Embedded System Design Based on Aspect-Oriented Executable UML

AKIRA TERUYA*, EIICHIRO IWATA*, MASAHITO SUGAI*, MASAHIRO KIMURA**
NURUL AZMA ZAKARIA*, NORIKO MATSUMOTO*, NORIHIKO YOSHIDA*
* Saitama University
Department of Information and Computer Sciences
255 Shimo-Ohkubo, Saitama 338-8570, JAPAN

** Toshiba Solutions Corp.
Musashidai, Fuchu 183-8532, Japan

Abstract: System-level design methodology is one of the design paradigm for embedded systems. As embedded systems are getting much larger and more complex recently, the research to improve design productivity of system-level design is being required. In order to solve this issue, we have proposed a methodology of language independent system-level design using Executable UML (xUML) and a scheme for the procedure of stepwise refinement by the refactoring technique. However, the refactoring rules are informal and sometimes ambiguous because they are defined in a natural language. Consequently, we are aiming at formalizing the stepwise refinement procedure based on refactoring to automatize it. In this paper, we propose a formalization scheme based on the aspect-oriented technique to the system-level design in xUML.

Keywords: Aspect-oriented design, Executable UML, Refactoring, Embedded systems

1 Introduction

The methodology of system-level design, a stepwise refinement [1], is one of the design paradigm for embedded systems. This aims at deriving interactively and cooperatively hardware implementation and software implementation from an abstract specification.

As embedded systems are getting much larger and more complex recently, system level design methodologies using UML (Unified Modeling Language) are being researched actively to improve design productivity. However, UML does not have any testing mechanisms. When a designer specifies a model of a system in UML, she/he must re-describe it in a system-level design language to verify its correctness. That is, a designer can use UML only in the first stage in specification, and in the rest of the stepwise refinement process, she/he must use a concrete programming language.

In order to reduce this kind of dependence on the design language, we have proposed a methodology of language independent system-level design [2] using Executable UML (xUML) [3]. Executable UML is based on Model Driven Architecture (MDA) [4], and has testing mechanisms. In our proposal, designers can verify models and consistently represent the whole process in stepwise refinement based on xUML(Figure 1).

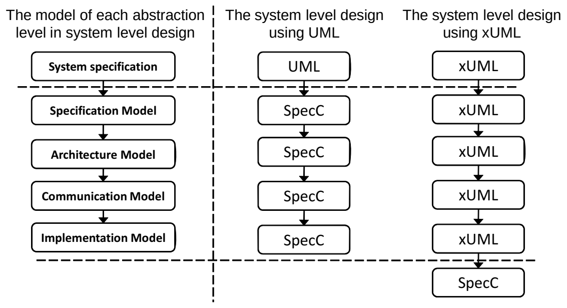


Figure 1: System-level design by using xUML

In system-level design, design productivity and quality still rely on designers' skills and experiences. To address this issue, we proposed a formalization scheme for the procedure of stepwise refinement by the refactoring [5] technique [2]. Refactoring is a restructuring procedure of system structures, while preserving its functions. However, this technique still has an issue that refactoring rules are informal and sometimes ambiguous because they are defined in a natural language.

Consequently, we are aiming at formalizing the stepwise refinement procedure based on refactoring to automatize it. In this paper, we propose a formalization scheme based on the aspect-oriented technique [6] to the system-level design in xUML. In particular, we formalize the model transformation of xUML using aspects, and define the refactoring rules in aspects.

This paper is organized as follows. Section 2 and 3 summarize Executable UML and Aspect-Oriented Programming respectively. Then, Section 4 presents our proposal, Section 5 describes some details on aspects for xUML model transformation, and Section 6 shows some experiment results. Section 7 overviews related works, and Section 8 contains concluding remarks.

2 Executable UML

Executable UML(xUML) is a profile of UML which enables rigorous definition of system behaviors. xUML comprises of an action language based on Action Semantics [7] and testing mechanisms of a model. A designer describes a model using the class chart, the state-machine chart, and the action language. Since the defined model is executable, we can test and verfy its correctness.

In this research, we use iUML [8] as a tool for xUML modeling.

3 Aspect-Oriented Programming

Aspect-Oriented Programing (AOP) was proposed as a technique for improving separation of crosscutting concerns in software systems.

A crosscutting concern is a procedure whose codes are scattered among several objects in Object-Oriented Programming (OOP), such as logging, synchronization, etc. Crosscutting concerns is difficult to modularize in OOP, and they make maintenance of program and recycling difficult.

Thus, AOP attempts to solve this problem by allowing a programmer to express crosscutting concerns in single modules called “aspects”. The mechanism of AOP is expressed in Join Point Model (JPM) which is composed of join points, advices, and pointcuts.

A join point is a well-defined location in a program execution (e.g. method calls, the execution of an exception handler block). An advice is a code of a crosscutting concern that should run at each join point in pointcuts. A pointcut specifies how to pick out a set of join points based on a certain criterion.

With AOP, we begin programming by implementing codes for classes using an OO language (e.g., Java), and then we separately define codes for aspects. Finally, both the code for classes and aspects

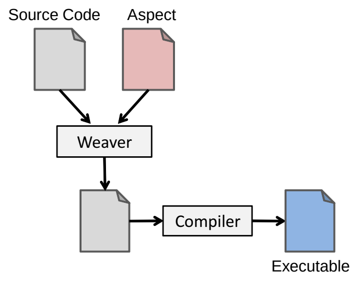


Figure 2: Aspect weaving

are combined into another code using an “Aspect Weaver”(Figure 2). As a result, a single aspect can contribute to the implementation of a number of methods, modules, or objects, increasing both reusability and maintainability of the code.

4 Outline of Our Methodology

This section presents an overview of our methodology. Our goal is to describe strict rules for the stepwise refinement procedure for the system-level design in xUML, based on refactoring defined in aspects, and to automate the work of stepwise refinement.

Therefore, we propose the method of the procedure formulation based on aspect oriented techniques in this research. The basic idea of this method is as follows.

(1) Defining operations for xUML model transformation in aspects

One operation for model transformation of xUML (e.g., addition of the class, addition of the method, etc) is described as an aspect (Figure 3-(a)). As a result, the model transformation procedure (equal to the stepwise refinement procedure) can be defined in aspects (Figure 3-(b)).

(2) Automating the work of model transformation

By using an aspect weaver to weave aspects to an xUML model, the work of model transformation can be automated.

(3) Aggregating aspects into hierarchies

As mentioned above, the stepwise refinement procedure can be expressed in aspects. So, the mechanism of aggregating the aspects into a hierarchy is necessary in order to improve modularity, readability, and reusability of the stepwise refinement procedure (Figure 3-(c)). We apply an aggregating technique of aspects for UML (not xUML) [9] to xUML.

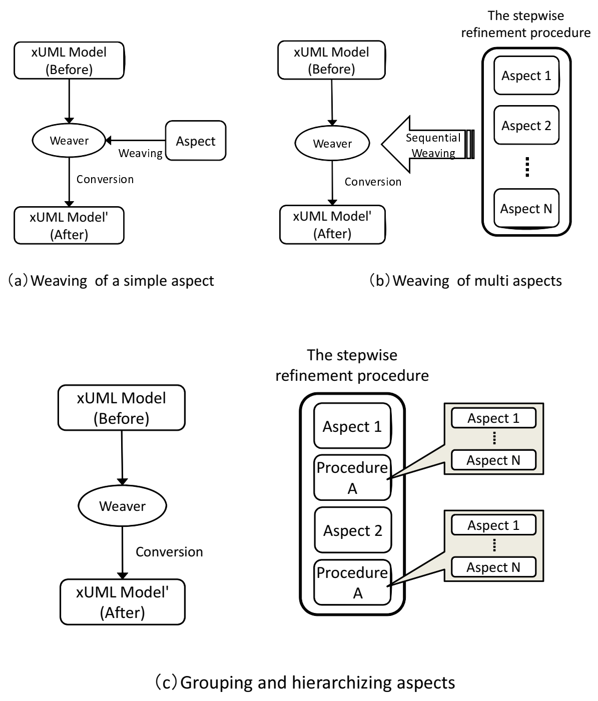

xUML Model (Before)
Weaver
Aspect
Weaving
Conversion
xUML Model' (After)
(a)Weaving of a simple aspect
The stepwise refinement procedure
Aspect 1
Aspect 2
Aspect N
Sequential Weaving
(b)Weaving of multi aspects
Procedure A
(c)Grouping and hierarchizing aspects

Figure 3: The model conversion using the aspect

By realizing the above idea, the stepwise refinement procedure for the system-level design in xUML can be described strictly and the reuse of the procedure becomes easy. As a result, the improvement of the design efficiency can be expected.

5 Aspects for xUML Model Transformation

This section explains how to realize aspects which can apply to xUML models.

5.1 Join Point and Advice for xUML Model Transformation

In order to realize the aspects that can be applied to an xUML model, it is necessary to define the Join Point Model corresponding to the xUML model.

So, we have categolized join points required for the model transformation out of components in xUML models (the class chart, the state-machine chart, and the action language). And, we have defined the type of an advice applied in each join point (Table 1). The definition of type of the advices is based on the operation in the model transformation. As a result, designers can define xUML model transformation processes in a formalized and structured manner using aspects.

Table 1: A list of xUML join point and available advices

Type of join point	Available type of advices
domain	add-class
	remove-class
	add-association
class	add-attribute
	add-operation
	add-state
	add-signal
	add-transition
	rename
attribute	remove-attribute
operation	remove-operation
	add-parameter
	add-action
state	remove-state
	add-action
	rename
transition	remove-transition
	replace-transition-signal
signal	remove-signal
	add-parameter
	rename
action	remove-action
	replace-action
association	remove-acsociation
prameter	remove-parameter
	replace-transition-signal

5.2 The description form of an aspect

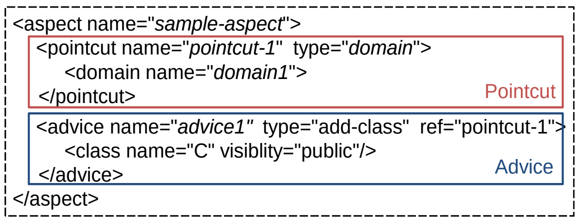
We have decided to use XML to describe an aspect, because XML has an advantage that it is an internaitonal standard, and its structure of tags and attributes makes desiging and constructing an aspect weaver easy. The description form of an aspect is as follows.

1. A single aspect

Aspect is composed of its name, pointcuts, and advices (Figure 4).

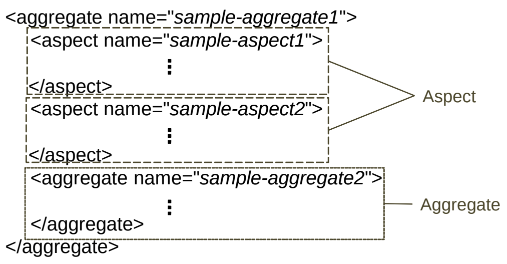
A single aspect contains zero or more pointcut(s). The definition of one pointcut consists of its name, type, and some definition body of the pointcut. The type of a pointcut expresses the type of join points which specifies a destination of aspect weaving.

A single aspect contains zero or more advice(s). The definition of an advice consists of its name, type, the pointcut name to apply it, and the definition body of the advice. In the type of advice, the kind of processing contents performed by the pointcut is described. When the definition of two or more advices is described, the order of



```
<aspect name="sample-aspect">
  <pointcut name="pointcut-1"  type="domain">
      <domain name="domain1">
  </pointcut>                                        Pointcut
  <advice name="advice1"  type="add-class"  ref="pointcut-1">
      <class name="C" visiblity="public"/>
  </advice>                                          Advice
</aspect>
```

Figure 4: The description example of a single aspect



```
<aggregate name="sample-aggregate1">
  <aspect name="sample-aspect1">
          ⋮
  </aspect>                                      Aspect
  <aspect name="sample-aspect2">
          ⋮
  </aspect>
  <aggregate name="sample-aggregate2">
          ⋮                                      Aggregate
  </aggregate>
</aggregate>
```

Figure 5: The description example of the grouping of aspects

applying the advices is assumed to be the same as the order of the definition.

2. The grouping of aspects

When two or more aspects exist, they can be grouped as "Aggregate". Aggregate is expressed with an "aggregate" tag. An aggregate contains zero or more aggregate and aspect respectively(Figure 5).

As a result, aspects can be hierarchically described, and can be reused easily. The hierarchical aspects are applied in the appearance order of the aspect definition.

5.3 The flow of weave to xUML model

Figure 6 shows the outline of aspect weaving for the xUML models. The flow is as follows.

1. Translate the target xUML model from the diagram form to the XML form.
2. Weave aspects to the xUML model in XML.
   (a) Translate the aspect from the XML form to XSLT form.
   (b) Apply this XSLT to the target model using the aspect weaver.
3. Translate the target xUML model from the XML form to the diagrom form.

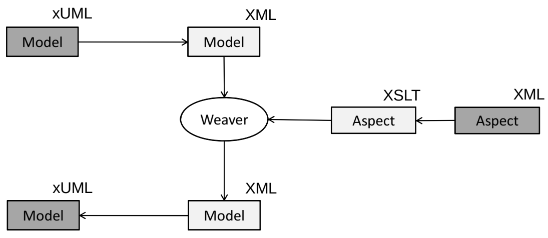


Figure 6: Outline of weaving aspect to xUML model

This aspect weaver has been implemented and reported elsewhere [10].

6 Experiment for xUML Model Transformation with Aspects

In order to confirm that the stepwise refinement design of xUML model can be performed using the proposed aspect framework, we defined aspects for the xUML model transformation [11] of "the separation of a communication and an operation" which is one of the refactoring rules defined in [2].

Figure 7 shows the class chart of the xUML model before the aspect application. We made 49 aspects for the model transformation operation which met "the separation of a communication and an operation", and these aspects were continuously woven into the xUML model. As a result, the model translation equivalent to application of the refactoring rules can be done automatically (Figure 8).

By this experiment, we confirmed that the proposed aspects are able to formalize the stepwise refinement procedure for xUML and that the automation of the stepwise refinement works well.

Refactoring rule of "the separation of a communication and an operation"

1. **The definition of a channel class (encapsulation of an attribute and an event)**
   Define the channel class which encapsulated the attribute and the event.
2. **The addition of a channel class**
   (a) Replace attribute and event with the channel class, and relate the channel class to behavior class of sending and receiving side.
   (b) Move the attribute and the event used in the channel class to the behavior class that calls the channel class as a local value.
3. **Update of a behavior class's communication**

Replace a description of the event access with a description of the access to the channel class.

The aspect description of "the separation of a communication and an operation"

1. The definition of a channel class

   aspect1 Adding channel class C2 to the class chart (add-class!K

   aspect2 Adding attribute *buf* to channel class C2 (add-attribute)

   aspect3 Adding attribute *flag* to channel class C2(add-attribute)

   aspect4 Adding operation *send* to channel class C2 (add-operation)

   aspect5 Adding parameter *flag* to operation *send* (add-parameter)

   aspect6 Adding the action description to operation *send* (add-action)

   ⋮

   aspect35 Adding the action description to state *End* (add-action)

2. The addition of a channel class

   aspect36 Deleting the relation R5 between behavior class B2 and B3 (remove-association)

   aspect37 Adding relation R5 to behavior class B2 and channel class C2 (add-association)

   ⋮

   aspect42 Adding attribute *v2* to behavior class B3 (add-attribute)

3. Update of a behavior class's communication

   aspect43 Change the name of state *Start* in behavior class B2 to *StartC2_Send* (rename)

   aspect44 Change the action description of state *StartC2_Send* in behavior class B2 (replace-action)

   ⋮

   aspect49 Chage *Event_e2_Signal* in behavior class B3 to *returnSignal* (modify-transition-signal)

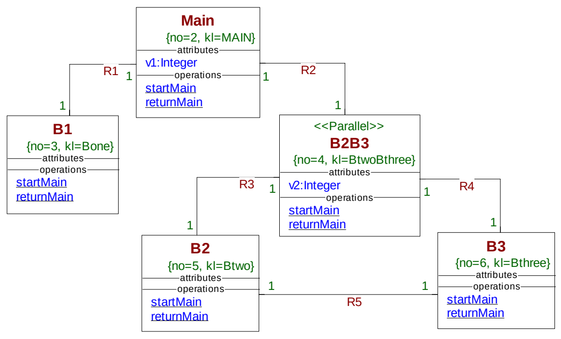


Figure 7: Class diagram(Before aspect application)

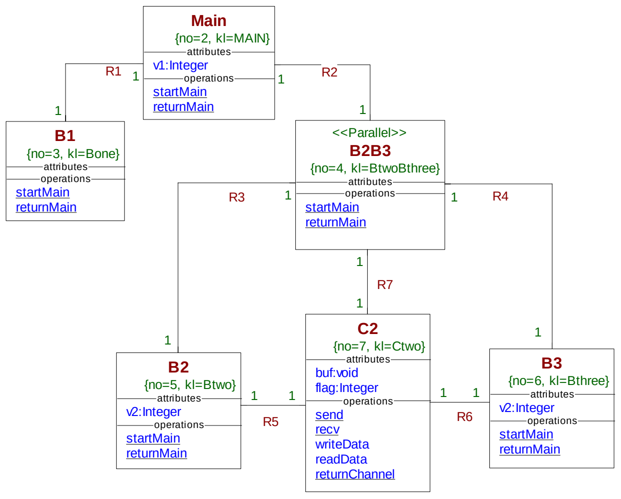


Figure 8: Class diagram(After aspect application)

7 Related Works

Yamazaki et al. presented attempts to introduce aspect-oriented techniques to system-level design in the SpecC language [12]. In this research, aspects express the stepwise refinement procedure of the SpecC models, and they enable the designer to automate the stepwise refinement work of the SpecC model. On the other hand, our research attempts to introduce aspect-oriented techniques to system-level design in xUML. Aspects can express the stepwise refinement procedure for the xUML model. This enable the designer to automate the stepwise refinement work in xUML as well.

Ubayashi et al. proposed a method of making a MDA model compiler using the aspect-oriented techniques [9]. In MDA, a model compiler transforms a Platform-Independent Model (PIM) into a Platform-Specific Model (PSM) which includes platform specific information. In this research, aspects express the platform information and this is weaved to UML model. So, the model compiler is implemented as the aspect processing system for UML model. We made the aspect processing system for xUML [10]. Our research uses this system for the stepwise refinement of xUML models.

L. Fuentes et al. presented an idea to introduce the AOP technology to xUML model in the field of software design [13]. The essence of this idea is that the aspect can be described in a xUML diagram as well and that the dynamic join point is introduced. Our research uses the AOP technology for the system-level design. The essence of our idea is that the aspect is described in XML, that the static join point is used, and that aspcts can be hierarchical. Using the static join points, we can transform easily the structure of xUML models rather than using dynamic join points. Also, hierarchical aspects facilitates the reuse of stepwise refinement procedures.

8 Conclusions

In order to describe strict rules of the stepwise refinement procedures and to automate the procedures, this research proposes the formalization framework based on aspect-oriented techniques. Through some experiment, we confirmed that the stepwise refinement design with aspect is well perfomed in an automatic manner.

Future works include making aspect templates for the stepwise refinement procedure to reduce the labor of new aspect definition.

Acknowledgments: This research was supported in part by Joint Research Project with Nippon Signal Co. Ltd..

References:

[1] Daniel D.Gajski, J. Zhu, R. Domer, A. Gerstlauer, and S. Zhao, *SpecC: Specification Language and Methodology,* Kluwer Academic Publishers, 2000.

[2] M. Kimura, N. A. Zakaria, A. Teruya, N. Matsumoto, N. Yoshida, *Stepwise Refinement Based on Refactoring of Executable-UML,* Proc. IPSJ/IEICE Forum on Information Technology 2008, Vol.1, pp.39-42.

[3] Stephen J. Mellor, Marc J. Balcer, *Executable UML: A Foundation for Model-Driven Architecture,* Addison-Wesley Publishers, 2002.

[4] Stephen J. Mellor, K. Scott, A. Uhl, D. Weise, *Mda Distilled: Principles of Model-Driven Architecture,* Addison-Wesley Publishers, 2004.

[5] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring,* Addison-Wesley, 1999.

[6] S. Chiba, *Aspect-Oriented Programing(in Japanese),* Gijutsu-Hyohron Co., Ltd, 2005.

[7] *UML Action Semantics,* http://www.omg.org/cgi-bin/doc?ptc/02-01-09

[8] *iUML,* Kennedy Carter Ltd., http://www.kc.com/

[9] N. Ubayashi, T. Tamai, S. Sano, Y. Maeno, S. Murakami, *Model Compiler Construction Based on Aspect-Oriented Mechanisms,* 4th ACM SIGPLAN International Conference on Generative Programming and Component Engineering (GPCE 2005), Lecture Notes in Computer Science, Springer-Verlag, vol.3676, pp.109-124 , 2005.

[10] E. Iwata, *XSLT-based Implementation of Aspects in Executable UML,* (in Japanese,) Graduate Thesis, Saitama University, 2008.

[11] A. Teruya, *System Level Design Based on Aspect-Oriented Executable UML,* Graduate Thesis, Saitama University, 2008.

[12] R. Yamasaki, K. Kobayashi, N. A. Zakaria, S. Narazaki, N. Yoshida, *Application of Aspect-Oriented Techniques to System Level Design,* IEICE/IPSJ Information Technology Letters, Vol.5, September 2006.

[13] L. Fuentes!$P. Sanchez!$*Designing and Weaving Aspect-Oriented Executable UML models*!$ Journal of Object Technology!$2007.