

재사용 가능한 컴포넌트 기반의 임베디드 시스템 통합 프로토타이핑 기법

Husni Teja Sukmana[†] · 이 정 배^{**} · 임 기 옥^{**} · 황 영 섭^{**} · 김 영 진^{***} · 안 성 순^{****}

요 약

최근에 임베디드 시스템 개발자들을 위한 프로토타이핑 시스템 도구가 시장에 많이 출시되고 있다. 일찍부터 임베디드 시스템 설계에서 사용자의 요구사항을 분석하고 사용자 편의 인터페이스를 만드는데 있어서 프로토타입은 중요한 요소이다. 프로토타이핑 도구는 다음과 같이 Physical Prototyping (PP), Virtual Prototyping (VP), Modeling Prototyping (MP) 3가지로 나눌 수 있다. 이러한 프로토타이핑 기술을 통합하면 개발에 소모되는 비용내지 시간이 절약되므로써 시장진입시점이 빨라지고 제품의 경쟁력이 향상된다. 하지만 이러한 프로토타이핑을 통합하는데 있어서 여러가지 문제점이 존재한다. 본 논문에서는 여러 프로토타이핑 도구를 통합하는데 있어서 Javabean, ActiveX등과 같이 유연하고 재사용성이 용의한 컴포넌트를 이용하여 문제점을 해결하는 모습을 보여준다. 이러한 기술은 여러 임베디드 시스템을 개발시 재사용성과 유연성을 통한 편리성을 가질 수 있다.

키워드 : 가상 프로토타이핑, 모델링 프로토타이핑, 실물 프로토타이핑, 통합 프로토타이핑, 임베디드 시스템

Embedded System Integrated Prototyping Mechanism Based on Reusable Component

Husni Teja Sukmana[†] · Jeong-Bae Lee^{**} · Kee-Wook Rim^{**} · Young-Sup Hwang^{**}
Young-Jin Kim^{***} · Sung-Soon Ahn^{****}

ABSTRACT

Recently, there are many embedded system prototyping tools for helping embedded system designers to trial their product before it releases to the market. A prototype is very important for early embedded system design to grasp the desire functions, to get a good performance, to create delightful user interface, and to increase the valuable of the product. Prototyping tools can be classified by three categories: Physical, Virtual and Modeling prototyping. The integration of these prototyping tools becomes valuable for speed up time-to-market and for decrease design cost when design embedded system. The problem comes up because these tools sometime do not provide an instrument for communicating each other. In this paper, we propose a flexible and reusable mechanism for integrate these tools base on JavaBeans and ActiveX technology. We show how this mechanism can be employed in various prototyping tools.

Keywords : Virtual Prototyping, Modeling Prototyping, Physical Prototyping, Integrated Prototyping, Embedded System

1. Introduction

Nowadays, embedded system products have been used in every sectors of our life, for example in daily activities we use mobile phone, MP3, electronic dictionary, etc. They

become more advanced, complicated and sophisticated hardware, software and middleware combination. To satisfy this situation, designers must concern about some metrics such as time-to-market, production cost, and etc [1]. However, just dealing with those metrics still not guarantees that end-users like their products. Thus, designers also must concern about trend and users requirements.

To deal with these problems, it is necessary to develop prototypes before making real products. A prototype may play a significant role not only for hardware and software designers but also for end users and marketing people.

* 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음. (IITA-2009-C1090-0902-0020)

† 준 회 원 : Sun Moon University, computer science and engineering

** 종 신 회 원 : 선문대학교 컴퓨터공학부 교수

*** 정 회 원 : 선문대학교 컴퓨터공학부 전임강사

**** 준 회 원 : 선문대학교 전자계산학과 석사과정

논문접수 : 2009년 4월 7일

수정일 : 1차 2009년 5월 18일

심사완료 : 2009년 5월 22일

Furthermore, it can detect prone error in design step, and become easy to debug mistakes. Basically, there are three prototyping tools: Physical Prototyping (PP), Virtual Prototyping (VP), and Modeling Prototyping (MP) [2].

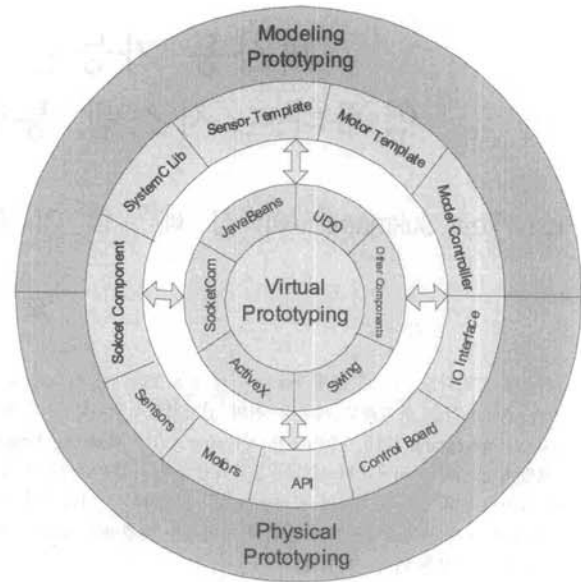
Embedded system design tools have been developed to handle VP, MP, and PP. VP, as its name implies, demonstrates a real picture of embedded system products in computers before their being coded in low-level hardware language. Furthermore, by using VP tools, front-end designers may create good-looking 2D or 3D interfaces to showing their functions to end-users.

On the other hand, a PP tool delivers the “looks-like” prototyping that offers an embedded system closer to the actual product. A typical example of a PP tool is the Embedded System Prototyping Suite (ESPS) [4], which has been tested in our laboratory previously, and which will be used in this study. Moreover, while PP only builds a miniature version of the product, we suggest that it is necessary to describe the hardware at an abstract level, the software algorithms, and the hardware architecture, by using MP tools. A popular modeling language, with an object-oriented base, is SystemC [5]. SystemC has the ability to build a natural programming language such as C or C++ into a hardware design language by adding some libraries. It can be used to describe system level designs, software algorithms, and hardware architectures.

There are many VP tools such as RapidPlus [6], Eclipse [7], and NetBeans [8]. Each VP tool has its own advantages and disadvantages. For example, RapidPlus can model end user prototyping easily, because it is bundled with particular objects for designing user interfaces. One of Eclipse plug-ins has a system-on-chip front end interface for SystemC simulations. The plug-ins can generate waveforms such as a clock. This paper will focus on making a supplement for VP tools to enable integration with PP and MP. To achieve this goal, we assume that it is necessary to develop flexible and reusable supplements, which can be included in several VP tools.

Here, we propose a mechanism to integrate PP, MP and VP which later we are called it Integrated Prototyping (IP), focusing in flexibility and reusability. IP offers interfaces based on JavaBeans and ActiveX to integrated PP, VP and MP. In this way, user-level interface designers can choose freely which tool they may want to utilize when they undertake integrated prototyping.

The paper is organized as follows: Section 2 will mention related work, and Section 3 and 4, we describe our motivation and contributions of the paper. In Section



(Fig. 1) Overview of IP's Components

5, we present the system design and implementation in detail. In Section 6 we will show how IP can be used as flexible and reusable in various embedded system design tools. Finally, we will conclude with summary and future work this paper in Section 7.

2. Related Works

PP, as already mentioned above, has the task of delivering a miniature version of embedded system products, such as real visualizations of factory automation, e-car systems, and vending machine systems. A research contribution that implements VP with Java Swing, together with PP, can be found in [12]. In [3], design, test, and analysis information for VP is integrated via a “d.tool”. The d.tool includes hardware components such as sensors and actuators, and software to track a video user’s interaction. A similar tool has been used by [15], but this focuses on the automatic discovery of distributed physical interfaces.

Some of this research involves similar ideas to our study, in terms of utilizing VP for controlling or modeling PP. However, we found that the research did not tell us whether it is possible to employ PP in several VP tools. We have an advantage over previous research by making components that can be plugged into different VP tools. On the other hand, we suggest that it is also necessary to bond the PP and VP to MP, and we did find some research that tried to integrate PP and VP with MP. We have discovered that MP is an important aspect of

speeding up time-to-market and can be used for user-level prototyping. Therefore, we bundle MP into VP to make an integrated prototyping system.

Because we want to integrate PP, VP, and MP, we investigated studies that deal with MP, especially for integrating and making VP in SystemC. An example is the use of QT, multiplatform GUI toolkit, which is based on OpenGL [9]. The limitations of output presentation in MP can be covered in QT by displaying a clock signal, a list of modules, and a list of signals. The implementation of this research enables designers to add menus to control simulation, display the simulation time, choose the signals they want to display, and link them in a new signal window. This work requires modification to the SystemC kernel. Because SystemC still needs some improvement, we think it is better not to modify the SystemC kernel as an alternative to making independent graphical modules. An IDE for MP has been developed in [10], which supports an abstraction level, beyond RTL, to implement complex system on chip design in MP. This raises the issue of new techniques to support debugging, system exploration, and verification.

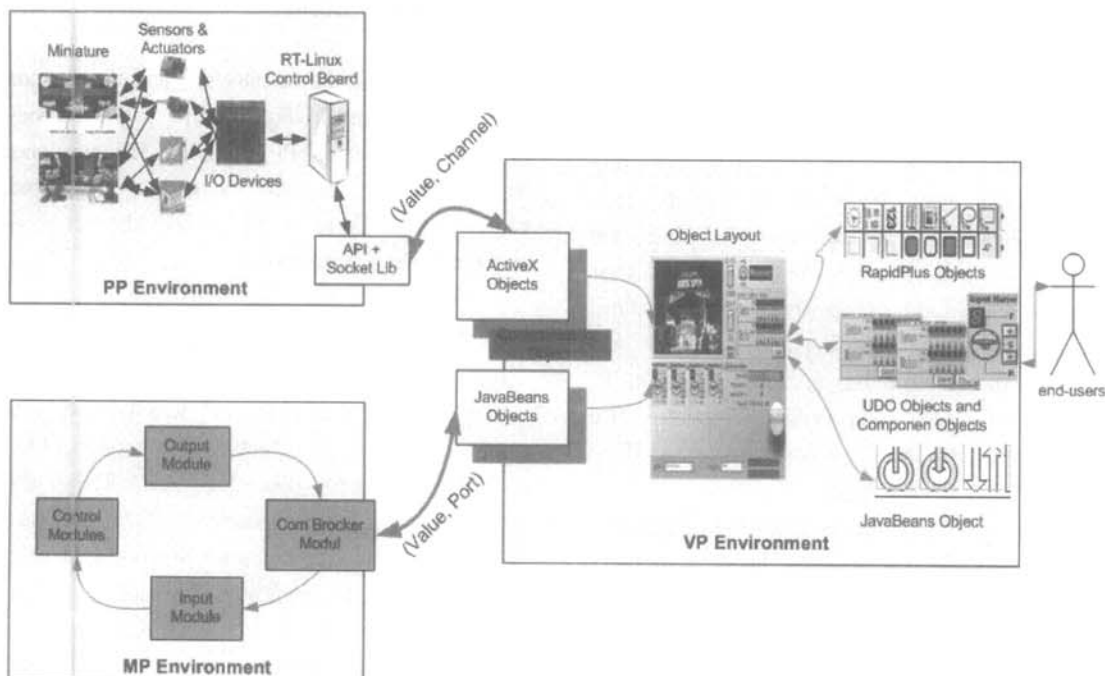
Our proposed system tries to integrate VP, PP, and MP. This can be considered as an extended version of our previous work [16] that includes many improvements. Integrated prototyping comprises a client side and a server side. The client side is VP because we assume that VP deals with the GUI used by the end user to

input data or view the desired results. Many GUI tools have been created for supporting embedded system development, including RapidPlus, Virtio [17], and standard GUI tools such as NetBeans and Eclipse. Each of these has advantages and disadvantages. The JavaBeans and ActiveX component that we have created is inserted in the client side because we want to offer GUI designers flexibility when choosing GUI tools.

3. Motivation

In general, PP and MP have been delivered to the market without VP. Thus, it is difficult to demonstrate them from a user level perspective. For example, the SystemC in our MP only supports textual output via *printf* and *cout* to show results. It is inefficient to trace the output in this way when a simulation runs. Although this problem can be mitigated by using the trace commands provided by SystemC, it still needs VP to run them. Several researchers have tried to improve MP's capability [9-12].

OpenGL has been used to enhance visualization of MP by generate timing diagrams [9]. The SystemC IDE has been constructed to support debugging, system exploration, and verification [10]. A user-level VP based on MP [11] has been presented, together with a case study. This involves a library, called the SystemC Java Library (SCJLib), which manages the communication between two



(Fig. 2) Overall Architecture of IP

independent programs, a VP program based on the Java Swing toolkit and an MP program based on SystemC. Because VP tools will continue to evolve, it is difficult to integrate MP into VP without flexible and reusable tools. Therefore, this issue is our first motivation for implementing IP.

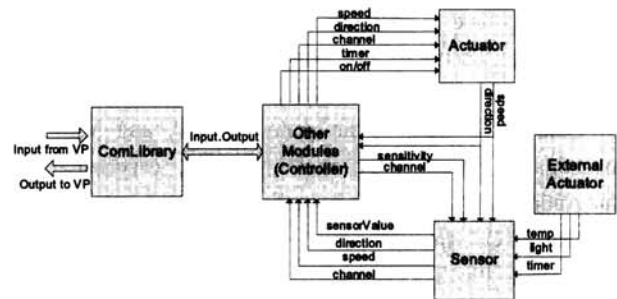
The second motivation runs parallel to the first, in that it involves communication between VP and PP. The ESPS tools, which are our PP tools, do not provide a front-end interface. Instead, the ESPS toolkit includes some API library functions, I/O boards, sensors, actuators, and Lego. It runs on real-time Linux, which is text-based. Unlike SystemC, the ESPS tools are used to create a miniature version of the hardware that employs some actuators and sensors. In [12], PP was simulated with VP based on Java Swing. However, reusable mechanisms were not used for the intercommunication, preventing application with other VP tools such as RapidPlus or even Eclipse.

Finally, these motivations can be used to address the problem comprehensively. We can use IP between VP and PP, or to connect VP and MP. Furthermore, we can integrate all three prototyping categories. Suppose that someone wants to make a controller in MP, and wants to show that the controller can control the actuators or sensors in PP. In this case, IP will have a role as the linker for integrated prototyping.

4. Contributions

To realize integrated embedded system prototyping, we have implemented IP, a flexible and reusable implementation for integrated prototyping. We intend IP to run in various VP tools, particularly for user-level prototyping. By using IP, VP designers can choose freely the VP tools that they need. When designers choose their own VP tools, the most important factor is that the tools should fit their requirements in the aspect of speedy prototyping time. Hence, the solution should be simple, but flexible and reusable so that the tool can run in many VP tools. To this end, we built two components which are tied harmonically within IP. Those components are based on JavaBeans and ActiveX technologies.

In IP, a JavaBeans component is realized to communicate between VP and MP. On the other hand, an ActiveX component is implemented in IP to link VP and PP. Both components utilize the same procedure for which type of data will be delivered and use also similar TCP/IP socket mechanisms for the communication process. Using these



(Fig. 3) MP Architecture

components IP cooperates organically with the built-in objects which are provided by VP tools user define objects (UDOs).

In addition, IP provides a template module for actuators and motors in MP to minimize the effort of their implementing. The template module can be used in a way similar to creating an object in object oriented programming since MP is based on C++ programming. For demonstrative simulations, we also implement a UDO in a specific VP tool (i.e, RapidPlus) for maintaining sensor and motor modules in VP.

Finally, to show how effectively IP can operate combining PP, MP and VP, we have added to various VP tools and run various programs. For such simulations, we realized MP programs which run on different VP tools: a traffic light module for RapidPlus and a waveform simulation for traffic light for Netbean.

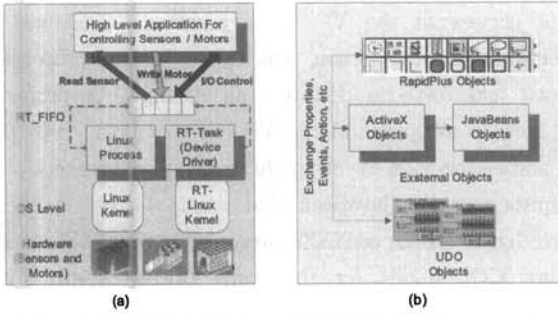
5. IP System Design

In this section, the architecture and the components of IP will be presented. (Fig. 1) depicts the overview of IP tools which contains PP, VP and MP components and objects. On (Fig. 2) show the general architecture of IP implementation. Detail of PP, VP and MP will be explained in the following subsections.

5.1 IP Architecture

5.1.1 Physical Prototyping

We used ESPS tools for PP, which involves hardware components such as actuators and sensors, I/O boards, and API functions (See Fig. 4.a). There are many actuators and sensors available in ESPS, such as motors, lights, touch sensors, temperature sensors, light sensors, and rotor sensors. Each element to be used in an experiment must be connected to an I/O board as a plug-ins for I/O board channels. In addition, there is an API included in ESPS tools. The API provides functions to control motor



(Fig. 4) (a) PP Architecture (b) VP Object Components

and sensor values, such as controlling the speed of motor rotation, changing the motor direction, and setting the sensor sensitivity. Some functions have a parameter to identify the particular motor or sensor used. This parameter must be set to the slot or port used by the sensor or motor to connect to the I/O board. As shown in (Fig. 4.a), the PP environment points up the relation between the various sensors and motors, the I/O board, and the control board.

To communicate with IP in VP, we apply a socket mechanism to receive data from, and send data to, IP. There are three parameters required, the first data item being the channel, the second being the value and the third being the direction. The channel identifies which sensor or actuator is being used. Because each I/O board restricted only for 16 channels (8 channels for actuators and 8 channels for sensors), it is necessary to check the channel hardware plugged into the I/O board and then set the channel numbers referring to that hardware configuration. The second data item is the value for the speed of the actuator or the sensitivity of the sensor. The third data item specifies a direction, such as right or left rotation, for the actuator or sensor.

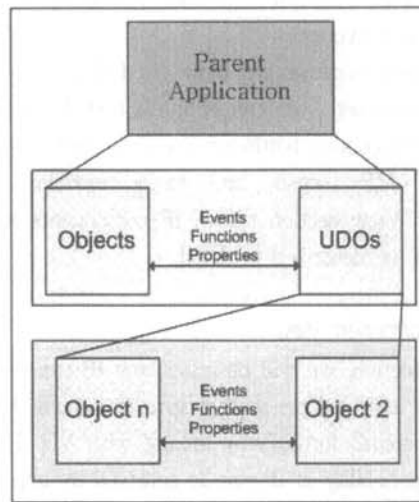
5.1.2 Modeling Prototyping

MP exchanges information with IP by employing the SCJLib used in [11]. This library contains an input module and an output module. The input module implements the function of receiving data from IP to send to the simulation module. If the simulation wants to give data to IP, it must be sent to the output module. The modules communicate with each other by using the *sc_channel*. The *sc_channel* will link signals such as *sc_in*, *sc_out*, and *sc_inout*, from one module to other modules.

In addition, we provide template modules for sensors and actuators. Each module has input and output ports for receiving signals from, and sending signals to, other modules. (Fig. 3) shows the details of the MP architecture.

<Table 1> The Beans Component Classes and Definition

Name	Definition
SCJBroker	Deal with Input, Output, and Tunnel
SCJInput	Send Data to Server
SCJOutput	Receive Data From Server
SCJInputListener	Implemented by Broker
SCJOutputListener	Implemented by Output
SCJInputEvent	Used by Input to Broker
SCJOutputEvent	Used by Broker to Output
SCJBrokerBeanInfo	Specific properties/method/icon for Broker
SCJInputBeanInfo	Specific properties/method/icon for Input
SCJOutputBeanInfo	Specific properties/method/icon for Output



(Fig. 5) Relation among Objects, UDOs and Parent Application

The aim of these modules is to simulate the actuators and sensors. MP designers can implement these modules in particular cases by using appropriate mathematical equations to model the speed, motor power, and sensitivity of sensors, for example.

To match the sensor and actuator models in PP, these modules should have input ports for changing the speed, reversing the direction, setting the channel, and varying the sensitivity, and output ports for informing controller about the results of those functions. The ports and functions of these modules are based on the API functions provided by ESPS. Finally, to get as close as possible to a true simulation, additional modules, namely the independent modules, should be implemented to give values to the sensor modules.

5.1.3 Virtual Prototyping

In VP, IP provides a flexible and reusable interface for communicating with MP or PP. Despite (Fig. 2) only depicting IP in one VP element, it is the aim of IP to be able to add additional VP (if necessary) for handling MP

and PP to achieve integrated prototyping. IP will act as the interface not only to the VP itself but also between many objects within VP modules (see Fig. 4.b). Retrieving an object's value could be triggered by the end user activating a button object, switch toggle, or rotary sensor. There are three scenarios for using IP according to this architecture. These scenarios arise from the viewpoint of integrated prototyping definition. First, IP simply maintains the communication between PP and VP. In this case, we just need IP based on ActiveX to be utilized. The second scenario employs IP with the JavaBeans component to join VP and MP. The third scenario occurs when we consider combining all tools in integrated prototyping.

In addition, because we have used RapidPlus for the testing environment, we can define a new UDO. We also employ it to control hardware (sensors and motors) and to control MP (sensor and motor modules). In the Component View section below, IP components and other objects will be described in detail.

5.2 IP Component View

In this section, we will describe how IP can extend the function of two independent programs, namely ESPS for PP and SystemC for MP, by linking with VP. The flexibility and portability of IP can be achieved by implementing with JavaBeans and ActiveX. In addition, some components in VP, particularly in RapidPlus, are also performed by creating a UDO for sensor and actuator components, called a Sensor Actuator Object (SAO).

5.2.1 JavaBeans

The JavaBeans components implemented in IP tools come from three main classes in the SCJLib (SCJBroker, SCJInput, and SCJOutput) [11]. To make IP flexible and reusable in many VP tools, we extend its ability by using the JavaBeans mechanism. <Table 1> shows the classes and their descriptions. As shown in <Table 1>, in addition to the three main classes, the JavaBeans components also need two Listener Classes (SCJInputListener and SCJOutputListener) and two Event Classes (SCJInputEvent and SCJOutputEvent). The last three classes are used as BeanInfo class (SCJInputBeanInfo, SCJOutputBeanInfo, and SCJBrokerBeanInfo).

In the realization of our idea, these classes must follow JavaBeans mechanism by implementing java.io.Serializable. The Serializable class allows applications and frameworks to save, store, and restore the bean state in a virtual machine, reliably and platform-independently. To register

these classes in the VP tools, JavaBeans facilitates a process called introspection, which describes the properties, events, and methods that a bean contains. A property that wishes to appear in the VP has to employ both the get and set methods by using the naming conventions.

Furthermore, a JavaBean that wants to be an event source has to add an addXXXListener and removeXXXListener, where XXX stands for the name of the event. Two classes, SCJInput and SCJBroker, will instance the object that called the event source, because they will generate event-to-event listeners. Therefore, SCJInput class has to insert addSCJInputListener and removeSCJInputListener to add and remove interested listeners. In this case, SCJBroker is the class that desires to register the event that is generated by the SCJInput class. On the other hand, following the same method, the SCJBroker class becomes the event source for the instance of the SCJOutput class. Finally, the event source has to add a method to ourbean that will notify all listeners by using the notifyinput() function.

To complete the JavaBeans components, two elements remain to be implemented. The first element is the BeanInfo class that will be used by VP to describe exactly the properties, events, names to display, property editors, icons, etc. All classes that want to produce BeanInfo have to extend SimpleBeanInfo class. We created three BeanInfo classes, for SCJBroker, SCJInput, and SCJOutput. Using the naming convention, the name of the class should precede BeanInfo, giving the class names SCJBrokerBeanInfo, SCJInputBeanInfo, and SCJOutputBeanInfo. The second element is a Manifest file. The VP can identify the JavaBeans components by reading the Manifest file. Among the classes, only three classes (SCJBroker, SCJInput, and SCJOutput) became JavaBeans. The classes that want to register as JavaBeans simply need to set a single code JavaBean: True below their class name.

5.2.2 ActiveX Component

ActiveX, a Microsoft technology used for developing reusable object-oriented software components [13], is used in IP as the socket mechanism for VP to transfer data to PP. It uses an asynchronous mechanism to transport data to PP, because PP has to run anytime, even without additional data from VP. For example, when a driver moves the car key to the start position, the engine will run automatically; without additional activities by the driver, the engine will still run and wait further instructions.

We composed various classes that can be packaged in the ActiveX component. The CRClientOCXApp is a subclass

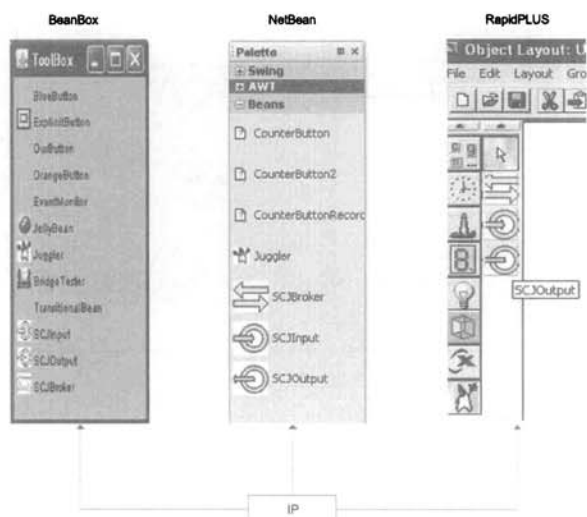
from ColeControlModule. The CWinApp is the main class, which has the function of broker for input and output data. Before the main class can apply its function, socket communication must be established by CConSocket class. Because ActiveX is a type of Windows component that is usually included in many interfaces, we need CSocketMgmt and CAnycSocket classes for linking interface modules. Finally, tCRClientOCXCtr will be placed to receive data from other objects in VP.

5.2.3 UDO and SAO Components

UDO is a group of objects that tie together for specific purpose. RapidPlus supports external objects that, once registered into RapidPlus, cannot be distinguished from other RapidPlus objects. All RapidPlus objects have properties, events, actions, and activities, so do UDO. (Fig. 5) illustrates the relation among objects, UDOs and parent application. Basically parent application consists of many objects such as default objects that are provided by RapidPlus, UDOs and other external object such as JavaBean, ActiveX, etc. These objects collaborate each other by exchange their properties, events and functions.

We follow UDO components from [14] where they create UDO to control sensors and actuators from PP to VP. In this case, we use the same structure for control the sensor and actuator from MP and VP. This UDO has properties with specific tasks for handling channel, direction, and speed of sensors and actuators. UDO, however, cannot transfer data to PP and MP directly, without a contribution from ActiveX and JavaBeans communication components. These components have the main function of maintaining a socket tunnel between VP and either PP or MP.

The UDO was designed as an SAO. This object derives from three classes: namely, SensorMotor, Motor, and Sensor. Once the sensor or motor module is defined in PP or MP, VP must also identify the module via an instance of the Motor or Sensor class. The Motor class holds the properties related to actuators: namely, channel number, direction, and speed. The Sensor class keeps channel number and direction. These classes are an aggregation from the SensorMotor class. There are several steps to implementing these classes in the real simulation. First, some objects from the classes should be added to the RapidPlus object layout. Other objects can also be dragged to the object layout, depending on the application to be constructed. After defining the entire object, a mode tree should be composed for changing the state of the objects. Finally, for transforming a state in the mode tree,



(Fig. 6) Usages of IP in different VP Tools

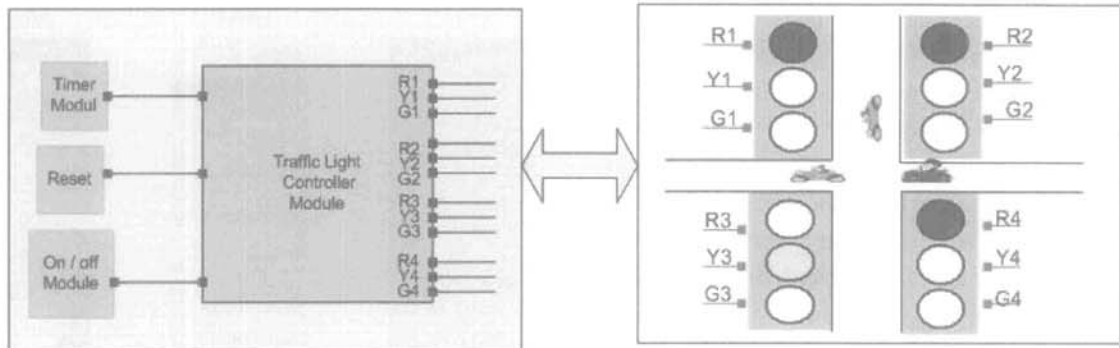
RapidPlus provides the triggers, actions, and activities. A trigger acts as an action for changing the state condition from one state to another state. The activities are applications performed on objects when a specific mode is activated, and the actions are similar to the activities, except for taking place only during transitions.

6. Flexibility and Reusability of IP

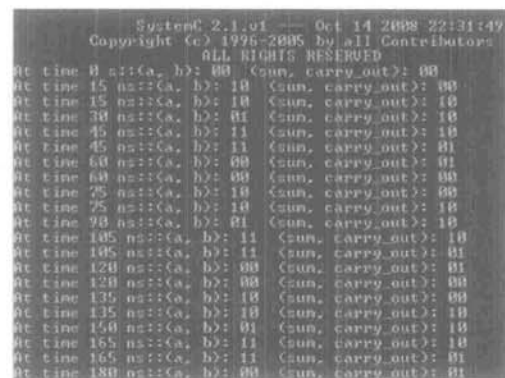
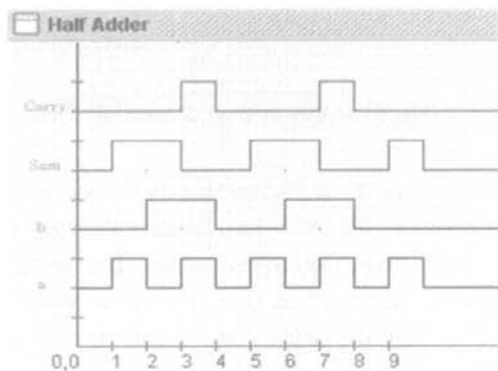
In this section, we demonstrate how flexibly IP can be reused for different embedded design tools. As shown in (Fig. 6), IP can be plugged into various VP tools such as RapidPlus, Netbean, and BeanBox. When IP is plugged into the VP, it requires three components namely, the Broker, Input, and Output. The components should be dragged to object layout in order to communicate to other GUI components that are provided by each tool. When they have been dragged to object layout, these components are categorized as non-graphic objects.

To test IP, we implemented a user-level prototype for a traffic light controller in MP and showed the VP by using RapidPlus (see Fig. 7.a). In MP, we define a controller module which has some input signals such as timer, reset and power. The timer module will send a signal based on a clock cycle which it could be ON (signal value is 1) and OFF (signal value is 0). The reset module is used for the controller to reset all output signals which are used by the LED. The module on/off has a function as a switch from electrical power. By using the SCJ library in MP, the value of signals out can be sent to GUI.

In GUI (in this case we implement in RapidPlus), instead of IP module (input, output and broker module),



(a)



(b)

(Fig. 7) (a) User-level Traffic Light Simulation, (b) Waveform Traces

some objects from default object should be added for showing how signal from MP can be visualized by end users. In this case, some lamp objects have been utilized in object layout. Those objects will receive value from MP by using IP module. For demonstrate as a real situation in traffic light, some car objects have been drawn.

Another example (Fig. 7.b) demonstrates how the Netbean can utilize IP to create waveforms. We compared the waveform when it runs in MP (text mode) with the graphical mode. The signals come from data A and data B, which have been generated by the driver module. In this case we will implement half adder combination logic, thus the output signals should sum $(A \wedge B)$ and carry $(A \& B)$.

7. Conclusions and Future Work

This work attempts to add an advantage for embedded system design tools by developing a flexible and reusable interface which can be embed to these tools. Thus, stand alone tools can communicate each other for creating a satisfying integration prototyping. By integrating these

tools, embedded system designers can deal with some embedded system design metrics such as time-to-market, low production cost, and can meet with user's functional requirement and trend. In this paper, the flexible and reusable interface will be added to VP, MP and PP by utilizing JavaBeans and ActiveX technologies. In addition, we have demonstrated that IP can be ported to various VP tools easily and flexibly. We think that IP can give a satisfaction support for VP designers when they choose their own VP tools.

Modeling the whole hardware components required in MP is very complex, thus as future work we plan to distribute them across PP and VP efficiently to mitigate such complexity. Then, we expect that hardware model designers will have more chances to focus on designing, developing, and maintaining a controller in MP rather than other hardware components.

References

[1] Vahid F, Givargis T. "Embedded System Design: A Unified Hardware/Software Introduction". Wiley, 2002.

- [2] Sukmana TS, Lee JB, Rim KW, Nam YJ. Extending Capability of SCJLib with JavaBeans for Integrating between SystemC and GUI Tools. In Proc of 7th APIS, 2008; 465-468.
- [3] Hartmann B, Klemmer SR, Bernsten M. Reflective Physical Prototyping through Integrated Design, Test, and Analysis. In Proc. DIS, PA, 2007.
- [4] ESPS Tools Page. www.artssystem.co.kr/product.php
- [5] System C Community Website. <http://www.systemc.org>
- [6] RapidPlus Development Tools Page. <http://www.sky-mobile-media.com/main.php?func=devtools>
- [7] SystemC Plug-in for Eclipse. <http://www.fzi.de/sim/sc-visualisation-plugin.html>
- [8] NetBeans Tools Page. <http://www.netbeans.org/>
- [9] ReidM, Charest L, TsiknovichA, Aboulhamid E, Bois G. Implementing a Graphical User Interface for SystemC. In Proc. HDLCon, USA, 2002; 224-231.
- [10] Rogin F, Genz C, Drechsler R, Rulke. SyCE: An Integrated Environment for System Design in SystemC. In The 16th IEEE International Workshop, 2005; 258-260.
- [11] Satria H, Kwon JB. Adding a Java GUI to SystemC Simulation for Virtual Prototyping of Embedded SystemC. In WSEAS Transactions on Computer Research, 2006; 1(1) 264.
- [12] Jong-Il Kim. A Design and Implementation of Real Time Monitoring and Control System for Shoes Production Process Automation. Master's Thesis, SMU, 2003.
- [13] ActiveX in <http://www.en.wikipedia.org/wiki/activex>.
- [14] Jung YJ, Lee JB, Kwon JB, Rim KW, Cho SY. An Embedded Integration Prototyping System based on Component Technique. LNCS, 2007, SpringerIn Berlin / Heidelberg; 4761:171-180.
- [15] Marquardt N, Greenberg S. Shared Phidgets: A Toolkit for Rapidly Prototyping Distributed Physical User Interfaces. In Proc. TEI, Louisiana ACM-Press, 2007.
- [16] Sukmana HT, Lee JB, Yang JS, Han HW, Jung YJ. The Components for Integrated Embedded System Prototyping. In IEEE International Symposium (ISCE), 2008.
- [17] Virtual Platforms Embedded Systems Emulators for Software Development Tools Page. <http://www.virtio.com/home.aspx>



Husni Teja Sukmana

e-mail : husniteja@yahoo.com
 1996년~2001년 University of Indonesia, Chemical Engineering (bachelor)
 2003년~2005년 Sun Moon University, computer science department (Master of Science)

2006년~현 재 Sun Moon University, computer science and engineering

관심분야: Embedded System, Integrated Prototyping



이 정 배

e-mail : jblee@sunmoon.ac.kr

1981년 2월 경북대학교 전자공학과 전산전공 (공학학사)

1983년 2월 경북대학교 전산전공(공학석사)

1995년 2월 한양대학교 전자공학과(공학박사)

1982년~1991년 한국전자통신연구원 선임 연구원

1991년~2002년 부산외국어대학교 컴퓨터공학과 교수

1996년~1997년 U.C.Irvine 객원교수

2002년~현 재 선문대학교 컴퓨터공학부 교수

관심분야: 임베디드 시스템, 실시간 시스템, 실시간통신 프로토콜



임 기 욱

e-mail : rim@sunmoon.ac.kr

1977년 인하대학교 전자공학과(공학학사)

1987년 한양대학교 컴퓨터공학과(공학석사)

1994년 인하대학교 컴퓨터공학부(공학박사)

2000년~현 재 선문대학교 컴퓨터공학부 교수

관심분야: 데이터베이스, 임베디드 시스템



황 영 섭

e-mail : young@sunmoon.ac.kr

1989년 서울대학교 컴퓨터공학과(공학학사)

1991년 포항공과대학교 컴퓨터공학과(공학석사)

1997년 포항공과대학교 컴퓨터공학부(공학박사)

1997년~2002년 한국전자통신연구원 선임연구원

2003년~현 재 선문대학교 컴퓨터공학부 교수

관심분야: 컴퓨터공학, 바이오공학



김 영 진

e-mail : youngkim@sunmoon.ac.kr
1997년 서울대학교 전기공학부(학사)
1999년 서울대학교 전기공학부(공학석사)
1999년~2003년 한국전자통신연구원 연구원
2003년~2008년 서울대학교 전기·컴퓨터공
학부(공학박사)

2008년~현 재 선문대학교 컴퓨터공학부 전임강사
관심분야: 임베디드 시스템, 저전력 소프트웨어 기법, 모바일 저장
장치 시스템 및 기법, 성능 및 전력 분석 도구



안 성 순

e-mail : ssAhn83@gmail.com
2008년 9월~현 재 선문대학교 전자계산
학과 석사과정
관심분야: 임베디드 시스템, 3차원기반 프로
토타이핑