New Educational Approach to Direct Experience through Embedded System Design and Implementation

Kenji Ohmori, Xiao Fan, and Ryo Mizutani
Computer and Information Sciences, Hosei University, Tokyo 184-8584, Japan
Email: ohmori@hosei.ac.jp, {xiao.fan.6b, ryo.mizutani.5g}@stu.hosei.ac.jp

***Abstract*—Direct experience is necessary in the information age, where virtual and indirect experiences are becoming increasingly common. Designing and implementing embedded systems give innovative, effective, and importantly, direct experience to students of computer science and information technology. Communicating sequential processes (CSP), which are effectively implemented by an event-driven and multi-thread processor, have an advantage in understanding the design and implementation of an embedded system because design concepts are implemented in a natural way. We propose a new educational approach, in which a formal development method called incrementally modular abstraction hierarchy (IMAH) is employed. As an example, a line-tracing car is designed and implemented using IMAH.**

***Index Terms*—embedded systems, communicating sequential processes, event-driven and multi-thread processor, incrementally modular abstraction hierarchy**

I. Introduction

In the information age, indirect and virtual experiences are useful, efficient, and economical educational tools. The social networking service makes it possible for people to communicate with one another over the Internet without actual face-to-face encounters. Moreover, Internet education allows teachers to teach their students without ever meeting them in person. Hardware experiments are often carried out by simulation without handling physical components. Nevertheless, indirect and virtual experiences are simulated; therefore, the need for direct experience becomes invaluable.

In particular, students of computer science and information technology have little chance to design and implement real-world systems because economical and user-friendly software simulators have been developed extensively, thus, denying them the unique satisfaction of implementing a real system while encountering new challenges. Therefore, it is important to provide courses that can provide students a direct experience of designing and implementing a system by handling physical components.

Manuscript received June 24, 2013; revised August 22, 2013.

We propose a new educational approach for designing and implementing embedded systems. An embedded system includes software and hardware designs; inputs, output, and control systems; combined implementation of hardware and software components; soldering and programming; testing and verification; and adjustment and factoring.

A LEGO NXT robot [1], which provides an environment of developing a robot using LEGO components and a programmable computer, known as the NXT Intelligent Brick, is a useful tool for teenagers. For students of computer science and information engineering, the LEGO robot is neither sophisticated nor instructive. Although visual software blocks allow the students to operate their robots, they are too simple for use in the study of programming or controlling of embedded systems. As the Intelligent Brick is equipped with a conventional computer, the students can develop their custom robots using C programming language. It is difficult for them to understand how their programs work within the whole system because the main environment is controlled by the operating system. Their programs are only part of the operating system, which hides the main behavior of the system.

The embedded system is characterized by inputs, outputs, and a controller. An input introduces an event into the system. By receiving the event, the controller computes its function and sends a response as an output. It is important that event handling be implemented in a natural way in a system. Interruption is used in the conventional computer so that processes running on the computer are performed sequentially. In the Intelligent Brick, event handling works as a sequence of complicated operations with interruptions, and is not implemented in a natural way. It is not easy to understand, and causes serious faults as it is sometimes used mistakenly.

Thus, instead of using the Intelligent Brick, we propose to use an event-driven and multi-thread processor. In this study, an XMOS processor [2] is used. The event-driven and multi-thread processor allows a designer to utilize threads, which are divided into three types in our design. The first type receives inputs from input devices such as sensors. The second type sends outputs from output devices such as motors. The third type controls the

doi: 10.12720/jiii.1.3.174-178

system by receiving messages from a first type thread and sending messages to a second type thread.

The threads in the system work in a concurrent and distributed way where a thread sends messages to another thread, which may, in turn, send messages to another thread. This behavior is described in a natural way using communicating sequential processes (CSP), which was proposed by Prof. Hoare [3].

As the students are not familiar with the proposed architecture using the event-driven and multi-thread processor and communicating sequential processes, they may encounter many challenges. To avoid unnecessary challenges, they are recommended to use a formal method when designing and implementing their systems. The formal method used in this approach is called an incrementally modular abstraction hierarchy (IMAH) [4-8]. IMAH is characterized by abstraction hierarchy, where the students proceed to design and implement their systems by ascending or descending abstraction levels. At a higher level, an abstract model is designed. At a lower level, a concrete model is designed. The concrete model inherits a property of the abstract model. By descending the abstraction hierarchy, properties are incrementally added to a new concrete model. As a result, students can avoid the so-called combinatorial explosion, which occurs as the students have to select an appropriate combination among several choices.

In addition to the formal method, each student utilizes agile software development, where the most important part of the system is first designed and implemented. Next, the second most important part is designed and implemented, and combined with the first part. This process is repeatedly continued until the whole system is completed. Two parts are combined based on the mathematical concepts of pushouts and pullbacks, which also provide a formal way of designing a system.

II. Design Steps

A. Robot System Specification

At the first stage, students consider the kinds of robots to design, and the possible behavior of the robots, such as line-tracing cars, soccer-playing robots, room-clearer robots, or helicopters. The first step is enjoyable, innovative, and creative.

After the kinds of robots are decided, figures (i.e., mechanical, humanoid, or miniature) and equipment (i.e., sets of sensors and motors) of the robots are considered. The students may draw designs of their robots, too.

The hardware system of a robot is composed of input and output devices. The software system is composed of agents. Each agent is implemented as a thread, and provides a service for a device or the robot itself.

B. Assembling LEGO Blocks

The figures of the robots that the students determined at the first stage are completed by assembling LEGO blocks. When figures are original, this stage is performed by trial and error.

The innumerable combination of LEGO blocks causes combinatorial explosion, which should avoided when designing a system. Therefore, when designing hardware and software, IMAH, which is a formal method of designing and implementing a system while avoiding combinatorial explosion, is used. Nevertheless, experiencing combinatorial explosion in assembling LEGO blocks is a means for students to understand the usefulness of the formal method at the later stage.

C. Hardware Design and Implementation

After an XMOS processor, an event-driven and multi-thread processor, has been installed, the equipment consisting of input and output devices is installed to the robot. Most devices transmit digital signals using Inter-Integrated Circuit (I^2C), which is a multi-master serial single-ended computer bus. I^2C is composed of a pair of serial clock line (SCL) and serial data line (SDA).

In our design, each device is controlled by a separated agent. Two input-output lines are separately provided for each device. Usually, I^2C has the capability to accommodate multiple masters and multiple slaves. In our design, the function of I^2C is limited to accommodate only one master and one slave, which simplifies the design of communication between the processor and the device. The individual agent that receives inputs or sends outputs becomes the master of its I^2C, and the device that sends its data (inputs) to the agent or receives its data (outputs) from the agent becomes the slave. Therefore, the number of pairs of SCL and SDA is equal to one of the input and output devices. That is, two input-output pins of the processor are provided physically to each agent and connected to the corresponding pins of the device.

Using I^2C, an agent sends its device commands, initiated by which the agent sends data to the device or reads data from the device. As commands and data are transmitted via an I^2C digital interface, the students can easily connect devices to their robots once they understand the I^2C principle.

Some devices are equipped with analog interfaces. In such cases, these devices are connected through analog-to digital converters, against which the students struggle if they do not have enough knowledge about the analog technology.

D. Software Design and Implementation

Each agent is completed by installing program codes. An agent for an input device receives data from the input device and sends a message to the control agent. An agent for an output device receives a message from the control agent and sends it to the output device. The control agent receives a message from an input agent, performs its function, and sends a response to an output device.

The software system is designed using IMAH. IMAH is divided into two parts: abstract and concrete. In the abstract part, the sequences of events and state transition diagrams are designed. In the concrete part, communicating sequential processes and programming codes are implemented. We will describe it more precisely in the following section:

E. Verification and Testing

In our development method, the verification of the software system is carried out before testing. As processes are described in the form of communicating sequential processes, students can use verification tools for CSP, such as PAT 3. Once the software system is verified, the processes are translated into programming codes, which run on the XMOS processor. Programming language XC is provided to the XMOS processor. XC is an extension of C, and enhances capabilities of message communication and concurrent processes. By running the verified software system, the hardware system is finally verified.

III. Design Example

A. Line-tracing Car

We assume that a student wants to design and implement a line-tracing car that runs on the black line while avoiding barriers. When the car encounters a barrier, it turns around and goes the opposite way. In addition, the black line has branches, which the car may take when it encounters them.

Furthermore, we assume that the car is equipped with four devices: a line-sensor array that detects a line; an ultra-sonic sensor that detects a barrier; and two motors that rotate the right and left wheels.

Moreover, we assume the designed software system provides five services: controlling the car; detecting the black line; detecting barriers; rotating the left wheel; and rotating the right wheel.

B. Applying the IMAH Method

After assembling LEGO blocks and designing and implementing the hardware system, it is a stage for the students to design and implement the software system. The development of the software system is carried out using agile software developmen.

The software system of the line-tracing car provides five services. Among them, the service for controlling the car is considered as the most important one. It is further divided into three tasks: running the car; avoiding a barrier; and selecting a path when encountering a branch. Let us try to design the task of avoiding a barrier.

IMAH consists of the following seven abstraction levels:

- The homotopy level: The most fundamental shape of the structure of the developing system is defined by including the number of connected spaces and the fundamental group of each space. Hardware components and software agents in the developing system are treated as separate spaces.
- The set theoretical level: Each space is configured as a set. The functions of a hardware component or the services of a software agent become elements of its set.
- The topological space level: A topology is induced into each set. The set becomes a topological space, which gives a strong mathematical foundation when designing the system.
- The adjuncting space level: The static and dynamical behavior of the developing system is clarified so that relations among hardware components and software agents (separated spaces) are defined.
- The cellular space level: Realistic images of a hardware component or a software agent are clarified using CW complexes, in which hardware components or software agents are configured as n-dimensional entities. A CW complex represents a state transition diagram in a sophisticated way.
- The presentation level: This level is the starting point in traditional architecture and modeling. CSP is defined for the software agents and hardware components.
- The view level: Finally, program codes and logical circuits are obtained.

IMAH is divided into two parts: abstract and concrete. The abstract part includes five levels: ranging from the homotopy level to the cellular space level. The concrete part consists of the presentation level and the view level. The abstract part generates common models in hardware and software. In contrast, the concrete part creates specific models of hardware and software.

Along with IMAH, the task of avoiding a barrier is designed as follows:

At the homotopy level, it belongs to the fundamental group 0 since it consists of only one entity.

At the set theoretical level, the following set is obtained:

T_B = {*normal-state, barrier-state, on-black-line, off-black-line, barrier, non-barrier, rotate-right-wheel, inverse-rotate-right-wheel, rotate-left-wheel, inverse-rotate-left-wheel*}

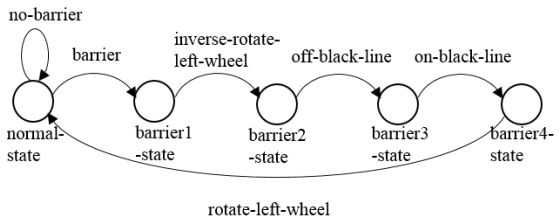

Figure 1. The task of avoiding a barrier is a sequence of events, which constitute a topological graph.

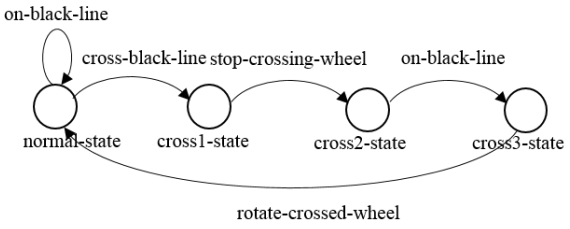

Figure 2. Sequence of events for the task of running the car.

In the above set, possible states and events are included. At the next level, some of them are unnecessary and further states and events may be added.

At the topological space level, the sequence of events is determined, which constitutes a topological graph. The task of avoiding a barrier is depicted in Fig. 1.

Similarly, the task of operating the car is obtained, as shown in Fig. 2.

At the adjuncting space level, the two tasks are attached together using an attaching function. Before describing the attaching function, we explain a pushout and a pullback, which are mathematical properties.

A pushout is defined as follows: Given two morphisms f: $A \rightarrow X$ and g: $A \rightarrow Y$, the pushout of the morphisms f and g consists of an object P and two morphisms i_1: $X \rightarrow P$ and i_2: $Y \rightarrow P$ such that $i_1 \circ f = i_2 \circ g$.

In the above definition, a morphism refers to a structure preserving mapping from one space to another. In set theory, a morphism is a function. In topology, it is a continuous function.

A pullback is defined as follows: Suppose that there are two morphisms f: $X \rightarrow Z$ and g: $Y \rightarrow$ Z. The pullback of the morphisms f and g consists of an object P and two morphisms p_1: $P \rightarrow X$ and p_2: $P \rightarrow$ Y such that $f \circ p_1 = g \circ p_2$.

Logical-and is an example of a pushout, and logical-or is one of a pullback: the commutative diagram is depicted in Fig. 3. As the pushout joins two spaces into one space, it is utilized as a bottom-up approach, which assembles components for a system. In contrast, a pullback is utilized as a top-down approach.

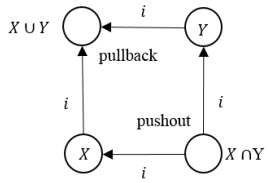

Figure 3. Logical-or and logical-and are examples of a pushout and a pullback, respectively

The attaching function is defined as follows: Suppose X is a topological space, which is attached by another topological space Y, then,

$$Y_f = Y \sqcup_f X = Y \sqcup X / \sim$$

is an attaching space obtained by attaching Y to X by an attaching map f (or by identifying each point $y \in Y_0 \mid Y_0 \subseteq Y$ with its image $f(y) \in X$ by a continuous map f). $\sqcup$ denotes a disjoint union. The attaching map f is a continuous map such that f: $Y_0 \rightarrow X$, where $Y_0 \subseteq Y$. Thus, the attaching space is a case of quotient spaces:

$$Y \sqcup X / \sim = Y \sqcup_f X = Y \sqcup X / (y \sim f(y) \mid \forall y \in Y_0).$$

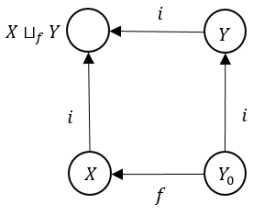

Figure 4. The attaching function is a pushout.

The identification map g, in this case, is

$$g: Y \sqcup X \rightarrow Y \sqcup_f X = Y_f = Y \sqcup X / \sim = (Y \sqcup X - Y_0) \sqcup Y_0.$$

A commutative diagram of the attaching map is shown in Fig. 4. As the attaching function is a pushout, it is utilized as a bottom-up approach. In Fig. 4, the two separated spaces that share parts of them are combined in one space by attaching sharing spaces.

Using the attaching map, the two tasks are attached together as shown in Fig. 5. The two tasks share the states *normal-state*, which are combined into one space. At the states *normal-state*, one task has a cyclic event *no-barrier* and the other task does *on-black-line*; these events are combined as an event *no-barrier* ∩ *on-black-line*.

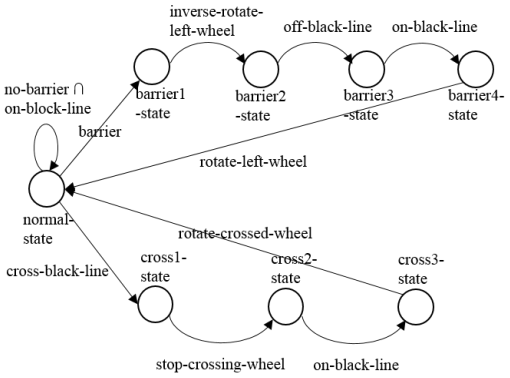

Figure 5. Two tasks are attached together by the attaching function

After the task of selecting a path when encountering a branch is added, a Meely machine, which is transformed from the sequence of events, is obtained as shown in Fig. 6. The Meely machine, which is a state transition diagram, belongs to the cellular space level. As the cellular space has dimensions, the nodes and the links are represented as 0-dimensional and 1-dimensional entities.

Four other services are designed similarly. These services are represented as independent state transition diagrams.

At the cellular space level, each service is represented by a Meely machine so that it is possible to implement the service not only by means of hardware but also by means of software. In case of hardware, each service becomes an independent sequential logical circuit. As services interact with each other, their sequential logical circuits also communicate with each other. In case of software, the services become processes (or threads), which communicate with each other.

At the presentation level, processes are obtained from the state transition diagrams. The following process C is obtained for the service of controlling the car, whose state transition diagram is shown in Figure 5:

C = (*no-barrier* ∩ *on-black-line* → C) [] (*barrier* → *inverse-rotate-left-wheel* →…→C) [] (*cross-black-line* → *stop-crossing -wheel* →…→C) [] (*in-black-line* → *cross-black-line* →…→C).

The service of detecting barriers is described by the following process B. The ultrasonic sensor sends a message indicating whether sound is reflected or not. By receiving it, the sensor sends it to the service of controlling the car.

B = (*reflection* → *barrier* → B) [] (*no-reflection* → *barrier* →B).

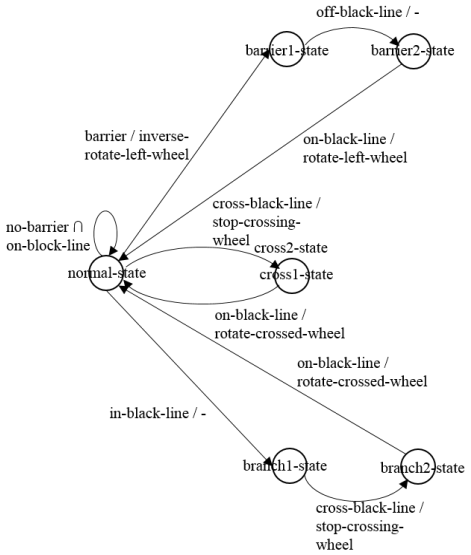

Figure 6. The Meely machine, which is a state transition diagram, is obtained at the cellular space level. It can be realized as a sequential logical circuit or CSP

The other services are also obtained in a similar way.

The software system of the line-tracing car is represented as concurrent processes:

$$SoftwareSystem = C \parallel B \parallel \ldots$$

At the view level, processes are transformed into programming codes. The car in Fig. 7 is a line-tracing car designed and implemented by one of our students. A line-sensor array, an ultra-sonic sensor, a color sensor, and two motors are installed on the car.

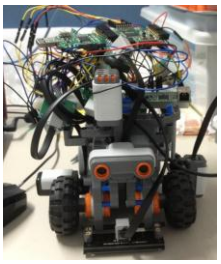

Figure 7. An implemented line-tracing car.

IV. Conclusion

A new approach to experiencing innovative study for students of computer science and information technology was proposed in this study. A line-tracing car was designed and implemented using the formal method IMAH: the car was designed by descending abstraction hierarchy, that is, by adding new properties linearly. CSP and XMOS made it possible to design and implement its hardware and software easily since each service was realized as an individual thread and an event was performed as message passing.

Our master's students designed and implemented their original LEGO robots. In spite of being unfamiliar with the setup, they thoroughly enjoyed their innovative experiences by finding original solutions when solving problems. In particular, they were thrilled to design and operate their own cars.

References

[1] *NXT User Guide*, LEGO Mindstorms education, 2006.
[2] D. May, *The XMOS XS1 Architecture*; Bristol: XMOS, 2009.
[3] C. A. R. Hoare, *Communicating Sequential Processes*. New Jersey: Prentice Hall, 1985
[4] K. Ohmori and T. L Kunii, "Pi-Calculus modeling for cyberworlds systems using the fibration and cofibration duality," in *Proc. International Conference on Cyberworlds*, 2008, pp. 363–370.
[5] K. Ohmori and T. L Kunii, "A formal methodology for developing enterprise systems procedurally: Homotopy, pi-calculus and event-driven programs," in *Proc. International Conference on Cyberworlds*, 2010, pp. 223–230.
[6] K. Ohmori and T. L. Kunii, "Visualization of joinery using homotopy theory and attaching maps," *Transactions on Computational Science XVI, Lecture Notes in Computer Science*, vol. 7380, pp. 95–114, 2012.
[7] K. Ohmori and T. L. Kunii, "Mathematical foundations for designing 3-dimensional sketch book," *Transactions on Computational Science XVI, Lecture Notes in Computer Science*, vol. 7848, pp.41–60, 2013.
[8] K. Ohmori, "Lego robot design using incrementally modular abstraction hierarchy," in *Proc. Eighth IASTED International Conference on Advances in Computer Science*, 2013, pp. 407–419.

Kenji Ohmori was born at Anjo, Aichi, Japan in 1945. He received a BS in Mathematical Engineering and a PhD in Information Engineering from the University of Tokyo in 1969 and 1984, respectively, and an MS in EECS from the University of California, Berkeley, in 1972. He is currently Professor of the Faculty of Computer and Information Sciences at Hosei University. He was the founding dean of the Faculty of Computer and Information Sciences from 2000 to 2004. Before joining Hosei University in 1985 as a professor of the Engineering Faculty, he was with the Central Research Laboratory of NEC, where he studied architecture of multiprocessor systems and object oriented languages. His current research area is a design method based on algebraic topology.

Prof. Dr. Ohmori is members ACM, the Computer Society of IEEE, IEICE and IPSJ. He received a Best Paper Award in 1985 from the Information Processing Society of Japan.

Xiao Fan was born at Suzhou, JiangSu, China in 1988. He received a BS in Electronic Information Science and Technology from University of Mining and Technology of China and an MS in Computer Science from Hosei University in Tokyo, Japan in 2013. He is currently Master Student of University of Science and Technology of China. He is now persuing his graduate study on embedded system design.

Ryo Mizutani was born at Ohta, Tokyo, Japan in 1990. He received a BS in Computer Science from Hosei University in 2011. He is currently Master Student of Graduate School of Computer and Information Sciences at Hosei University. He studies design and implementation of embedded systems based on CSP, and complex (in particular, traffic) system simulation using Netlogo.