A Library-Based Approach to Translating OCL Constraints to JML Assertions for Runtime Checking
Carmen Avila, Guillermo Flores, Yoonsik Cheon

Automatic MDA (Model Driven Architecture) Transformations for Heterogeneous Embedded Systems
Woo Yeoil Kim, Hyeon Seung Son, Young Bom Park, Byung Ho Park, C. Robert Carlson, Robert Young Chul Kim

Story Cards Process Improvement Framework
Chetan Kumar Patel, Muthu Ramachandran

A Frame Work for Software Engineers to Support Collaboration
Samina Jadoon, Kashif Hesham Khan, Ijaz Ahmad

SESSION: THEORETIC APPROACHES

Model-based Object-oriented Requirement Engineering and its Support to Software Documents Integration
William C. Chu, Chih-Hung Chang, Chih-Wei Lu

Integrating Z in DEVS: a case study Lift Control System
Mohamed Wassim Trojet, Mââmar El-Amine Hamri, Claudia Frydman

Mostafa Madieesh, Guido Wirtz

Software Complexity for Computer Communication and Sensor Networks Using Binary Decision Diagrams
Harpreet Singh, Adam Mustapha, Arati Dixit, Kuldip Singh, Grant Gerhart

Model Checking Consistency Between Sequence and State Diagrams
Kuang-Nan Chang

An Expert System for Pi-Calculus and Api-Calculus Automated Reduction
Shahram Rahimi, John Dillards, Bidyut Gupta

Qualitative Comparison of B, VDM and Z in Specifying Requirements of Safety Critical Systems
Ishrat Sami, Brian Dupee

Queue Design and Implementation Based on Service Level of NQS
Young-Joo Lee, Chan-Yeol Park, Sung-Jun Kim, Jin-Woo Sung, Sang-Dong Lee, Joong-Kwon Kim
Automatic MDA (Model Driven Architecture) Transformations for Heterogeneous Embedded Systems

Woo Yeol Kim¹, Hyun Seung Son¹, Y. B. Park⁵, B. H. Park¹, C. R. Carlson⁶, R. Young Chul Kim¹
¹Dept. of Computer & Information Comm., Hongik University, Korea
²Dept. of Informatics, Dankook University, ³The Armed Forces Medical Command, Korea
⁴Dept. of ITM, Illinois Institute of Technology, Chicago, USA
{joun, sun, bob}@hongik.ac.kr

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 processes, 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 numerous 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 it is just code oriented development. 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.
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 to the different versions of 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 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.

In Figure 2a., 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.

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 2a. Mapping diagrams at TIM phase

Figure 2b. Mapping diagrams at TSM phase
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].

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].
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 basis of the class diagram, concurrent message diagram, and concurrent state diagram of TSM. Figure 6 shows the meta template model for these diagrams.

![Figure 6. Meta template model](image)

<table>
<thead>
<tr>
<th>Class Name: string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package List: List</td>
</tr>
<tr>
<td>Parent List : List</td>
</tr>
<tr>
<td>Interface List : List</td>
</tr>
<tr>
<td>Association List: List</td>
</tr>
<tr>
<td>Attribute List: List</td>
</tr>
<tr>
<td>Function List: List</td>
</tr>
<tr>
<td>Head</td>
</tr>
<tr>
<td>Body</td>
</tr>
</tbody>
</table>

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](image)

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](image)

Table 1 describes the information comparison of two heterogeneous SUGV systems.
Table 1. Hardware Information for SUGV Systems

<table>
<thead>
<tr>
<th></th>
<th>SUGV1</th>
<th>SUGV2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>Ubicom SX48AC 28MHz</td>
<td>Hitachi HB3292 16MHz</td>
</tr>
<tr>
<td>RAM</td>
<td>32 KByte</td>
<td>512 KByte</td>
</tr>
<tr>
<td>EEPROM</td>
<td>32 KByte</td>
<td>16 KByte</td>
</tr>
<tr>
<td>Sensor</td>
<td>Two ultrasonic sensors</td>
<td>Two light sensors</td>
</tr>
<tr>
<td>Display</td>
<td>Text LCD</td>
<td>Text LCD</td>
</tr>
<tr>
<td>Servo-motor</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JVM</td>
<td>H/W</td>
<td>N/A</td>
</tr>
<tr>
<td>language</td>
<td>Java</td>
<td>C/C++</td>
</tr>
</tbody>
</table>

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 Javelina for SUGV1, then click the 'Generate' button to transform into one TSM. Figure 10 shows to choose Hitachi HB3292 and brickOS for other SUGV2 then also click the 'Generate' button to transform into other TSM.

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.
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