2006 International Conference on

Volume II

Hybrid Information Technology

Cheju Island, Korea
9-11 November 2006

Editors
Geuk Lee, Dominik Slezak, Tai-hoon Kim, Peter Sloot,
Haeng-kon Kim, Il Seok Ko, and Marcin Szczuka

IEEE COMPUTER SOCIETY

Organized and hosted by SERC and SERSC

IEEE

SERC
Security Engineering Research Center

SERSC

Proceedings

ICHIT 2006

Volume II

Table of Contents

Adapting Model Driven Architecture for Modeling Heterogeneous Embedded S/W Components

Woo Yeol Kim, R. Young Chul Kim
Dept. of Computer & Information Comm., Hongik University,
Jochiwon, 339-701, Korea
{john, bob}@selab.hongik.ac.kr

Abstract

In this paper, we propose 'MDA based approach for Modeling Embedded Software with xUML' to solve reusable problems of the heterogeneous embedded software. Through our proposed method, we produce 'Target Independent Meta Model' (TIM) which is transformed into 'Target Specific Model' (TSM) and which is generated 'Target Dependent Code' (TDC) via TSM. We would like to reuse a metamodel to develop heterogeneous embedded software systems. To achieve this mechanism, we extend xUML to solve unrepresented elements (such as real things about concurrency, and real time, etc) on dynamic modeling of a particular system. As a result, it is possible to develop s/w independent of hardware with reusability of models. So we adapt the MDA mechanism for embedded s/w development, which can reduce the costs and lifecycle of s/w development. It contains one example which illustrates the proposed approach.

1. Introduction

Today's demands for short product development cycles usually prohibit the manufacturing of a complete prototype as part of the development for embedded products [1]. Even if one or a few prototypes can be built, they cannot replace the hundreds of different models that are routinely developed for an average electronic product. The idea that a unified language cannot meet all requirements of embedded system modeling and specification tool has been absolutely accepted. The requirements for a language to model numerously diverse embedded systems are just too diverse to be supported neatly by one single modeling language. Many software researches of dealing with the complexity of embedded system are now in progress a bout these two needs. Our research focuses on reusing embedded software (such as design, model, code, and test) to develop heterogeneous embedded systems. But it is difficult to reuse 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 attempt to adapt the MDA mechanism [2] for embedded s/w development, which may reuse meta-model.

We adapt xUML [4,5] as embedded modeling language, instead of the previous UML 1.x [3]. Executable UML(xUML) is a graphical specification which combines with executable semantics and timing rules, and also runs just like a program [5].

Also we suggest detail activities of lifecycle for embedded s/w development and propose refined multiple V-model based on MDA against previous multiple V-model [6]. This will be possible to develop target independent software, may help to reuse software, and then reduce the cost of software development

This paper is organized as follows: In section 2, we describe the original concept of model driven architecture (MDA) and xUML, multiple V-model. In section 3, we show our adapting MDA structure to develop embedded s/w. In section 4, it shows the modeling example of heterogeneous embedded systems used in this paper.

2. Model Driven Architecture

The MDA[2] is the OMG proposed approach for system development. It primarily focuses on software development, but can be applied to system development. The MDA is based on one meta-model describing the systems to be built. A system description is made of numerous models, each model

This work is supported by grant No. B1220-0601-0031 from the UF Research Program of the Ministry of Information & Communication in Republic of Korea (2006 ~2007).

representing a different level of abstraction. The modeled system can be deployed on one or more platforms via model to model transformations [7].

2.1. Principles of MDA for the development

The original MDA approach creates good designs that cope with multiple-implementation technologies and extended software lifetimes and also brings the focus of software development to a higher level of abstraction, thereby raising the level of maturity of the IT industry[2]. The MDA is a framework for software development defined by the Object Management Group (OMG). MDA development process is the one that the first model (called a Platform Independent Model) with a high level of abstraction, which is independent of any implementation technology, is generated and then the PIM is transformed into one or more Platform Specific Models (PSM). Finally, it is the transformation of each PSM to code. The MDA process may look much like traditional development. Actually there is a critical difference. Traditionally, the transformations from model to model, or from model to code, are done mainly by hand. In contrast, MDA transformations are always executed by automatic tools. This is the obvious benefits of MDA[2,8]. But the original MDA process is not for embedded software development process.

2.1.1. MDA Development Process vs. Product Line
Nowadays, many software engineers make their efforts to apply for embedded software development fields.
The representative cases are that some software researchers attempt to work on with product line mechanism or with MDA mechanism.
We analyze and compared the difference between MDA and Product Line on Table 1.

Table 1. Comparison of Product Line with MDA

Product Line	MDA
1) After one development lifec ycle, possible to reuse core asse ts. 2) Product-Line based Develop ment to release high-quality em bedded systems at time-to-mar ket. 3) Feature Driven Method. 4) Easily Customizing. 5) Based on UML 2.0.	1) During one lifecycle, reuse a meta-model to some specific dep endent Models. 2) Heterogeneous Product based Development to release high-qua lity product at time-to-market. 3) Based xUML & UML 2.0 . 4) Possibly Tailoring. 5) Absolutely need the automatic model/code generation tool.

As a result of comparison Product Line[9] with MDA, we recognize that this important issue is reusability of artifacts, but the difference when and how to reuse. In table 1, Product Line mechanism may possibly develop a new system to reuse with core assets after development lifecycle of a system. MDA may develop new other systems to reuse with meta-model during one possibly development lifecycle.
These two processes may be applied for high quality of embedded s/w development at time-to-market.
Product Lines may easily develop one single family set of product, while MDA may easily develop heterogeneous product family. Only one critical problem of MDA is that the autonomic transformation form model to mode or from model to code should be done, not by hand. With this automatic tool, easily make each code for multiple platforms.

2.2. Modeling Language

The using of MDA leads to choose a modeling language. As suggested by the OMG, UML(Unified Modeling Language [3]) is a good candidate. UML is a standard for OO modeling. This language is extensible, so it can be adapted easily to specific constraints of a given domain. The extension in UML is done via stereotypes (elements whose semantic is extended) grouped into profiles. Once the language is chosen, it is necessary to define the modeling strategy [10].

In this paper, it will choose Executable UML (xUML) to represent embedded software modeling.

2.2.1. Executable UML. Executable UML[4,5] is a graphical specification language, which combines a subset of the UML 1.x with executable semantics and timing rules.
This language can be built a fully executable system specification consisting of class, state, and action models that run just like a program. Leon starr[4] said in xUML, the models are the code. Also the design benefits by staying separate from the specification. That is, leverage one design in multiple products.
We strongly believe that executable UML will be suitable to specification of complex real time, distributed, and embedded system.

2.3. Straightforward Multiple V-Model

The Multiple V-Model[6], based on the well-known V-model (Spillner, 2000) is a development model (model, prototype, final product) which is developed the different physical versions of a system.

In figure 1, the multiple V-model shows three consecutive V-shaped development cycles (model, prototype, final product). This is the development process for embedded systems where the prototype is first fully designed, then built, tested and made a final product.

The development of such a system starts with specification of the requirement at a high-level, followed by an architectural design phase where it is determined which both software and hardware are required to realize this. Software and hardware are then developed separately and finally integrated into a full system.

The left side of the V-model handles the decomposition of the system into its hardware and software. The middle part of the V-model consists of parallel development cycles for H/W and S/W. The right side of the V-model handles the integration of them.

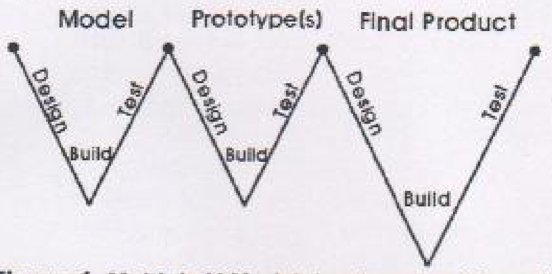


Figure 1. Multiple V-Model development lifecycle

In Multiple V-model for embedded systems it anticipates to reduce the cost and lifecycle of development because of correcting better at model phase than at Prototype phase or better at prototype phase than at final product when happens requirement changes and errors [8].

3. Adapting Modeling Approach

To solve reusable problems of the heterogeneous embedded software, we suggest to adapt MDA based approach with xUML for modeling embedded software. It may be possible to develop s/w independent of hardware with reusability of models, which can reduce the costs and lifecycle of software development.

3.1. Refined Multiple V-model on MDA

The original multiple V-model 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 solve problem of the original V-model.

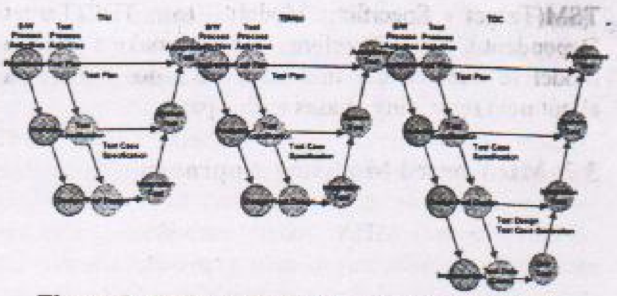


Figure2. Refined Multiple V-Model with MDA

In figure 2, it describes to adopt multiple V-model with MDA. The refined multiple V-model is also a development model (Target Independent Model, Target Specific Model, Target Dependent Code) which is developed the different physical 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 develop heterogeneous embedded systems with reusability on different target domains. Moreover it works 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.

But still we need the automatic tools such as automatic model transformation, code generation.

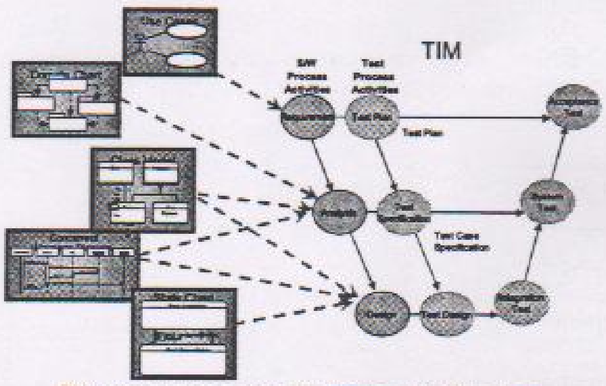


Figure 3. Mapping diagrams at TIM phase

In figure 3, it is just more detail described to map diagrams at the first V-model. Use case diagram is used during requirement to represent the functionality of the system from the user's point of view. During analysis, class diagram describes the structure of the system. Concurrent message sequence 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.

3.2. MDA based Modeling Approach

We propose MDA based 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 sequence diagram and concurrent sate diagram [11] represent the dynamic concurrent behavior of the system in figure 4.

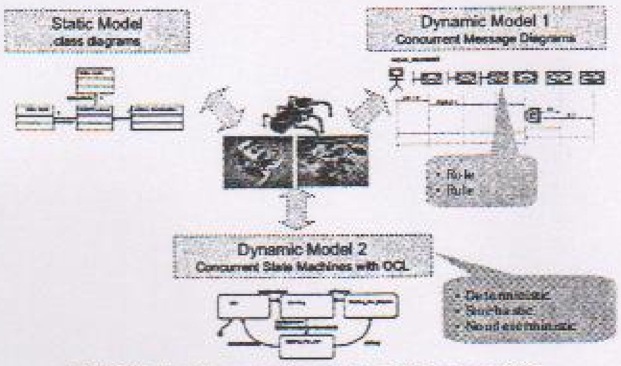


Figure 4. The relationship of the models

Figure 5 represents to map between CMD(concurrent message sequence diagram) and CSD(concurrent state diagram) in xUML which we extended.

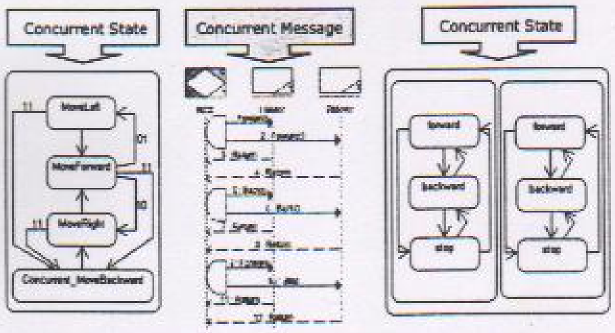


Figure 5. Mapping between diagrams

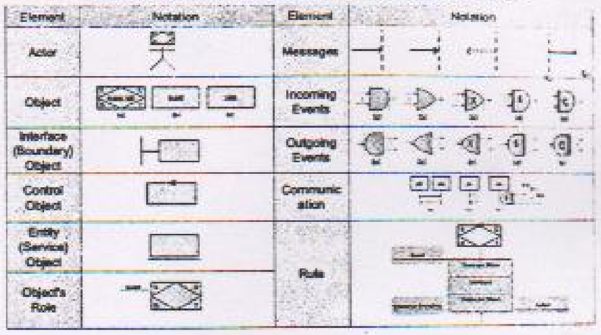
Actually CMD means interaction among objects, which is included with concurrent mechanisms (such as fork-join, reverse fork-join, etc) for representing real things. And CSD works some concepts which will be included with nested state, non-deterministic, deterministic, and stochastic mechanism for real things. We may use this diagram to generate actual executable code of the control object in the system.

Our extended xUML consists of use case diagram, class diagram, concurrent message sequence diagram, concurrent state diagram.

3.2.1. Class Diagram. Class Diagram describes the static structure of a system with role and rule mechanism. Its basic class may have the template types of role such as recognizer, decision, communication, and transaction. Also rule concept in the class is followed by ECA (Event/Condition/Action) in table 2.

3.2.2. Concurrent Message Diagram. We extend the basic sequence diagram with concurrent mechanisms (such as fork-join, reverse fork-join, etc) for handling real things. Table 2 describes the notations of extended xUML.

Table 2. Notations of Extended xUML



Element	Notation	Element	Notation
Actor		Messages	
Object		Incoming Events	
Interface (Boundary) Object		Outgoing Events	
Control Object		Communication	
Entity (Service) Object		Role	
Object's Role			

In extended xUML, the object in concurrent message diagram also is followed by Ivan Jacobson's stereotype such as interface, control, and service object. Interface object (or boundary object) just transfers a message to control object without any state. Control object makes a decision and mediates (or controls) between interface and service object. Service object transacts with data needed. Like class's role, its object may or may not have one or more roles of recognizer/ decision/ communication/ transaction.

In other words, the object has role(s) based on which ERCDT(Event/Recognition/Communication/Decision/ Transaction). Each object may or not have zero or more role(s). also the object is included with ECA (Event/Condition/action) rule. For example, one object comes in an event, checks the condition, then acts some service. Basically our approach is adopted synchronized message passing mechanism in concurrent message sequence diagram.

3.2.3. Concurrent State Diagram. Still on our research, CSD(Concurrent State Diagram) will be basically followed by nested mechanism with OCL(Object Constraint Language). This diagram will contain some mechanisms for deterministic/stochastic system. Next time, we will extend automatically to check and convert from nondeterministic to deterministic state diagram.

4. Case Study

4.1. Comparison of Heterogeneous systems

In this section, we make meta-model in TIM, then transform two specific models of heterogeneous embedded systems. Until now, we write two program codes based on two specific models by hand and port each code into each heterogeneous embedded system. We recognize that two different mobile devices move the same way under the same functionality. It shows demo examples of LEGO Mindstorm and Javelin as follows.

4.1.1. LEGO Mindstorm. At the core of the LEGO's RCX is a Hitachi H8 microcontroller with 32K of external RAM. The microcontroller is used to control three motors, three sensors, and an infrared serial communications port. An on-chip, 16K ROM contains a driver that is run when the RCX is first powered up. The on-chip driver is extended by downloading 16K of firmware to the RCX. Both the driver and firmware accept and execute commands from the PC through the IR communications port. Additionally, user programs are downloaded to the RCX as byte code and are stored in a 6K region of memory. RCX is programmed using NQC(Not Quite C). It is a simple language with a C-like syntax.

4.1.2. Javelin. The Javelin Stamp is a single board computer that's designed to function as an easy-to-use programmable brain for electronic products and projects. It is programmed using software on a PC and a subset of Sun Microsystems Java® programming language.

Some of the features that set the Javelin Stamp are:

- The instruction codes for the Javelin are fetched and executed from a parallel SRAM instead of a serial EEPROM.
- The Javelin has 32k of RAM/program memory with a flat architecture.
- The Javelin has built in Virtual Peripherals (VPs) that take care of serial communication, pulse width modulation and tracking time in the background.
- Serial communication is buffered as a background process. When writing programs, all you have to do is periodically check the buffer.

These LEGO Mindstorm and Javelin are two other embedded systems with different CPU, operational environment, and language. From next section, we would like to show step by step that generate each TSM to reuse with one meta-model of TIM.

4.2. Static Modeling

4.2.1. Target Independent Model. Figure 6 shows the static structure of the existing system with only the basic functions such as forward and backward moving. We attempt to change a control not with hardware, but with only software.

For example, we attempt to add two methods, MoveLeft() and MoveRight() to turn the right or the left. Also to handle the case for touching two sensors at the same time, add new Concurrent_MoveBackward() instead of Backward().

In figure 6, a dotted rectangle indicates the representation for modeling concurrent occurrences.

We show to model considering later extension for heterogeneous motors of LEGO and Javelin.

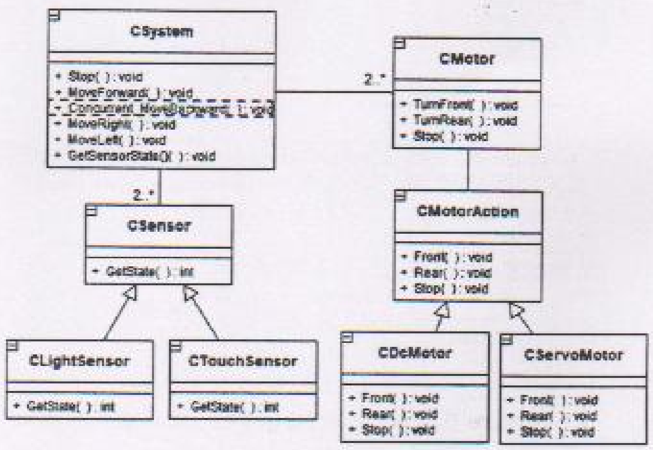


Figure 6. Target Independent Model

4.2.2. Target Specific Model. Figure 7 shows TSM of LEGO Mindstorm derived from a TIM, which is considered with its CPU and language.

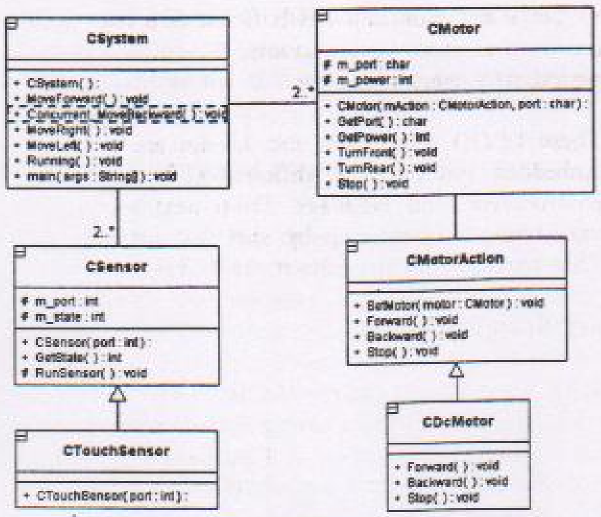


Figure 7. LEGO's Target Specific Model

Figure 8 also shows other TSM (Javelin) derived from the TIM, which is considered with other CPU and language like figure 7.

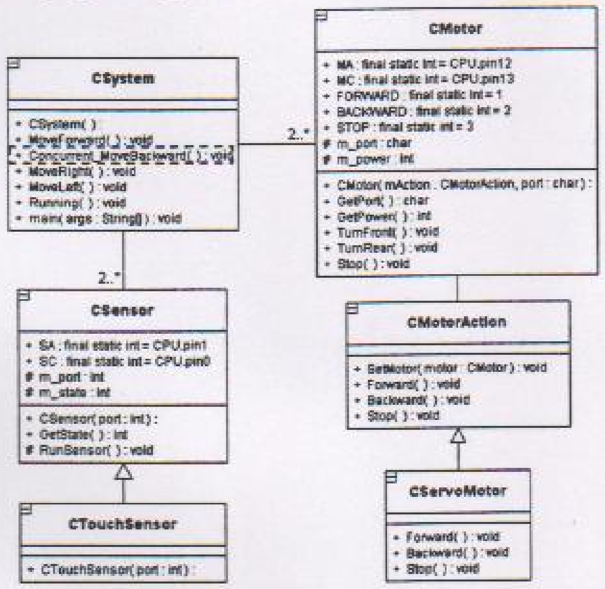


Figure 8. Javelin's Target Specific Model

4.2.3. Target Dependent Code. Until now, we can not help generating source code from each TSM by hand. But we are still implementing auto code generation tool.

In this section, we shows each other source code(that is, NQC, Java) from results of modeling TSM(s) in table 3,4.

Actually, it is impossible to reuse on level of code because each embedded system has different hardware structure. Therefore, we attempt to generate code with reusing MDA based models.

Table 3. LEGO's Code

NQC

```
task main()
{
 SetSensor(SENSOR_1, SENSOR_TOUCH);
 SetSensor(SENSOR_3, SENSOR_TOUCH);

 while (true)
  {
   move_forward();
   int sense_result = 0;
 if (SENSOR_1 == 1) { sense_result =1; }
 if (SENSOR_3 == 1) { sense_result =2; }
 if (SENSOR_1 && SENSOR_3) {sense_result =3;}

  switch (sense_result)
   {
    case 1: turn_right(); break;
    case 2: turn_left(); break;
    case 3: concurrent_move_backward();
            turn_right(); break;
   }
  }
}
```

Table 4. Javelin's Code

Java

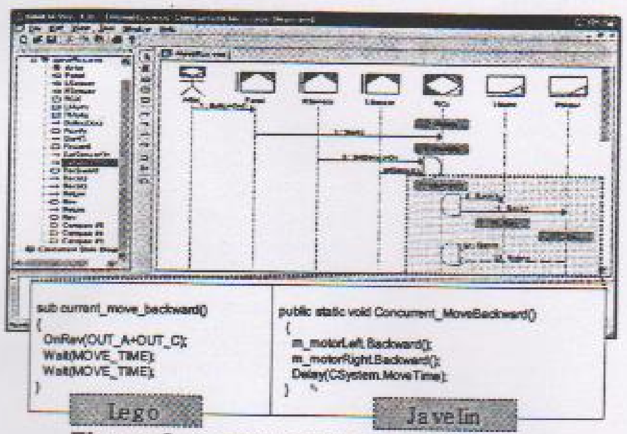
```
public static void main()
{
    Timer tSensing = new Timer();Timer tLoMotor = new
Timer();
    CSensor sensor_left = new CSensor(CSensor.SC);
    CSensor sensor_right = new CSensor(CSensor.SA);

            boolean left = true;boolean right = true;
            tSensing.mark(); tLoMotor.mark();

            while (true)
            {  if(tLoMotor.timeout(CSystem.SPEED))
              {  if( (!left && right) )
                     CSystem.MoveLeft();
                  else if( (!right && left) )
                     CSystem.MoveRight();
                  else if( (!left && !right) )
          {  CSystem.Concurrent_MoveBackward();
                     CSystem.MoveRight();     }
                   else
                     CSystem.MoveForward();
         tLoMotor.mark(); }
               if(tSensing.timeout(CSystem.SENSING))
               {  left = sensor_left.GetState();
                  right = sensor_right.GetState();
                  tSensing.mark();    }
            }
}
```

4.3. Dynamic Modeling

We extend the original dynamic diagram to represent real things. Our CMD(Concurrent Message Sequence Diagram) is included mechanism of rule and role, and also added to represent concurrent occurrence.

Figure 9 shows concurrent message diagram of the system. A dotted box within figure 9 represents reverse Fork/Join mechanism of CMD. This means that when two sensors are touched at the same time, the controller should give a control signal to two motors simultaneously. Still we can not model this situation with the existing sequence diagram. This is why we needed to extend.

These two devices have the same movement, but definitely each different source code is represented due to heterogeneous embedded systems.



```
sub current_move_backward()
{
  OnRev(OUT_A+OUT_C);
  Wait(MOVE_TIME);
  Wait(MOVE_TIME);
}
```

```
public static void Concurrent_MoveBackward()
  {
    m_motorLeft.Backward();
    m_motorRight.Backward();
    Delay(CSystem.MoveTime);
  }
```

Figure 9. Concurrent Message Diagram

Figure10 represents a polling algorithm for working the sensor. Also possible to represent stochastic mechanism with CSD. Still working on conversion from nondeterministic to deterministic model.

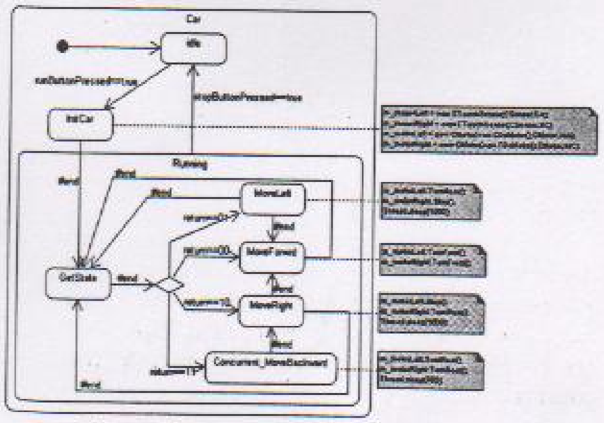


Figure 10. Concurrent State Diagram

5. Conclusion

Embedded software is difficult to reuse the existed design and code due to the hardware dependent systems, that is, heterogeneous embedded systems.

This paper is introduced to modeling heterogeneous embedded software system based on MDA (Model Driven Architecture). This method is possible to develop target independent software, which helps to reuse software. As a result, this leads to reduce the cost and lifecycle of s/w development.

But this approach has a big problem that it should be necessary to implement the automatic tools for model transformation and code generation.

Still working on our embedded modeling tool and code generator.

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.