International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:7, 2009

Stepwise Refinement in Executable-UML for Embedded System Design: A Preliminary Study

Nurul Azma Zakaria, Masahiro Kimura, Noriko Matsumoto, Norihiko Yoshida, *Member, IEEE*

Abstract—The fast growth in complexity coupled with requests for shorter development periods for embedded systems are bringing demands towards a more effective, i.e. higher-abstract, design process for hardaware/software integrated design. In Software Engineering area, Model Driven Architecture (MDA) and Executable UML (xUML) has been accepted to bring further improvement in software design. This paper constructs MDA and xUML stepwise transformations from an abstract specification model to a more concrete implementation model using the refactoring technique for hardaware/software integrated design. This approach provides clear and structured models which enables quick exploration and synthesis, and early stage verification.

Keywords—Hardaware/Software Integrated Design, Model Driven Architecture, Executable UML, Refactoring

I. Introduction

TO accelerate hardaware/software integrated design processes for embedded systems, several design methodologies and description languages for them, called system-level design languages, have been proposed such as SpecC, SystemC and SystemVerilog [1], [2], and are gradually being used in practice. However, the fast growth in complexity coupled with requests for shorter development periods are bringing demands towards a more effective, i.e. higher-abstract, design process.

In Software Engineering area, Model Driven Architecture (MDA) and Executable UML (xUML) [3] has been accepted to realize higher level of abstraction to bring further improvement in software design. The research goal is to formalize transformation in xUML from an abstract specification model to a more concrete implementation model in the MDA process for hardaware/software integrated design. This paper tries to construct xUML stepwise transformations using the refactoring technique [4]. Fig. 1 shows the flow of the proposed approach in comparison with typical system-level design and MDA flow.

Section 2 summarizes MDA and xUML, and Section 3 introduces refactoring and stepwise transformation. Section 4 explores the suitability of this technique using a channel partitioning example [1]. Section 5 mentions some related research works, and Section 6 contains some concluding remarks.

II. Modeling in Executable UML

Models in MDA are formal representation of the function, behavior and structure of the system. MDA comprises two model abstractions. Platform Independent Model (PIM) is a model that describes all requirements of the system, while being free from implementation specific aspects. Since PIMs are expressed as executable models, they are also verifiable. On the other hand, Platform Specific Model (PSM) comprises all the functionality expressed in the PIM with the added design concerns on a specific platform. A PSM is derived from its corresponding PIM by applying a set of systematic transformation. Each of the models is represented in xUML which consists of class and state machine diagrams. The former describes static structures of a system, and the latter depicts dynamic behaviors of the classes. Within a state machine diagram, action semantics is defined using Action Specification Language (ASL) [3] to allow model execution. An MDA tool iUML [3] is used to execute and verify models in xUML.

N. A. Zakaria, M. Kimura, N. Matsumoto and N. Yoshida are with Department of Computer Science, Saitama University, Saitama 338-8570, Japan e-mail: {azma|masahiro|noriko|yoshida}@ss.ics.saitama-u.ac.jp.

Manuscript received April 19, 2009; revised April 20, 2009.

III. Refactoring-based Transformation

A. Refactoring Rules

Refactoring [4] is a technique for restructuring an existing body of code, altering its internal structure without changing

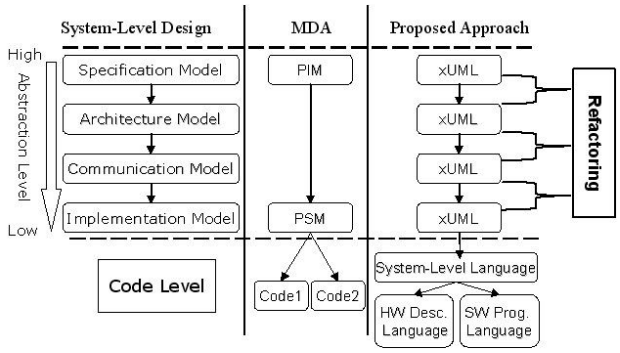

Fig. 1. Proposed design flow

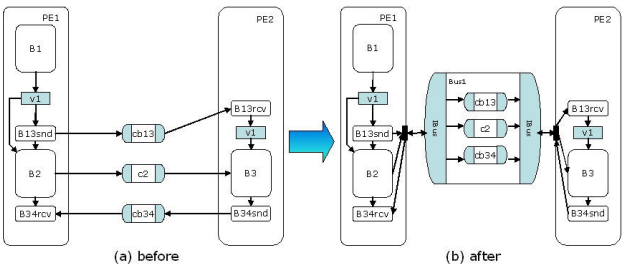

Fig. 2. Channel partitioning task

its external behavior. It is a series of small behavior-preserving transformations. A series of step-by-step refactoring rules involved in "channel partitioning" are shown below as an example. "Channel partitioning" is one of the fundamental and important processes in hardware/software integrated design, which allocates an abstract communication channel between components to an implementation of concrete wired buses. This does not appear in the original MDA and xUML for software (not hardware/software integrated) design.

1) Add another level of hierarchy to the existing class diagram to model its bus structure.
   - Create a new class to represent the bus structure.
   - Add relationship to Main class.
2) Bind channel classes to the bus class and group communication.
   - Remove relationship from Main class.
   - Add relationship to the bus class.
3) Renew access towards channel classes.
   - Remove relationship from channel classes.
   - Add relationship to the bus class

B. Transformation

Fig. 2 shows a schematic diagram of the transformation process in "channel partitioning". The class organizations are shown in Fig. 3 and Fig. 4 (at the end of the paper) which correspond to Fig. 2 (before) and (after) respectively. Each function or behavior is assigned to a class, without any consideration on concrete implementation. In the initial model, Main class represents top-level hierarchy of the design which connects to subclasses such as PE1, PE2 and C2. A new class named Bus1 is created and linked to Main class to model the bus structure.

Some intermediate refactoring is applied to the initial model which finally derives xUML definition of a more concrete model as shown in Fig. 4. The xUML model becomes more detailed due to change in connectivity among classes such as Bus1, B13Rcv and B13Snd. What is important is that the functions and behaviors of the original specification model are strictly preserved, while its structure is modified so as to make the model more concrete.

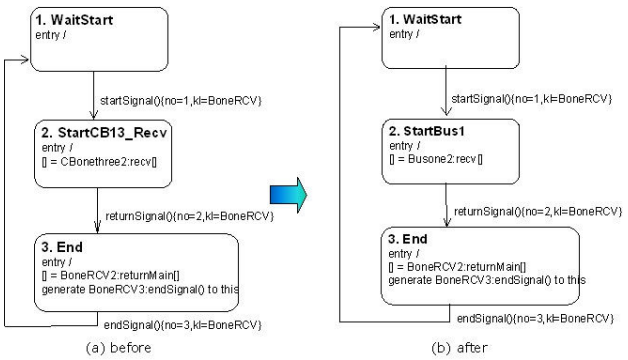

Fig. 5. State transition of B13Rcv class

Figure 5 depicts the state transition which relates to transformation illustrated by Fig. 3 and Fig. 4. The chart for B13Rcv class is shown as an example. This type of state transition definition is attached to every class presented in the class diagram.

Initially, startSignal instantiates the second state which executes startMain operation of CB13 class but in final transformation startSignal activates startMain operation of Bus1 class instead. This is due to changes occurred in Fig. 4. Finally, after all statements are executed returnSignal starts returnMain operation of B13Rcv class to end the process in both cases.

IV. Related Works

There are already some related studies on application of UML or MDA to systems design [5], [6], [7]. However, there has been none yet on xUML transformation based on refactoring for system-level, hardware/software integrated design.

V. Concluding Remarks

A modeling framework has been introduced in which designs can be specified using xUML notations so as to support specification and modeling of embedded system. This paper presented some refactoring rules which acts as guideline in design process. This paper have verified the approach by simulating the models using iUML suite. This approach provides clear and structured models which enables quick exploration and synthesis, and early stage verification. Future works will include modeling remaining tasks of stepwise refinement to enable generation of complete system-level design using the framework and apply them to real applications.

Acknowledgment

This research was supported in part by Nippon Signal Co. Ltd.

References

[1] D. D. Gajski, J. Zhu, R. Doemer, A. Gerstlauer and S. Zhao, *SpecC: Specification Language and Methodology*, Kluwer, 2000.
[2] P. Marwedel, *Embedded System Design*, Springer, 2006.
[3] C. Raistrick, P. Francis, J. Wright, C. Carter and I. Wilkie, *Model Driven Architecture with Executable UML*, Cambridge University Press, 2004.
[4] M. Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.
[5] E. Riccobene, P. Scandurra, A. Rosti and S. Bocchio, "A SoC Design Methodology Involving a UML 2.0 Profile for SystemC", Proc. Design, Automation and Test in Europe, 2005.
[6] T. Schattkowsky and W. Muller, "Model-Based Design of Embedded Systems", Proc. 7th IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing, 2004.
[7] Proc. 2006 Workshop on UML for SoC Design, in conjunction with ACM/IEEE 43th Design Automation Conf., 2006.

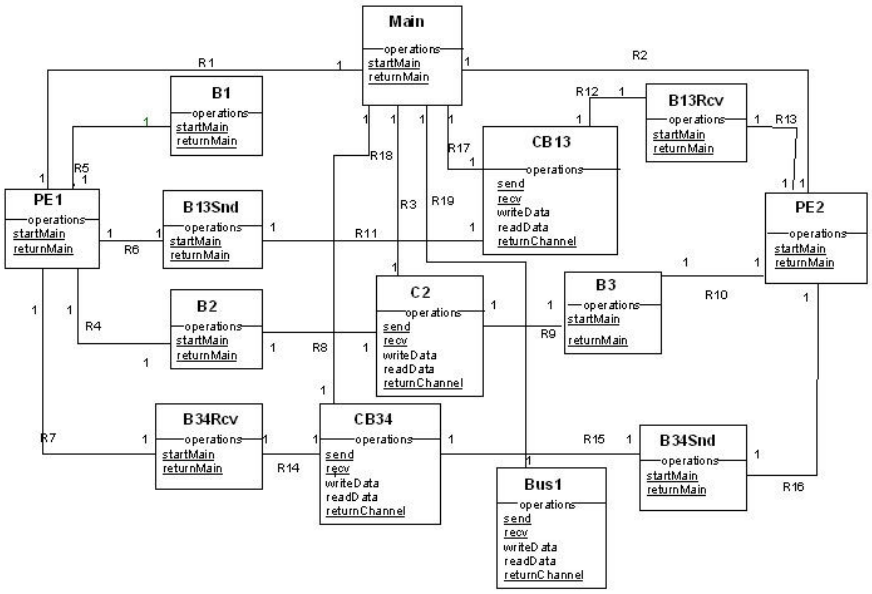

Fig. 3. Initial xUML model

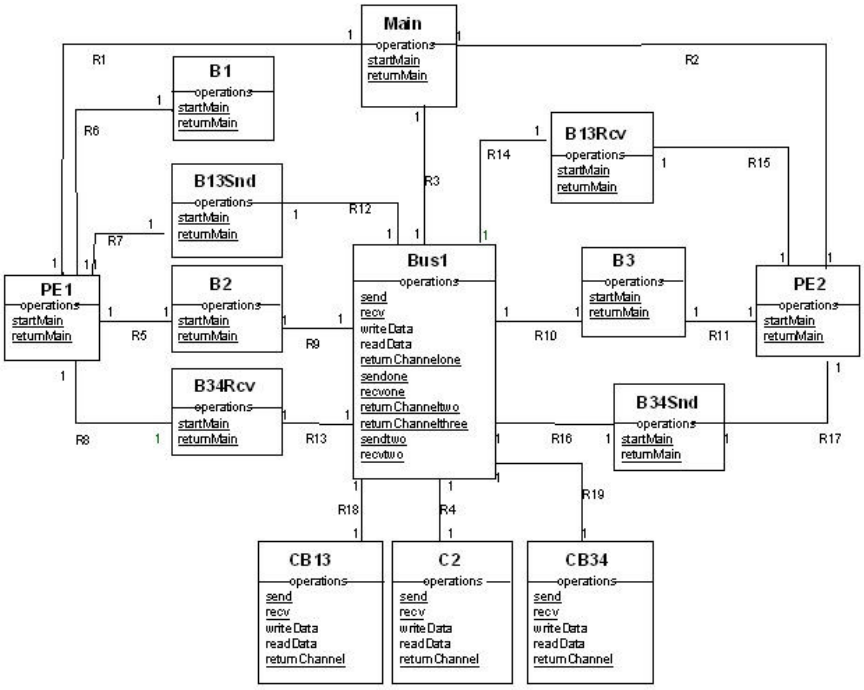

Fig. 4. Final xUML model