Ubiquitous Chip: a Rule-based I/O Control Device for Ubiquitous Computing

Tsutomu Terada[1], Masahiko Tsukamoto[1], Keisuke Hayakawa[2], Tomoki Yoshihisa[1], Yasue Kishino[1], Atsushi Kashitani[2], and Shojiro Nishio[1]

[1] Graduate School of Information Science and Technology, Osaka University, Japan
[2] Internet System Research Laboratories, NEC Corp., Japan

Abstract. In this paper, we propose a new framework for ubiquitous computing by rule-based, event-driven I/O (input/output) control devices. Our approach is flexible and autonomous because it employs a behavior-description language based on ECA (Event, Condition, Action) rules with simple I/O control functions. We have implemented a prototype ubiquitous device with connectors and several sensors to show the effectiveness of our approach.

1 Introduction

As a result of the development of computer software/hardware technologies, the processing power and storage capacity of personal computers are rapidly increasing. At the same time, technological advances are contributing to the continued miniaturization of computers and component devices, such as microchips, sensors, and wireless modules[5][9][10]. In the near future, these devices will be embedded into almost any artifact and provide various services to support human daily life. This computing style is called *ubiquitous computing*. In ubiquitous computing environments, we can acquire various services with multiple interconnected computers that are embedded everywhere[4][12][13].

These embedded computers should automatically perform information exchanges and physical actions in response to surrounding circumstances. Although these computers may have low processing power and small memory, they must have the flexibility to change their function dynamically. Consequently, we apply rule-based technologies to describe the behavior of these ubiquitous computers. In this paper, we propose a new style of computing with rule-based I/O (input/output) control devices for constructing ubiquitous computing environments. We call this device the *ubiquitous chip*. The remainder of this paper is organized as follows. Section 2 outlines rule-based ubiquitous computing, presents several related works, and describes the design of the behavior description language for the proposed devices. Section 3 explains the software/hardware architectures of the ubiquitous chip, and Section 4 describes the application development environments for the ubiquitous chip. Section 5 presents several examples of its application and Section 6 sets forth the conclusion and planned future work.

2 Rule-based Ubiquitous Computing

In conventional computing, a user operates systems with input devices such as a mouse and a keyboard, and acquires computational results with output devices such as a display and speakers. On the other hand, since embedded computers are essentially invisible and must work without these conventional input/output devices, they need to exchange information and perform physical actions automatically in response to surrounding circumstances. Here, we define the following three characteristics, which are requirements for ubiquitous computers:

1. **Autonomy**: computers work automatically without human operation
2. **Flexibility**: computers are applied to various purposes
3. **Organic cooperation**: complex behaviors are achieved by organic coordination with multiple computers

Most previous prototyps of ubiquitous computing environments did not completely fulfill these requirements. For example, although the devices of Aware Home Project[1] and the Active Badge system[11] realize important applications for ubiquitous computing, they have been developed for one special purpose and are not intended for reuse in other applications.

On the other hand, there are several projects to construct a common framework and device. Smart-Its[2] is small computing device that consists of two independent boards, a core board that consists mainly of processing and communication hardware and a sensor board containing a separate processing unit, various sensors, and actuators. Motes[3], MICA[7], and U-cube[6] are also small ubiquitous devices which are separated into two units; a core unit and other (sensor) units. These devices have enough flexibility because we can customize the system configurations by changing attached sensors and other devices. However, in these devices, since running programs are closely related to device configurations, we cannot change their functions or attached devices dynamically while the programs are running. Therefore, it is difficult to customize the behaviors of embedded devices in response to the user. Moreover, since applications are developed in a C-like programming language, it is difficult for public users to program or to customize applications. From this point of view, previous devices lack simplicity in programming and the flexibility/autonomy in terms of changing the functionality of a device dynamically.

To construct flexible, scalable, and easy exploitable ubiquitous computing environments, there is a need for a general device and an architecture that fulfills above three requirements. Consequently, to present our new device for ubiquitous computing, we employ a system design philosophy that consists of two characteristics: the separation between the I/O control and the attachments, and the rule-based approach.

As for the former one, the separation helps to achieve a flexible system configuration by changing attached devices (other devices in related works have the same advantage). Moreover, the proposed device plays the role of a network hub that receives signals from multiple devices such as sensors, and sends signals

to multiple devices such as actuators. We can develop the device with a low-processing-power chip because the most of primary responsibility of the device is limited to a circuit switching.

In regard to the latter one, we apply the rule-based (event-driven) principle to the behavior description of ubiquitous computers because a person generally comprehends an event in the real world as a causal relation (event and action). We use ECA rules for the event-driven programming language. An ECA rule consists of the following three parts:

EVENT(E): Occurring event
CONDITION(C): Conditions for executing actions
ACTION(A): Operations to be carried out

ECA rules have been used to describe the behaviors of active databases. An active database is a database system that carries out prescribed actions in response to a generated event inside/outside of the database[14]. Using ECA rules, we can achieve the following advantages:

- As a consequence of their simplicity, we can program applications easily and intuitively. Anyone can construct and change applications for various devices embedded everywhere to enrich daily-life.
- We can change a full/part of a program dynamically because applications are described as a set of ECA rules, and each rule is stored independently in the device. This characteristic also enables users to customize behaviors of devices easily in response to users' requests.
- ECA rules in our system are described as a short-bit string. Therefore, we can send ECA rules as a message to devices via the network.
- Since ECA rules can be processed with a step-by-step approach, we can process rules with high efficiency (see 3.2).

2.1 Language Design of an ECA Rule

In conventional active databases, database operations such as *SELECT*, *INSERT*, *DELETE*, and *UPDATE* are considered events in ECA rules. Further, active databases can carry out actions only concerning database operations. Since ubiquitous computers may have little processing power and small memory, we must simplify the language specification of ECA rules while maintaining the ability to fulfill various requirements in ubiquitous computing environments.

Consequently, we decided that ECA rules are almost always used for I/O control. In other words, the on/off states of an attached switch and inputs from a sensor are handled as "input signals." In addition, output operations such as ringing a buzzer and turning on an LED (Light Emitting Diode) are handled as "output signals." In this way, a ubiquitous chip works almost for I/O control, and various devices are attached to the ubiquitous chip, as shown in Figure 1. In this figure, a ubiquitous chip evaluates the input from these sensors and devices, and outputs signals to connected devices. Moreover, a ubiquitous chip features

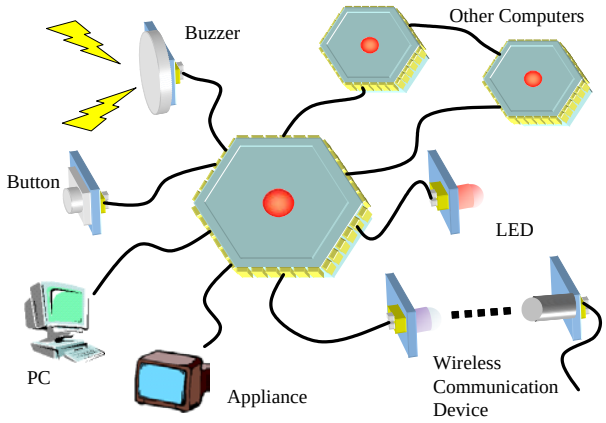

Fig. 1. Ubiquitous chip with connected devices

registers for storing the internal state and includes a flash memory for storing ECA rules. In addition, it has serial ports to communicate with other ubiquitous chips and has multiple timers for flexible timer functions.

Based on this principle, we define the events available in our system as shown in Table 1. There are two types of inputs to a ubiquitous chip: one is packet reception via a serial port, while the other is an input from a sensor via a normal port. When a ubiquitous chip receives a packet, the system generates a *RECEIVE* event. As for inputs from sensors, the system deals with the port state as conditions to execute actions. Therefore, the system allows ECA rules to be described without any events for executing actions by depending on the port state only.

We can specify the port and internal states in the condition part of the ECA rules. The system allows the description of multiple conditions and executes actions when all of the conditions are satisfied.

Actions provided by the system are shown in Table 2. The *OUTPUT* action changes the on/off state of each port to control connected devices such as a buzzer and an LED. The *OUTPUT_STATE* action changes the internal states, while the *TIMER_SET* action creates a new timer by specifying the interval of the timer and a once/repeat flag. The *SEND_MESSAGE* action sends a message that has a specific ID via a serial port. Since another ubiquitous chip generates a *RECEIVE* event on receiving this message, cooperation between ubiquitous chips is achieved using this event and the action. The *SEND_COMMAND* action sends the control commands shown in Table 3. The *HW_CONTROL* action controls the hardware. This action controls a general LED (described in Section 3.1), power saving functions, and a relay to switch serial ports. We can describe multiple actions with an ECA rule, and these actions are executed in a sequential order.

Table 1. Events

Name	Contents
RECEIVE	Data reception via the serial port
TIMER	Firing a timer
NONE	Evaluate conditions at all times

Table 2. Actions

Name	Contents
OUTPUT	On/off control of output ports
OUTPUT_STATE	On/off control of state variables
TIMER_SET	Setting a new timer
SEND_MESSAGE	Sending a message
SEND_COMMAND	Sending a control command
HW_CONTROL	Hardware control

Table 3. Commands for the SEND_COMMAND

Name	Contents
ADD_ECA	Adding a new ECA rule
DELETE_ECA	Deleting specific ECA rule(s)
REQUEST_ECA	Requesting a specific ECA rule

2.2 Binary Coding

ECA rules are translated to the binary format according to regulations shown in Figure 2. Basically, an ECA rule consumes four bytes (two bytes for Event and Condition, two bytes for Action). However, since the system allows the description of multiple conditions and multiple actions, we can describe a maximum of two conditions in a rule by switching on a multi-condition flag. We can also describe any number of actions by switching continue flags on. The format of each action varies by type. The second bit of the action discriminates the *OUTPUT/OUTPUT_STATE* actions from other actions. If the *OUTPUT/OUTPUT_STATE* actions are selected, the next 14 bits specify the output state. Otherwise, the next three bits specify the type of action and the remaining 11 bits are used to describe the action's content. For example, an ECA rule that states, "When INPUT1 and INPUT5 are ON and INPUT2 is OFF, the system fires a timer five seconds later," is translated as the four-byte binary format "0x91 0x13 0x49 0x15."

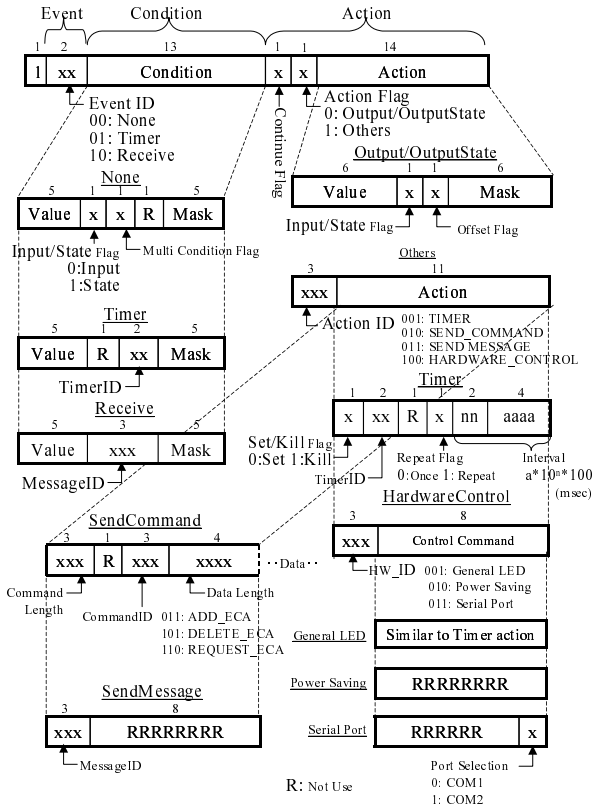

Fig. 2. Binary coding formula

3 Prototype of the Ubiquitous Chip

Based on the design described in the previous sections, we developed a prototype device of the ubiquitous chip. In the following sections, we explain the prototype device, focusing on the hardware architecture, the software architecture, and the attachments.

3.1 Hardware Architecture

The prototype device consists of two parts: the core-part (Figure 3, left) and the cloth-part (Figure 3, right). The core-part (34 mm in diameter) contains a microprocessor (PIC16F873), a power on/off switch, and a general LED that indicates the state of the ubiquitous chip. As shown in Figure 4, the core-part has six input ports (IN1-6), 12 output ports (OUT1a-6a, 1b-6b), six power-supply ports (VCC), and two serial ports (COM1-2). The cloth-part (59 mm in diameter) operates as a converter between the core-part and external sensors/devices.

Fig. 3. A prototype device

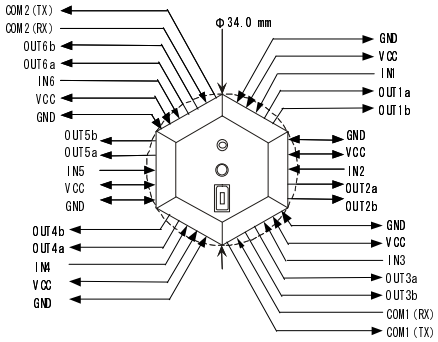

Fig. 4. I/O ports of a prototype device

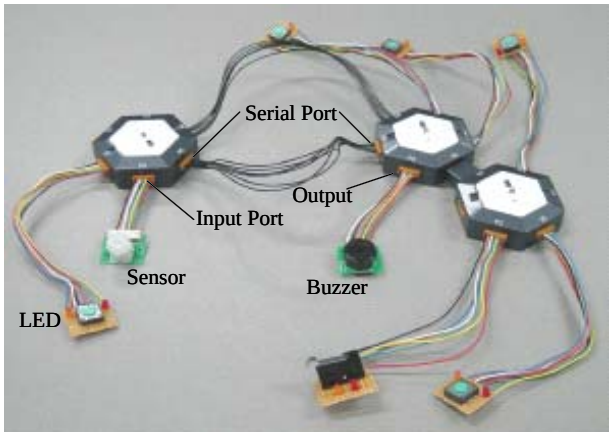

Fig. 5. An example showing connections

The cloth-part houses a Li-ion battery, connectors for attaching sensors/devices, and input/output ports as well as the core-part. Figure 5 shows a connection example between the prototype devices, sensors, and other devices. Table 4 shows the hardware specifications of the ubiquitous chip.

3.2 Software Architecture

Our prototype device uses the PIC16F873, which is a programmable RISC (Reduced Instruction Set Computer) type processor. Although this processor is cheap and easily programmed, its memory size is small and processing power is low. Hence, it is necessary to implement a rule-based system to improve efficiency. For example, our prototype stores ECA rules into the rule-base and processes the rules every two bytes. The microprocessor reads a condition part

Table 4. Hardware specifications

CPU	PIC16F873
Operating voltage	2.9 – 6.0V(3.3V)
Weight	11g (51g includes cloth)
Power resource	300mAh(4.2V)
Program memory	4000 words
RAM	192 bytes
EEP-ROM	128 bytes

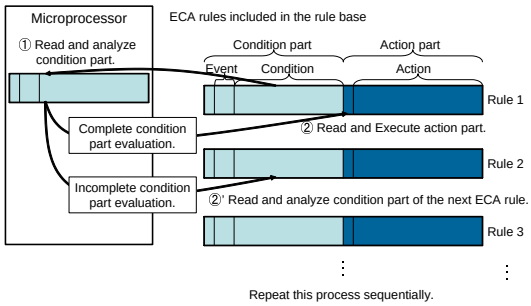

Fig. 6. A process for ECA Rules of the prototype

of the first rule from the rule base, and if the condition is satisfied, the microprocessor reads the action part; otherwise, it reads the next rule.

This processing formula is illustrated in Figure 6. Therefore, by reading just the necessary data from the rule base, the memory size required for rule processing is only two bytes.

Figure 7 shows a block diagram of the rule processing in the prototype. The rule-processing part reads and executes rules, and the communication-processing part checks the serial ports. When the system receives data from a serial port, it generates a *RECEIVE* event. Since the required memory size to process rules is small, even the PIC16F873 can drive the system easily.

3.3 Attachments

As shown in Figure 8, we have developed attachments for the ubiquitous chip. These attachments include various types of connectors for connecting one ubiquitous chip to another, sensors such as an infrared sensor, a pyroelectric sensor, an ultrasonic sensor, ans ultraviolet sensor, as humidity sensor, a light sensor, an acceleration sensor, a pressure sensor, a thermal sensor, a geomagnetic sensor, and an audio sensor. Moreover, we have developed input devices such as various types of buttons, switches, rheostats, and actuators such as a vibrator, buzzers, and motors.

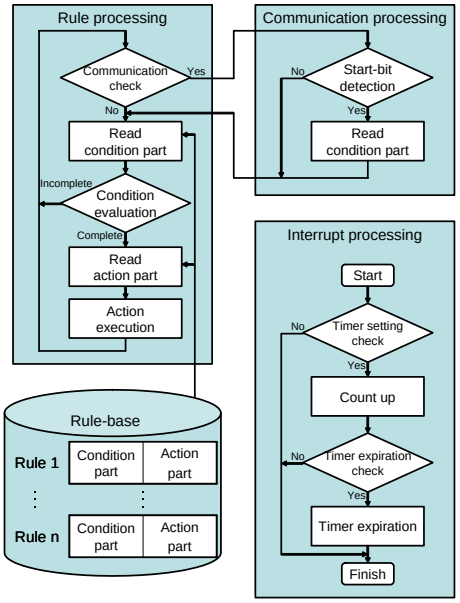

Fig. 7. A block diagram of the prototype

In addition to these attachments, we have developed devices that enhance the cloth-part of the ubiquitous chip, as shown in Figure 9. One is a battery box for AAA rechargeable batteries that have long battery-life and are easy to obtain. The other is a wireless unit that makes the serial port wireless. This device employs the unprocedure communication via RF. Using this device, a ubiquitous chip can communicate other chips in wireless.

4 Development Environment

Since an ECA rule is represented in the binary format, as shown in Figure 2, it is difficult to describe ECA rules directly[8]. Therefore, we implement two development tools: the rule editor and the rule writer. Using these tools on a PC, we can easily describe ECA rules and store them in ubiquitous chips. Moreover, we implement a ubiquitous chip emulator that achieves cooperation between PCs and ubiquitous chips.

4.1 ECA Rule Editor and ECA Rule Writer

Figure 10 shows a screen shot of the ECA rule editor. In this application, we can make ECA rules by specifying events, conditions, and actions graphically with easy mouse operations. When we specify a rule, bit sequences of this rule

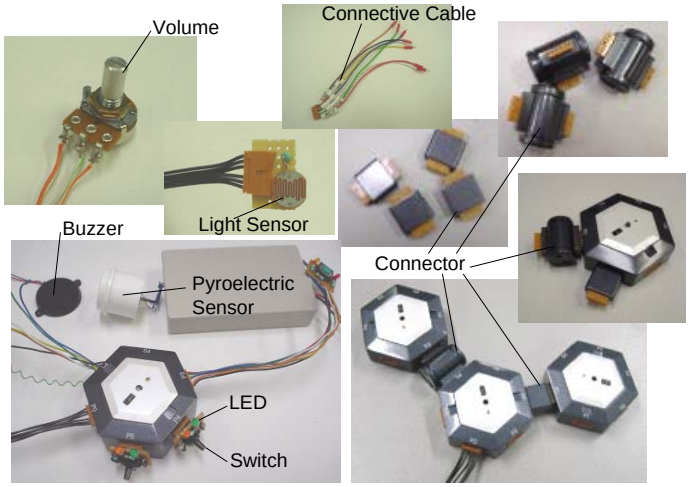

Fig. 8. The developed attachments

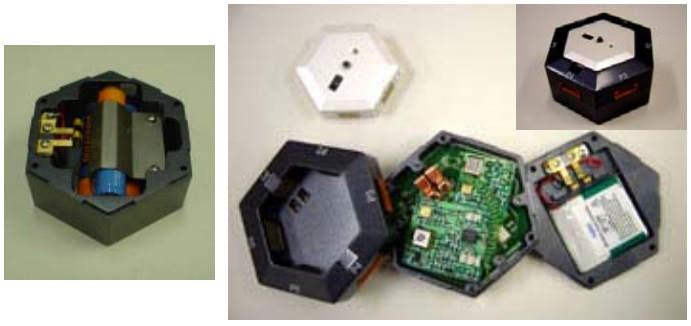

Fig. 9. The battery box (left) and the wireless unit (right)

are displayed on the binary display part and contents of this rule are displayed in natural language on the rule monitor part.

Figure 11 shows a screen shot of the ECA rule writer. This application reads, analyzes, and displays rules from a binary file, or a ubiquitous chip that is connected via a serial port. This application also writes displayed rules into ubiquitous chips via a serial port. We can add, delete, and modify ECA rules on ubiquitous chips freely using this application.

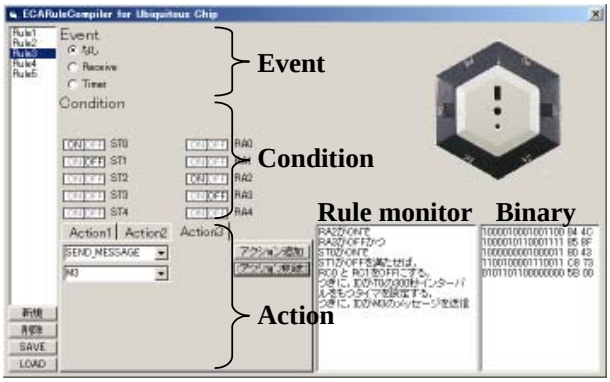

Fig. 10. A screen shot of the Rule Editor

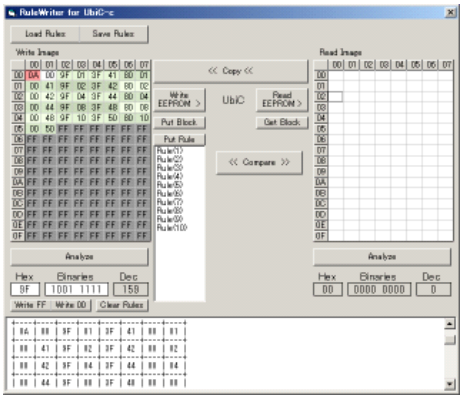

Fig. 11. A screen shot of the Rule Writer

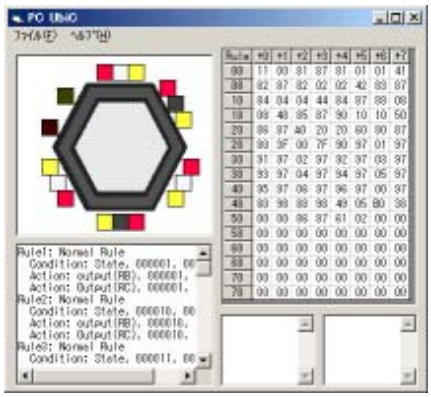

Fig. 12. A screen shot of the Emulator

4.2 Ubiquitous Chip Emulator

We also implement the ubiquitous chip emulator as shown in Figure 12. This emulator is used for simulations of rules/applications. It is also used to achieve cooperation between PCs and ubiquitous chips.

It is possible to use the same rules employed for ubiquitous chips to operate the emulator. The on/off states of the input/output ports on an emulated ubiquitous chip are represented by lighting tones, and we can control the input states by a click operation on each port. Figure 13 shows an example of a connection between a PC and a ubiquitous chip. In this way, a user can collect/utilize information from sensors connected to ubiquitous chips.

Moreover, since the emulator does not have any restriction on memory size, the emulator can not only simulate a ubiquitous chip, but also store ECA rules for another ubiquitous chip and deliver them to connected ubiquitous chips.

Fig. 13. Example of a connection between a PC and a ubiquitous chip

5 Applications

In this section, we show two examples of applications using ubiquitous chips. The first application is the room automation system illustrated in Figure 14. This application covers the following scenario:

- The user downloads the rule for the room control to his ubiquitous chip by reserving the room on his PC.
- When the user goes to the room and inserts his ubiquitous chip to the slot on the door, the door is automatically unlocked.
- When the door opens, the ubiquitous chip works to customize the room, such as by turning on the room light and the air-conditioner in cooperation with other ubiquitous chips embedded in the room.
- After once shutting the door, the system sounds a buzzer if the door is left open for more than one minute.
- The ubiquitous chip controls the air-conditioner with its temperature sensor.
- When the telephone rings, if there is a person in the room, it plays the message automatically. Otherwise, it records or transfers the message.

Table 5 shows the principals of ECA rules for these services. *RULE 1-5* are the downloaded rules for *ubiquitous chip A*. *RULE 6-8* and *RULE 9-11* are stored rules in *ubiquitous chip B* and *ubiquitous chip C*. *RULE 1* detects the door opening and sets a one-minute timer. *RULE 2* sounds the buzzer when the timer fires. If the door is closed within one minute, *RULE 3* resets the timer. *RULE 4* requests the door to unlock and *RULE 5* notifies *ubiquitous chip B* of the entry. *RULE 4* is activated when this message arrives. This rule resends the entry message to *ubiquitous chip C* and turns on the room light. *RULE 7* and *8* control the air conditioner using the information from the temperature sensor.

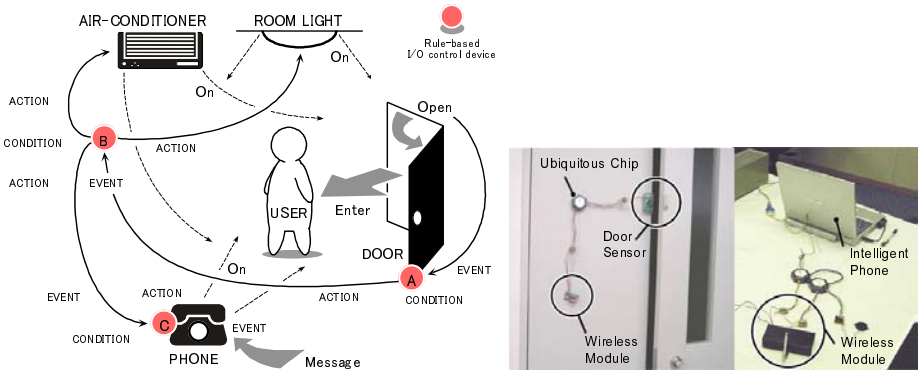

Fig. 14. An example of Application (1)

Table 5. The rule set for Example (1)

RULE 1	RULE 2	RULE 3
E:	E: TIMER	E:
C: I1=0, S1=0	C:	C: I1=1, S1=1
A: S1=1, TIMER(1min)	A: O1=1	A: S1=0, TIMER(0), O1=0
RULE 4	RULE 5	RULE 6
E:	E: RECEIVE(M2)	E: RECEIVE(M3)
C: I2=1	C:	C:
A: SEND(M1)	A: SEND(M3)	A: O2=1, SEND(M4), S2=1
RULE 7	RULE 8	RULE 9
E:	E:	E: RECEIVE(M4)
C: I3=1, S2=1	C: I3=0	C:
A: O3=1	A: O3=0	A: S3=1
RULE 10	RULE 11	
E:	E:	
C: I4=1, S3=1	C: I4=1, S3=0	
A: O4=1	A: O4=0	

I1: door sensor	O1: buzzer	S1: door flag	M1: unlock request
I2: key sensor	O2: room light	S2: enter flag (for chip B)	M2: unlock complete
I3: heat gauge	O3: air conditioner	S3: enter flag (for chip C)	M3: enter (to chip B)
I4: phone call	O4: phone		M4: enter (to chip C)

RULE 10 and *11* change the behavior of the telephone according to the state of the user's presence.

This type of application produces a "smart" space. We consider that most of the requirements in the smart space are simple, and easy realizable using our device. Of course, the flexibility of our device enables the smart space to change its functionality in response to different situations by using the dynamic change of rules.

The second example, illustrated in Figure 15, is the application for wearable computing environments. This application covers the following scenario:

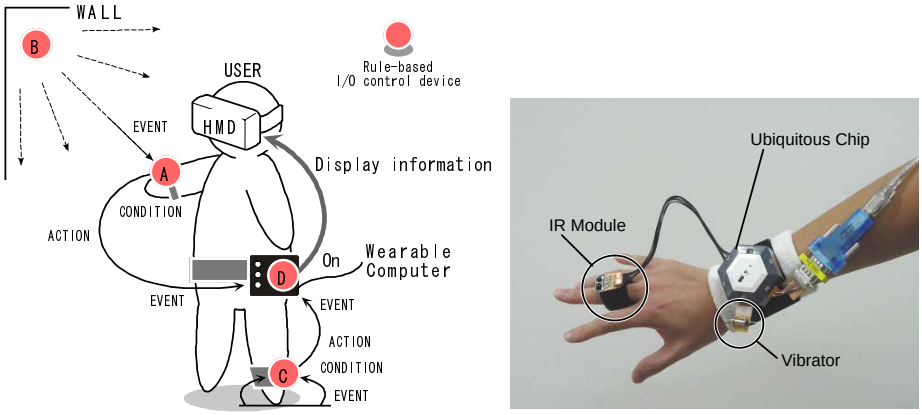

Fig. 15. An example of Application (2)

Table 6. The rule set for Example (2)

RULE 1	RULE 2	RULE 3
E: RECEIVE(M1)	E: RECEIVE(M3)	E:
C:	C:	C: I1=1
A: SEND(M2)	A: O1=1	A: SENDECA
RULE 4	**RULE 5**	**RULE 6**
E: TIMER	E:	E: RECEIVE(M2)
C:	C: I2=1, I3=1	C: I4=1
A: SEND(M1)	A: O2=1	A: SEND(M3), SEND(M4)

I1: button	O1: vibrator	M1: hello packet
I2: right foot	O2: notify of stay	M2: detection
I3: left foot		M3: vibration request
I4: connected with O2		M4: detection to wearable PC

- The user can acquire local information disseminated from a wall when he points his fingers at the wall. However, this acquirement is performed only when he stops walking.
- The user can change the wall light's blinking pattern by pushing a button on his wrist when he points his fingers at the wall light.

Table 6 shows the ECA rules for this application. *RULE 1-3* are stored in *ubiquitous chip A*, while *RULE 4, 5,* and *6* are stored in *ubiquitous chip B, C,* and *D*. *RULE 4* disseminates the beacon at a regular interval, and *RULE 5* sends the status of the user's feet to *ubiquitous chip D*. *RULE 1* detects the beacon from the wall (*ubiquitous chip B*) and notifies *ubiquitous chip D*. If *ubiquitous chip D* receives the notification and he stops walking, *RULE 6* sends the detection of the disseminated beacon to the wearable computer and *ubiquitous chip A*. *RULE 2* then turns on the vibrator. *RULE 3* sends the specific ECA rules to *ubiquitous*

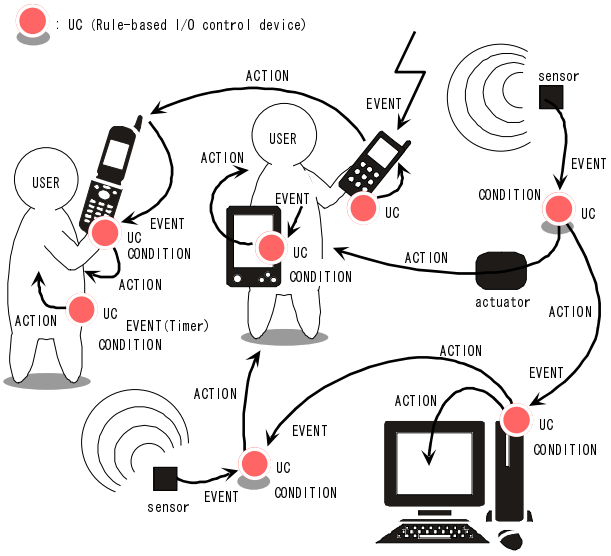

Fig. 16. A ubiquitous computing environment with rule-based I/O control devices

chip B via the infrared module when the user pushes the button on his wrist.

In this way, using our devices, we can construct a ubiquitous computing environment that integrates embedded computers, wearable computers, artefacts, and users. Such a system is illustrated in Figure 16.

6 Conclusion

In this paper, we have described the design and implementation of the ubiquitous chip, which is a rule-based I/O control device for ubiquitous computing. The two characteristics of our device, (1) the separation between I/O control and attachments and (2) the rule-based approach, work effectively to construct applications in ubiquitous computing environments. Moreover, we have presented examples of services that use our devices. These applications show the possibility of integration of ubiquitous computing environments and wearable computing environments by using ubiquitous chips.

In the future, we plan to construct ad-hoc networking functions with ubiquitous chips, and various other functions such as analog data processing. Moreover, in its current state of implementation of development environment, it is still rather difficult for users to develop/customize the system behaviors and to develop larger systems composed of hundreds of ubiquitous chips. Therefore, we plan to provide new development tools for end-user programming and larger systems.

References

1. G. D. Abowd, C. G. Atkeson, A. F. Bobick, I. A. Essa, B. Macintyre, E. D. Mynatt and T. E. Starner: The Future Computing Environments Group at the Georgia Institute of Technology, In Proc. of the 2000 Conference on Human Factors in Computing System.
2. M. Beigl, H. Gellersen: Smart-Its: An Embedded Platform for Smart Objects, Smart Objects Conference (sOc) 2003.
3. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister: System architecture directions for networked sensors, In Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 93–104, 2000.
4. L. Holmquist, F. Mattern, B. Schiele, et al: Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts, In Proc. of 3rd International Conference on Ubiquitous Computing (UbiComp 2001), pp. 116–122, 2001.
5. J. Kahn, R. Katz and K. Pister: Mobile Networking for Smart Dust, In Proc. of ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom99), pp.271–278, 1999.
6. Y. Kawahara, M. Minami, H. Morikawa, T. Aoyama: Design and Implementation of a Sensor Network Node for Ubiquitous Computing Environment, In Proc. of VTC2003-Fall 2003.
7. MICA, http://www.xbow.com/products/Wireless_Sensor_Networks.htm
8. M. Resnick and S. Ocko: LEGO/Logo: Learning Through and About Design, http://llk.media.mit.edu/papers/1991/11.html.s.
9. P. Saffo: Sensors: The Next Wave of Infotech Innovation, Ten-Year Forecast, Institute for the Future (IFTF), pp. 115–122, 1997.
10. K. Sakamura: TRON: Total Architecture, In Proc. of Architecture Workshop in Japan'84, pp. 41–50, 1984.
11. R. Want, A. Hopper, V. Falcao and J. Gibbons: The Active Badge Location System, ACM Transactions on Information Systems 10(1), pp. 91–102, 1992.
12. M. Weiser: The Computer for the Twenty-first Century, Scientific American, Vol. 265, No. 3, pp. 94–104, 1991.
13. P. Wellner, E. Machay, R. Gold, M. Weiser, et al: Computer-Augmented Environments: Back To The Real World, Communications of the ACM, Vol. 36, No. 7, pp. 24–97, 1993.
14. J. Widom and S. Ceri: ACTIVE DATABASE SYSTEMS, Morgan Kaufmann Publishers Inc, 1996.