A Study on the Development of Embedded System Software for Ubiquitous Sensor Networks using UML

Jong-Won Choi*, Dong-Jin Lim**

**Department of Electrical Engineering & Computer Science, Hanyang University, Ansan, Korea (e-mail: ventiano@hotmail.com)*
***Department of Electrical Engineering & Computer Science, Hanyang University, Ansan, Korea (e-mail: limdj@hanyang.ac.kr)*

Abstract: This paper shows the results of an ongoing study into the application of model-driven methods for embedded system software in ubiquitous sensor networks. In this study, Unified Modelling Language (UML) is used to make a model for the embedded software. A UML software tool with code generation capability is employed and a simple wireless sensor network system built; additionally, a portion of the embedded software is developed using the UML model. Memory usage, timing performance, and power consumption of the code generated from the UML model are compared with those of handwritten code to show that this approach is feasible.

Keywords: UML, Model-driven, Embedded systems, Wireless, IEEE 802.15.4, USN

1. INTRODUCTION

Recently, wireless sensor network technology has attracted a great deal of attention with regard to the applications of home networking and ubiquitous sensor network systems. To build wireless sensor network systems, it is necessary to develop embedded system software for sensor network systems. Currently, there are several solutions available for wireless sensor networks, such as TinyOS and ZigBee. However, no one solution meets all the requirements for the various applications, and sometimes it is necessary to have customized embedded software, which is specified for a certain application. As with large scale system software, developing embedded system software for sensor network systems is a rather complicated process.

Traditionally, programs for microprocessors are coded using conventional programming languages such as C or assembly programming language. Before the actual coding is done, documents for requirements and specifications are usually written, and programs are coded based on those documents. However, sometimes the meaning of the written documents can be misinterpreted, and this misinterpretation will result in errors in the programs. The likelihood of these kinds of errors increases as the system become more complex. The functionality of embedded systems is increasing, and it is almost impossible to describe all the functions completely in the written documents. In order to describe the functions of modern embedded systems, it is therefore necessary to have ways to describe the specifications of systems using other than text documents. One of the possible solutions is to use a model-driven method for developing embedded systems software. In a model-driven method, models are used to describe the specifications of a system instead of using text-based documents. Since there are many advantages in using a model-driven method for embedded systems, it has attracted much recent attention (Ibrahim 2006).

In the model-driven method, Unified Modelling Language (UML) is usually used to make a model of the system (Martin 2002). There are several software UML tools on the market, and some tools have capabilities for generating code automatically from UML models. With this code generation capability, it is possible to execute the UML models. When using a model-driven method for the embedded system software, it is very important to use a tool with automatic code generation capabilities. If the code has to be written by hand after the system is designed using models, there is no guarantee that the actual code will reflect the model exactly. If a UML software tool with the automatic code generation is used, it is possible to implement a seamless development process from the abstract level design to the implementation level (Khan 2006, Mura 2008).

There are several benefits in using UML for designing embedded applications. A designer is able to develop embedded software using the powerful notation of UML diagrams. If several designers are involved, they may be able to exchange ideas easily thanks to the graphical representation. Some UML software tools are able to simulate the software before the actual implementation. However, the method also has disadvantages in that the program generated from the model may not be as efficient as handwritten code. In this study, a simple application for wireless sensor network is developed using UML and compared with the handwritten code in terms of memory usage, timing, and power consumption to find out if this approach is feasible. For the target wireless platform IEEE 802.15.4 was used. Due to its low power and low cost, IEEE

802.15.4 is gaining popularity, not only in home networking but also in sensor networks and in industrial automation systems. In order to have seamless integration of the model-driven method, it is necessary to have a UML software tool with automatic code generation capability. There are several commercial UML software tools in the market, and some are more suitable for embedded system software design than others. Among them, Rhapsody from Telelogic is known to be among the most advanced and appropriate for embedded system software design (Krasner 2004). This paper shows the results of an ongoing study into the application of model-driven approaches to sensor network systems using Rhapsody (although similar results would be possible using other commercially available software tools). In this paper, a simple star topology wireless sensor network is built, and the embedded software for a coordinator node is developed using UML with automatic code generation.

2. SOFTWARE MODELLING

The code generated by Rhapsody is supposed to run on operating systems that include Windows XP, VxWorks, and Windows CE. Between the operating system and the code generated from the model, there is a framework library for running UML models and an operating system adaptor layer, which enables the code from the model to run regardless of the operating system used. Figure 1 shows the conceptual structure of the software. Rhapsody generated code can also be run on small systems without an operating system, thanks to the Interrupt Driven Framework (IDF). As can be seen in Figure 2, IDF replaces the operating system and the interface layer. With some limitations, almost all the features of UML can be used with IDF.

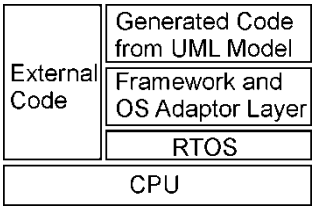

Fig. 1. Software structure with an operating system

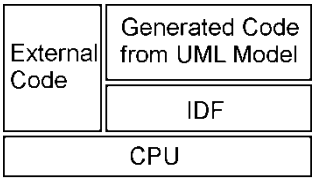

Fig. 2. Software structure without an operating system

The test node system used in this study has one coordinator node and three end device nodes, as shown in Figure 3. Each node is equipped with an Atmel ATmega128L microcontroller and a TI Chipcon CC2420 RF transceiver. Chipcon CC2420 works at 2.4GHz and is compatible with the IEEE 802.15.4 standard. To run the test scenario, each node has a switch, an LED, and a potentiometer. In the test scenario, the user is able to change the channel by pressing the switch. When the switch is pressed, an interrupt is generated to change the communication channel. Also, when the potentiometer is turned, the brightness of the LEDs on the other nodes with the same channel is changed.

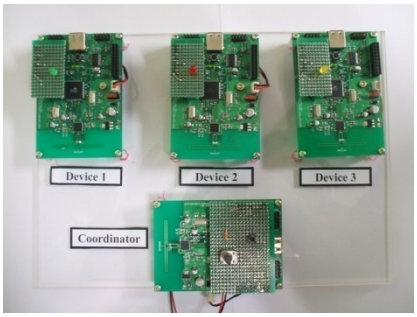

Fig. 3. IEEE 802.15.4 Test nodes

The test programs for end device nodes are written in C, while the program for the coordinator node is developed from the UML model. The UML model for the coordinator has four objects, whose names are Rf_init, Main, Basic, and Switch. Figure 4 shows the object model diagram for the coordinator node.

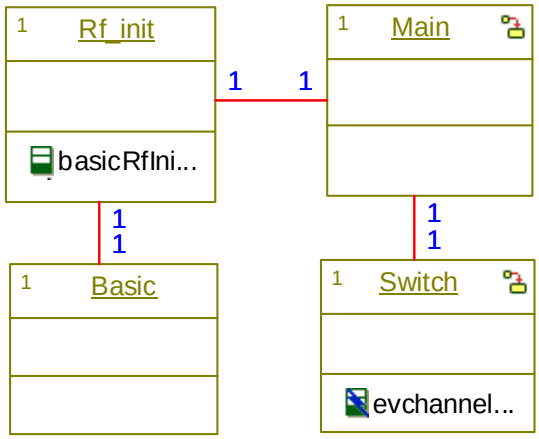

Fig. 4. Object model diagram for the coordinator

The objects shown in Figure 4 are responsible for defining the structure of MCPS-DATA.request, which is specified in IEEE 802.15.4 MAC layer, initializing CC2420 for radio communication, and checking the potentiometer and the switch for user interactions. In the Rf_init object, the structure type of RF Channel, PanID, and myAddr are defined, and initialization of CC2420 is made. Parameter variables for MCPS-DATA.request are defined in the Basic object. The Main object is responsible for checking the potentiometer. Handling of the switch interrupt for changing the channel is done in the Switch object.

In order to describe the behaviour of the objects, state charts can be included for the objects. In this test model the Main object and the Switch object have state charts to describe their behaviours. Figure 5 shows the state chart for the main object. After the initialization, it checks the variation in the potentiometer with a 10 msec interval.

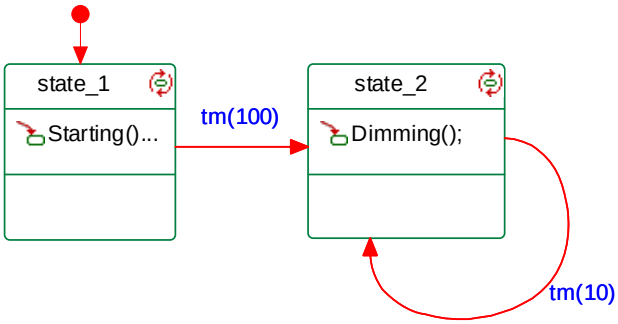

Fig. 5. State chart for the main object

Figure 6 shows the state chart for the Switch object. When the switch is pressed, an interrupt is generated. In the interrupt service routine for the switch interrupt, a macro function to generate an event is called. When the event is generated, a state transition occurs and the operation to change the channel is called. With this test model, one of three channels can be selected by successive pressing of switches.

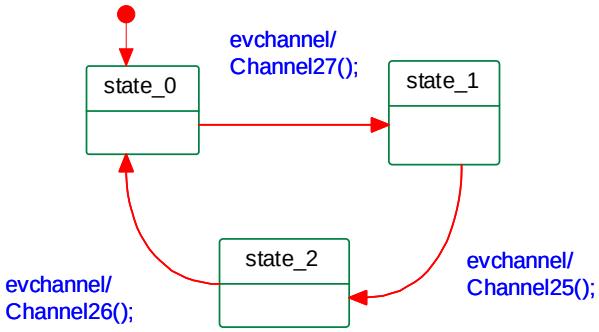

Fig. 6. State chart for the Switch object

When developing embedded system software using UML, the entire code is not necessarily generated from the model. The software engineer who has chosen to use a UML model may prefer to use routines written in C for some part of the program. For example, if good driver routines for peripheral hardware are already available, they can be combined with the code generated from the model. In this project, since the basic driver routines written in the C programming language for the TI CC2420 chip are available from Texas Instruments, they were included in the model. Rhapsody allows developers to include user source code in the model. Figure 7 shows the list of source files written in C for this project.

Since UML tools such as Rhapsody are merely a code generation tool and do not have their own compilers, an external compiler must be used to compile the generated codes. There are several commercial compilers for the microcontroller selected in this project, and IAR's EWAVR compiler was used. IAR's Integrated Development Environment (IDE) allows a user to download the executable file and debug it using source code files through the JTAG interface. After the code generation is completed, the program can be compiled and debugged in the same way as with handwritten source code.

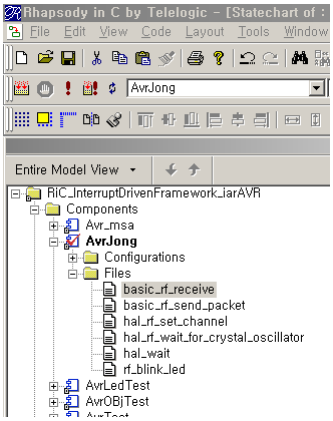

Fig. 7. List of C source codes included

3. MEMORY USAGE, TIMING, AND POWER CONSUMPTION

There are several things to be considered when using a model-driven method for embedded system software. One of the most important things is memory usage. Compared with handwritten code, it is unavoidable that some increase in memory usage will be seen when a model-driven method is employed. Given the fact that the amount of internal memory of microcontrollers is increasing all the time, the memory usage may not be a critical issue in the near future. Also, the cost of memory chips is decreasing, and extra memory chips may be added to accommodate the memory requirements. However, when the small microprocessor systems, such as a sensor node, are under consideration, it is always necessary to check the memory usage requirements.

In order to compare the memory usage of the code generated from the UML model with the handwritten code, the coordinator for the same test scenario was programmed by hand using the C programming language. Table 1 shows the size of the code memory and the size of the data memory used for each implementation.

Table 1. Size of the code and of the data (bytes)

	Handwritten Code	Code from Model	IDF	Model without IDF
Code	2569	8721	3631	5090
Data	359	1383	923	460

The column labelled 'Handwritten Code' is for the memory usage for the handwritten code, and the column labelled 'Code from Model' is for the code generated from UML model. To run the code generated from the model without an operating system, the IDF library must be included all the time. The column labelled 'IDF' in the table is the memory requirement for IDF library alone. The numbers in the last column are obtained by subtracting numbers in the IDF column from the 'Code from Model' column. Therefore, the last column represents the increase of memory caused by the addition of UML models for this test scenario. Since the ATmega128 has 128KB of flash memory, and 4KB of RAM, the source code generated from the model has no problem in running. The code generated by Rhapsody takes up less than 7% of the available code memory, and the data takes up 35% of the available data memory.

In order to implement complex communication protocols, it is necessary to add many objects and state charts in the model. To check the memory increase due to the added objects and state charts, a dummy project with a single object including a state chart with a single state was created. Then, objects with the same state chart were added one by one, and memory sizes checked. The size of source code memory increased by 73~95 bytes for each added object, while the size of data memory increased by 19 bytes. To check the memory increase due to the added states, states were added one by one to the initial dummy project, and the memory size checked. The first addition of an extra state increased the size of the source code memory by 144 bytes, but after the successive addition of states the code memory increase was reduced to 18 bytes per state. The size of the program memory was not changed by the addition of extra states. From the above results, the implementation of a rather complex protocol, with many objects and states, would be possible with the internal memory of the microcontroller. A new family of Atmel AVR series microcontrollers with 256KB of flash memory and 8KB of RAM has recently become available. With the new higher memory chips, it becomes even more feasible to have complex protocols, which can be accommodated in the internal memories of a single microcontroller.

It is also necessary to compare the timing performance of the code generated from the UML model with the handwritten code. To do so, a coordinator node is programmed to send a packet, and an end device node is programmed to return a packet as soon as it receives one from the coordinator. The time for the predetermined number of round trips was measured, and table 2 shows the results. As can be seen from the table, the increase in time is only about 2–3%.

Table 2. Measured Timing

Number of round trips	50	100	150	200
Handwritten	0.84sec	1.69sec	2.53sec	3.38sec
UML	0.87sec	1.73sec	2.57sec	3.44sec

Finally, the power consumption of each implementation is compared. A coordinator node was programmed to send out packages continuously, and the average power consumption measured for each case. The power consumption of the node with the code generated from the UML model is slightly higher than for the handwritten code, however, the increase is less than 1%.

4. CONCLUSIONS

This paper shows the results of an ongoing study into the application of model-driven methods for embedded system software for ubiquitous sensor networks. In this study, UML was used to make a model for the embedded software, and code generation was employed. A simple wireless sensor network system was built, and a portion of the embedded software developed using the UML model. Memory requirements, timing, and power consumptions of the code generated from the model were compared with those from handwritten code. The results show that this approach may be feasible for wireless sensor network applications.

In the future, a wireless communication protocol, such as MAC layer protocol of IEEE 802.15.4 or ZigBee, will be implemented with the model-driven approach. The embedded software developed using this model-driven approach will be compared to handwritten software in terms of memory usage, timing performance, and power consumption.

ACKNOWLEDGEMENT

This work is supported by Gyeonggi-do Regional Research Center Program funded by Gyeonggi-do, Korea.

REFERENCES

Ibrahim, A.E., Zhao, L., and Kinghorn, J. (2006). 'Embedded Systems Development: Quest for Productivity and Reliability', *Proc. ICCBSS 2006*, pp. 13-16.

Khan, M.U, Geihs, K., Gutbrodt, F., Gohner, P., and Trauter, R. (2006). 'Model-Driven Development of Real-Time Systems with UML 2.0 and C', *Proc. MBD/OMPES 2006*, pp. 33-42.

Krasner, J.L. (2004). 'Reducing OEM Development Costs and Enabling Embedded Design Efficiencies Using UML', *2004 Embedded Market Forecasters*, American Technology International, Inc.

Martin, G. (2002). 'UML for Embedded Systems Specification and Design: Motivation and Overview', *Proc. Design Automation and Test in Europe Conference and Exhibition 2002*, pp. 773-775.

Mura, R. (2008). 'Code Generation from Statecharts: Simulation of Wireless Sensor Networks', *Proc. Digital System Design Architectures, Methods and Tools 2008*, pp. 525-532.