PROCEEDINGS OF
THE 2008 INTERNATIONAL CONFERENCE ON
SOFTWARE ENGINEERING RESEARCH & PRACTICE

SERP'08

Volume II

Editors

Hamid R. Arabnia
Hassan Reza

Associate Editors

Emanuel Grant, Jalal Karam
Vincent Schmidt
Ashu M. G. Solo

WORLDCOMP'08
July 14-17, 2008
Las Vegas Nevada, USA
www.world-academy-of-science.org

SESSION: THEORETIC APPROACHES

Automatic MDA (Model Driven Architecture) Transformations for Heterogeneous Embedded Systems

Woo Yeol Kim[1], Hyun Seung Son[1], Y. B. Park[2], B. H. Park[3], C. R. Carlson[4], R. Young Chul Kim[1]
[1]Dept. of Computer & Information Comm., Hongik University, Korea
[2]DanKook University, [3]The Armed Forces Medical Command, Korea
[4]Dept. of ITM, Illinois Institute of Technology, Chicago, USA
{john, son, bob}@hongik.ac.kr[1]

Abstract - *We adopt the MDA mechanism for embedded s/w development, which reduces the lifecycle of s/w development. In this paper, we propose the automatic MDA (Model Driven Architecture) transformations to develop the heterogeneous embedded software. We first model 'Target Independent Meta Model' (TIM) through Requirement analysis. With the automatic MDA transformations, then automatically produce some 'Target Specific Model's (TSMs) selecting the different OS APIs and/or different processors, and then possible generate 'Target Dependent Code' (TDC) such as Java, C++, or C per each specific TSM. As a result, it is possible to port a specific TDC into the target system. We show one example which illustrates the proposed approach.*

Keywords: Model Driven Architecture, Embedded System, Small Unmanned Ground Vehicle, Unified Modeling Language

1 Introduction

Today's demands for numerously diverse embedded systems are on the very rapidly and greatly increase in recent years. Some industrial software developers are now in progress about the huge complexity of embedded system. Our researches focus on automatic development tool for embedded software (such as design, model, code, and test) to develop the heterogeneous embedded systems. But it may be hard automatically to develop embedded software because the embedded software mechanism is dependent on the particular hardware system and also is just code oriented development [1]. To solve these problems, we propose the automatic MDA transformations mechanism [2] using a general meta-model for embedded s/w development. We suggest detail activities of lifecycle for embedded s/w development and refined multiple V-model on MDA [6]. This will be possible automatically to generate target dependent code per each of target specific models via target independent software. With this transformation, we can help automatically to reuse software by product (such as design, model, code, and test), and to reduce the lifecycle of heterogeneous embedded software development.

This paper is organized as follows: in section 2, we describe the embedded model driven architecture (MDA) and refined multiple V-model. In section 3, we show our automatic MDA transformations for develop embedded s/w system. In section 4, we show the modeling example of heterogeneous embedded systems used in this paper.

2 Embedded Model Driven Architecture

The original MDA[2] is the OMG proposed approach for system development. It primarily focuses on software development platform. The MDA is based on one meta-model describing the systems to be built. A system description is made of numerous models, each model representing a different level of abstraction. The modeled system can be deployed on one or more platforms via model to model transformations [7].

We mention to adopt MDA mechanism into embedded software development as follows.

2.1 Refined Multiple V-model on MDA

The original multiple V-model[8] is focused on developing (or modeling) a system based on a particular target domain. This is difficult to apply for other target domains. So, we attempt to refine multiple V-model on MDA, which solves problems of the original V-model.

This research is financially supported by the Ministry of Commerce, Industry and Energy (MOCIE) and Korea Industrial Technology Foundation (KOTEF) through the Human Resource Training Project for Regional Innovation (2008)

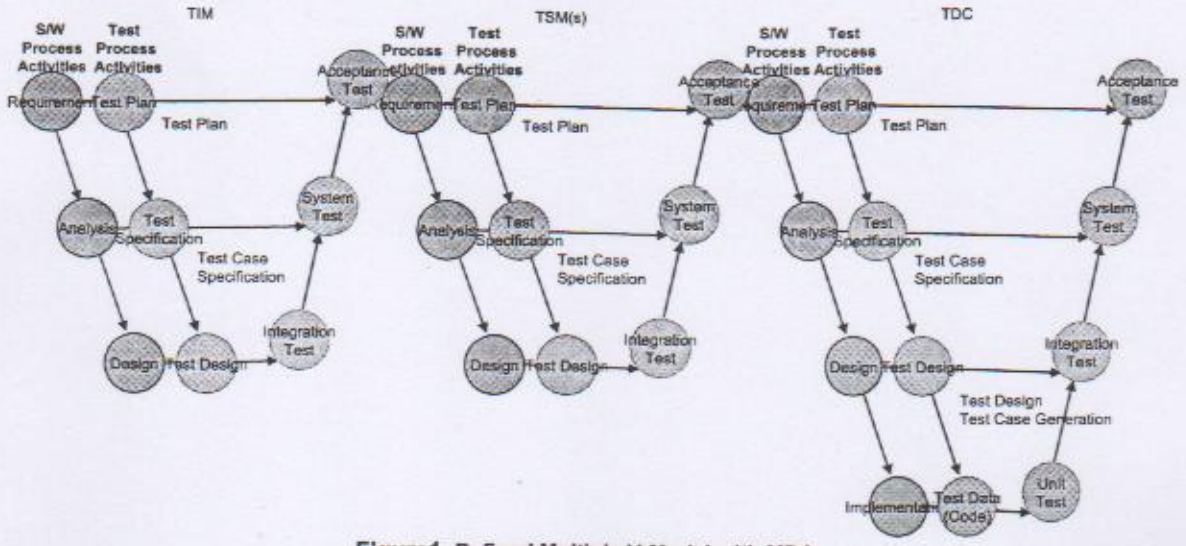

Figure1. Refined Multiple V-Model with MDA

In figure 1, it describes to adapt multiple V-model with MDA. The refined multiple V-model is also a development model process (Target Independent Model, Target Specific Model(s), Target Dependent Code(s)) which is developed the different versions of the heterogeneous systems. The first V-model is focused on the target independent model. The middle V-model is focused on the target specific model(s). The last V-model is focused on the target dependent code(s).

The refined one may usefully develop heterogeneous embedded systems with reusability on different target domains. Moreover it can also work parallel with both of s/w development process and test process. Due to these double processes, it may be possible to develop more safe and reliable software components.

So we develop the automatic tools for automatic MDA transformations, and code generations such as Java, C++, or C.

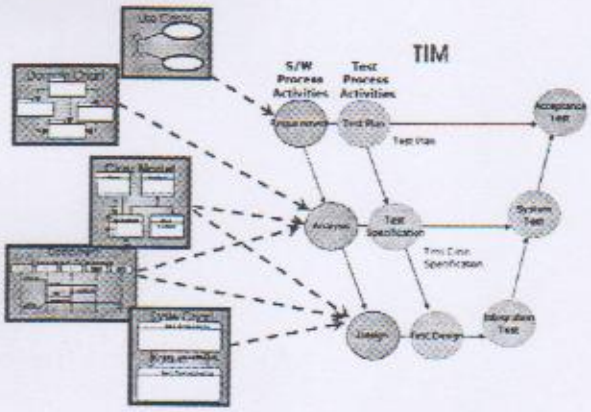

Figure 2.a. Mapping diagrams at TIM phase

In figure 2.a,b.c, it is just more detail described to map diagrams at the first V-model. The use case diagram is used during requirement to represent the functionality of the system from the user's point of view. During analysis, the class diagram describes the structure of the system. The concurrent message diagram and concurrent state diagram describe the internal concurrent behavior of the system during design.

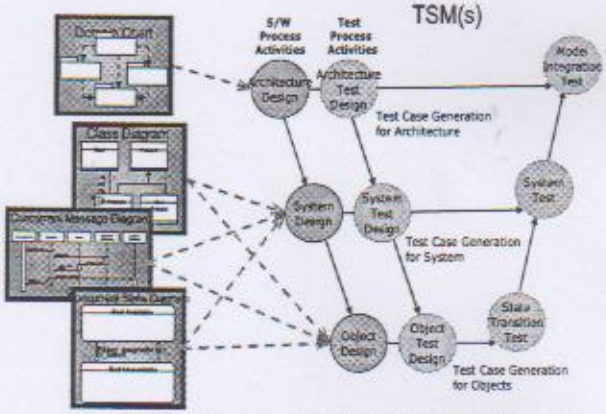

Figure 2.b. Mapping diagrams at TSM phase

It is necessary work to make the automatic transformation from TIM(Target Independent Model) phase to TSM(Target Specific Model) or from TSM(Target Specific Model) to TDC(Target Dependent Code). Therefore, it should make a reliable model of TIM at the first step. We don't mention about next remaining phases in this paper.

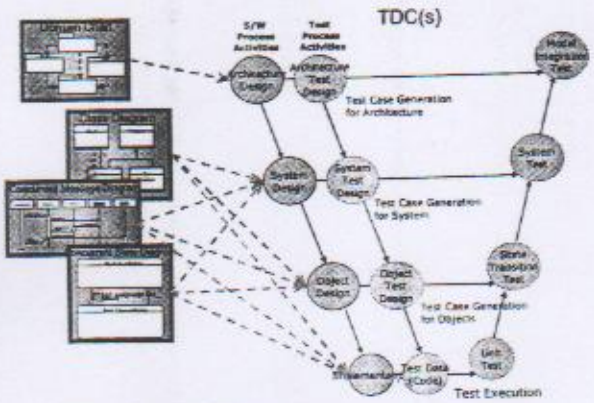

Figure 2.c. Mapping diagrams at TDC phase

2.2 MDA based Modeling Approach

We apply MDA to embedded software modeling approach. Our modeling approach consists of static modeling and dynamic modeling. Static model uses class diagram to represent the static aspect of a system. In dynamic model, concurrent message diagram and concurrent state diagram [11] represent the dynamic concurrent behavior of the system in figure 3.

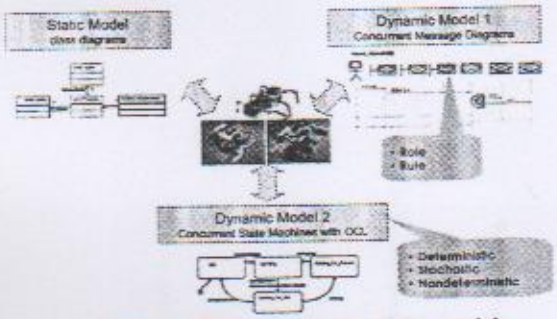

Figure 3. The relationship of the models

3 MDA Transformation Process

Our defined MDA Transformation Process [5,10] consists of TIM (Target independent model) stage, TSM (Target specific model) stage, and TDC (Target dependent code) stage. TIM defines a general model independent of the particular target domains. After requirement analysis, we may design TIM with extended xUML[4,5,11] and UML profile[3] at this TIM stage.

TSM defines a specific model dependent of the OS, and hardware within the particular target domains.

At this TSM stage, we can model more complete with additional functions on TSM automatically transformed with TIM. TDC defines the source code per the specific target system. At TDC stage we can automatically generate codes whatever we need such as Java, C++, or C. But we manually need to write a little part of the particular function codes that do automatically impossible. Finally we can port complete executable code(s) into the target system(s).

Our MDA Transformation Process consists of two transformation steps T1, T2(or T2', T2'') in figure 4.

The first transformation T1 is automatically transformed TIM (which merges with the predefined Processor profile and OS profile) into TSM. The second transformation T2 is automatically generated the right executable TDC per each different target system. Its possible generable code languages are C, C++, and Java. The code generating method is automatically generated using conversing text from model [6].

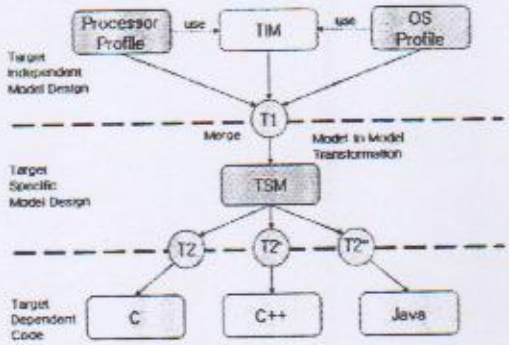

Figure 4. MDA Development Process

3.1 Transformation T1: Transforming from a TIM to TSM(s)

To transform TIM into TSM(s), we mention about four layers such as application layer, service layer, operating system layer, and processor layer. Each Layer is described as follows: Application Layer is the top layer for modeling a system. The user can do modeling with services on service layer. But we can not change the service name for T1 transformation[9].

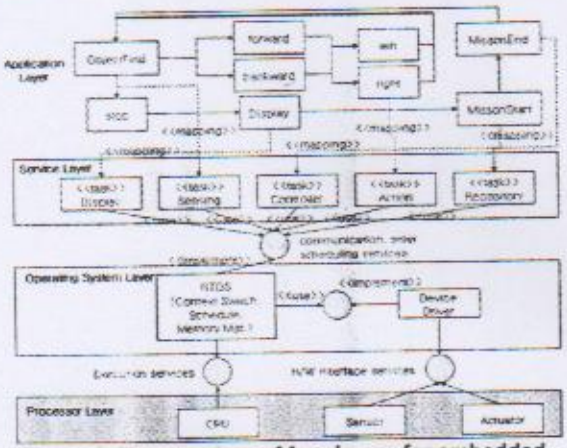

Figure 5. The structure of four layers for embedded system development

In figure 5, there is mapping the behaviors of application layer into services of service layer. Then associate with operating system layer and processor layer. The user will possible develop a system having only knowledge of services without detail implementation. Operating System Layer consists of APIs of OS. Just work with context switch, scheduling, memory management, timer service of OS on the processor layer. The last processor layer is the bottom layer dependent of hardware.

3.2 Transformation T2: Transforming from TSM(s) to a TDC(s)

To transform the actually executable code from TSM through T1 transformations, we should do execute T2 transformations.

T2 is generated a language with meta-template model on the basic of the class diagram, concurrent message diagram, and concurrent state diagram of TSM. Figure 6 shows the meta template model for these diagrams.

<table>
<tr><td colspan="2">Class Name: string</td></tr>
<tr><td colspan="2">Package List: List</td></tr>
<tr><td colspan="2">Parent List : List</td></tr>
<tr><td colspan="2">Interface List : List</td></tr>
<tr><td rowspan="2">Association List: List</td><td>Attribute</td></tr>
<tr><td>SetFunction</td></tr>
<tr><td colspan="2">Attribute List: List</td></tr>
<tr><td rowspan="2">Function List: List</td><td>Head</td></tr>
<tr><td>Body</td></tr>
</table>

Figure 6. Meta template model

Meta template model consists of all elements for the actual code generation in figure 6. "Class Name" stores the strings of class name. "Package List" stores a type of linked list to include package or library. "Parent List" is the above one of current class, which stores a type of linked list. "Interface List" stores interface name. "Association List" also stores a type of linked list about association relationship between other classes. " Attribute List" also stores a type of linked list about the class attributes. "Function List" stores a type of linked list about the class methods.

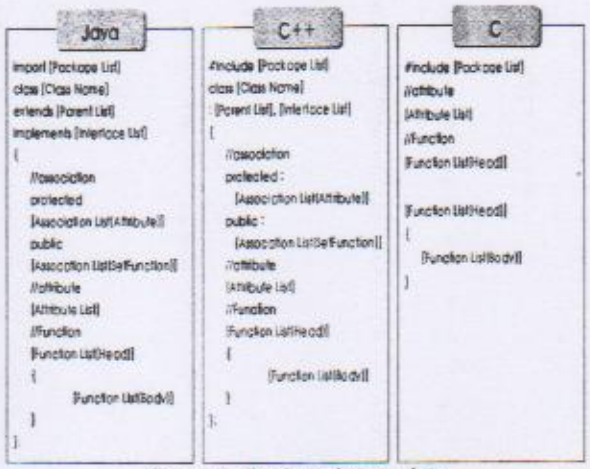

Figure 7. Each code template

After storing all information of diagrams through meta template model, it is transformed like figure 7.

It shows that each of meta template names in figure 6 matches each code template in figure 7. As a result, we can execute T2 transformation automatically to generate each code with collecting information on Meta template model of TSM in figure 6.

4 The automatic MDA transformation tool

We would like to use one example of small unmanned ground vehicle system (SUGV) in figure 8.

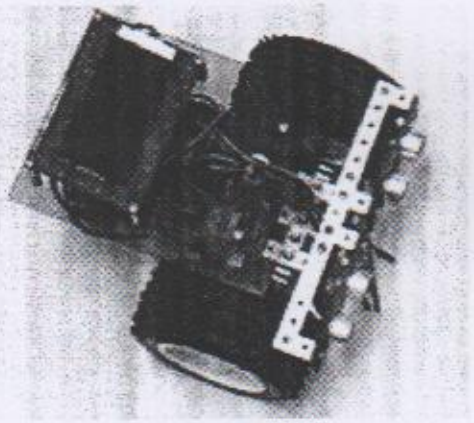

Figure 8. Small Unmanned Ground Vehicle System

Table 1 describes the information comparison of two heterogeneous SUGV systems.

Table 1. Hardware Information for SUGV Systems

	SUGV1	SUGV2
Microcontroller	Ubicom SX48AC 20MHz	Hitachi H8/3292 16MHz
RAM	32 KByte	512KByte
EEPROM	32 KByte	16KByte
Sensor	Two ultrasonic sensors	Two light sensors
Display	Text LCD	Text LCD
Servo-motor	2	2
JVM	H/W	N/A
language	Java	C/C++

4.1 Transformation T1: Transforming from a TIM to TSM(s)

The first transformation T1 is automatically transformed TIM (which merges with the predefined Processor profile and OS profile) into TSM in Figure 9, 10. Figure 9 shows to choose Ubicom SX48AC and Javeline for SUGV1, then click the '*Generate*' button to transform into one TSM. Figure 10 shows to choose Hitachi H8/3292 and brickOS for other SUGV2 then also click the '*Generate*' button to transform into other TSM.

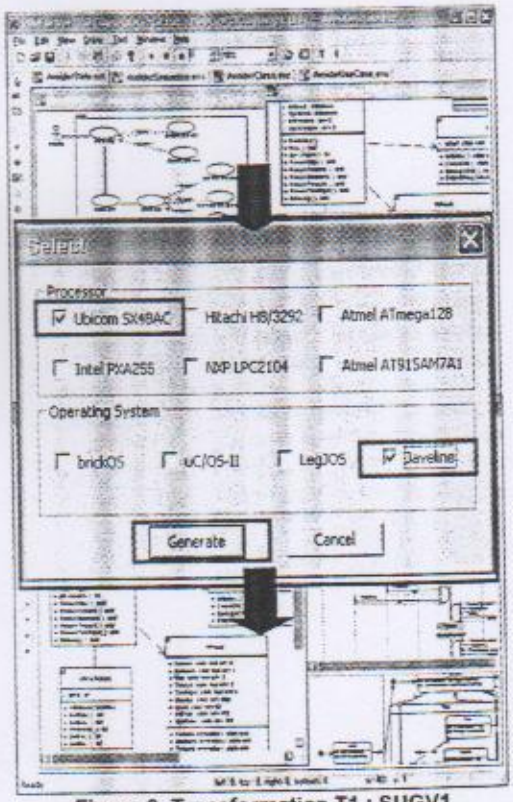

Figure 9. Transformation T1 : SUGV1

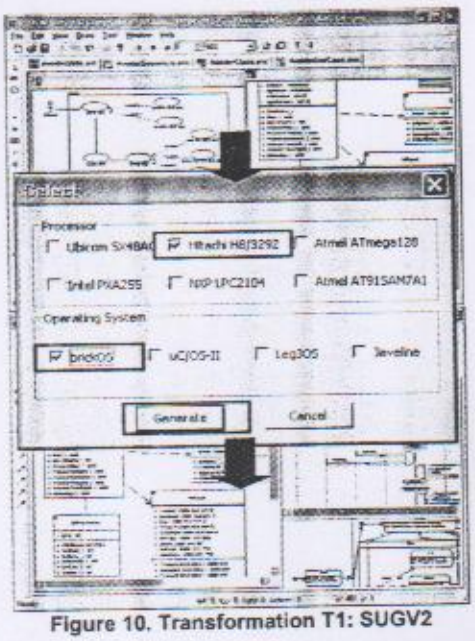

Figure 10. Transformation T1: SUGV2

4.2 Transformation T2: Transforming from TSM(s) to TDC(s)

The second transformation T2 is automatically generated the right executable TDC per each different target system. Its possible generable code languages are C, C++, and Java.

Figure 11, 12 shows automatically to transform into code through code template after modeling a TIM. All transformations are automatically executed on the tool. In the case of SUG1, java code can be generated as clicking 'Java' button while C++ be generated in SUGV2.

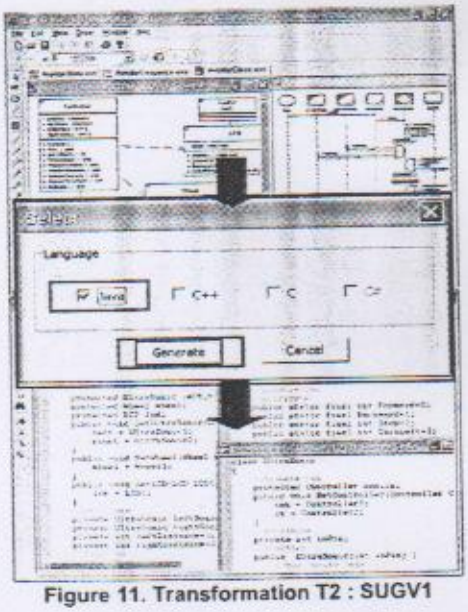

Figure 11. Transformation T2 : SUGV1

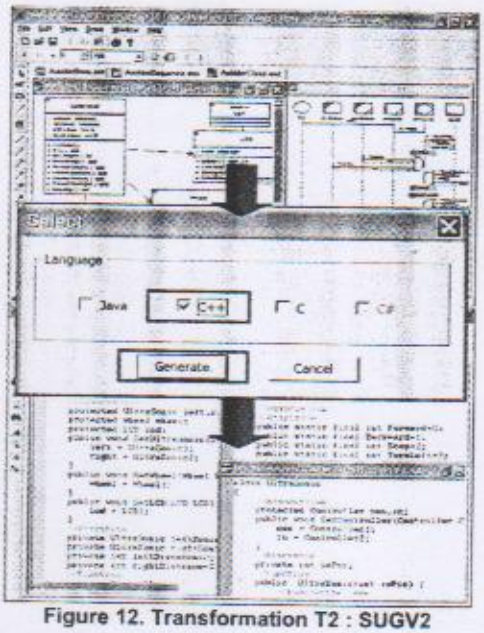

Figure 12. Transformation T2 : SUGV2

On T2 transformation in Table 2, it shows Forward(), Backward(), Stop() functions of motor class. Motor class links to one other motor to work the servo motor.

So, it rotates each opposite direction if the same code to move the left servo motor and the right servo motor. As a rersult, we can store the location status information with 'm_chPosition' variable. We show to compare the underline on three functions generated with the tool, and to transform into the right code for the heterogenous embedded systems.

Table 2. The comparison of codes on each SUGV

SUGV1(Javeline)	SUGV2(MindStorm)

public void Forward()
{
if("left" == m_chPosition)
CPU.pulseOut(STOP -m_speed, m_nPin);
else
CPU.pulseOut(STOP + m_speed, m_nPin);
m_status = "forward";
}
public void Backward()
{
if("left"==m_chPosition)
CPU.pulseOut(STOP+m_speed, m_nPin);
else
CPU.pulseOut(STOP-m_speed, m_nPin);
m_status = "backword";
}
public void Stop()
{
CPU.pulseOut(STOP,m_nPin);
m_status="stop";
}
	public void Forward()
	{
	if("left"==m_chPosition)
	Motor.forward(m_speed);
	
	else
	Motor.reverse(m_speed);
	
	m_status="forward";
	}
	public void Backward()
	{
	if("left"==m_chPosition)
	Motor.reverse(m_speed);
	
	else
	Motor.forward(m_speed);
	
	m_status = "backword";
	}
	public void Stop()
	{
	Motor.brake();
	m_status="stop";
	}

5 Conclusion

It may be hard to reuse embedded software products (such as model, design, and code) for the hardware dependent systems, that is, heterogeneous embedded systems.

We propose the automatic MDA (Model Driven Architecture) transformation to develop the heterogeneous systems. With this transformation method, we can possible develop target independent model, then automatically generate target specific model(s), which also are generated target dependent code(s).

To solve a problem for embedded software development, we implement the automatic tools for model transformation and code generation.

As a result, we can lead to reduce the lifecycle of the heterogeneous embedded software development.

6 References

[1] Axel Jantsch, *Modeling Embedded System and SOCs*, Mogan Kaufmann, 2004.

[2] A. Kleppe, J.Warmer, W.Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, Addison-Wiseley, 2003.

[3] Object Management Group, *OMG Unified Modeling Language Specification (draft) Version 1.3*, June 1999.

[4] Leon Starr, *Executable UML: How to Build Class Model*, Prentice Hall, 2002.

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

[6] Bart Broekman, Edwin Notenboom, *Testing Embedded Software*, Addison-Wesley, 2003.

[7] Pierre Boulet, Jean-Luc Dekeyser, Cedric Dumoulin, and Philippe Marquet, "Mda for Soc Design, Intensive Signal Processing Experiment", In FDL'03, Frankfurt, September 2003. ECSI.

[8] D. Kim, W. Kim, Robert Y. Kim, "A Study on Design for Embedded S/W based on Model Driven Architecture", *Journal of IWIT*, Korea, Vol. 6, No. 1, March 2006.

[9] Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe, "Feature-Oriented Product Line Engineering," *IEEE Software*, Vol. 9, No. 4, Jul./Aug. 2002, pp.58-65.

[10] Jean-Louis Houberdon, Jean-Philippe Babau, "MDA for embedded systems dedicated to process control," Workshop on MDA in SIVOEES, in conjunction with UML'2003, October 2003.

[11] W. Kim, Robert Y. Kim, "A Study on Extension of Executable UML for Modeling Real-time Embedded Software", Proceedings of the 25th KIPS Spring Conference, Korea, Vol. 13, No. 1, May 2006, pp.231-234.