# Application development environment for event-driven ubiquitous devices

Sagara, Ryouhei
Kishino, Yasue
Terada, Tsutomu
Nishio, Shojiro

(Citation)
International Journal of Pervasive Computing and Communications,5(2):87-103

(Issue Date)
2009

(Resource Type)
journal article

(Version)
Accepted Manuscript



(URL)
https://hdl.handle.net/20.500.14094/90002093

# Application Development Environment for Event-driven Ubiquitous Devices

Ryohei Sagara
*Graduate School of Information Science and Technology, Osaka University*
*1-5 Yamadaoka, Suita, Osaka 565-0871, Japan*

Yasue Kishino
*NTT Communication Science Laboratories*
*2-4 Hikaridai Seika-cho Soraku-gun,*
*Kyoto 619-0237, Japan*

Tsutomu Terada*
*Department of Electrical and Electronic Engineering,*
*Graduate School of Engineering, Kobe University*
*1-1 Rokkodai, Nada Ward, Kobe, Hyogo 657-8501, Japan*

Shojiro Nishio
*Graduate School of Information Science and Technology, Osaka University*
*1-5 Yamadaoka, Suita, Osaka 565-0871, Japan*
(Dated: March 14, 2008)

In this paper, we propose a new application development environment for ubiquitous computing environments. In ubiquitous computing environments, not only programmers but also general users come to develop/customize applications. Moreover, the style of application development is dramatically different from conventional environments. Therefore, we clarify the requirements for application development environments of ubiquitous computing environments. Then we design and implement a prototype of a development environment that fulfills these requirements. We also present an example of an application development to show the effectiveness of our approach.

## I. 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 [7][10][11]. In the near future, these devices will be embedded into almost any artefact and provide various services to support 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 [5][14][15].

Recently, many researches have been achieved such services [6][9][13]. Embedded input/output (I/O) control devices play an important role, such as acquiring information from the real world with sensor devices (i.e., if a trash box becomes full, the embedded LED is turned on), controlling various output devices in response to surrounding circumstances, and exchanging information with other embedded devices. Such I/O control devices can be embedded into almost all artefacts such as desks, chairs, blackboards, windows, and roads to provide such effective services as the following:

- A trash bin gives a caution signal when it reaches capacity;
- A desk light adjusts its brightness to suit the user;
- A buzzer sounds when a friend walks by the user;
- A pet collar records an animal's activities, such as movements and meeting other animals;
- A medicine chest cautions about the harmful effects of simultaneously ingesting multiple medicines;
- A refrigerator tracks the expiration date of foods stored in it;
- A chair shakes when the user becomes drowsy;
- A shopping cart calculates the total cost of products placed in it;
- A wallet lights up when its owner needs to find it.

From the characteristics of embedded I/O control devices, the style of constructing such applications for ubiquitous computing environments is greatly different from conventional software development. In concrete terms, application programmers have to manage many devices and their connection relationships, allocate programs to all distributed embedded devices, and check behaviors by actually operating devices and tackling the complicated debugging. Therefore, only programmers who have expert knowledge can construct applications. On the other

*Electronic address: tsutomu@eedept.kobe-u.ac.jp

hand, since these applications for ubiquitous computing environments highly depend on users' daily lives, they want to construct/customize applications by themselves.

In response to these requirements, we propose a new application development environment for ubiquitous computing environments that simulates the behaviors of multiple I/O control devices and helps general users create programs. Moreover, the most characteristic feature of the proposed development environment is the function for associating a virtual ubiquitous device in the development environment with a real ubiquitous device. Using this function, we can control a virtual device by operating the associated real ubiquitous device. Correspondingly, we can also control the actual ubiquitous device by operating the associated virtual ubiquitous device. This function enables users to test applications using actual sensors and to change part of an application by changing the virtual part of an application. Moreover, this development environment has a topology detection mechanism that collects the information of surrounding ubiquitous devices and displays it on a screen. We can customize applications by modifying the rules stored in these ubiquitous devices.

The remainder of this paper is organized as follows. Section II explains the requirements to application development environments for ubiquitous computing environments and presents related works. Section III outlines the ubiquitous chip, a rule-based I/O control device selected as an example of a ubiquitous device. Section IV describes the design of the proposed development environment, and Section V shows its evaluation. Section VI sets forth the conclusion and planned future works.

## II. APPLICATION DEVELOPMENT FOR UBIQUITOUS COMPUTING ENVIRONMENTS

As described in Section I, we assume that I/O control devices are embedded into almost all artefacts and that they cooperate with each other to provide various services, as shown in FIG. 1. In this research, we suppose an application development style where a user brings her PDA or laptop, which automatically recognizes ubiquitous devices in the room, and displays the current application status and their connection relationship. Using her terminal, she can customize applications graphically even though dozens of ubiquitous devices in the room and several applications are running together. This style enables the following application development scenarios.

- When a new piece of furniture, which includes an embedded ubiquitous device, is added to a room, the program on the device is modified in the room.
- When a user enters a room with her PDA and some ubiquitous devices, she incorporates these ubiquitous chips into previous applications to support her by buzzers and vibration motors. Her PDA discovers the network topology among ubiquitous devices, including additional ones, and modifies the stored program on the previous ubiquitous devices to collaborate with the new ones.
- When a user changes her routine, she adjusts the applications.
- A user employs actual ubiquitous devices to check the behavior of applications that run on the application development environment in PDAs.

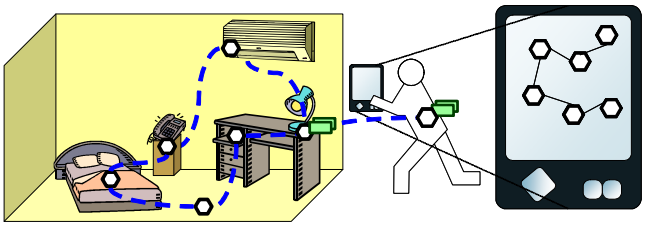


FIG. 1: Application development model

To fulfill these requirements and scenarios, application development environments have to help both programmers and general people intuitively develop/customize applications. Here we define the following characteristics, which are required for application development environments:

1. **Easy programming**
   In our assumed environments, not only programmers but also general users will develop/customize applications. Therefore, the development environment must provide easy programming interfaces.
2. **Detection of current status**
   When developing/customizing applications, users would like to know what kind of equipment and what kind of applications are in-service.
3. **Programming awareness of network connections between multiple ubiquitous devices**
   Since applications are realized by cooperation among multiple ubiquitous devices in our assumed environment, users would like to program in consideration of network connections among them.
4. **Debugging with cooperation among real/virtual environments**
   This function enable users to intuitively check the behavior of applications.

Many previous researches for constructing ubiquitous computing environments or small ubiquitous devices exist, and some systems have their own development environments. MICA [4] is a platform for wireless sensor networks, and Smart-Its [3] is a tiny computer system embedded into everyday objects. Although these development environments provide GUI, they employ C-like programming languages. Patch Panel [1], a system

that controls output devices in responce to the operation of input devices, provides a graphical user interface like a patch panel to configure applications. This mechanism enables general users to develop applications easily. MINDSTORMS [8] and ROBOT WORKS [2] also have their own development environments. Users can easily program applications by allocating blocks that represent conditions and operations.

Although these systems have their own features for easy developing, they do not fulfill all of the foregoing requirements. Smart-Its and MICA do not fulfill the 1st requiment because these devices are developed with C-like programming language, so it is difficult for general users to develop and customize applications. Moreover, Smart-Its, PatchPanel, MINDSTORMS, and ROBOT WORKS do not fulfill the 2nd, 3rd, and 4th requirements because they do not consider application developments on devices already embedded in environments. MICA satisfies the 2nd requirement because its development environment can detect network topology. However, we cannot utilize the topology information for application development. Moreover, it does not fulfill the 3rd and 4th requirements. Therefore, we need an application development environment that fulfills all requirements for general users to develop applications.

## III. UBIQUITOUS CHIP

We employ ubiquitous chip as the target ubiquitous device for constructing an actual development environment that fulfills the requirements described in the previous section. Ubiquitous chip is a rule-based I/O control device that we previously proposed in [12]. Since it achieves hardware/software flexibility that enables us to construct various types of applications, we expect that ubiquitous chips will be embedded into most artefacts and will be suitable for the ubiquitous device of our development environment. In this section, we outline the ubiquitous chip and its functions. Note that we use ubiquitous chip version 2, which possesses enhanced functions compared with version 1 described in [12].

### A. Overview

As shown in FIG. 2, a ubiquitous chip consists of a core part, which is the main unit, and a cloth part that has connectors and a rechargeable battery. It has five digital input ports, two serial communication ports (exclusive use), and a multi-purpose LED. FIG. 3 shows various attachments for ubiquitous chips such as sensors, input devices, and actuators. Using these attachments, we can flexibly change their hardware configurations.

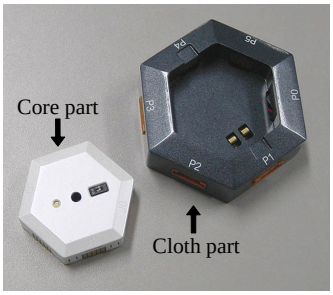



FIG. 2: Ubiquitous chip

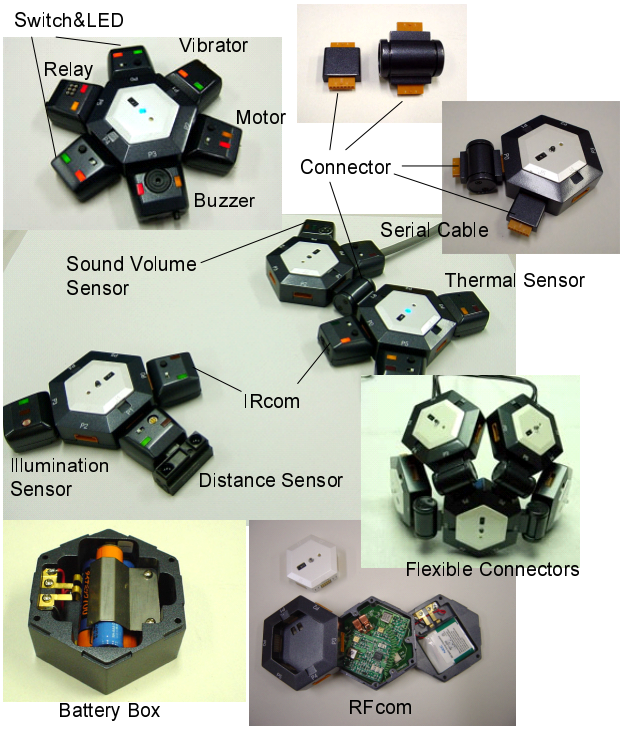



FIG. 3: Attachments for ubiquitous chip

### B. ECA Rule

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 the database [16]. 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

Using ECA rules, we can achieve the following advantages: (1) Due to their simplicity, we can program applications easily and intuitively. (2) We can change a full/part of a program dynamically because applications are described as a set of ECA rules. (3) Since ECA rules in our system are described as short-bit strings, we can

TABLE I: Events

| Name | Contents |
|---|---|
| INPUT_EVENT | Change of input ports' state |
| A_INPUT_EVENT | Change of analog input ports' state |
| STATE_EVENT | Change of internal variables |
| DEVICE_EVENT | Connection/disconnection of I/O device |
| TIMER_EXPIRE | Firing a timer |
| RECEIVE_MESSAGE | 8 types of message reception |
| RECEIVE_DATA | 1 byte data reception |
| RECEIVE_DEVICE | REPLY_DEVICE command reception |
| NONE | Evaluating conditions constantly |

TABLE II: Conditions

| Name | Contents |
|---|---|
| INPUT_CONDITION | On/off state of input ports |
| STATE_CONDITION | Value of internal variables |
| DEVICE_CONDITION | Connected device IDs |

TABLE III: Actions

| Name | Contents |
|---|---|
| OUTPUT | On/off control of output ports |
| OUTPUT_STATE | Setting a value of state variables |
| TIMER | Setting a new timer |
| SEND_MESSAGE | Sending a message |
| SEND_DATA | Sending 1 byte data |
| SEND_COMMAND | Sending a command |
| HW_CONTROL | Hardware control |
| COPY | Copy input/state to state/output |

TABLE IV: Commands

| Name | Contents |
|---|---|
| ADD_ECA | Adding a new ECA rule |
| DELETE_ECA | Deleting specific ECA rule(s) |
| ENABLE_ECA | Enabling specific ECA rule(s) |
| DISABLE_ECA | Disabling specific ECA rule(s) |
| DEMAND_DATA | Requesting data on EEPROM |
| REPLY_DATA | Sending data on EEPROM (reply to DEMAND_DATA) |
| CHECK_DEVICE | Confirming the existence of a specific device |
| REPLY_CHECK | Sending device information (reply to CHECK_DEVICE) |
| DEMAND_DEVICE | Requesting device ID connected to a specific port |
| REPLY_DEVICE | Sending list of device IDs (reply to DEMAND_DEVICE) |

send ECA rules as messages to devices by the network.

Using ECA rules, some requirements described in Section 1 are expressed as:

- A trash bin gives a signal when it reaches capacity.
  E: Trash bin detects new gabage
  C: Full?
  A: Gives visual/audio warning
- A buzzer sounds when a friend approaches.
  E: A person moves close to the user
  C: Acquaintance?
  A: A buzzer sounds
- A shopping cart calculates the total cost of products placed in it.
  E: A new product is placed in the car
  C: N/A
  A: Calculates cost and displays it

### C. Language Specification

Each rule has one byte of rule ID, and Tables I, II, and III show the lists of events, conditions, and actions that can be used in ubiquitous chips.

TABLE I shows a list of available events. There are two types of inputs for ubiquitous chips: one is packet reception by a serial port and the other is an input from a sensor by a normal port. In addition, we can describe the expiration of a timer, a change of device configuration, and a change of internal variables as an event. The situation of input ports, internal variables, and device configuration can be used in a condition, as shown in TABLE II, and TABLE III shows actions include output ports, sending message/data/command, changing internal variables, and setting timers.

### D. Communication Functions

The communication function is one of the most important features of ubiquitous chips. A ubiquitous chip can communicate with other chips by its serial communication ports. These ports transmit information by serial cables or wireless modules. We provide various wireless communication units for the ubiquitous chip, such as Infrared (IR), Radio Frequency (RF), and bluetooth unit. All of these units work with low battery consumption and have communication areas ranging from several centimeters to dozens of meters. These characteristics contribute to the flexible ubiquitous computing environments of ubiquitous chips.

**Actions and Commands for Communication**

A ubiquitous chip communicates with other ubiquitous chips by serial communication ports. We can use the SEND_MESSAGE action, the SEND_DATA action, and the SEND_COMMAND action as communication functions. The SEND_ MESSAGE action sends a message with a specific ID (0-127). The SEND_DATA action sends one byte data specified in the rule or input voltage of the analog port. The SEND_COMMAND action sends a command to remotely manage ECA rules stored

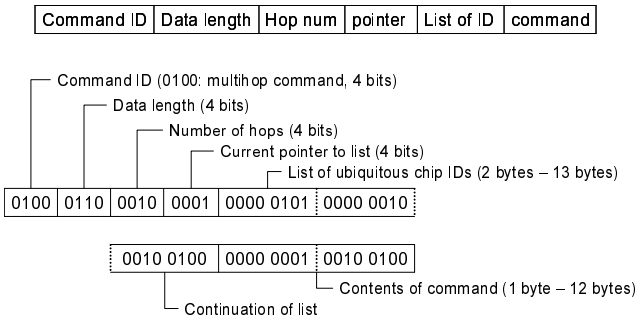



FIG. 4: Format of multihop commands

in ubiquitous chips. TABLE IV shows the lists of commands that can be sent by the SEND_COMMAND action. The DEMAND_DATA command demands the one byte data specified in the address of the memory in the ubiquitous chip. When a ubiquitous chip receives a DEMAND_DATA command, it returns the required data as a REPLY_DATA command.

**multihop Function**

The ubiquitous chip has two communication modes: *single-hop* and *multihop*. In the former, the ubiquitous chip processes a packet immediately after receiving it. On the other hand, in the latter, a packet is sent to its destination based on specified communication pathways. This method enables a ubiquitous chip to send a message to another ubiquitous chip located outside its direct communication area.

**Step 1:** The ubiquitous chip checks the ID of the current pointer. If it matches the ID of the ubiquitous chip or the broadcast ID and the ID of a ubiquitous chip not contained in the list, it goes to the next step. Otherwise, the ubiquitous chip ignores the packet.

**Step 2:** If the current pointer and the length of the list are not the same (the ubiquitous chip is not at the end of the pathway), the ubiquitous chip increments the current pointer and sends the packet to ubiquitous chips in the next hop. Otherwise, it goes to the next step. When the current pointer indicates the broadcast ID, the ubiquitous chip inserts its own ID into the list instead of the broadcast ID to avoid packet circulation.

**Step 3:** If the last ID of the ubiquitous chip matches its own ID or the broadcast ID, the ubiquitous chip processes the contents of the packet.

### E. I/O Device Detection

Another important function of ubiquitous chips is the device detection function that can locate attached devices, this function is added to ubiquitous chip version 2. For this function, each I/O device has a tiny CPU that sends its own device ID, which consists of device type and serial number, to the ubiquitous chip. Device ID can be used in the event, condition, and action parts of ECA rules.

In the previous ubiquitous chip version, programmers had to specify the port number to which the target I/O device connects to control the device; but the current version can specify target device by device ID or type such as "When a toggle switch is turned on, a blue LED lights up" and "When a buzzer is connected to the ubiquitous chip, some rules are enabled." We use the CHECK_DEVICE command to ask the ubiquitous chip to verify whether the specific I/O device is connected.

## IV. DESIGN OF DEVELOPMENT ENVIRONMENT

In this section, we describe the design and implementation of the proposed development environment that fulfills the requirements for development environments in ubiquitous computing environments, as described in Section II. The proposed development environment has the following characteristics:

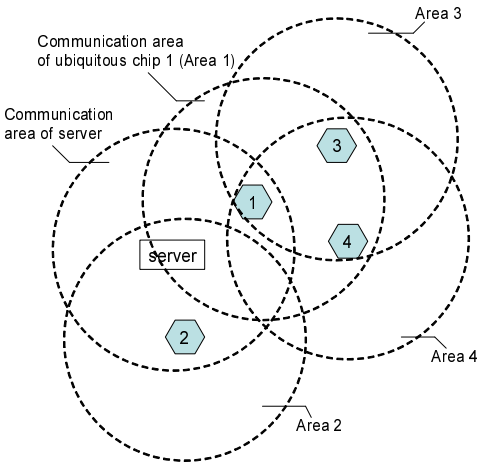
1. Network topology discovery based on a rule-based approach
2. Attached device detection from embedded ubiquitous chips
3. Simulation with virtual ubiquitous chips
4. Cooperation among real/virtual ubiquitous chips

The details of each are described below.

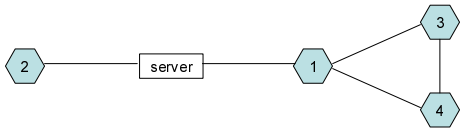
### A. Network topology discovery

The application development environment must grasp the network topology among real ubiquitous chips, especially when the user customizes applications that are in-service on real ubiquitous chips. Ubiquitous chips do not have functions for managing network topology because they have little computational power and memory for calculating topology and maintaining routing tables. It is unrealistic that to enhance the functions of ubiquitous chips only for the development environment. Therefore, we propose a topology discovery mechanism by only using ECA rules.

We realized a method to analyze communication topology among ubiquitous chips. Since the protocols are realized using ECA rules, they can be easily implemented on ubiquitous chips. On the other hand, at the discovery of network topology, our method requires a server who modifies the ECA rules in ubiquitous chips at the location, sends inquiry messages to them, receives reply messages from them, and analyzes their network topology.




(a): Allocation of ubiquitous chips in real world




(b): Mesh topology of ubiquitous chips

(c): Contents of each node

| ID | parent | children | other |
|---|---|---|---|
| server | - | 1, 2 | - |
| 1 | server | 3, 4 | - |
| 2 | server | - | - |
| 3 | 1 | - | 4 |
| 4 | 1 | - | 3 |

FIG. 5: Example of topology discovery

FIG. 5 shows an example of topology discovery. When ubiquitous chips are allocated, as illustrated in FIG. 5(a), the server manages ubiquitous chip IDs as a mesh topology (FIG. 5(b)). Each node of the mesh has information of its ID, the ID of its parent, the IDs of its children, and other IDs that can communicate with the ubiquitous chip (FIG. 5(c)).

In the following, we explain the details of the proposed method that can discover a mesh topology among ubiquitous chips.

TABLE V shows the ECA rules for the direct method. The procedure for discovering a network topology is as follows:

**Step 1** All ubiquitous chips have WAIT_UCID rules beforehand.

**Step 2** The server sends an ADD_ECA command to store REPLY_MESSAGE rules in the ubiquitous chips that can directly communicate with the server.

**Step 3** It sends the REQUEST_UCID message to them.

TABLE V: Rule set for topology discovery

| WAIT_UCID | |
|---|---|
| E: State 0=1 | |
| C: N/A | |
| A: State 0=0, T(ID×100ms) | |
| **REPLY_MESSAGE (2 rules)** | |
| E: RM(7) | E: Timer |
| C: N/A | C: N/A |
| A: State 0=1 | A: SM_M(6) |

T: Setting a timer for a proportional time to its ID
RM: RECEIVE_MESSAGE event
SM_M: SEND_MESSAGE (multihop mode) action
Message 7: REQUEST_UCID message
Message 6: REPLY_MESSAGE message
State 0: Flag for timer

**Step 4** It receives REPLY_UCID messages that are replies to the REQUEST_UCID message from them.

**Step 5** It records the IDs in replies and adds them to the mesh topology information.

After these operations, the server begins to discover ubiquitous chips that can communicate with the server by two-hop communication. The server can discover ubiquitous chips in the second hop, as in steps 2–5.

**Step 6** The server selects a ubiquitous chip as the *current* ubiquitous chip from those handled in the previous hop, which has the minimum ID.

**Step 7** The server sends the ADD_ECA command to store the REPLY_MESSAGE rules in the ubiquitous chips within a hop from the current ubiquitous chip. At this point, the server sends a multihop command with a multihop header, which includes the ID of the current ubiquitous chip as the first hop and the broadcast ID as the second hop.

**Step 8** It sends REQUEST_UCID message to these ubiquitous chips.

**Step 9** It receives REPLY_UCID messages from them.

**Step 10** It adds these IDs to the mesh topology. If an ID has already been in the topology, the server records the ID in the direct communication list for the current ubiquitous chip. Otherwise, the server records it as a child of the current ubiquitous chip.

**Step 11** It deletes the REPLY_MESSAGE rules stored in Step 7 to prevent sending superfluous REPLY_UCID messages.

**Step 12** If another ubiquitous chip has not been selected as the current ubiquitous chip, it is selected as the new current ubiquitous chip, and the procedure returns to Step 7.

**Step 13** The server selects a new current ubiquitous chip that has the minimum ID among the children of the previous hop. In the same way, the server repeats this operation over the third hop.

**Step 14** If the number of hops exceeds the limit or all ubiquitous chips are discovered, the server finishes the procedure.

### B. Attached device detection

The proposed development environment can grasp network topology by using the previously described method. On the other hand, catching the functionality of ubiquitous chips is also important for constructing applications. Therefore, we realize the detection of attached I/O devices for each ubiquitous chip by using ECA rules.

After acquiring to network topology, the server sends DEMAND_DEVICE queries to all ubiquitous chips to collect their attached device information.

### C. Simulation with virtual ubiquitous chips

Services in ubiquitous computing environments are realized through cooperation among multiple ubiquitous chips. In such situations, it is difficult for users to grasp the existing configurations and construct applications that consider the relationships among multiple ubiquitous chips. Therefore, our application development environment has a function that simulates multiple virtual ubiquitous chips that process their ECA rules in the same way as real ubiquitous chips.

FIG. 6 shows a screenshot of the development environment. In the proposed development environment, the behaviors of ubiquitous chips are simulated by virtual ubiquitous chips. A virtual ubiquitous chip is illustrated as a hexagon, and circles indicate I/O ports, serial ports, and a multi-purpose LED. One input port and two output ports are placed along each edge of the hexagon, as in a real ubiquitous chip. The state of the I/O ports and the multi-purpose LED are expressed by color differences. A user operates the virtual ubiquitous chip in the following ways:

- places multiple virtual ubiquitous chips in the simulation area
- toggles input ports
- checks the state of the output ports and the multi-purpose LED
- connects I/O and serial ports to other ubiquitous chips
- checks the value of the internal variables and the stored ECA rules

TABLE VI: Formula for creating control rules

| Original Rule | Control Rule |
|---|---|
| E: (Any event)<br>C: (Any condition)<br>A: OUTPUT | E: RECEIVE_DATA(00≤Data≤3F)<br>C: NONE<br>A: COPY(ReceivedData→Output) |
| E: (Any event)<br>C: (Any condition)<br>A: HW_CONTROL (V_LED=Off) | E: RECEIVE_DATA(Data=0x40)<br>C: NONE<br>A: HW_CONTROL(V_LED=off) |
| E: (Any event)<br>C: (Any condition)<br>A: HW_CONTROL (V_LED=On) | E: RECEIVE_DATA(Data=0x41)<br>C: NONE<br>A: HW_CONTROL(V_LED=On) |
| E: INPUT_EVENT<br>C: (Any condition)<br>A: (Any action) | E: INPUT_EVENT<br>C: NONE<br>A: SEND_DATA(Input) |
| E: A_INPUT_EVENT<br>C: (Any condition)<br>A: (Any action) | E: A_INPUT_EVENT<br>C: NONE<br>A: SEND_MESSAGE |
| E: DEVICE_EVENT<br>C: (Any condition)<br>A: (Any action) | E: DEVICE_EVENT<br>C: NONE<br>A: SEND_MESSAGE |

- adds new ECA rules using the ECA rule editor (FIG. 7)

Users simply use a mouse to achieve the above operations.

### D. Cooperation among real/virtual ubiquitous chips

The application development environment has a function for constructing applications through cooperation between virtual ubiquitous chips and real ubiquitous chips. It manages the state of a real ubiquitous chip identically as a virtual ubiquitous chip by linking their states. For example, as FIG. 8 shows, when a user pushes the button connected to the real ubiquitous chip, the input port of the associated virtual ubiquitous chip is turned on. Likewise, when the output port of the virtual ubiquitous chip is turned on, the output port of the real ubiquitous chip is also turned on.

This function achieves the following implementation styles:

**Case 1** A user customizes applications that are in-service on real ubiquitous chips.

**Case 2** A user checks the behaviors of real I/O devices at the final step of application development.

In the former case, cooperation is achieved as follows:

**Step 1** A user connects a real ubiquitous chip to a PC.

**Step 2** A user places a new virtual ubiquitous chip in the simulation area.

**Step 3** The application development environment reads the ECA rules stored in the real ubiquitous chip and adds them to the virtual ubiquitous chip.

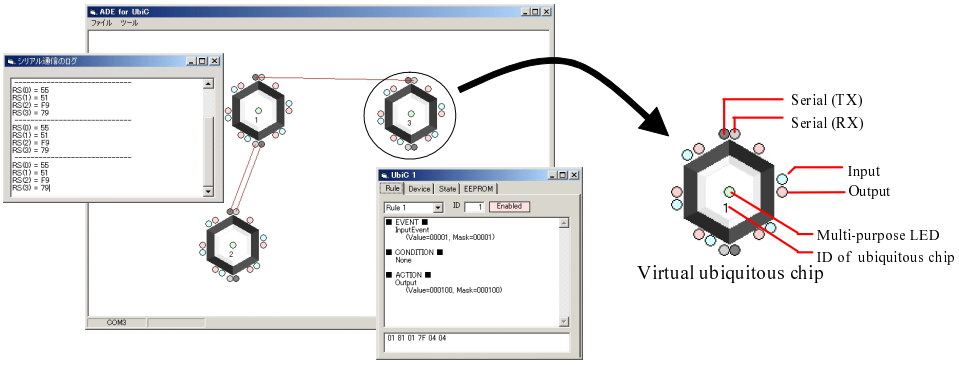



FIG. 6: Screenshot of application development environment

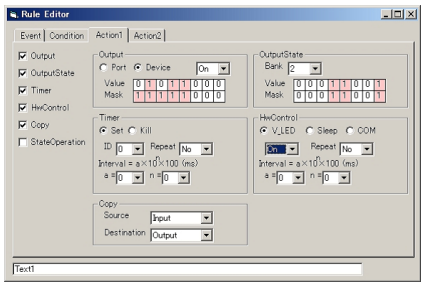


FIG. 7: ECA rule editor

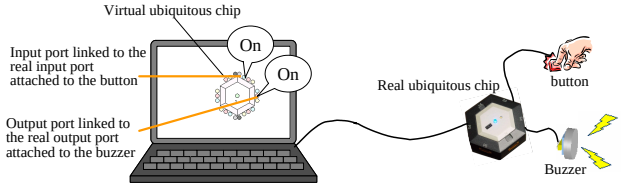



FIG. 8: Cooperation among real/virtual ubiquitous chips

**Step 4** The development environment sends a DELETE_ECA command to the real ubiquitous chip to delete all stored ECA rules, which prevents conflict among the ECA rules.

**Step 5** The development environment writes the rules on the real ubiquitous chip, which allows the real ubiquitous chip to behave identically as the virtual ubiquitous chip.

In the latter case, cooperation is realized by only performing Steps 4 and 5 of the above procedure.

TABLE VI shows the formula for creating control rules. When the state of a virtual output port or a versatile LED changes, the development environment sends one byte data to the associated real ubiquitous chip. After receiving the data, it changes the output port or versatile LED the same as the virtual ubiquitous chip. On the other hand, when the state of a real input port changes or I/O devices are connected/disconnected, the real ubiquitous chip sends a message or one byte data to the development environment. After receiving the data, it changes the state of the associated virtual input port or virtual I/O devices.

## V. EXAMPLE OF APPLICATION DEVELOPMENT

In this section, we give an example where a user customizes an application by using the proposed application development environment. The sample development senario is as follows:

- In a room, an application behaves as "when a user sits on a chair, and the desk lamp automatically turns on." A user only wants the desk lamp to turn on when is is not bright enough. Therefore she customizes the application by adding a ubiquitous chip and a illumination sensor.

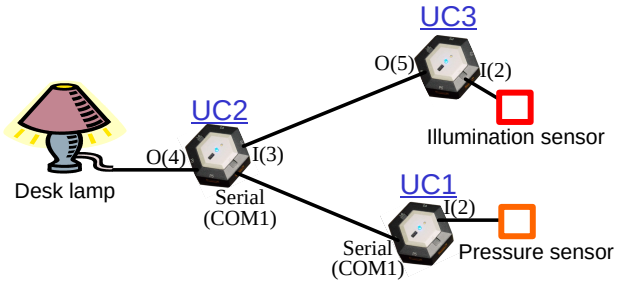
In this application, we use three ubiquitous chips: UC1, UC2, and UC3. UC1 is attached to the chair and has a pressure sensor that detects when the user is sitting. UC2 is attached to the desk and connected to the desk lamp in order to control it. UC3 is newly attached to the wall and has an illumination sensor. FIG. 9 shows the connection relationship of the ubiquitous chips.

The user customizes the application as follows:

1. When the user enters the room, the development environment collects the information of real ubiquitous chips in the room, and the virtual ubiquitous chips (UC1 and UC2), their communication link, and the I/O devices connected to them are displayed.

2. The user positions the virtual ubiquitous chip corresponding to UC3.

3. The user connects UC1 and UC3, as shown in FIG. 9.




I(2), I(3): Input ports
O(4), O(5): Output ports

FIG. 9: System structure of sample application

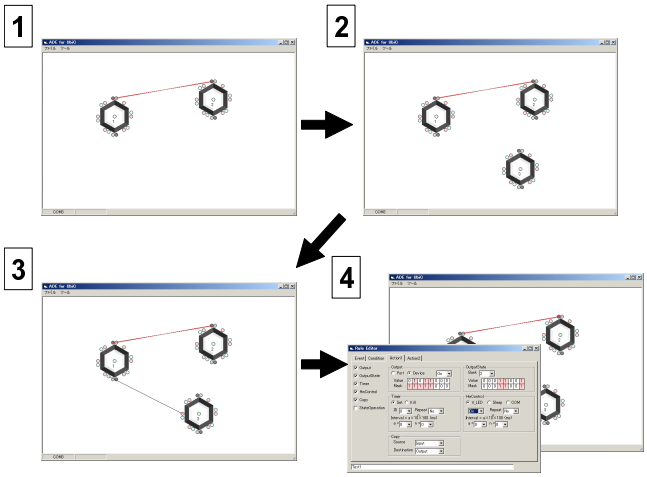



FIG. 10: Screenshots in application customization process

4. The user adds/customizes the ECA rules shown in TABLE VII using the ECA rule editor.

5. The user checks the behavior of the ubiquitous chips by toggling their input ports. If bugs are found, the user modifies the rules.

6. When the user wants to confirm the application behavior with a real illumination sensor, he connects a real ubiquitous chip to a PC and links UC3 and the real ubiquitous chip. In this case, the control rules shown in TABLE VII are automatically added to the real ubiquitous chip. The user changes the brightness of the room and checks the behavior.

7. After completing the application, she writes the ECA rules on the real UC3 and attaches it to the wall.

FIG. 10 shows screenshots of the development environment in the customization process.

Here, supposing that a user customizes applications without this development environment, it becomes difficult for the user. First, she has to detach the ubiquitous chips from the environment and edit their ECA rules by connecting them to the PC one at a time. It is hard to describe ECA rules without a rule complier because they must be translated to binary based on a complicated format. It is also difficult to develop applications without the information about the connection relationship between ubiquitous chips. Debugging applications using real devices is also difficult for many causes, such as a bug of the control program, and a communication error. In contrast, since the proposed development environment has various specialized functions to develop applications for ubiquitous computing environments, we can construct applications easily by using the development environment.

TABLE VII: Rule set for the sample application

| | | |
|---|---|---|
| Rules for UC1 (2 rules) | | |
| E: I(2)=On | E: I(2)=Off | |
| C: N/A | C: N/A | |
| A: SM(0) | A: SM(1) | |
| Rules for UC2 (4 rules) | | |
| E: RM(0) | E: RM(1) | E: I(3)=On |
| C: State 1=1 | C: N/A | C: N/A |
| A: O(4)=On | A: O(4)=Off | A: State 1=1 |
| E: I(3)=Off | | |
| C: N/A | | |
| A: State 1=0 | | |
| Rules for UC3 (2 rules) | | |
| E: I(2)=On | E: I(2)=Off | |
| C: N/A | C: N/A | |
| A: O(5)=On | A: O(5)=Off | |
| Control rule for UC3 | | |
| E: INPUT_EVENT | | |
| C: N/A | | |
| A: SEND_DATA(Input) | | |

I(2), I(3): Input port
O(4), O(5): Output port
RM: RECEIVE_MESSAGE event
SM: SEND_MESSAGE action

## VI. CONCLUSION

In this paper, we described the requirements of an application development environment for ubiquitous devices. Moreover, we described the design and implementation of an application development environment for ubiquitous chips. The proposed development environment provides various functions for users to intuitively develop/customize applications.

In the future, we plan to construct functions for developing large-scale applications, for automatically generating ECA rules using I/O devices. We also plan operational tests and further evaluation of the application development environment.

**Acknowledgments**

This research was partially supported by The 21st Century Center of Excellence Program "New Information Technologies for Building a Networked Symbiotic Environment" and Grant-in-Aid for Scientific Research (A)(17200006) from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

[1] R. Ballagas, A. Szybalski, and A. Fox: A. Patch Panel: Enabling Control-Flow Interoperability in Ubicomp Environments, in Proc. of the Second IEEE International Conference on Pervasive Computing and Communications, pp. 241–252, 2004.

[2] BANDAI: ROBOT WORKS, http://www.roboken.channel.or.jp/borg/.

[3] M. Beigl and H. Gellersen: Smart-Its: An Embedded Platform for Smart Objects, Smart Objects Conference (sOc), 2003.

[4] J. Hill and D. Culler: MICA: A Wireless Platform For Deeply Embedded Networks, IEEE Micro, Vol. 22, pp. 12–24, 2002.

[5] L. Holmquist, F. Mattern, B. Schiele, et al: Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts, in Proc. 3rd International Conference on Ubiquitous Computing (UbiComp 2001), pp. 116-122, 2001.

[6] C. Kidd, R. Orr, D. Abowd, et al: The Aware Home: A Living Laboratory for Ubiquitous Computing Research, In Proc. of the Second International Workshop on Cooperative Buildings (CoBuild'99) Position paper, 1999.

[7] 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.

[8] LEGO: MINDSTORMS, http://mindstorms.lego.com/.

[9] T. Okoshi, et al: Smart Space Laboratory Project: Toward the Next Generation Computing Environment, in Proc. of IEEE Third Workshop on Networked Appliances (IWNA 2001), pp. 115–121, 2001.

[10] P. Saffo: Sensors: The Next Wave of Infotech Innovation, Ten-Year Forecast, Institute for the Future (IFTF), pp. 115–122, 1997.

[11] K. Sakamura: TRON: Total Architecture, In Proc. of Architecture Workshop in Japan'84, pp. 41–50, 1984.

[12] T. Terada, M. Tsukamoto, K. Hayakawa, T. Yoshihisa, Y. Kishino, S. Nishio, and A. Kashitani: Ubiquitous Chip: a Rule-based I/O Control Device for Ubiquitous Computing, in Proc. Int'l Conf. on Pervasive Computing (Pervasive 2004), pp. 238–253, 2004.

[13] 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.

[14] M. Weiser: The Computer for the Twenty-first Century, Scientific American, Vol. 265, No. 3, pp. 94–104, 1991.

[15] 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.

[16] J. Widom and S. Ceri: ACTIVE DATABASE SYSTEMS, Morgan Kaufmann Publishers Inc., 1996.