then, through verifications in the model executions, we confirmed that the model was transformed accurately.

Here only mentioned the refinement above due to limitations of space, but we also verified that other refinements (to the time-triggered and centralized/decentralized models) can be automated by weaving aspects in the same manner.

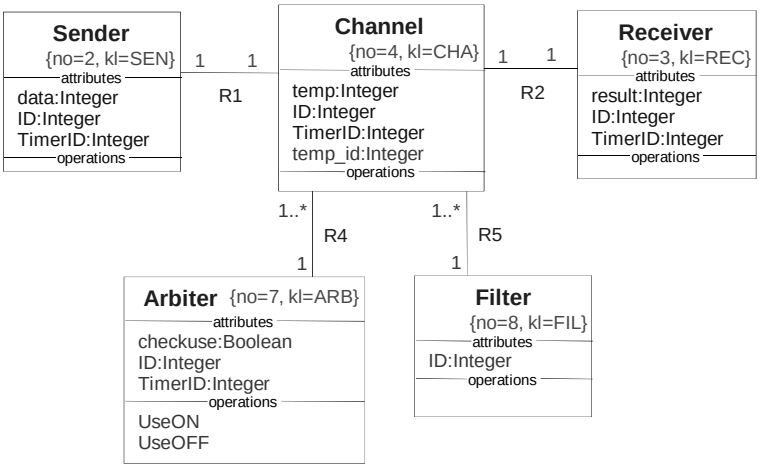

Figure 5. Event-triggered model (after aspect weaving)

6. CONCULUSIONS

In this paper, we proposed to describe the communication models in xUML, and also to formalize the refinement procedures of communication model exploration in aspect-oriented xUML. The former approach benefits to encourage design property reuse. Toward the realization of the latter, we described the refinement procedures of communication model exploration as aspects, and then confirmed that the refinement works can be automated by weaving aspects.

Future works include categorizing communication models further, applying this study to real-world applications.

REFERENCES

Chiba, S., 2005. *Aspect-Oriented Programming.* Gijutsu-Hyohron Co., Ltd, Tokyo, Japan.

Fowler, M. et al, 1999. *Refactoring*. Addison-Wesley Professional, Indiana, USA.

Gajski, D.D. et al, 2000. *SpecC: Specification Language and Methodology.* Kluwer Academic Publishers, Massachusetts, USA.

Gerstlauer, A. et al, 2002. *System Design: A Practical Guide with SpecC.* Kluwer Academic Publishers, Massachusetts, USA.

Iwata, E., 2008. XSLT-based Implementation of Aspects in Executable UML. *Graduate Thesis, Saitama University*. Saitama, Japan.

Kennedy Carter Ltd. iUML. *http://www.kc.com/*.

Kimura, M. et al, 2008. Stepwise Refinement Based on Refactoring of Executable UML. *Forum on Information Technology 2008*. Kanagawa, Japan, pp. 39-42.

Kinoshima, T. et al, 2007. Communication Model Exploration for Distributed Embedded Systems and System Level Interpretations. *EUC 2007 Workshops,* pp. 355-364.

Kinoshima, T., 2009. Communication Model Exploration in Executable UML. *Master Thesis, Saitama University*. Saitama, Japan.

Kobayashi, K. et al, 2007. Exploration of Communication Models in Design of Distributed Embedded Systems. *IEEJ Translations on Electrical and Electronic Engineering*, Vol. 2, No. 3, pp. 402-404.

Mellor, S.J. and Balcer, M.J., 2002. *Executable UML: A Foundation for Model-Driven Architecture.* Addison-Wesley Professional, Indiana, USA.

OMG (Object Management Group), 2000. UML 2.0. *http://www.uml.org/*.

OMG (Object Management Group), 2001. UML Action Semantics. *http://www.omg.org/cgi-bin/doc?ptc/02-01-09*.

Teruya, A. et al, 2008. Embedded System Design Based on Aspect-Oriented Executable UML. *Proceedings of 8th International Conference on Applied Computer Science*. Venice, Italy, pp. 247-252.