The Development of Training Course for Embedded Middleware Design

Hewijin Christine Jiau and Kuo Chen Wu
Institute of Computer and Communication Engineering
Department of Electrical Engineering
National Cheng Kung University, Tainan, Taiwan, R.O.C.
jiauhjc@mail.ncku.edu.tw, kuchenwu@nature.ee.ncku.edu.tw

ABSTRACT

In order to promote embedded software education and establish fundamental related research, Embedded Software (ESW) Consortium was initiated by Taiwan Ministry of Education in 2004. Embedded Middleware Design Curriculum is the advanced course for graduate and undergraduate students to develop and practice embedded middleware. In this work, course design, expected results and the course result analysis are presented.

Categories and Subject Descriptors

D.3.3 [**Computer Milieux**]: Computer and Information Science Education

General Terms

Human Factors

Keywords

Embedded Software, Education, Embedded Middleware

1. INTRODUCTION

The industry of Information Technology (IT) and Integrated Circuit (IC) are the main focuses in Taiwan's industry. In order to keep the competition with worldwide IT/IC industry, well-trained engineers play the key factor. Therefore, the Advisory Office of the Ministry of Education (AOMOE) of Taiwan established the VLSI Circuits and Systems Education Program in 1996. Embedded Software (ESW) Consortium [1] was initiated by the Ministry of Education under the VLSI Circuits and System Education Program in 2004 to promote ESW education and establish fundamental related research for embedded software technologies [3, 13].

Embedded Middleware Design Curriculum is one of the curricula created by ESW Consortium and the purpose is to develop the advanced embedded training course for graduate and undergraduate students. This training course is designed to teach students to design and build embedded applications by merging existing embedded systems by various hardware vendors. The course design and material preparation assumption is for students with basic knowledge of embedded systems. The detail course design is in section 2. Expected results are described in section 3. Section 4 is the course result analysis to make sure the course performance. Finally, section 5 provides the conclusion of this work.

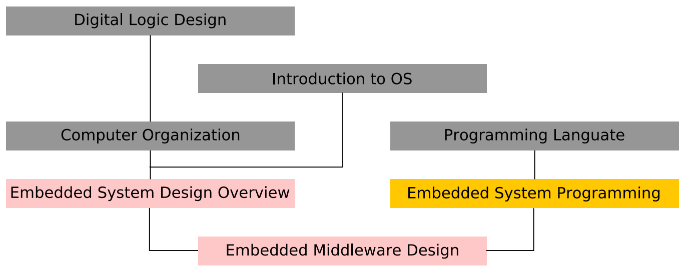
Digital Logic Design
Introduction to OS
Computer Organization
Programming Languate
Embedded System Design Overview
Embedded System Programming
Embedded Middleware Design

Figure 1: Course Relation.

2. COURSE DESIGN

According to the course arrangement plan of Embedded Software Consortium [1, 13], students who take this course must have taken the prerequisite courses. The required prerequisite courses are illustrated in Figure 1. Because students have taken the prerequisite courses, students are assumed to have some pre-required background knowledge. These background knowledge includes basic concept of design issues of embedded systems, experiences of implementing embedded systems, understanding tools with different purposes for embedded system development, and computer organization. Because students already have capabilities to implement the basic embedded system, the design of this course will only focus on teaching students the analysis of applying issues in embedded middleware and the integration concerns for middleware in final embedded systems.

Embedded systems are widely accepted on many different domains. Those variations trigger all kinds of design issues, such as requirement analysis and architecture specification. Establishing proper strategies to deal with these issues can improve the quality of embedded systems and decrease the overall development cost [14, 15]. Middleware helps embedded system developers decrease development complexity by separating application logic from physical environment. Because separation of different concerns is necessary in embedded middleware, object-oriented paradigm is introduced to define system boundary, model physical environment. Model-driven approach is also applied for development process.

To combine embedded middleware design and object-orientation paradigm together is the main new aspect of this course. The course is divided into seven chapters: *Socio-technical Systems*, *Critical Systems*, *USB Software and UML*, *Embedded Software for Transportation Applications as Example*, *Development Issues*, *Memory in Embedded Systems*, and *Migrating Your Software to a New Processor Architecture*. The detail of these chapters are described in the followings.

2.1 Chapter 1: Socio-technical Systems

To help students realize the differences between embedded middleware systems and embedded systems is the first step. The differences between embedded middleware systems and general embedded systems are due to the application domain influences. The decision of applying embedded middleware is made by the characteristics of application domains. It is very important for developers to pick the suitable embedded middleware applying context. Chapter 1 focus on teaching students socio-technical systems to help them analyze system characteristics.

One of essential characteristics of socio-technical systems is emergent properties. Therefore, the definition, the characteristics, and what and how emergent properties influence the development of embedded middleware systems is introduced in this chapter. Socio-technical systems include operational processes and people who use and interact with the technical system [11]. It means that socio-technical systems are governed by organizational policies and rules. After providing the definitions of technical computer based systems and socio-technical systems, students would realize that embedded middleware systems are naturally socio-technical systems. They also learned the concerns and issues are different in these systems. It is very important to teach students the meaning of system engineering after introducing the emergent properties. System engineering is the activity of specifying, designing, implementing, validating, deploying, and maintaining socio-technical system. The purpose of teaching system engineering is to cooperate with the services provided by an embedded middleware, constraints on construction of an embedded middleware system, and the interaction between embedded middleware and other modules in the final system.

2.2 Chapter 2: Critical Systems

In addition to socio-technical systems, embedded middleware systems also belong to critical systems in many application domains [10]. Students have to learn the definition and recognize different types of critical systems. They should further learn the most important emergent property of critical systems is dependability. The dependability of a system reflects the user's degree of trust in that system and the success of an embedded middleware system highly relates to it. Through the explanation of dependability, students will learn four dimensions of dependability which includes availability, reliability, safety, and security.

After leading students to consider the development of embedded middleware by the viewpoint of socio-technical systems and critical systems, it is assumed that students will accumulate more background knowledge.

- *Embedded middleware system is not only the system which connects various hardware and software components by middleware, but also the system has multiple integration issues during developing.*
- *Embedded middleware system is a socio-technical system. Emergent properties should be considered during the development.*
- *The applications of embedded middleware systems are usually critical systems. Dependability should be considered during the development.*

2.3 Chapter 3: USB Software and UML

The goal of this chapter is to teach students the detail development of an embedded middleware system. In order to help students get the whole picture, USB software and UML are demonstrated as examples to compare with the principles of embedded middleware development. There are several similar development points between USB software and embedded middleware. Through these comparison results, students can find out the best way to develop an embedded middleware. One similar point is that both USB and embedded middleware are embedded in different devices which support specified functions. Students could detect how a middleware is embedded in specific devices as USB does. Another similar point is that there are hosts in both USB and embedded middleware to use those functions provided by specific devices to achieve typical application goal. An embedded device or a USB device usually plays both roles, a device which supports specified functions and a host which handles application logic. Students can conclude how to separate physical environments and application logics after the observation of USB devices. Finally, the detail definition of USB software layer can help students understand the communication among other embedded systems by embedded middleware and what an embedded middleware system should have.

Model integration and transformation have been applied on embedded software development [4, 12]. Students should understand the meaning of *Model* and the influence when applying model driven approach to embedded middleware system. Besides, UML 2.0 has paid special attention on embedded system design [2, 5, 6, 7, 8, 9]. For these reasons, UML and how to apply UML in the development of embedded middleware system are introduced formally in chapter 3. The UML development concept is divided into three parts. The first part is that students must understand that model is not a precise or complete unit for final system and model is for communication purpose only. If students understood this property of model, they would further know the reason how UML can help developers in the work of analysis and design, and decrease the overall cost in integration, testing, and deployment. Developers could concentrate in more complex issues and improve the total productivity. The second part is that students should understand that a model of software works as blueprint of architecture. This property shows that model can describe the decisions made by developers, such as problem analysis, solution design, and implementation detail. The third part is that students could understand it is not necessary to distinguish a model from *real software*, and a model could describe the whole software. Because of this property, model compiler is introduced in this chapter. UML can be translated to C/C++ code by model compilers. Model compiler can encapsulate embedded software design parameters when translating a model. It shows that model itself or using model compiler is separated from the application development. Then, design can be split from the application and the architecture. The design of application can be done by domain experts and the design of architecture can be done by embedded experts.

Students learn the concepts and benefits of model driven approach and the process of applying UML to the design of embedded middleware systems. Especially, students would know how UML can integrate embedded systems in the design phase and avoid the complexity from further implementation.

2.4 Chapter 4: Embedded Software for Transportation Applications as Example

The overview of development technologies of embedded middleware systems is introduced in chapter 3. To help students evaluate the principles learned from previous chapters, transportation application is provided in chapter 4. There are several reasons for choosing transportation application.

A transportation system is not only a socio-technical system, but also a critical system. Course content taught in previous chapters could be reviewed through example. Besides, because a transportation system is a distributed system, an middleware is needed to provide communication services for multiple microprocessors or microcontrollers in the transportation system. Communication among these microprocessors or microcontrollers is a typical issue for students to consider real world application of middleware. Communication issue is not the only important issue in transportation system design. To integrate various devices and architectures into a whole transportation system are also typical critical issues. Since all units must be integrated into a final transportation system, finding the most effective method to handle and integrate different architectures is the top mission and students will learn a lot from the discussion in class. Real-time issue is also another famous issue when developing a transportation system. How to make a transportation system predictable and reliable with real-time constraint influences the design of embedded middleware system. Except reviewing the concept of the previous three chapters, related technologies, such as *microprocessor technology*, *system architecture, design composition*, *software content*, *programming language*, and *software team size*, are also mentioned in this chapter.

The major content of chapter 3 and 4 was the overview of development of embedded middleware system. In chapter 3, USB software was used as example to help students understand how to communicate with other embedded systems by embedded middleware and the fundamental modules of an embedded middleware system. Then, model driven approach was taught as a key technology for the development of embedded middleware system. A transportation system was demonstrated in chapter 4 to conclude the concepts previous three chapters.

2.5 Chapter 5: Development Issues

It is assumed that students are qualified to develop an embedded middleware system after the teaching of four chapters. To guide students to further development issues due to the characteristics of embedded software is the purpose of this chapter.

The major difference between embedded middleware system and general middleware system is *Software/Hardware Trade-offs*. Students must know that software design and hardware design are tightly coupled in the development of embedded middleware systems. This issue make the development of embedded middleware system different with general purpose middleware system in several typical aspects, such as organization and process. Besides software/hardware trade-offs, there is one more issue that students should pay special attention to when they choose development assistant tools. Suitable tools can speed up the development process and improve the quality of the final product. Different development tools will equip with different features but also with different development strategies.

It was assumed that students could accumulate the knowledge of what kind of problems they would possibly face during development process and what kind of tools they could use to solve those problems. In addition to these issues, students should also focus on the influence from hardware design decisions. For this purpose, students would learn the meanings of memory in embedded systems in chapter 6 and consider the portability of embedded software in different processors.

2.6 Chapter 6: Memory in Embedded Systems

Students must understand memory has different definitions in different contexts and it would be mixed in the development of embedded middleware systems. Students should understand memory architecture in embedded systems and the different definitions of memory in different contexts. After learning these background knowledge, students will be taught how to use various memory sections for various purposes. After chapter 6, students are expected to utilize their knowledge to consider constraints from the viewpoint of memory variation to make right design decisions.

2.7 Chapter 7: Migrating Your Software to a New Processor Architecture

The design of embedded middleware system highly depends on the specification of processors. To handle the migration issues between different processors was the purpose of this chapter. Different processors have different instruction and registers. Although high level programming languages, such as C and C++ code, have portability, language compilers still depend on the processor. Besides, there are different real-time profiles in different processors. Especially for interrupt and in-house RTOS are also the hindrance to portability. Students should practice their knowledge and accumulate experience to solve these issues. Embedded application binary interface (EABI) was introduced in this course to demonstrate a possible solution for students to solve these issues from migrating different processors. In order to help students practice the knowledge they learned in this course, there was a final project to let students develop an embedded middleware application for evaluation.

3. EXPECTED RESULTS

It is expected that students can get knowledge in four aspects after they finish the course.

- *Students should identify the **embedded middleware applying context**.*
- *Students should establish their background knowledge for **embedded middleware system architecture design**.*
- ***Environment issues handling** is necessary for developers to build the desired embedded middleware systems due to the various application domains of embedded middleware.*
- ***Embedded middleware practice** is necessary for students to utilize knowledge learned in this course to design an embedded middleware.*

3.1 Embedded Middleware Applying Context

This aspect focuses on whether students understand the real world factors of applying embedded middleware. Students should establish their own knowledge to identify the right context by following three principles. The first principle is the difference between embedded systems with embedded middleware and embedded systems without embedded middleware. In order to evaluate students' understanding of this principle, students are required to answer some questions after the teaching of chapter 1. If students could find out the answers correctly, it is very possible for them to identify suitable embedded middleware applying context. The second principle is analyzing emergent properties from an embedded system with embedded middleware. After finishing the teaching of chapter 1, students are required to analyze the scenarios of their final projects and find the major emergent properties in the chosen domain. It is important to evaluate the analyzing results to know whether they have capabilities to find the emergent properties of an embedded system with embedded middleware or not. If students have capabilities to find the emergent properties, they have insight of the characteristics of embedded middleware. The third principle is the dependability of an embedded middleware system. The applications of embedded systems with embedded middleware are usually critical systems. The analysis result of dependability has strong impacts in applying the embedded middleware and the characteristics of the embedded middleware. If students are able to analyze dependability of an embedded middleware system, they can have a clearer picture of their final progress and the embedded middleware.

3.2 Embedded Middleware System Architecture Design

Students should learn how to design an embedded middleware in required software products after this course. Students should have two capabilities to achieve this goal. The first capability is to integrate embedded middleware into an embedded system. Embedded middleware integrates various software components and hardware components into a whole system. It also separates application logic and physical environments to help developers pay attention on application logic only. Therefore, it is important to evaluate students' understanding of this capability. The evaluation is done by observing the design documentation of the final projects. A good architecture design of the final project shows that students have better understanding of how to integrate components by embedded middleware. The second capability is the understanding of model driven development. Model driven development is useful when developing embedded system with embedded middleware and can help developers avoid the complexity of environment constraints in early design. However, it is not easy to evaluate this capability by normal questions or documentation. The better way is to observe the development process of the final project with related documentation during the development. If students produce high quality documentation and define suitable development process, they must have better understanding of model driven development.

3.3 Environment Issues Handling

Students should focus on four design issues when dealing with environment issues. These issues are *software/hardware codesign*, *tool choices*, *memory variation*, and *architectures variation*. Students should analyze the impacts from environment and practice the process of software/hardware codesign after this course. Although application logic and physical environment are separated by embedded middleware, developers are still responsible for handling the impact from the physical environment to make effective design decisions. Again, the design documentation is the only base to detect whether students are able to practice software/hardware codesign or not. Another capability that students need is to choose suitable tools for different platforms. There are two methods to proceed the evaluation. One is to lead students to answer questions about the concepts in chapter 5, and the other is to observe the tools they choose during the development of final projects. Memory variation handling is not software/hardware codesign. Software/hardware codesign focuses on picking suitable process to handle software/hardware codesign but memory variation handling is the capability to handle the issues from memory constraints to produce better design. Besides memory variation, students could learn architecture variation handling technique. Students should be able to migrate their software among different architectures and designs. In order to check whether students can handle all above issues or not, the evaluation of final project documentations would include all of them.

3.4 Embedded Middleware Practice

The practice plays an important role in this course. We can observe the practice result to detect students' study performance. Besides, students could figure out that qualified documentation does improve the quality of final products through the practice.

The documentation produced in specification phase must clearly present the speciality of the system. Students won't be forced to follow any commercial standard to produce their documentations in this phase but only pay attention on the

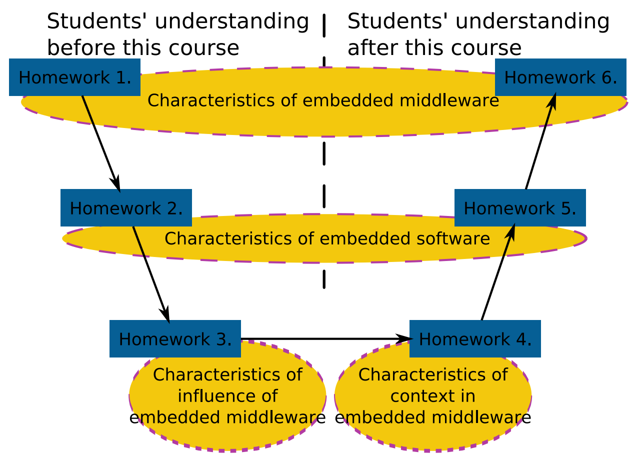
Students' understanding before this course
Students' understanding after this course
Homework 1.
Homework 6.
Characteristics of embedded middleware
Homework 2.
Homework 5.
Characteristics of embedded software
Homework 3.
Homework 4.
Characteristics of influence of embedded middleware
Characteristics of context in embedded middleware

Figure 2: Homework Design.

expressiveness of the documentations. After specification phase, students are required to produce design documentations by utilizing UML. Students can learn how model driven development approach can improve development process of embedded middleware system by practicing UML. After design phase, students would choose suitable tools according to the specific project to speed up the development process or decrease the development complexity.

Students can get the evidence of the knowledge learned in this course from the embedded middleware practices. The practices are well observed to understand the study performance as the base to improve the course.

4. COURSE RESULT ANALYSIS

The execution of this course was from September 2007 to January 2008. Eleven graduate students took this course. The homeworks and final projects which students produced during this courses were the course results. The analysis of the course results can reflect the performance of this course and is also as the base for further improvement of this course. The following sections present detail description of homeworks and final projects.

4.1 Homework Analysis

To match the original goal of this course, six homeworks are designed to help students understand the application domain of embedded middleware and the influence of embedded middleware. This section describes homework design and the homework analysis result.

4.1.1 Homework Assignment Design

The arrangement of six homeworks is illustrated in Figure 2. In order to track the progress of students in the course, homeworks are separated into two phases. Students finished homework 1 in the beginning of this course and homework 2 during the teaching process of chapter 1. The results of homework 1 and 2 reflected students' original background of embedded middleware and embedded software. Then, students finished homework 3 and homework 4 after the teaching of chapter 4. Homework 3 required students to answer the influence of embedded middleware during development process and homework 4 required students to consider the

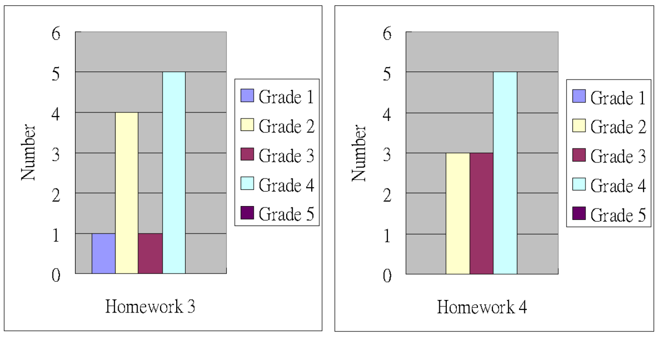

Figure 3: Homework 3 and 4 result.

suitable context for embedded middleware. Both homework 3 and 4 helped students review previous content and conclude their own ideas of embedded middleware. Students finished homework 5 and homework 6 in the last two classes of this course. Homework 5 and 6 required students to answer the same problems again in homework 1 and homework 2. The purpose was to force students to evaluate themselves through the same questions and compare the answers to double check what they have learned.

According to homework assignment design, there are four indicators to evaluate the performance of this course. To grade the answer in homework 3 is the first indicator. This indicator detects understanding about the embedded middleware influence. The second indicator is to grade the answers in homework 4 and this indicator detects the understanding of the embedded middleware applying context. The third indicator detects the improvement of understanding of embedded software design after the teaching of course content. The difference in scores between answers in homework 2 and homework 5 represents the third indicator. The fourth indicator detects the improvement of understanding about the characteristics of embedded middleware after this course. Detection the difference in scores between homework 1 and homework 6 is the representation of fourth indicator. Analysis results of homework is according to the detection of these indicators and is described in the following section.

4.1.2 Homework Analysis Result

The scores of homework can be mapped into five grades. Grade 1 is the lowest score, grade 5 is the highest. Figure 3 displays the result of homework 3 and homework 4. According to the results shown in Figure 3, the mean of scores is 2.91 but almost half answers are under grade 3. The result shows that only half of students have right vision in homework 3 and the overall performance is not good before homework 3. In homework 4, only three students score grade 2 and others score above grade 3. The result represents that students have better understanding about the applying context of embedded middleware.

The comparison between homework 2 and homework 5 is illustrated in Figure 4. As illustrated in Figure 4, mean and median of scores in homework 2 are 3.27 and 4. Mean and median of scores in homework 5 are 3.36 and 4. The results of homework 2 are similar with results of homework 5. The progress between homework 2 and 5 is little. The reason

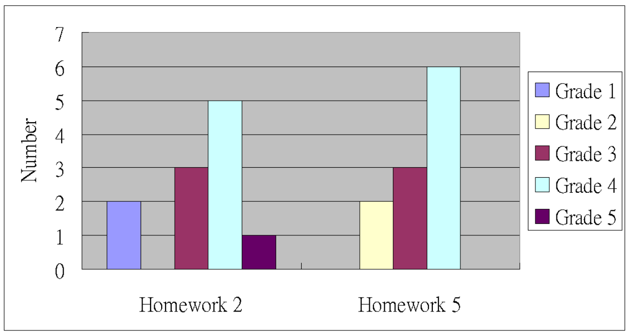

Figure 4: Homework 2 and 5 comparison.

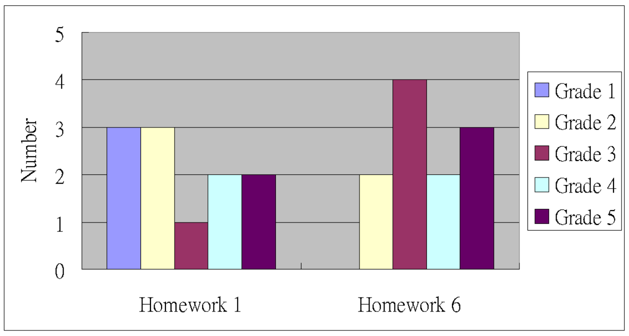

Figure 5: Homework 1 and 6 comparison.

may caused by that most students have enough background knowledge of embedded software. Although the mean and median of scores in both homeworks are similar, students get grade 1 in homework 2 both improved their scores in homework 5. This phenomenon shows the positive influence of this course.

Figure 5 illustrates the comparison between homework 1 and homework 6. Over half students could score under grade 3 in homework 1. Then, mean and median of scores in homework 1 are 2.73 and 2. This phenomenon shows that students took this course had no clear ideas about embedded middleware. Result of homework 6 displays the progress of students. Most of answers in homework 6 score above grade 3 and mean and median of scores in homework 6 are also better than scores in homework 1. The comparison result of homework 1 and homework 6 shows positive results of the course performance.

4.2 Final Project Analysis

There are six teams organized in final projects. Each team would find out a topic to fit the characteristics of embedded systems and decide which part should be embedded middleware. This section describes the support devices for students to develop final projects and the analysis of final projects.

4.2.1 Support Devices

In order to support students' final projects, there are several equipments are prepared in this course. The purpose of the preparation is to support two different communication modes. The first mode is client-server mode so students can build a networked embedded system within it. Equipments

Table 1: Embedded Middleware Server Specifications

	Computation Power	Space	Mobility
PC	High	High	Low
Notebook	Medium(1.66GHz)	Medium	Medium
Sony VGN-27TN	Medium(1.33GHz)	Medium	High
Dopod U1000	Low(624MHz)	Low	High

for middleware server are presented in Table 1. Equipments for middleware server can be classified according three attributes: computation power, space, and mobility. Students can choose suitable equipments as middleware servers to develop the final projects. Equipments for middleware client are also provided. According to Table 2, equipments for middleware client can be classified by four attributes: processor, memory, mobility, and development cost. Besides equipments for middleware server and client, equipments for wireless equipments are also provided because most of networked embedded systems communicate by wireless network.

The second mode is peer to peer mode. Mobile devices are one of most import applications of embedded system. Most mobile devices support ad hoc network and communicate in peer to peer mode. For this reason, equipments support both peer to peer mode and mobile devices in this course. Equipments for peer to peer mode is from the equipments for client-server mode because all equipments can support both client-server mode and peer to peer mode. Students could use these equipments to simulate the application in client-server mode or peer to peer mode according to the application domain.

4.2.2 Final Project Summary

- Group 1: Amusement Facilities Routing Assistant (AFRA)

  This project aims to build an embedded middleware for amusement facility routing assistant. The routing assistant suggests routing paths of the amusement park to visitors based on the preferences made by tourists to decide scheduling strategies. Through the help of AFRA, visitors can play as many amusement facilities provided by the park as possible. Considering the facility queuing information is scattering and varying with time, a middleware is necessary to aggregate and distribute real-time queuing information for routing path suggestions. With the middleware, a uniform abstraction of queuing information is introduced so the communicating complexity between diverse facilities and different visitors is reduced.

- Group 2: Service-oriented Bus Embedded Middleware (SOBUS)

  This project aims to build an embedded middleware for service-oriented city bus systems (SOBUS). SOBUS provides automatic travel plan service for visitors according to the destinations they want to go by taking buses. Every destination contains different set of services. According to the characteristics of available services, SOBUS will reconfigure the plan dynamically for efficient travel. The reconfiguration requires communication and coordination between service providers and

Table 2: Embedded Middleware Client Specifications

	Processor	Memory	Mobility	Development Cost
Notebook	1.66GHz	1GB	Medium	Low
Sony VGN-27TN	1.33GHz	1GB	High	Low
Dopod U1000	624MHz	128MB	High	Medium
Nokia N95	330MHz	64MB	High	High
iRex iLaid ER-0100	400MHz	64MB	High	High

SOBUS. In order to fulfill these requirements, SOBUS is built. SOBUS deals with the communication and coordination among services in various destination and mobile devices with each visitor. In addition, overall service quality and service resource utilization can be improved by coordinating all service requests.

- Group 3: Mobile Home Automation (MHA)

  This project is to build an embedded middleware for people to facilitate their household appliances with remote control strategy. MHA mediates the control of household appliances and hides the heterogeneity between various household appliances when different communication protocols are applied. With MHA, people can control their household appliances remotely by any mobile device through the internet. By providing a standard interface, MHA improves the flexibility when people add/delete any household appliances with different communication protocols in the future.

- Group 4: Bulletin System (BS)

  In Taiwan, there are a lot of information exhibition markets available. Normally, each exhibition will contain more than thousands of people. A group of friends are often separated in each environment. An embedded bulletin system in mobile device could help group communication in such context. However, different types of information, heterogeneous bulletin services by the market, and security of group member information are the issues of the environment. Group 4 proposed a middleware embedded in the exhibition bulletins (BS), that provides a communication base for group members getting various pricing information and share their own message securely through their own mobile devices. The middleware separates low level communication issues from application logics of service on bulletin system. Thus services of group communication could easily be developed without handling physical communication and security issues.

- Group 5: A Remote Control Used in a Literature Room (RCULR)

  There are many devices with different operations and controllers in lecture room. For speakers or professors who are unfamiliar with those operations, they have to operate devices one by one respectively to finish a particular activity during speech. The RCULR aims to solve the complexity and develops a middleware that is distributed over devices and remote controllers attached to microphone. The middleware is responsible to transmit operations to corresponding devices by decomposing users' commands on microphone. After devices finish their operations, RCULR validates received feedbacks and displays current status of devices to users. In other words, users can control many devices immediately by performing predefined commands on microphone. By using RCULR, unnecessary interaction with device maintainer will be reduced and it will make the whole speech process with smoothness during speech. The presented middleware-based approach hides the heterogeneity of devices and makes device replacement or extension easier in future. On the other hand, users can avoid the complexity of what and how devices actually operate and spend their effort on current conditions interpreted by middleware in lecture room.

- Group 6: Push-to-Talk System on Group (PTS)

  This team provides functionality, walkie-talkie, on mobile devices and walkie-talkie supports group communication. In order to complete walkie-talkie, an embedded middleware (PTS) is proposed. This middleware implement walkie-talkie functions by the combination of SIP protocol, RTP protocol, and RTCP. The major benefit of PTS is to provide a standard interface for group communication and improve the efficiency when building up application bases on group communication.

4.2.3 *Final Project Analysis*

Final project results could reflect on expected results described in section 3, but it is not easy to directly reflect the observation to expected result. For this reason, there are six indicators proposed to evaluate final projects. These indicators are utilized in the discussion of course performance. The evaluation result shows in Table 3. The first indicator is *Proposal Validation.* This indicator validates that each team chooses a suitable topic and exposes enough evidence in the proposal to explain the topic with course goal. The evidence includes system features should be embedded and application logic should be separated by the embedded middleware. This indicator detects whether students understand the meanings of embedded middleware. If students understand the meanings of embedded middleware, they would choose related topics and show enough evidence to prove it. The second indicator is *Domain Analysis Verification.* This indicator verifies domain analysis result of each team. Students must proceed domain analysis to make sure which system features are fixed in the environment and which system features belong to application logic. Results of this indicator represent the separation between physical environments and application logic in final projects. The third indicator is *Model Driven Approach Verification.* Students are taught to utilize model driven approach to develop embedded middleware systems. This indicator observes the documentation to verify the degree of applied model driven approach in the development process. The fourth indicator

Table 3: Final Project Evaluation

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6
Proposal Validation	4	5	2	3	3	2
Domain Analysis Verification	3	5	5	3	4	4
Model Driven Approach Verification	3	5	2	2	2	2
Environment Issues Verification	2	4	2	2	2	3
Completeness of Documentation	3	5	2	3	3	2
Evaluation of Final Presentation	4	5	2	2	2	3

Table 4: Final Project Analysis

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Average
Characteristics of Embedded Middleware	4	5	2	3	3	2	3.17
Architecture Design of Embedded Middleware	3	5	3.5	2.5	3	3	3.33
Environment Issues Handling	2.5	4.5	3.5	2.5	3	3.5	3.25
Development Process	3	5	2	2.5	2.5	2	2.83

is *Environment Issues Verification*. This indicator observes the documentation to distinguish that students made design decision according to environment issues. The fifth indicator is *Completeness of Documentation*. Evaluation result of completeness of documentation can expose the development process of each project and understand whether each team choses suitable process. The sixth indicator is *Evaluation of Final Presentation*. Evaluation result of final presentation represents the quality of each project. The purpose is not only to judge quality of each project but also provide the base to understand the relation between other indicators and project quality.

According to the evaluation results of final projects, the performance of this course is exposed in Table 4. *Proposal Validation* represents the degree of understanding of characteristics of embedded middleware because students who know characteristics of embedded middleware can also provide a suitable proposal. According to the description in section 3.2, learning performance of architecture design of embedded middleware system should be represented by the average of variables of *Domain Analysis Verification* and *Model Driven Approach Verification*. Development issues from embedded design are regarded as how to handle design issues from environments and indicators, *Domain Analysis Verification* and *Environment Issue Verification*, represent this result. Finally, *Development Process*, which presents process quality of final projects, is the average of *Model Driven Approach Verification* and *Completeness of Documentation*. *Model Driven Approach Verification* is chosen because this indicator reflects the degree of applying model driven approach on development process.

Except *Development Process*, the average score of each indicator is over grade 3. This result shows that students don't follow an identified process to develop final projects. Besides, we observe that *architecture design* and *environment issue handling* are the best indicators among the rest indicators. Both indicators are for the purpose of application domain understanding. Except group 6, other groups didn't choose familiar application domains in their final projects. Therefore, the better performance in domain understanding is caused by the emphasis on domain analysis in this course. After observing the results in each indicator in Table 4, the comparison among evaluation of final presentation and indicators in Table 3 is proceeded to find the reason of better project quality to be the base of next course. The comparison shows two phenomena. One is that groups with better project quality have better scores in indicator *Model Driven Approach Verification*. The other is that group 6 only performs better result in *Environment Issue Verification* and that is the reason why the project performs better than other projects with low scores. These two phenomena imply that in order to improve final project quality, model driven approach and environment issue handling should be emphasized during the development.

5. CONCLUSION

In this work, the course design of Embedded Middleware Design Curriculum is presented. The proposed course content is from the viewpoint of software engineering and the course design focuses on which system features should be embedded, which system features should be separated from embedded middleware, and the influence of embedded middleware during software development. Besides course design, expected results and course result analysis are also presented. Embedded Middleware Design Curriculum is an ongoing project. The goal is to collect more teaching-related materials and experiences to help other universities in Taiwan to develop embedded middleware design related courses more effectively.

6. ACKNOWLEDGMENTS

This research was supported in part by the Taiwan ESW Consortium, and in part by the Taiwan National Science Council (NSC) under contracts NSC 96-2628-E-006-024-MY2 and NSC 97-2221-E-006-177-MY3.

7. REFERENCES

[1] The Embedded Software Consortium, VLSI Circuits and System Education Program, Ministry of Education, Taiwan, http://esw.cs.nthu.edu.tw/e-index.php.

[2] R. Damasevicius and V. Stuikys. Application of UML for hardware design based on design process model. In *Proceedings of the 2004 Conference on Asia South*

Pacific Design Automation: Electronic Design and Solution Fair, pages 244–249, January 2004.

[3] T.-Y. Huang, C.-T. King, Y.-L. S. Lin, and Y.-T. Hwang. The embedded software consortium of Taiwan. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(3):612–632, August 2005.

[4] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145–164, January 2003.

[5] P. Kukkala, J. Riihimäki, M. Hännikäinen, T. D. Hämäläinen, and K. Kronlöf. UML 2.0 profile for embedded system design. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 710–715, March 2005.

[6] G. Martin. UML for embedded systems specification and design: Motivation and overview. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 773–775, March 2002.

[7] G. Martin, L. Lavagno, and J. Louis-Guerin. Embedded UML: a merger of real-time UML and co-design. In *Proceedings of the Ninth International Symposium on Hardware/Software Codesign*, pages 23–28, April 2001.

[8] A. T. Murray and M. Shahabuddin. OO techniques applied to a real-time, embedded, spaceborne application. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, pages 830–838, October 2006.

[9] M. F. S. Oliveira, L. B. de Brisolara, L. Carro, and a. R. W. Fl˙Early embedded software design space exploration using UML-based estimation. In *Proceedings of the Seventeenth IEEE International Workshop on Rapid System Prototyping*, pages 24–32, June 2006.

[10] D. C. Schmidt. Middleware for real-time and embedded systems. *Communications of the ACM*, 45(6):43–48, June 2002.

[11] I. Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, 8th edition, 2007.

[12] J. Sztipanovits and G. Karsai. Model-integrated computing. *IEEE Computer*, 30(4):110–111, April 1997.

[13] S.-L. Tsao, T.-Y. Huang, and C.-T. King. The development and deployment of embedded software curricula in Taiwan. *ACM SIGBED Review*, 4(1):64–72, January 2007.

[14] J. W. Valvano. *Embedded Microcomputer Systems: Real Time Interfacings*. Thomson-Engineering, 2nd edition, 2006.

[15] C. Walls. *Embedded Software: The Works*. Newnes, 2005.