NAIST-IS-DD0461027

Doctoral Dissertation

Design and Implementation of Middleware for Facilitating Development of Ubiquitous Systems

Koji Nishigaki

August 24, 2006

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Koji Nishigaki

Thesis Committee:

Professor Mirotu Ito	(Supervisor)
Professor Kenichi Matsumoto	(Co-supervisor)
Associate Professor Keiichi Yasumoto	(Co-supervisor)

Design and Implementation of Middleware for Facilitating Development of Ubiquitous Systems*

Koji Nishigaki

Abstract

This thesis summarizes the work of the author as a master/doctor student of Graduate School of Information Science, Nara Institute of Science and Technology on the middleware and the framework for developing ubiquitous systems.

Recently, there has been an increase of research on the ubiquitous system, aiming to realize the ubiquitous society. In the ubiquitous society, it is expected that anytime 24 hours a day, anywhere in the world, anyone can enjoy useful services depending on context (his/her location, states of his/her surroundings, and so on) unaware of underlying systems or mechanisms working behind to provide the services.

To achieve this goal, many research efforts have been made to realize the ubiquitous application and system. In particular, there has been an increase of research on the mobile application that offers services through mobile terminals as well as the context-aware system that offers services depending on contexts. Unfortunately, these applications and systems have not become common among ordinary people yet. The main reason is that there is difficulty in developing such applications and systems. Therefore, in order to realize the ubiquitous society, it is inevitable to design and implement middleware and framework for facilitating development of the ubiquitous application and system. In this thesis, we have focused on the mobile and home appliance systems and have studied on the

* Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0461027, August 24, 2006.

middleware of these systems because these systems are very popular and could be widely spread in the future.

This thesis provides the following two research topics.

First, we propose a middleware library for efficiently developing distributed cooperative applications consisting of a large number of cellular phone users. Our middleware provides (1) a dynamic group formation mechanism depending on users' locations and preferred subjects and (2) a group communication mechanism called multi-way synchronization for multicasting, synchronization and mutual exclusion. Most of Java executors on cellular phones do not support direct communication among user programs. Usable resources are also restricted. Therefore, in our middleware, most parts of user programs are executed on their servers as agents. Group formation and group communication mechanisms are implemented as inter-process communication on the server, and only the user-interface parts are executed on the cellular phones. From some experiments, we have confirmed that group applications consisting of ten thousands of cellular phones can be easily developed using the middleware, and that their group communication performance is reasonable for practical use.

Secondly, we propose a new framework for context-aware computing systems with home appliances (devices). In an ordinary home, each user wants to personalize multiple devices based on his/her preference. For example, an ordinary family consisting of three inhabitants: Son, Mother and Father, they may want to personalize device control in living room as follows : Son: "play music loudly and turn on room lamp with gloomy illuminance", Mother: "turn on TV and room lamp with bright illumination when a cooking program starts", Father: "turn on TV when a sport program starts". For above purpose, in order to connect sensors and home appliances via network and make these devices work cooperatively based on the current context, it is natural to specify a rule consisting of a condition and an action where the condition specifies in what context the action should be executed, and the action does how to control the target device. There have been many academic and industrial efforts to realize this kind of context-aware systems. However, due to various reasons, it would be still difficult to apply the current context-aware system to the ordinary home. Specifying feasible rules with an appropriate combination of sensors and actions is difficult for home users, and

there is no good way when multiple users want to control the same device at the same time in different ways. For these problems, our framework facilitates (1) personalization of devices, (2) intuitive specification of rules, and (3) consistency check and conflict detection in multiple rules. Our framework allows users to define new simple phrases so that a new phrase indicates a complex condition with multiple sensors or a complex action with multiple devices. It also provides intuitive user interfaces for retrieving sensors/devices to specify rules and for detecting a conflict over multiple rules. Through experiments with our prototype implementation, we show that our framework is useful for context-aware computing at home.

Keywords:

middleware, framework, context-awareness, mobile application, home appliance

Contents

List of Figures

List of Tables

1. Introduction

Recently, there has been an increase of research on the ubiquitous system, aiming to realize the ubiquitous society. In the ubiquitous society, because many small computers are embedded everywhere and cooperate with each other, it is expected that anytime 24 hours a day, anywhere in the world, anyone can enjoy useful services depending on context (his/her location, states of his/her surroundings, and so on) unaware of underlying systems or mechanisms working behind to provide the services [43] [39].

To achieve this goal, many research efforts have been made to realize the ubiquitous application and system. As the study of the small device and operating system for ubiquitous system, MOTE and TinyOS have been developed [18] [42] [36]. In Refs. [33] [34], SpaceTag has been proposed and implemented as the study of mobile application. Several techniques have been proposed to achieve context-awareness in ubiquitous computing systems. In Ref. [5], a location sensitive university campus guiding system has been implemented and evaluated. For home environments, home network systems using home appliances (devices) have been developed, and some of them are available on the market [37] [17]. Refs. [20] [21] propose a home server which can be carried by each user to personalize nearby devices. Ref. [13] is a study of home networking, and it proposes a distributed service with cooperation of home appliances based on the service oriented architecture. In Ref. [35], automatic, flexible and systematic cooperation of devices is realized by separating I/O and attachments, and introducing rule-bases. In Ref. [31], Roy et al. have developed a predictive framework for smart home based on game theory.

In spite of these academic and industrial efforts, ubiquitous applications and systems have not become common among ordinary people. The main reason is that there is difficulty in developing such applications and systems. Therefore, in order to realize the ubiquitous society, it is inevitable to design and implement middleware and framework for facilitating development of the ubiquitous application and system as well as easy utilization of the system.

In this thesis, we have focused on the mobile and home appliance systems and have studied on the middleware of these systems because these systems are very popular and could be widely spread in the future.

This thesis provides the following two research topics.

First, in Chapter 2, we describe middleware for mobile application [24] [26] [25]. Recently, the progress of cellular phones is remarkable. They can execute games and Java programs, take photographs and movies, inform their locations obtained from GPS (Global Positioning System) to other persons, and so on. Under these circumstances, distributed cooperative applications consisting of a large number of cellular phone users have received much attention where users with a common subject form a group dynamically and communicate with each other. For example, while traveling, tourists may want to contact with local persons who are familiar with local information such as tourist attractions, restaurants, shops, and so on, and obtain the recent information including photos. Also, in an exhibition, each shopkeeper may want to inform special information to the potential guests in order to gather them to his/her shop. Such information exchange may be available via group communication facilities among cellular phone users. Mass games using cellular phones are also considered as typical applications. For development of such cooperative applications, it is desirable that we can use a middleware library, which provides (1) a dynamic group formation mechanism depending on users' geographical locations and preferred subjects and (2) a group communication mechanism for multicasting, synchronization and mutual exclusion. In Ref. [40], Umedu et al. have proposed Java middleware which provides a dynamic group formation mechanism among programs (agents) on multiple mobile terminals in wireless networks and a group communication mechanism called *multi-way synchronization* of LOTOS language [14].

Extending the technique of Ref. [40] for a cellular phone, we propose a new middleware library, which provides a dynamic group formation mechanism and a group communication mechanism by *multi-way synchronization* [14] to user programs on cellular phones. Here, Java executors on the current cellular phones have the following problems : (1) they cannot use direct communication facilities among cellular phones; (2) they have a limited amount of resources (e.g., CPU, memory, battery, and so on); and (3) they can use only HTTP as communication protocols. Moreover, since there are so many potential cellular phone users, (4) we need a mechanism to efficiently select users who are interested in common subjects.

In order to solve the above three problems (1) – (3), the proposed middleware makes the most of users' programs run on their servers as agents where message exchange realizing multi-way synchronization mechanism is implemented as inter-process communication on their servers. Only the user-interface parts (UI parts) such as I/O operations are executed on the cellular phones where each agent and its UI part communicate with each other via Servlet on the server using HTTP. For problem (4), the relevance among given keywords is expressed numerically as the *context distance*, and it is used in the condition to form a group.

From some experiments, we have confirmed that group applications consisting of a few thousands of cellular phones can be easily developed using the middleware, and their group communication performance is reasonable for practical use.

Secondly, in Chapter 3, we describe framework for context-aware computing using home appliances [23]. Recently, context-aware computing [7] using home appliances is one of the most important research topics. In context-aware computing systems, devices are automatically controlled based on the *context* obtained from various sensors such as user's positions, room temperature and so on. In order to make context-aware systems work properly, we need to identify the current context of the environment (including users) and to retrieve the actions which can be executed on the context. As techniques to discover specific devices in a ubiquitous environment, UPnP [41] , Jini [32] and DLNA[9] have been standardized. Moreover, it is now becoming easy to set up a network between home appliances using short-range wireless communication such as ZigBee [44] and Bluetooth [4], power line communication [6], and so on. As described above, in an ordinary home, many sensors and home appliances will be connected via network in the near future. When such a networked home environment is available, inhabitants would hope to realize convenient services supporting daily life by automatically controlling home appliances and sensors, and making them cooperate with each other.

In order to connect sensors and home appliances via network and make these devices work cooperatively based on the latest context, it is natural to specify a *rule* consisting of a condition and an action where the condition specifies in what context the action should be executed, and the action does how to control the target device. As a study on context-aware control of devices based on rules, Ref.

[35] has been proposed. In addition, in order to specify a rule, it is required for a user to search sensors related to the required context and the target device which is executed as specified in the action part. Moreover, in order to execute the specified rule, it is required for a user to assign addresses and/or identifiers to the sensors and the devices specified in the rule with the corresponding addresses. As described above, it would be too difficult for users in an ordinary home to describe a feasible scenario for making the system work in their expected ways. Also, at a home environment, multiple users may want to control the same device simultaneously in different ways.

In this chapter, we propose a framework for allowing ordinary home user to easily describe scenarios for context-aware computing systems including various home appliances and sensors. Here, we suppose ordinary home user to be usual PC user, since PC has become common at many homes now, and for the future more. Our framework facilitates (1) personalization of devices, (2) intuitive specification of rules, and (3) consistency check and conflict detection among multiple rules. For these purposes, first, we define a language called CADEL (Context-Aware rule Description Language) to specify rules. CADEL has similar syntax and semantics to natural languages. In CADEL, each user can define new phrases (e.g., hot-and-stuffy) to indicate specific contexts sensed from multiple sensors. This functionality helps users to easily and intuitively specify similar rules to multiple devices and/or in different rooms. Secondly, our framework provides a guidance function to users during rule description, with which users can retrieve the nearby sensors and devices through GUI and obtain the useful information such as the allowable actions of a device and the value range of a sensor. Thirdly, our framework provides a mechanism to automatically detect a conflict among multiple rules, which happens when the conditions of multiple rules hold at the same time and they try to perform different actions to the same device. If many potentially conflicting rules are registered in the system, it may take a lot of computation time to detect conflict among those rules. So, we introduce a mechanism to detect conflict only when a new rule is registered in the system and let users to resolve the conflict by specifying the priority order among conflicting rules. This technique exempts the system from detecting conflict while the system is running. For this purpose, when a new rule is registered, our framework checks whether

the rule conflicts with existing rules. If it conflicts, our framework prompts users to specify the priority among the rules. Users can attach a specific context to the priority so that the priority works only on the context.

Through experiments, we confirmed that performance of our prototype implementation is practically good enough to retrieve sensors and devices and to detect conflicts over many rules. Moreover, we confirmed usefulness of our conflict detection and definition of new phrases using our prototype system.

2. Middleware for Cellular Phone Applications

2.1 Introduction

Recently, the progress of cellular phones is remarkable. They can execute games and Java programs, take photographs and movies, inform their locations obtained from GPS (Global Positioning System) to other persons, and so on. By using these functions, for example, while traveling, tourists can contact with local persons who are familiar with local information such as tourist attractions, restaurants, shops, and so on, and obtain the recent information including photos. Also, in an exhibition, each shopkeeper can inform a special information to the guests in order to gather them to his/her shop. Such information exchange may be possible via group communication facilities among cellular phone users. Mass games using cellular phones are also considered as typical applications.

For development of such cooperative applications, it is desirable that we can use a middleware library, which provides (1) a dynamic group formation mechanism depending on users' geographical locations and preferred subjects and (2) a group communication mechanism for multicasting, synchronization and mutual exclusion.

In this chapter, we propose and implement a new middleware library, which provides a dynamic group formation mechanism and a group communication mechanism by *multi-way synchronization* [14] to user programs on cellular phones.

2.2 Related Work

Recently there is a lot of research work for efficient group communication in P2P environments using application level multicast [11, 12, 29]. In wireless mobile ad-hoc networks, there are also research results for efficient group communication using multicast mechanisms based on location information of mobile terminals [8, 15]. However, in those applications using multicast mechanisms, users usually join a multicast group in order to receive some specific contents delivered from a source node. Such multicast mechanisms are not suitable for interactive group applications where many users who are interested in the same subject form a group autonomously and exchange messages each other.

As a middleware library for mobile terminals, a multi-agent based platform called FIPA has been developed [10]. Based on FIPA, some light-weight platforms such as LEAP [2] and Crumpet [30] have been developed so that cooperative applications can be efficiently executed on mobile terminals with restricted resources such as handheld PCs and cellular phones. Moreover, KDDI has also developed a FIPA based agent platform which runs on its cellular phones with Java executors [27] and agent platforms working on cellular phones (e.g. e-jumon[28] and picoPlangent[38]) are developed. Those researches are similar to our approach since both approaches make complicated programs run on their proxy servers. However, the above multi-agent based approaches do not offer powerful group communication facilities.

In Ref. [33], each information is treated as an object where its location range and live time range are appended as a tag so that only users within those ranges can access to the object. An object system based on this mechanism called SpaceTag has also been developed. In SpaceTag, the information is basically exchanged between a server and its client using one-to-one communication and broadcasting. On the other hand, in our approach, more powerful mechanisms for group communication among clients are supported. Those mechanisms can be used as an implementation environment for SpaceTag.

Moreover, various methods for forming a group have been proposed. Ref. [19] presents a method to form a mobile community in real-time so that mobile users in a specified geographical region and have similar interest/knowledge form a group. This method is different from our proposed method in the sense that our method

can handle distance not only geographical distance between positions of users but also logical distance between interests of the users and we offer highly interactive communication functions in the group. In Ref. [22], an information retrieval technique using preference information, meaning information and P2P technology is realized. This method is different from our proposed method in the sense that we offer a group formation method and highly interactive communication functions in the group.

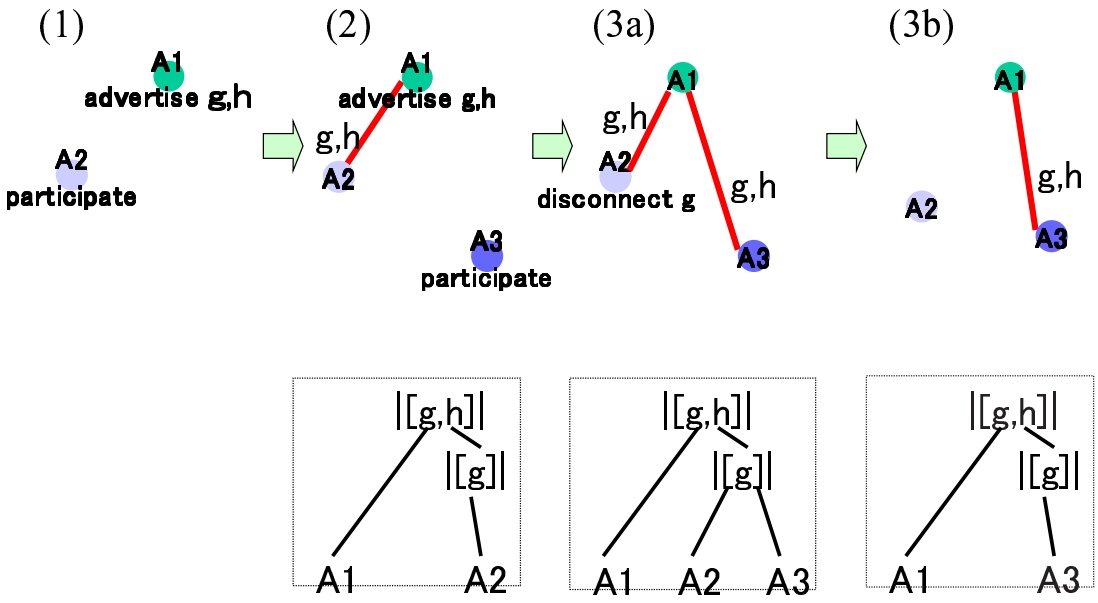

Figure 1. Establishment and Disconnection of Communication Relation in Proposed Middleware

2.3 Proposed Middleware for Cellular Phones

In this chapter, we propose a new middleware library, which provides a dynamic group formation mechanism and a group communication mechanism by multi-way synchronization [14] to user programs on cellular phones. We show our middleware's API in Table 1.

2.3.1 Dynamic Group Formation Mechanism

We call the instance of the program on each cellular phone an **agent**. Our middleware provides mechanisms that agents dynamically form a group using the method of advertising for group members $advertise(ch\_rel, val\_list, cond, max)$ and the method of participation in group $participate(val\_list, cond)$. As ch_rel in the above methods, we specify the **communication relation** (in which multicast, mutual exclusion and their combinations can be defined based on the syntax of LOTOS language[14]) among group member is defined. As val_list and $cond$,

Table 1. API of Middleware

API	function
advertise	advertise for group members with conditions
cancel	cancel advertising for group members with group ID
participate	participate in group with conditions
disconnect	break away from group
DisconnectException	notify breakaway members left from group
executeEvent	exchange data or synchronize among group member

a list of data values and a condition for restricting group formation are specified, respectively. Argument max denotes the maximum number of members.

Outline of LOTOS Here, LOTOS[14] is a formal description language for communication protocols developed by ISO and has a powerful syntax to briefly express behavior of a system including concurrency. In LOTOS, s system is described as a set of multiple parallel processes and behavior of each process (**behavior expression**) is defined as a sequence of interaction with the outside of process (input and output of a value) through a gate (it is interaction point with environment, hereinafter called channel). This interaction is called **event**. In LOTOS, by using a parallel operator $(A1|[g]|A2)$, we can describe behavior that multiple processes carry out the event on the specified gate at the same time and perform data exchange (this is called **multi-way synchronization**). In LOTOS, multi-way synchronization is described in a syntax of a binary tree such as $A1|[g,h]|((A2|[g]|A3)|[g]|A4)$, whereas in this chapter we denote a set of processes connected to the same channel by $|[g]|\{A1,...,An\}$. When we connect a process denoted by $|[g]|\{A1,...,An\}$ with another process P through synchronization parallel operator $|[g,h]|$, it is written by $P\ |[g,h]|-|[g]|\{A1,...,An\}$.

Definition of Communication Relation A *communication relation* is defied in the form of $\sharp 1|[g,h]|-|[g]|\{\sharp 2\}$ (here, $\{...\}$ denotes a set of processes). In the relation, g and h are channel names, and $|[g,h]|-|[g]|$ represents the type of communication (e.g., multi-cast, mutual exclusion, synchronization, and so on). The details of communication type are explained in Sect. 2.3.2. $\sharp 1|[g]|\{\sharp 2\}$

expresses that process (agent) $\sharp 1$ synchronizes with the set of processes (agents) in $\{\sharp 2\}$ through channel g. If we want to specify an asynchronous channel, $/as$ is attached to the channel name such as g/as. In $\sharp 1|[g,h]| - |[g]|\{\sharp 2\}$, the symbol "$-$" expresses that a relation of synchronization is established between channels, and agents in $\{\sharp 2\}$ synchronize with each other through g and then agent $\sharp 1$ synchronizes with agents in $\{\sharp 2\}$ through g or h. $\sharp 1$ is the location to which the agent executing method *advertise* is connected, and $\{\sharp 2\}$ is the set to which multiple agents executing *participate* are added.

By executing a pair of methods *advertise* and *participate* which satisfy the group formation conditions, the specified communication relation is shared by the corresponding agents, and a group called the *agent group* is formed. After that, when another agent executes *participate* and the condition holds, the agent can join the group. Like this way, the group size can be increased. While executing an application, agent can continue advertising for group members, and group ID is given back as a return value by executing method *advertise*. Agent can limit a recruitment period and the maximum number of members by specifying time and a number in conditions. In order to stop advertising additional members, we execute method *cancel* with the group ID. Agent can try participating in a group by executing method *participate*. And availability of participating in the group is given back as a return value (if available, group ID is given back).

The connection point of each agent in the communication relation is identified by the ID obtained at group formation. So, each agent can leave from the group by executing method *disconnect* with ID, scrapping the communication relation. When an agent left from a group, the other agents in the group receive exception $DisconnectException$.

The set of agents who are candidates for group members is called the *application domain*. When a cell phone user executes the application using this middleware, an agent corresponding to the user is added to the application domain. For example, in Fig. 1, assume that two agents $A1$ and $A2$ are in the application domain, $A1$ executes method *advertise* with communication relation $\sharp 1|[g,h]| - |[g]|\{\sharp 2\}$ and group formation condition $C1$, and that $A2$ executes method *participate* with condition $C2$. If conditions $C1$ and $C2$ hold between $A1$ and $A2$, they form an agent group and share communication relation

$A1|[g,h]| - |[g]|\{A2\}$. Then, when a new agent $A3$ which has just been added to the application domain executes *participate*, $A3$ joins the group of $A1$ and $A2$ and communication relation $A1|[g,h]| - |[g]|\{A2, A3\}$ is created and shared, as shown in Fig. 1 (3a). When method *disconnect*(group ID) is executed, the communication relation is scrapped as shown in Fig. 1 (3b).

Members in an agent group can use powerful group communication primitives explained in Sect. 2.3.2.

Conditions for Group Formation In order to efficiently form a group among nodes with the same interest in a large number of cell phone terminals, our middleware uses context information consisting of (1) physical distances between nodes based on their GPS-based locations, (2) logical distances calculated from interesting keywords of nodes, and (3) other specific conditions with date, time, profiles, and so on.

We normalize the logical distance between two nodes as the following formula.

$$CDist(A, B) = 1 - \frac{| A \cap B |}{| A \cup B |}$$

Here, A and B denote sets of keywords, and $|X|$ denotes the number of elements in set X. the value of $CDist(A, B)$ varies between 0 and 1. The distance will be smaller as more keywords are common between A and B. For example, when $A = \{kidswear, jeans\}, B = \{jeans\}$, the *context distance* $CDist(A, B) = 0.5$. We can define the context distance considering meanings of keywords and the implication among them, for example, using the existing text mining techniques.

2.3.2 Group Communication

Hereafter, we call input/output actions at each channel as *event*s. $g!f(x)$ denotes the event which outputs the value of $f(x)$ to channel g. We assume that members in each group basically communicate with each other synchronously.

For example, assume that four agents have formed a group with communication relation $A4|[g,h]| - |[g]|\{A1, A2, A3\}$. In this case, channel g is used as a shared bus among these agents. Multicasting a data to all members in a group is achieved as shown in Fig. 2. When $A1, A2$ and $A3$ are executing events to receive something from a channel, and $A4$ executes an event to send a value to

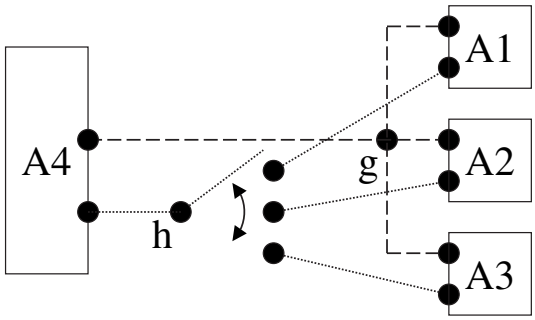

Figure 2. Multi-cast communication and exclusive control

the channel, the value is substituted to variables specified in events of $A1, A2$ and $A3$.

On the other hand, channel h is used as a switching bus between the sub group (consisting of $A1, A2$ and $A3$) and the agent $A4$, as shown in Fig. 2. This mechanism allows agents to compete to use the shared resource. That is, when agents $A1, A2$ and $A3$ try to send a data to the same channel simultaneously, only an agent is allowed to send the data so that the receiver $A4$ receives and stores the data to the variable.

Furthermore, if all agents $A1, A2, A3$ and $A4$ are ready to exclusively execute events on channel g and h, one of them is selected nondeterministically.

Primitives for group communication We have defined a method (**executeEvent**) which allows agents communicate with each other through the communication relation shared among them. Each agent can use synchronization, mutual exclusion and multicast among group members by executing method **executeEvent** with a channel name.

2.3.3 Example Application for Cellular Phone

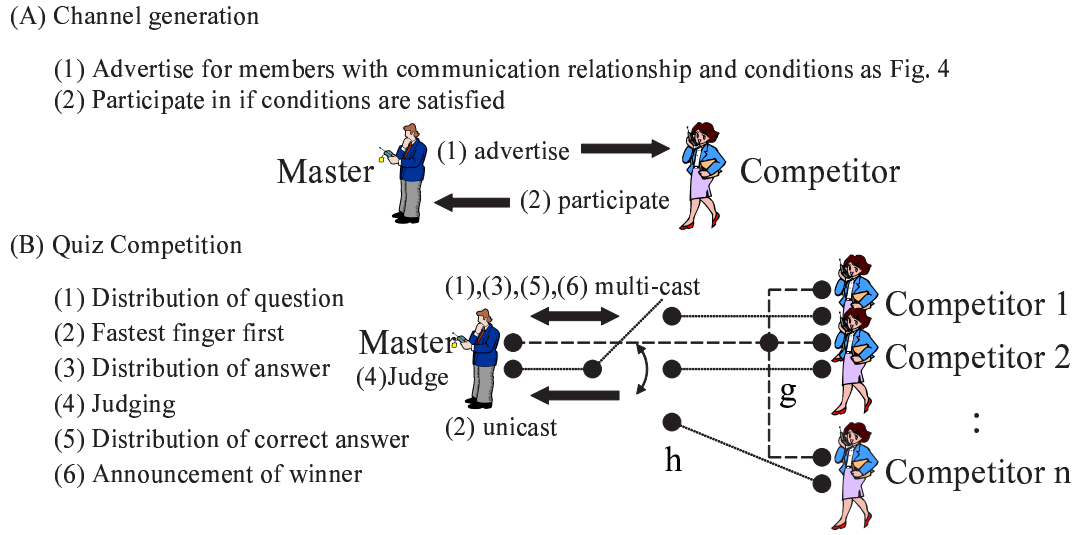

Figure 3. Example of Application (Quiz Competition)

We illustrate the quiz game application in Fig. 3. This application restricts the number of participants and uses mutual exclusion among competitors.

The whole system consists of one master and multiple (n persons) competitors where the master advertises for participants of the quiz game with communication relation shown in Fig. 10, the allowable location range and a quiz theme as a keyword. Similarly, the competitors participate in the game with its geographical location and the preferred quiz theme as a keyword. If the physical distance and the context distance calculated based on the given quiz themes are within the ranges which the master and the competitor have specified, the competitor can participate in the game. Here, the master advertises for members during a specified period, so that many competitors can participate in the game.

In the case of participation, the channel g for data sharing and the channel h for mutual exclusion are shared based on the communication relation shown in Fig. 10. In this application, the distribution of questions is described as the output event to channel g from the master (Fig. 3(B)-(1)). We assume that each agent has a unique ID, and then at most one competitor can answer the question

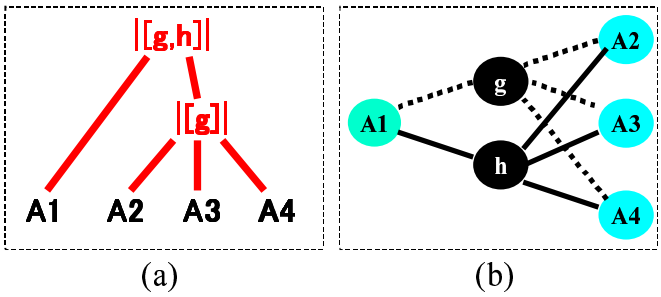

Figure 4. Channel Relation

at one time. So, the answer to the question is described as the output event to channel h with the ID of the competitor (Fig. 3(B)-(2)). If two or more persons try to perform the answer, one competitor's output event is chosen and execution is permitted based on the communication relation (Fig. 10). The terminal of the competitor who was permitted to answer distributes the answer via channel g to all terminals containing the master's and other competitors'. Then, each terminal receives and displays the answer (Fig. 3(B)-(3)).

The implementation using our middleware is very simple as shown in Fig. 5(a) and (b).

We have implemented this application as MIDlet. In Fig. 6, we show snapshots when the application is executed. The first snapshot (leftmost) shows the situation that the master advertises for participants with a keyword (NAIST). The second shot shows the situation that a group is formed among four users. The third shows the situation that each competitor tries to answer the question, and the last one shows the winner.

```
// Quiz Master
// Advertising for group member
advertise(communication relationship,
  {location, quiz theme}, condition,
num of group);
// 10 questions
for(int i=0;i<10;i++){
    // Distribution of question
    executeEvent(g!question);
    // Reception of answer
    executeEvent(h?x);
    // Judging
    ……;
    // Distribution of correct answer
    executeEvent(g!correct);
}
// Announcement of winner
executeEvent(g!winner)
```

(a)

```
// Competitor
// Participation in group
participate({location, quiz theme},
condition);
// 10 questions
for(int i=0;i<10;i++) {
    // Reception of question
    executeEvent(g?x);
    // Sending answer
    executeEvent(h!answer);
    // Reception of correct answer
    executeEvent(g?y);
}
// Reception of winner
executeEvent(g?z)
```

(b)

Figure 5. Pseudo Code

Figure 6. Snapshots of Application Execution

2.4 Basic Policy to Implement Middleware

When implementing each primitive of the proposed middleware of Sect. 2.3 on a cellular phone, we must solve the following problems: (1) only HTTP can be used as a communication protocol and direct communication between terminals cannot be used; and (2) resources (e.g., CPU, a memory, a battery, etc.) are not sufficient in the Java executor on the current cellular phone.

In the proposed middleware, we adopt a method to execute the great portion of the terminal side program on a server as an agent. Furthermore, in order to solve the above problem (1), we implement message exchanges between terminals required for multi-way synchronization by the internal communication between the agents on a server, and execute only an input/output interface program on each cellular phone. We reduce the whole communication amount by making each cellular phone communicate with the server only when the communication is required. Therefore, agents can continue communicating with each other on a server even when some of the agents cannot communicate with the corresponding cellular phones.

When we implement an application system consisting of multiple users using the proposed middleware, we implement a user program as two Java programs: one is a user interface program (called UI, hereafter) executed on a cellular phone and the other is an agent program executed on a server (Fig. 7). When a user starts the application on his/her cellular phone, UI is loaded to the cellular phone, and the corresponding agent is also loaded to the server, respectively. Each agent is executed as a java object started by Servlet. Then, the agent

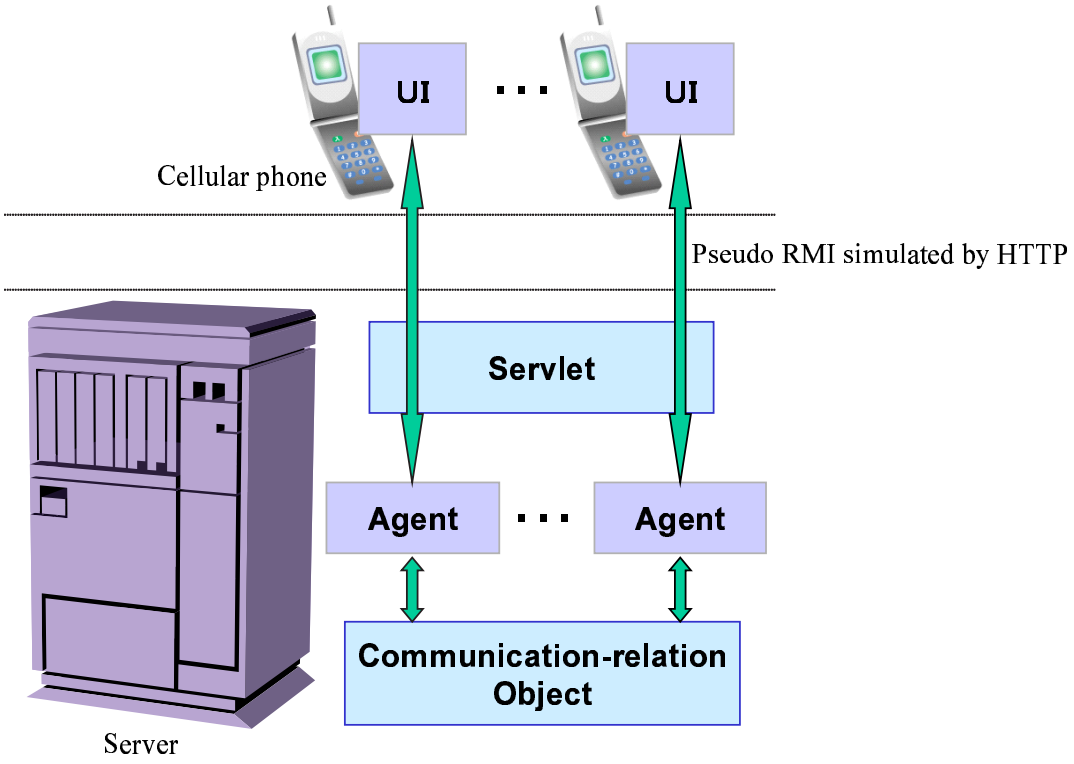

Figure 7. Outline of Middleware Architecture Using Servlet

and the UI communicate with each other by pseudo RMI (simulated by HTTP) through Servlet. Agents share a communication relation object (as more fully described hereinafter) at the time of group formation and execute synchronous communication by multi-way synchronization.

2.5 Implementation of Middleware

As described in Sect. 2.4, implementation of middleware is divided into the server side and the terminal side roughly. As shown in Fig. 8, we have implemented RMI to operate an agent program on a server from a user interface on a cellular phone, a group library to form a group and a multi-way synchronization library to communicate among group members. We describe below the details.

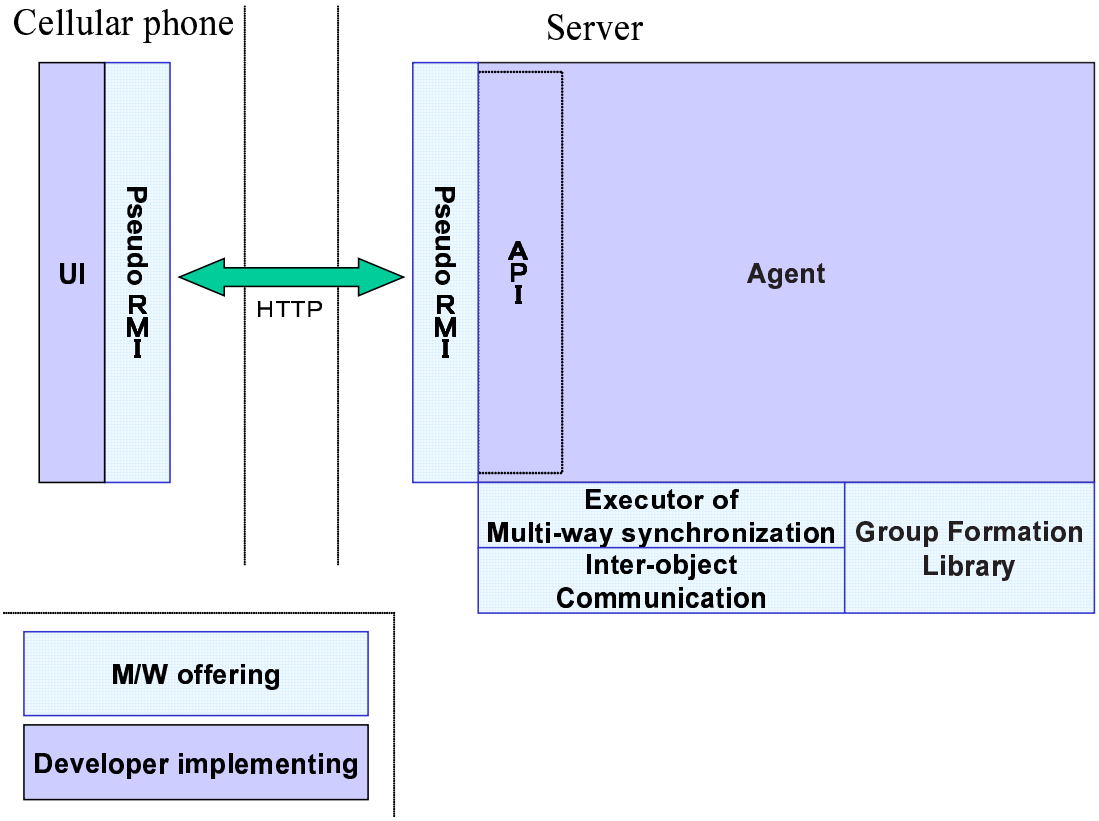

Figure 8. Block Diagram of Middleware

2.5.1 Implementation of Servlet

We have implemented class **LotosMServlet** that invokes and manages the agents. This class also acts as mediator between the agent and the user interface. In **Lo-**

tosMServlet, we define method **executeApp** to start the agent and method **executeMethod** to operate the agent.

Implementation of Methods executeApp/executeMethod We have implemented method **executeApp** in class **LotosMServlet**. Method **executeApp** is invoked from each user's UI with the options that specify the name of the agent to invoke and the ID of the user. Method **executeApp** instantiates and invokes the specified agent by using reflection functions of the standard Java APIs. The behavior of the invoked agent will be managed by object **LotosMServlet**.

We have implemented method **executeMethod** to access the agents from each user's UI. Method **executeMethod** is invoked with the parameters that specify the ID of the user, the name of the method to be invoked and a string where the parameters and their number are encoded as a comma separated list. Method **executeMethod** picks up the agent identified by the ID and invokes the target method by using the reflection functions.

2.5.2 Implementation of Agent

We have implemented a class **LotosMApp** as a base class in Java. All agents to be implemented must inherit this base class. In this class, APIs in Table 1 have been implemented as methods and exceptions.

Implementation of Group Formation We have implemented a class **GroupManager** for group formation. In our middleware, agent instances invoked by method **executeApp** are managed by an instance of **GroupManager** where each agent instance requests the group manager to register the member advertisement or the participation to a group.

As shown in Fig. 9, (1) an agent *A1* which executed method *advertise* registers advertising message m_1 (including keywords, location and so on) with group manager *M*. Then, *M* generates communication relation objects and gives pointers of the objects to *A1*. (2) An agent *A2* which executed method *participate* sends participating message m_2 (including keywords, location and so on) to *M*. (3) *M* checks values and conditions to be included in m_1 and m_2. If conditions are satisfied, *M* sends Ack message with pointers of communication relation objects. (4) Communication relation objects are shared among agents and a group is formed. (5) Repeat (2) to (4).

For a group which is formed by *advertise/participate* as shown in Fig. 10(a), communication relation objects as shown in Fig. 10(b) are made. *A1*, *A2*, *A3* and *A4* are agents and g is a shared channel and h is an exclusive channel.

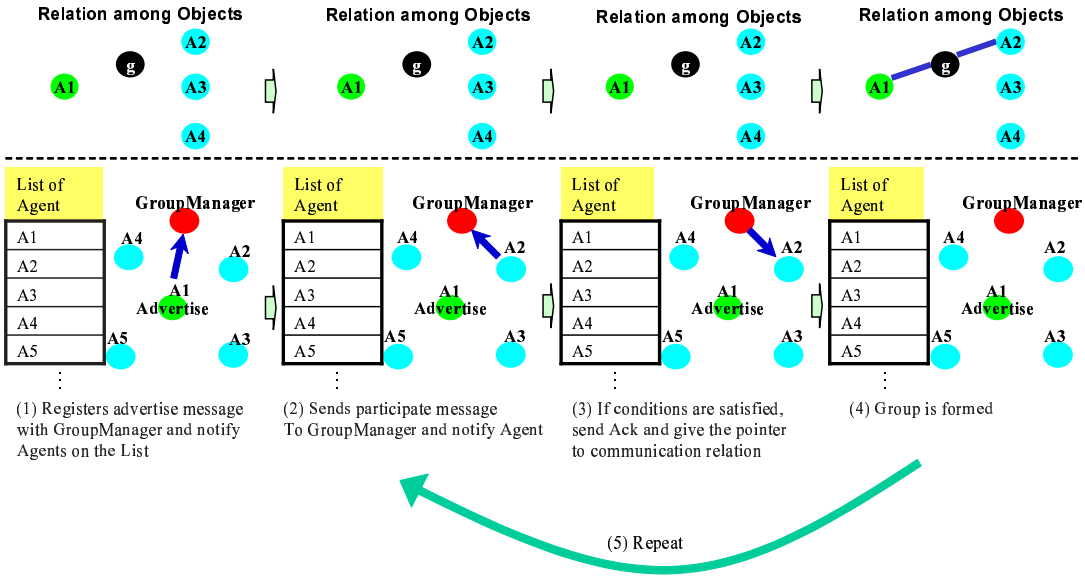

Figure 9. Advertising Process

Implementation of Group Communication In order to achieve good performance, we restrict the communication relation to be (i) $\sharp 1|[G]|\{\sharp 2\}$, or $\sharp 1|[G]|-|[H]|\{\sharp 2\} \wedge H \subseteq G$ (here, G and H are sets of channel names).

In our middleware, group communication among members is specified by method **executeEvent**. We have defined and have implemented three classes Synchronization, Exclusion and Multicast for group communication by multi-way synchronization. Through a communication relation object shared among agents, they can communicate based on one of the above three communication types.

For class Synchronization, an agent writes a data to the communication relation object and another agent in the same group reads the data and substitutes the data value to the variable specified in its event. In order to synchronize among all agents in the group, each member agent which finished the above operation

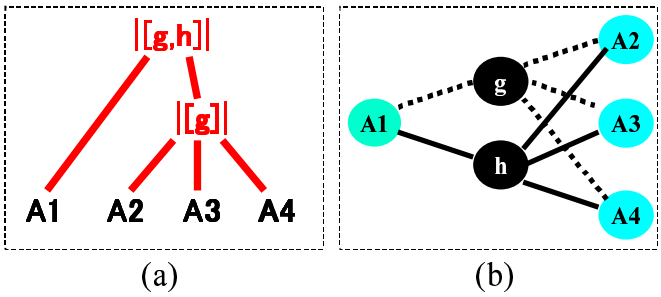

Figure 10. Channel Relation

blocks its behavior until all member agents finish this operation. For class Exclusion, among multiple agents which want to send data, only a member agent which could accessed the communication relation object first can write the data in the object. Then, the agents waiting for the data on the specified channel read the data and substitute it to the variables. Implementation of class Multicast is similar to the case of Synchronization except that each agent does not block its behavior during the data exchange process (i.e., asynchronous communication).

For example, we assume that there is a relation between agents as shown in Fig. 10. Then, if an output event to g (synchronous communication relation) is defined at A1 and an input event from g is defined at A2, A3 and A4, when A1 outputs a data to g and A2, A3, and A4 are ready to input data from g, the event is executed and the data is shared among all the agents. Moreover, if an input event from h (exclusive communication relation) is defined at A1 and an output event to h is defined at A2, A3 and A4, when A1 is ready to input data from h and the one of A2, A3 and A4 (e.g., A3) outputs a data to h, the event is executed and the data is shared between A1 and A3.

2.5.3 Starting and Operation of an Agent

Invocation and Management of Agent through HTTP We have implemented the user interfaces on the cellular phones as MIDlet and the communication between the UI and the agent on Servlet based on HTTP. As UI, we can use, for example, a web browser. To invoke an agent from a UI, an HTTP

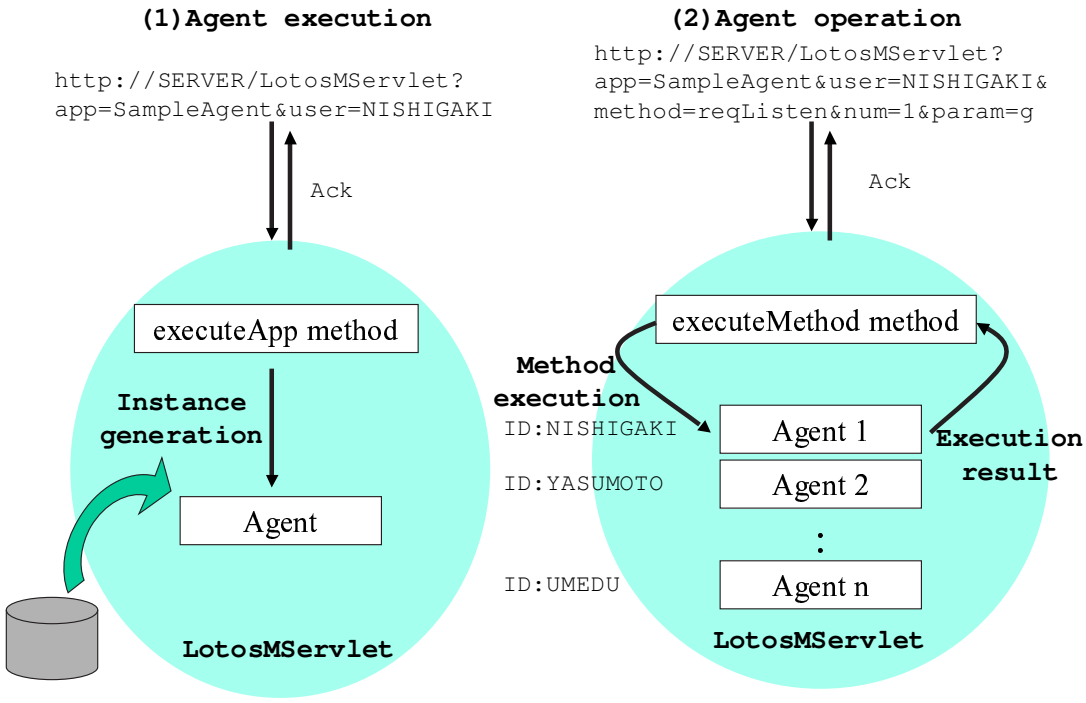

Figure 11. Starting and Operation of an Agent

request is sent to the Servlet like in Fig. 11(1). The request contain the parameters consisting of the name of the agent(SampleAgent) to invoke and the user ID(NISHIGAKI). The Servlet will invoke method **executeApp** according to the parameters. The result will be sent as the response of HTTP. Similarly the agent can be operated by the UI through HTTP like in Fig. 11(2). The parameter includes the name of the agent(SampleAgent), the user ID(NISHIGAKI), the name of method(reqListen), the number of parameters for method invocation(2) and the string that consists of comma separated list of the parameters for method invocation(p1,p2). The Servlet will invoke specified method and send back the result to the UI as an HTTP response. Here, **SOAP** is likely used for communication between UI and the agent. However, since the peer process is always determined and the required resource should be as small as possible, we adopted the original protocol lighter than SOAP.

To receive the messages from the agent to the UI except for result for the above invocation, when the UI wants to receive messages from the agent at arbitrary time, the UI must periodically send a dummy request to the Servlet. We are planning about more efficient implementation such that the agents are notified of the arrival of messages by short message services.

2.5.4 Load Distribution

It may lack scalability when we load and execute agents corresponding to all cellular phones on one server. Consequently, in this section, we describe a policy to realize load distribution by using multiple servers. As a basic policy, to perform group communication fast, we perform relocation of the agent at the time of the group formation, so that agents of the same group are executed on the same server.

As shown in Fig. 12, the group formation among agents residing in multiple different servers is as follows. (1) An agent A1 which executes *advertise* registers advertising message with group manager on the server on which A1 is running and then registers advertising message (including keywords, location and so on) with group manager on the other servers (Fig. 12(1)). (2) An agent A2 which executes *participate* sends participating message (including keywords, location and so on) to group manager on the server where A2 is running (Fig. 12(2)). (3) The

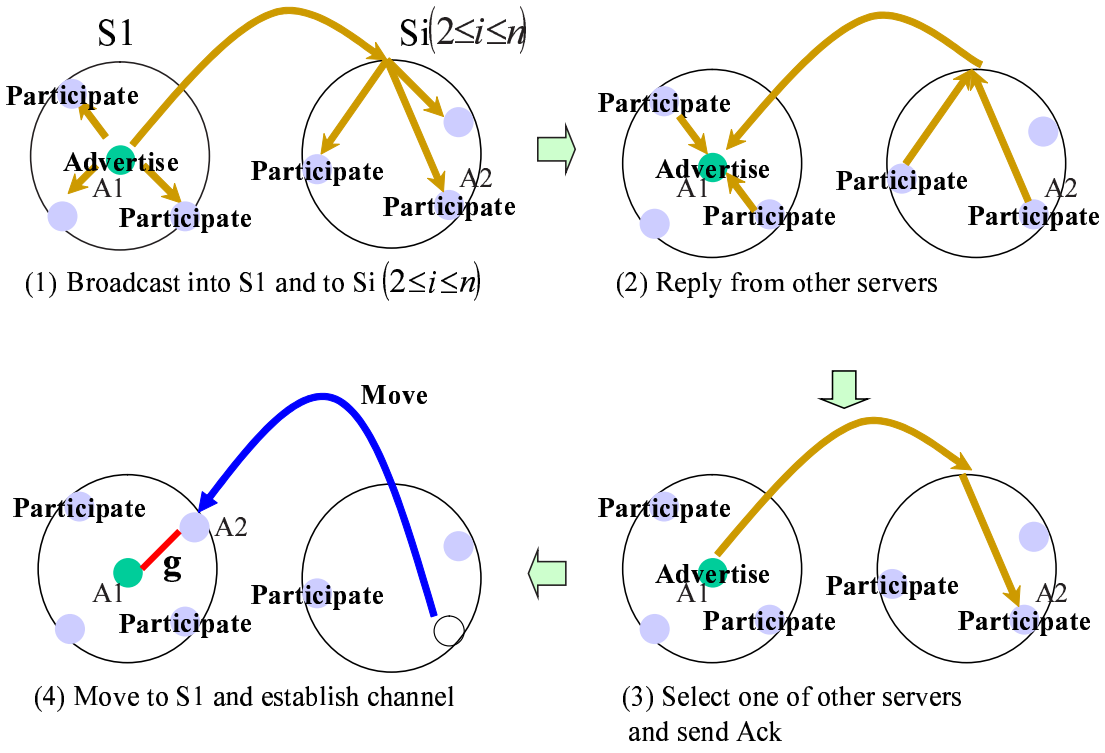

Figure 12. Image of Agent Migration

group manager checks values and conditions. If conditions are satisfied, group manager sends Ack (Fig. 12(3)). (4) A2 which received Ack message moves to the server where A1 is running and establishes the group communication channel (Fig. 12(4)).

2.6 Experimental Results

We have measured the communication performance of our group formation and group communication mechanisms, the communication performance between UI and the agent, and the overhead when using multiple servers. Moreover, we have measured communication latency at user terminals with a simple application developed with our middleware. We used the application that users form a group by setting multiple keywords and exchange simple texts. We used an ordinary PC (Athlon2200+) as the server, and used the cell phone emulator on the PC and a Java MIDlet on the real cellular phone terminal (A5303H) as UI.

In order to measure the communication performance, we measured the time of event execution among 10,000 agents, for synchronization, mutual exclusion and multicast. The result is shown in Table 2. From this result, even if the application contains the frequent interaction among 10,000 users, we see that communication among the agents within the server does not become a bottleneck. Here, since transfer of data between agents is realized as transfer of the pointer among objects in a same process running on a server, communication performance does not depend on data size (however, communication performance between an agent and UI is proportional to data size). Therefore, we think that there is no difference of group communication performance of this middleware by the difference of the data class and size that agents change.

Moreover, in order to measure the group formation performance, we measured the time of judging context distance to the agent with which the number of nodes is n=1,000, n=10,000, n=100,000 and n=1,000,000, and with which the number of elements of the keyword set is a=2 and a=4. The result is shown in Table 3. From this result, even if the application includes 100,000 agents, we see that it is able to judge the context distance in a reasonable time. However, when the application includes 1,000,000 agents, it needs about 17 seconds to calculate the context distance. Therefore, in order to reduce this time, it needs to limit the number of the agents on the one server by using the load distribution mechanism as shown in Sect. 2.5.4.

Similarly, in order to measure the communication performance between UI and the agent, we measured the time of the remote method invocation from the real cellular phone (A5303H). We measured the time until the cell phone receives

a response from a server after it requested the server to invoke an agent. The result (average of 10 measurements) was 8825 msec. On the other hand, the time required for the method execution from the cell phone was 1280 msec (average of 10 measurements). Here, because the time is long at first time communication, (specification of the cellular phone (A5303H)), the time of starting agent is longer than the time of method execution. If a communication is executed before starting agent, there is no difference with the time of starting agent and the time of method execution. This result is almost the same as the time of HTTP request (1175 msec). So the performance is reasonable for practical use.

Finally, in order to measure the overhead when using multiple servers, we measured the time of selecting a server and the time of moving an agent. The time of selecting a server is shown in Table 4 and the maximum time was about 12 seconds. Moreover, the time of moving an agent (the class size was about 1K byte) was 109 milliseconds (in the case of 100 agents, the time was about 10 seconds). From these results, we think that the overhead when using multiple servers is reasonable for practical use.

Table 2. Performance of Event Execution ($m$sec)

relationship of communication	synchronization	exclusive control	multi-cast
n=10,000	11	40	11

Table 3. Performance of calculations and matching of context distances ($m$sec)

	n=1,000	n=10,000	n=100,000	n=1,000,000
a=2	60	202	1557	17712
a=4	70	214	1557	15182

From these results, the communication performance of the application using this middleware can be estimated as follows. The time of output event execution becomes sum of the sending time from UI to the agent (the time of method

Table 4. Performance of Selection of the Server ($m$sec)

Order of server selection	
Server1	8825
Server1 to Server2	10184
Server1 to Server2 to Server1	12050

execution from UI (1280 milliseconds)) and the time of event execution(Table. 2), that is about 1.3 seconds. Moreover, the time of input event execution becomes sum of the time of event execition(Table. 2) and the sending time from the agent to UI. Therefore, we think that this time depends on the polling period (a few seconds) (As shown in Sect. 2.5.3, the polling is necessary for the communication from the agent to UI). We implemented application in Sect. 2.3.3, and measured the time of event execution (distribution of question and fastest finger first). As a result, we confirmed that our middleware achieves performance as we expected.

2.7 Conclusions

In this chapter, we proposed middleware for developing distributed cooperative applications consisting of multiple cellular phones. With the proposed middleware, cellular phone users can dynamically form a group depending on their preferences and geographical locations, where group members can efficiently communicate with each other using group communication facility such as multicast and mutual exclusion of the multi-way synchronization mechanism.

3. Framework for Context-aware Computing at Home

3.1 Introduction

In an ordinary home, many sensors and home appliances will be connected via network in the near future. When such a networked home environment is available, inhabitants would hope to realize convenient services supporting daily life by automatically controlling home appliances and sensors, and making them cooperate with each other.

In order to connect sensors and home appliances via network and make these devices work cooperatively based on the current context, it is natural to specify a *rule* consisting of a condition and an action where the condition specifies in what context the action should be executed, and the action does how to control the target device. In addition, in order to specify a rule, it is required for a user to search sensors related to the required context and the target device which is executed as specified in the action part. Moreover, in order to execute the specified rule, it is required for a user to assign addresses and/or identifiers to the sensors and the devices specified in the rule with the corresponding addresses. As described above, it would be too difficult for users in an ordinary home to describe a feasible scenario for making the system work in their expected ways. Also, at a home environment, multiple users may want to control the same device simultaneously in different ways.

In this chapter, we propose a framework for allowing ordinary home user to easily describe scenarios for context-aware computing systems including various home appliances and sensors. Here, we suppose ordinary home user to be usual PC user, since PC has become common at many homes now, and for the future more.

3.2 Related Work

There are several studies on personalization of devices. Ref. [20] proposes a home server which can be carried by each user to personalize nearby devices. This home server is capable of personalizing various devices, controlling those devices according to users' preferences and situations. The server also has a mechanism to discover nearby devices in various locations (e.g., at home, at stations, in cars, on streets, and so on). Ref. [13] is a study of home networking, and it proposes a distributed service with cooperation of home appliances based on the service oriented architecture. In Ref. [35], automatic, flexible and systematic cooperation of devices is realized by separating I/O and attachments, and introducing rule-bases. These existing studies allow users to personalize devices to a certain extent. However, they suppose that each device is used by a single user at one time, and do not treat the case when more than one users use the same device simultaneously. They do not have a function to facilitate easy description of personalization scenarios for ordinary home users.

Several techniques have been proposed to achieve context-awareness in ubiquitous computing systems. In Ref. [5], a location sensitive university campus guiding system has been implemented and evaluated. Ref. [1] proposes a framework for facilitating development of context-aware applications. In Ref. [3], a framework for developing mobile context-aware applications has been proposed. In these existing frameworks, contexts are represented by multiple output data from different kinds of sensors. The frameworks help developers to easily implement context-awareness in the system with APIs to obtain the current context by sensing required information through sensors. The frameworks also provide a mechanism to assign a relationship among sensors, actuators and application components. As described above, these frameworks focus mainly on facilitating development of context-aware systems. Our proposed framework is different from these existing ones, since our framework allows ordinary home users to easily describe scenarios for controlling the home appliances in their expected ways.

3.3 Basic Ideas

3.3.1 Problems in Typical Context-aware Control of Home Appliances at Home

Let us suppose context-aware control of home appliances in a living room at an ordinary home, consisting of Tom and his parents (Alan and Emily). Also suppose that there are a stereo system, a flat-panel TV, a video recorder, a fluorescent light, floor lamps, and an air conditioner in the living room.

Typical context-aware control is as follows: When Tom comes to the living room, the floor lamps are turned on with half-lighted (e.g., indirect lighting by floor lamps), the stereo system starts to play his favorite music (e.g., jazz) with appropriate volume, the air conditioner regulates room temperature and humidity to his comfortable degrees (e.g., 25C and 60%). After a while, the TV is automatically turned on since a TV program on air includes a keyword which he is interested in. Then a pop-up menu is shown on the TV to let him decide switching off the stereo and watching the program or turning on the video recorder to record it.

It is possible to implement this kind of a system with the existing personalization methods such as in Ref. [20] and sensors which detect the information required to identify the current context (existence/location of the user, the current temperature and humidity, the current time, and the TV programs on air). In order to control devices appropriately, users need to describe rules indicating target devices, actions to be executed, and conditions to hold when executing those actions. This task would be difficult for ordinary home users.

On the other hand, it is also difficult to personalize devices when multiple users share the same space. For example, let us suppose that Alan (Tom's father) has got home from work while Tom is listening to jazz music in the above example. When Alan has registered a keyword "baseball game" and a game is currently on air, the system identifies Alan and turns on the TV. In that case, the TV sound might interfere with that of the stereo, resulting in unsatisfaction of the both users (we call this situation *semantic conflict*). Also, when Alan has preferences for the room lighting and temperature which greatly differ from those of Tom's, the system will have a trouble in deciding whose preference should be adopted to

the device (this situation is called *device conflict*). In an actual situation, conflict with the same device is likely to be solved by a negotiation among the conflicting users and by deciding a priority order with a certain policy. However, most of existing context-aware systems suppose that each device is used by one user at one time. They do not have mechanisms to detect conflicts or to support users to solve the conflicts.

3.3.2 Basic Ideas of Our Framework

For the problems in the previous section, we adopt the following ideas.

Avoidance of Device Conflict

If there are conflicts among rules specified by different users, the corresponding actions may not be executed as the users expect. Although the result of those conflicting rules depends on how the system is implemented, possibly, the oscillation among those actions may likely occur or one action (e.g., the first or the last action) may be executed. Therefore, in order to avoid this problem, each user should be able to notice whether the rule which the user tries to add conflicts with other rules. We believe that it is mandatory to prevent conflict among rules before activating a new rule. Also this technique exempts the system from detecting conflict among registered rules periodically while the system is running. Through experiments, we will investigate our conflict detection method needs practically small computation time in Sect. 3.5.

In the proposed framework, we basically solve the conflict by defining a priority among conflicting rules. However, users might want to change the priority depending on situations. So, we adopt a policy that users can control a device based on the priority given to rules containing the device.

We can give multiple different priorities for a set of conflicting rules for various situations. For example, it is possible to define priorities so that rule R1 is executed prior to R2 if condition A holds, and that R2 is executed prior to R1, otherwise. When two conflicting rules R1 and R2 are executable at the same time, either R1 or R2 is executed depending on whether condition A holds or not. Multiple conditions can be specified to generate multiple priority patterns for the same set of conflicting rules. However, those conditions must be disjoint (i.e., two or more conditions must not hold at the same time).

Intuitive Rule Description

In existing context-aware systems, users must be familiar with functionalities of sensors and devices as well as their locations, and specify precise values (or ranges) for sensors and action names for devices in each rule. Unlike these existing systems, our framework provides a lookup service for sensors and devices, which allows users to browse them (e.g., visually or by voice) and to see their functionalities, current values of sensors and so on.

Also, it is complicated to specify a condition in each rule as a compound context which is typically represented by a logical conjunction of inequalities for the values derived from multiple different sensors. Our framework provides a facility to define each compound context as a simple phrase.

The condition of each rule is expressed by the combination of value ranges of multiple sensors. So, we define each combination of the value ranges as a phrase such as *hot and stuffy* (e.g., the temperature over 25C and the humidity over 70%) and *half-lighting* (e.g., the range over 50 lx and under 300 lx).

This facility alleviates rule description when many rules have to be specified with similar conditions. In addition, phrases can be used as macro definitions in programming languages to change conditions of multiple rules by modifying only the definition of a phrase. Since user preferences may change from time to time, this facility would be convenient.

For example, when a user changes the definition of "hot and stuffy"" (temperature over 25C and humidity over 70%) to "temperature over 28 C and humidity over 75%", this change will be applied to all rules of the user specifying "hot and stuffy" as the conditions. Furthermore, there are usually many locations such as rooms and entrances where rules are executed. It is convenient to use phrases for specifying rules for specific locations such as children room and living room by combining phrases and locations (e.g., "if hot and stuffy in children room"). It is also convenient to use phrases for specifying similar rules with different time (e.g., 10:00 to 12:00 or after 16:00).

3.4 Proposed Framework

In order to achieve the facilities explained in Sect. 3.3.2, we define a language called CADEL (Context-Aware rule DEfinition Language), and propose a framework with user interfaces to easily specify rules in CADEL, a mechanism to automatically detect inconsistencies and conflicts of rules, and a mechanism for controlling devices based on rules.

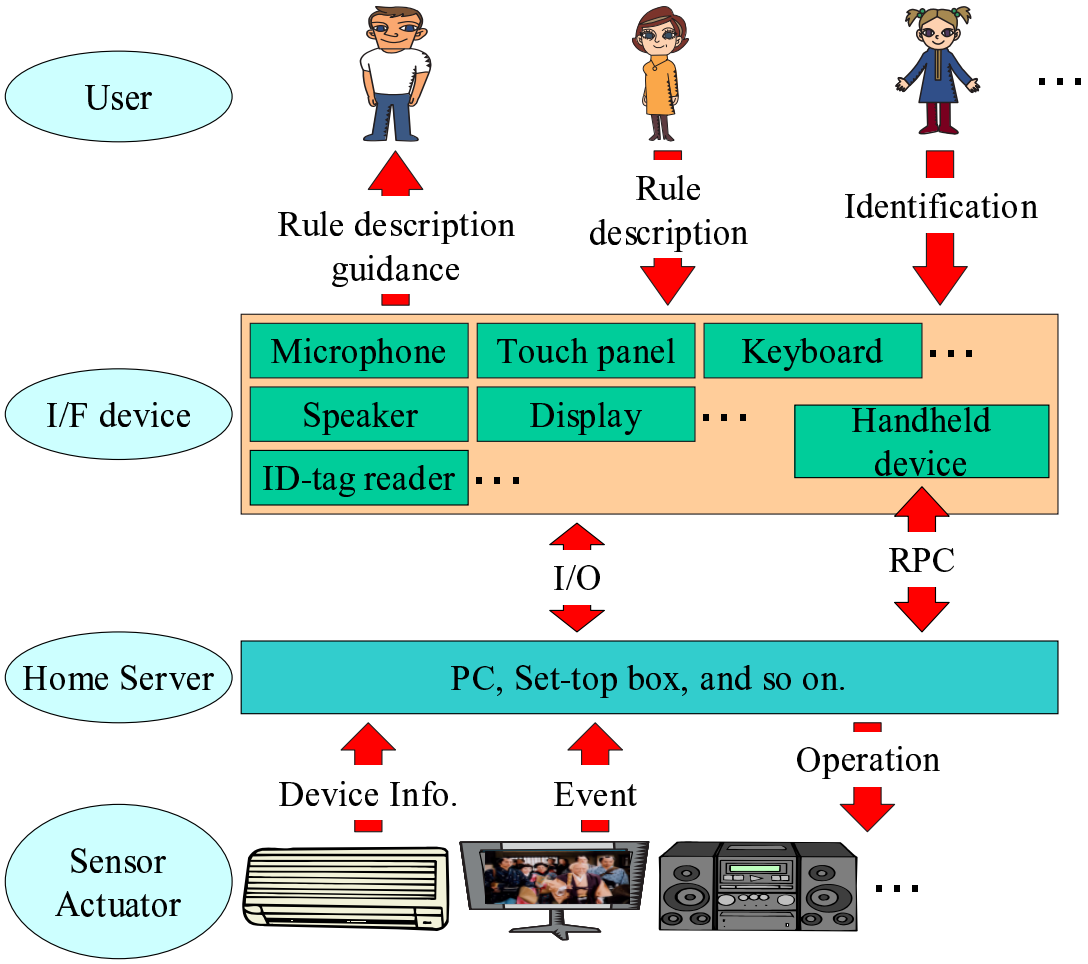

Figure 13. Environment of CADEL Framework

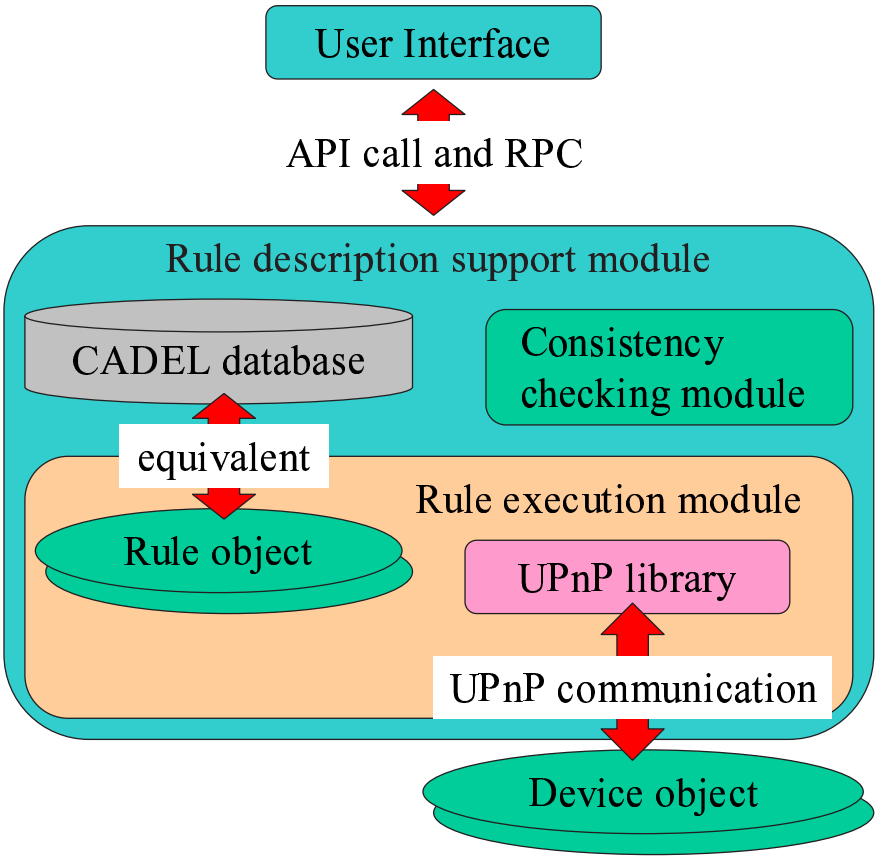

Figure 14. Structure of CADEL Framework

3.4.1 Overview

Target Environment

As shown in Fig. 13, our framework facilitates development of context-aware computing systems consisting of users with IDs (e.g., RFID tags), interface devices, a home server(s), sensors, and home appliances with actuators. We assume that these components are connected with each other through a wired or wireless physical network.

In the system, users behave in two ways. First, users describe rules for personalizing devices through interface devices. Secondly, users generate contexts such as IDs and locations and receive context-aware services according to the contexts and rules they specified. As interface devices, we suppose input devices such as microphones, touch panels, keyboards and mice, and output devices such as speakers and displays. Portable computing devices such as PDAs and cell phones can also be used as interface devices. We suppose that home appliances have capability of communication with a home server with standard platforms such as UPnP or Jini. Finally, we suppose that most functionalities of the proposed framework are implemented in a home server(s). Any PC or set-top box can be a home server.

Structure of Our Framework

As shown in Fig. 14, we compose our framework of the following modules: (1) rule execution module, (2) rule database, (3) rule description support module, (4) consistency checking module, and (5) communication interface module.

The rule description support module allows users to easily describe rules by communicating with interface devices and collecting information from sensors and devices through the communication interface module. Here, we suppose that the UPnP library is used as the communication interface module. Rules described by the rule description support module are represented as CADEL descriptions and stored in the rule database.

In order to separate the user interface from the rule description support mechanism, we implement all functions of the rule description support mechanism as APIs. If the user interface and the rule description support module are located on different computers, APIs are called using RPC from the user interface side.

The consistency check module checks whether a new rule is consistent or not

and whether the new rule conflicts with other rules registered by other users. The rule execution module executes rules. Each rule is described in CADEL, and the rule is translated to the corresponding "rule object" when the rule is registered. It receives events from external components in Fig. 13 and issues commands to devices through the communication interface module. Here, we use the UPnP library to retrieve sensors and actuators, to obtain data from the sensors, and to interact with actuators.

3.4.2 CADEL

In CADEL, we have adopted syntax similar to natural languages, aiming at intuitive rule description by ordinary home users (it can also be supported to specify rules by voice). Although we only give English-based syntax of CADEL in this chapter, the CADEL syntax can be defined based on any other languages. The syntax of CADEL in BNF notation is shown in Table 5.

In CADEL, we can describe the following rules, for example.

(1) Turn on the light at the second floor.

(2) If humidity is higher than 80%, turn on the dehumidifier.

(3) After evening, if someone returns home and the hall is dark, turn on the light at the hall.

(4) At night, if no one is at the living room and the light is turned on, turn off the light.

(5) If humidity is higher than 80 percent and temperature is higher than 28 degrees, turn on the air conditioner with 25 degrees of temperature setting.

(6) If baseball game is on air at channel 40, turn on the video recorder with 40 of channel setting.

(7) At night, if entrance door is unlocked for 1 hour, turn on the alarm.

In CADEL, new phrases can be defined by combining multiple actions of devices. For example, both VCR and TV have to be controlled when a user

Table 5. Syntax of CADEL

<Command> ::=	<RuleDef> | <ActionDef> | <CondDef> | <ConfDef>
<RuleDef> ::=	[<PreCondition>] <Action> [<PostCondition>]
<Action> ::=	<Verb> <Object> [<Configuration>]
	| <UserDefinedAction>
<Verb> ::=	"Turn on" | "Turn off" | "Record" | ...
<Object> ::=	[<Article>] <DeviceName> [<Modifier>]
<Article> ::=	"a" | "an" | "the"
<PreCondition> ::=	[<TimeSpec>] "if" <CondExpr> ["then"]
	| [<TimeSpec>] "when" <CondExpr>
	| <TimeSpec>
<PostCondition>::=	"if" <CondExpr> | "when" <CondExpr> | <TimeSpec>
<ActionExpr> ::=	<Action>
	| <ActionExpr> "and" <ActionExpr>
	| <ActionExpr> "or" <ActionExpr>
	| "(" <ActionExpr> ")"
<CondExpr> ::=	<Cond> [<PeriodSpec>] [<TimeSpec>]
	| <Cond> <TimeSpec> <PeriodSpec>
	| <CondExpr> "and" <CondExpr>
	| <CondExpr> "or" <CondExpr>
	| "(" <CondExpr> ")"
<Cond> ::=	<Sensor> [<Modifier>] <State> | <UserDefinedCond>
<Sensor> ::=	<DeviceName> | <Person> | <Place> | <Event> | "nobody" | ...
<Event> ::=	"baseball game" | ...
<State> ::=	[<Be>] "turned on" | [<Be>] "dark" | [<Be>] "is higher than" Temperature
	| [<Be>] "over" Percent
	| [<Be>] "hotter than"
	| [<Be>] "at" <Place>
	| "comes back" | "returns home" | ...
<Be> ::=	"is" | "are"
<Temperature> ::=	<Figures> "degrees" | <Figures> "degrees Celsius" | <Figures> "degrees Fahrenheit"
<Configuration>::=	"with" <RowOfConfs>
<RowOfConfs> ::=	<Setting> "of" <Parameter> "setting" | <RowOfConfs> "and" <RowOfConfs>
<Parameter> ::=	"temperature" | "channel"
<Setting> ::=	<Temperature> | <Channel> | ...
<Modifier> ::=	"at the second floor" | "at the living room" | ...
<TimeSpec> ::=	"after" <Time> | "at" <Time> | "until" <Time> | ...
<PeriodSpec> ::=	Period | "from" <Time> "to" <Time> | Period "after" <Time>
<Time> ::=	[DateSpec] <TimeOfTheDay>
<DateSpec> ::=	<Date> | "every" <DayOfTheWeek>
<Period> ::=	"for" <Figures> "seconds" | "for" <Figures> "minutes" | ...
<ActionDef> ::=	"Let's call the action that <ActionExpr> <UserDefinedAction>
<CondDef> ::=	"Let's call the condition that" <CondExpr> <UserDefinedCond>
<ConfDef> ::=	"Let's call the configuration that" <RowOfConfs> <UserDefinedConf>

wants to watch a TV program recorded in a VCR. In this case, it is convenient to define a new phrase "watch video" which is a combination of actions "play back the content in the VCR" and "project the signal from input 1 to the TV screen". Each defined phrase can be used as an action in each rule.

In CADEL, users can also define new phrases that can be used in conditions and/or in device configurations of rules by `<CondDef>` and `<ConfDef>` of Table 5.

For example, we can define a new phrase *hot and stuffy* as follows.

> Let's call the condition that humidity is higher than 60 % and temperature is higher than 28 degrees *hot and stuffy*

Moreover, for example, we can define a new configuration *high volume* as follows.

> Let's call the configuration that 80 of volume setting *high volume*

3.4.3 Rule Description Support Module

This module navigates users to describe rules in CADEL. When rules are described, they are compiled into rule objects to make the system easily process them.

This provides the following navigation functions.

(1) display of the current context, which helps a user to describe a condition of a rule by showing types and current values of sensors.

(2) definitions of a compound context as a simple phrase and a compound action as a simple phrase, which help users to easily describe rules.

(3) retrieval of devices, which helps users to know what devices are available nearby by specifying device types such as air-conditioning, and displays the available command list of each device.

(4) consistency check and conflict avoidance, which allow users to know (i) whether the specified rule is consistent or not, and (ii) whether the rule conflicts with other rules in the database. This function also allows users to specify priority orders among conflicting rules.

The navigation function is provided through GUI. Rules are described in the following steps.

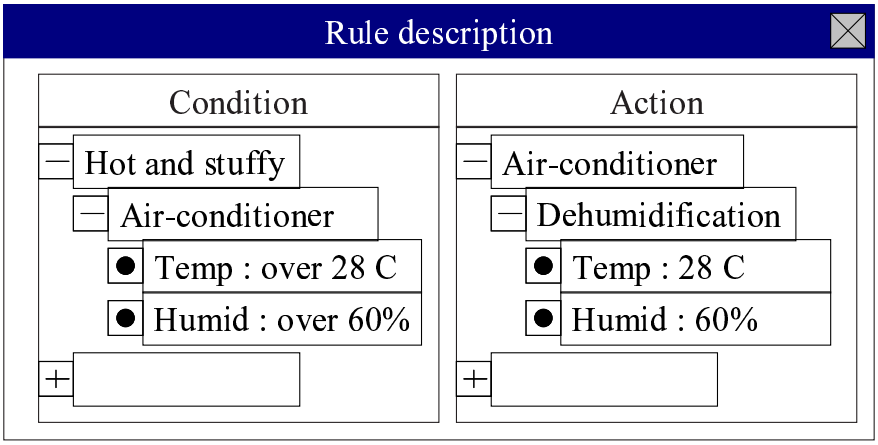

Figure 15. GUI for Rule Description

Rule Description

Basically, we use a GUI-based dialog box as shown in Fig. 15. We also support a voice recognition system to specify rules. The interface consists of two sub-interfaces: condition description I/F and action configuration I/F.

(i) Retrieval of Contexts and Conditions

In the condition description I/F, users can retrieve contexts and related sensors as shown in Fig. 17. Contexts and sensors can be retrieved by specifying combination of the following items: (1) keyword, (2) action, (3) sensor type, (4) sensor name, and (5) location. For example, the air-conditioner, the temperature meter and so on can be retrieved by specifying "temperature" as the sensor type. Sensors can also be retrieved by the user defined phrase (e.g., "hot and stuffy"). Inversely, information about sensor types and the user defined phrases can be retrieved by specifying sensors. Since device information is registered into the UPnP control point in the UPnP framework, retrieval of a specified device can be performed locally. Therefore, retrieval of devices finishes instantly.

(ii) Defining Compound Context as New Phrase

In the condition description I/F, users can define a new phrase to represent

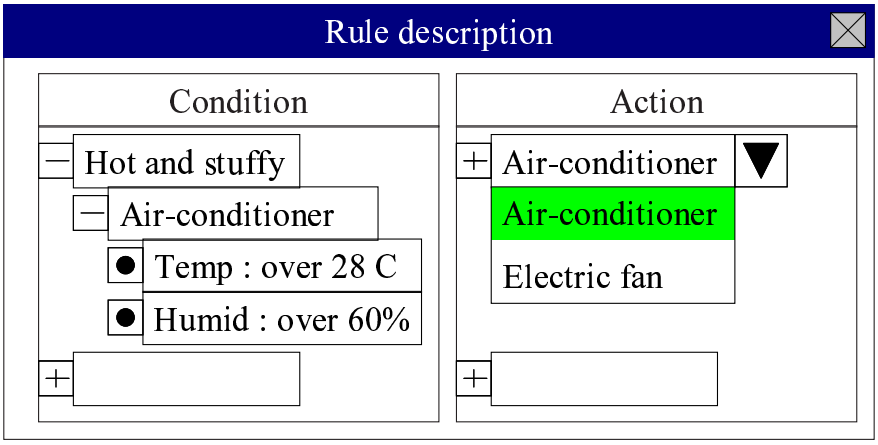

Figure 16. Action Selection by Retrieving Devices

a compound context through GUI. Fig. 15 shows an example to define a new phrase *hot and stuffy* for representing the context with temperature more than 28 degrees and humidity more than 60 %.

(iii) Retrieving Actions and Configurations

In the action configuration I/F, devices can be retrieved by specifying combination of the following items: (1) keyword, (2) context, (3) action, and (4) location. By selecting a specific device in the retrieved device list, the I/F shows what actions are allowed in the device.

Retrieval of actions as well as contexts can be executed instantly. It is because retrieval of a specified device can be performed locally since device information is registered into the UPnP control point in the UPnP framework.

(iv) Defining Series of Actions as New Phrase

With action configuration interface, new phrases can be defined by specifying multiple actions as described before. Defined phrases can be used as an action on action configuration interface.

(v) Import and Export of Rules

The navigation function of our framework greatly simplifies rule description. However, some users may still find it troublesome to describe rules for various devices at different locations. So, our framework provides an import/export mech-

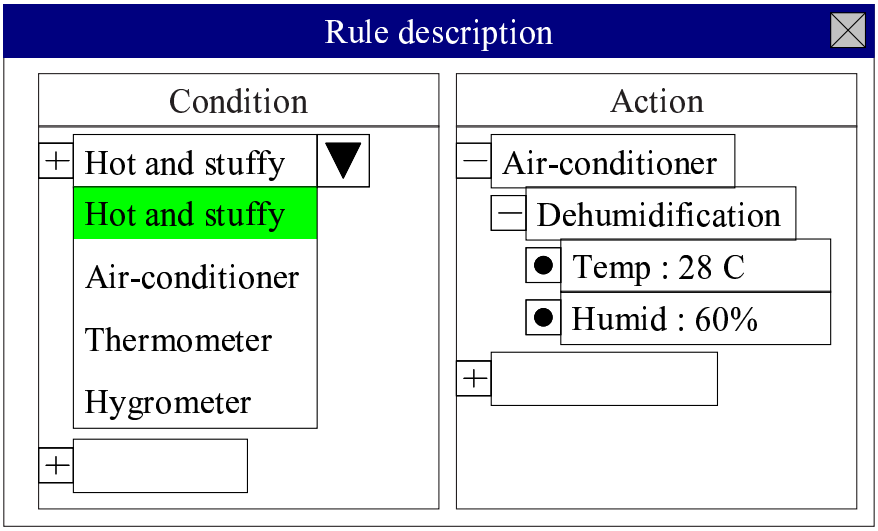

Figure 17. Condition Description by Retrieving Sensors

anism for rules. Users can import a rule registered in the database, and customize it to match their preferences. When a rule is imported, the system automatically searches sensors and devices that match the rule. If there are multiple sensors and devices matching the imported rule, users can select one of them.

3.4.4 Consistency and Conflict Check Module

In CADEL, since condition in each rule is described as a logical conjunction of inequalities, it can be calculated efficiently to check whether the condition can hold or not.

Inconsistency Check of Rules

Whenever a new rule is described and registered in the system, the module evaluates the condition in the new rule to check whether it can hold. If the condition cannot hold, the module warns the user to modify the condition in the rule.

Conflict Detection among Multiple Rules

If a new rule r_{new} is consistent and registered in the system, then the module checks whether r_{new} can conflict with other rules in the database in the following

steps.

First, the module extracts from the database the set of rules R which control the same device as r_{new}. Then, for each rule r in R, the module checks if there is a value (a combination of values) satisfying both conditions of r_{new} and r simultaneously.

For example, suppose the following two rules A and B.

> A: if temperature is higher than 30 degrees and humidity is higher than 60 %, then turn on the air conditioner with temperature of 28 degrees setting.
> B: if temperature is higher than 29 degrees and humidity is higher than 65 %, then turn on the air conditioner with temperature of 27 degrees setting.

In that case, the logical expression representing conditions for A and B are represented as follows.

> Condition for A: Temp $>$ 30 and Humid $>$ 60
> Condition for B: Temp $>$ 29 and Humid $>$ 65

Whether conditions for A and B hold for simultaneously depends on whether a logical product of A and B has a solution for variables $Temp$ and $Humid$. If there are values for $Temp$ and $Humid$ satisfying the both conditions, these conditions conflict with each other. In this particular case, A and B conflict with each other, since the above logical product has a solution.

When the module detects a conflict, it warns the user to modify the new rule or to specify the priority order among the conflicting rules.

Detecting Semantic Conflicts The proposed method has a mechanism to support detection of semantic conflicts between actions of different devices (e.g. air conditioner and heater). As we explained in Sect. 3.3.1, "semantic conflicts" happens when different devices do conflicting actions such as cooling and heating or listening to the stereo and watching TV in the same room.

This mechanism allows each user to (i) see what rules are currently executed when the user feels something wrong in the control of devices, and (ii) to notify the system that what combination of actions are semantically conflicting when

selecting the corresponding rules. The possible combination of the semantically conflicting actions is registered in the database so that the system can detect semantic conflicts and warns the user when new rules are specified later on.

For obvious conflicts like using air conditioner and heater at the same time, we can register combinations of such conflicting actions beforehand.

Specifying Priority Order

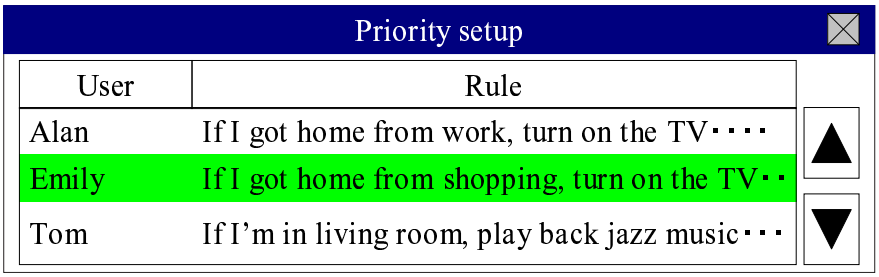

Figure 18. Interface to Specify Priority Order

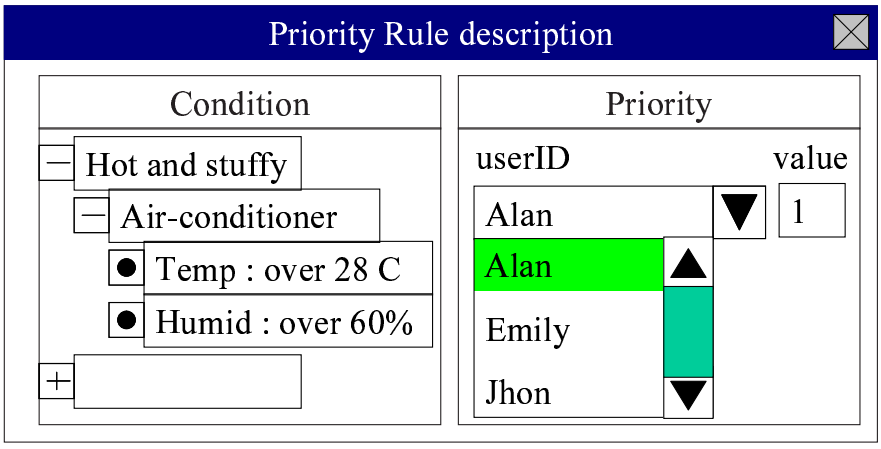

Figure 19. Interface to Specify Priority Rule

When the consistency check module detects a conflict, it shows conflicting rules in a dialog box as shown in Fig. 18. In the dialog box, users can specify the priority order among those rules. Here, rules are arranged in the decreasing

order of the priority degree [1] (i.e., the first rule (Alan) has the highest priority). If the priority order is already specified among some of them, users can modify the order.

For flexible conflict avoidance, our framework provides a mechanism to change the priority order when the conflict occurs. In that case, when the system detects the conflict, it shows the conflicting rules with the current priority order among them and lets users to follow or modify the current priority order. Our framework also allows users to attach the context to the priority order. So, when a user wants to change the priority order, he/she can keep the old priority order by attaching a context to it in which the user wants to apply this priority order in the future.

For this purpose, we allow users to specify a condition and numbers specifying the priority order of conflicting users as context. Users specify the condition and priority order using priority description I/F shown in Fig. 19.

[1] In general, the partial order should be defined among those conflicting rules. Here, we use the total order to simplify the interface.

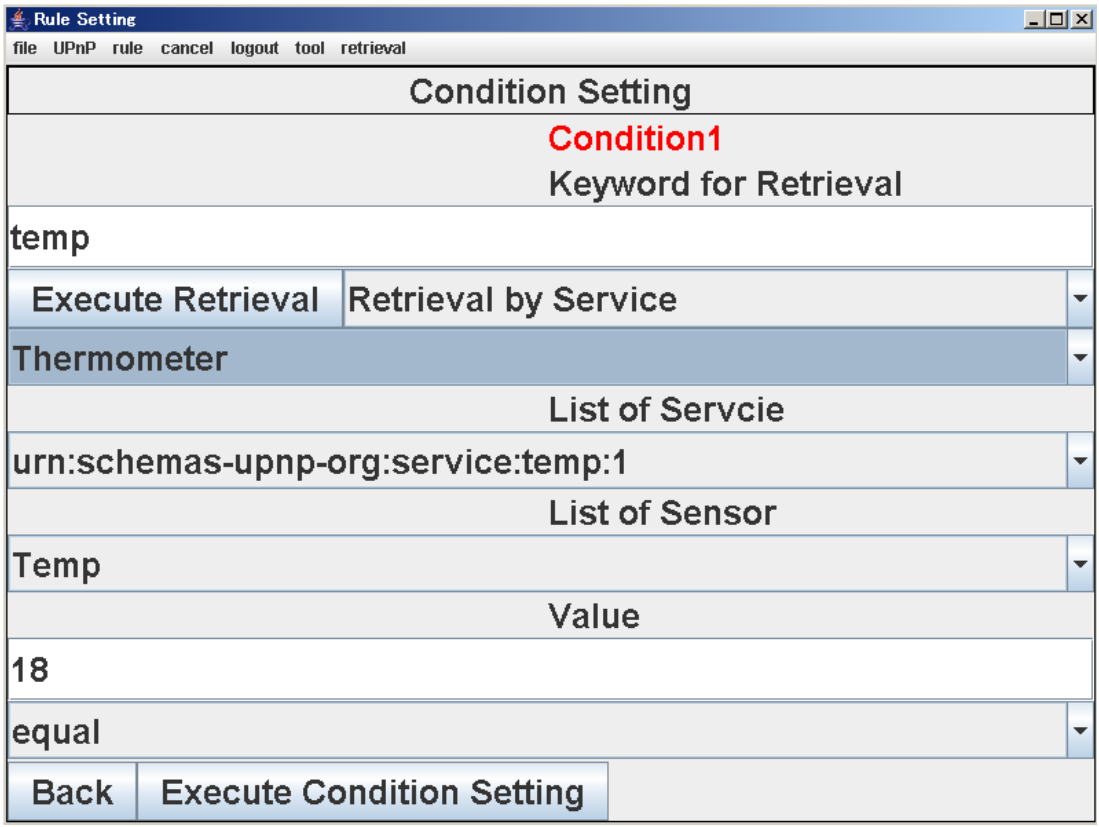

Figure 20. Snapshot of Prototype System

3.5 Evaluation

In order to evaluate usefulness of the proposed framework, especially for mechanisms for defining compound context and for detecting conflicts of rules, we developed a prototype system of the framework (Fig. 20), and let seven testees with various attributes use it. In order to categorize the testees, we asked them to answer the following questionnaire.

1. usual work
   1.1. using Web on PC (light PC user)
   1.2. writing text on PC (medium PC user)

1.3. programming (heavy PC user)

2. experience of usual work

   2.1. under one year (beginner)

   2.2. under three years (ordinal user)

   2.3. over three years (expert)

3. age

   3.1. twenties to fifties (adult)

   3.2. over the sixties (old age)

4. occupation

   4.1. information science student

   4.2. liberal arts student

   4.3. office worker

   4.4. IT professional

We divided the testees into the categories from the following three viewpoints.

- regularly using PC or other information equipments
- regularly using input devices like keyboard
- has experience of using logical expression

However, since it is difficult to prepare all combinations covering all attributes of the above categories due to costs and labors, we selected some attributes for the experiment.

We evaluated the time of the conflict detection and usefulness of defining compound context. As for conflict detection, we measured time taken by testees to perform conflict detection, and confirmed that the performance is sufficient for practical use. As for defining compound context, we confirmed that there is the significant difference between the usage before and after introducing the

definition of compound context. Furthermore, we confirmed that the introduction of defining compound context shortens difference of operation time among testees.

In our experiments, we used a PC with Athlon2200+ and 512M memory and JDK1.5.0 on Windows XP as the home server. We used CyberLink IPv6 for Java[16] as the UPnP library. A snapshot of the user I/F part of our system is shown in Fig. 20.

To detect conflicting rules, we implemented a C library for solving the satisfiability of given linear expressions using the Simplex Method. The experimental results are shown below.

Time for Detecting Conflicting Rules

When a user registers a new rule in the home server, the server searches the existing rules conflicting with the new rule. The server (1) extracts existing rules which specify the same device as the new rule, (2) for each extracted rule, constructs a logical conjunction of linear inequalities by concatenating it and the new rule, and (3) checks if this expression has feasible solutions or not.

In the experiment, we assumed the case that the server retains 10,000 registered rules, and that among them 100 rules specify the same device in their action parts. We also assume that the condition part of each rule contains a logical product of two inequalities. Thus, a logical product of four inequalities must be evaluated for each extracted rule.

We think that the above configuration includes most of possible situations in an ordinary home.

In our experiment, the time for extracting the rules with the same device was 10ms or less, and the time for evaluating a logical product of four inequalities 100 times was about 0.2ms.

We think that the above result is good enough for practical use.

Usefulness of Compound Context Definition

In order to evaluate usefulness of our compound context definition mechanism, we measured the actual time to describe rules with and without the mechanism. We measured the actual time taken by the seven testees to operate the system. Each one of profiles are shown in Table 6. Here, testees specified the rule “If it is hot

Table 6. Profile

	Profile		
	usual work (experience)	age	occupation
koji-ni	1.3(2.3)	3.1	4.4
kazuhiro	1.2(2.3)	3.2	4.3
muneo-n	1.3(2.2)	3.1	4.1
morihi-t	1.3(2.3)	3.1	4.1
tatsu-ta	1.3(2.3)	3.1	4.1
katsumoto	1.2(2.3)	3.1	4.3
terauchi	1.3(2.3)	3.1	4.1

and stuffy, turn on the air-conditioner" with the user I/F of our system for both cases that the phrase "hot and stuffy" is defined and not defined in the system. We measured time for testees to specify this rule with various contexts. Here, for example, we let the testees to specify the rule in the following case. "If the user exists in the living room and it is hot and stuffy, the air-conditioner is started up." The results are shown in Table 7. Table 7 shows that the mechanism reduced time to describe rules to a certain extent. Furthermore, when we performed the t-test to the averages, we obtained the statistical significance P as follows.

$$P = 0.004 < 0.01$$

Consequently, we confirmed that the difference of the amount of the work for specifying rule was significant in both cases with and without compound context definition. Furthermore, the root-mean-square deviations are 98 and 27, for both cases with and without compound phrases, respectively, so we confirmed that variance decreases when using compound context definition. Consequently, we believe that our compound context definition mechanism is useful for users who are not PC experts.

Table 7. Time for Describing Rule (Definition Compound Context)

	Time for Describing	
	Without Definition of Compound Context	With Definition of Compound Context
koji-ni	59s	38s
kazuhiro	306s	111s
muneo-n	275s	96s
morihi-t	122s	76s
tatsu-ta	230s	113s
katsumoto	354s	114s
terauchi	290s	120s
Average	234s	95s

3.6 Conclusions

In this chapter, we proposed a framework for context-aware computing systems with home appliances, which facilitates rule description and device configuration for ordinary home users. The novelty and contribution of this chapter reside in that the proposed framework provides (1) an easy and intuitive way in configuring the whole system to personalize devices, and (2) a support for detecting device conflicts among multiple users.

Through experiments, we confirmed that performance of our prototype implementation is practically good enough to retrieve sensors and devices and to detect conflicts over many rules. Moreover, we confirmed usefulness of our conflict detection and definition of new phrases using our prototype system.

4. Conclusion

In this thesis, the following two research topics on development of ubiquitous systems have been studied.

First, we proposed and implemented middleware for developing distributed cooperative applications consisting of multiple cellular phones. With the proposed middleware, cellular phone users can dynamically form a group depending on their preferences and geographical locations, where group members can efficiently communicate with each other using group communication facility such as multicast and mutual exclusion of the multi-way synchronization mechanism.

Secondly, we proposed and implemented a framework for context-aware computing systems with home appliances, which facilitates rule description and device configuration for ordinary home users. The novelty and contribution of this paper reside in that the proposed framework provides (1) an easy and intuitive way in configuring the whole system to personalize devices, and (2) a support for detecting device conflicts among multiple users.

Acknowledgements

I would like to thank all those people who made this thesis possible and an enjoyable experience for me.

First, I am deeply indebted to my supervisor Professor Minoru Ito, Professor Kenichi Matsumoto and Professor Hideki Sunahara for their valuable suggestions, advise and support.

I would like to express my sincere gratitude to Associate professor Keiichi Yasumoto for his adequate guidance, valuable suggestions and discussions throughout this work. This work could not be achieved without his support, encouragement and guidance.

I would like to thank Professor Teruo Higashino of Osaka University, Associate professor Naoki Shibata of Shiga University and Assistant professor Takaaki Umedu of Osaka University for their invaluable comments, valuable suggestions and support.

Finally, I would like to thank all of members of Ito Laboratory for their helpful advice and support.

References

[1] Jakob E. Bardram. Applications of context-aware computing in hospital work: examples and design principles. In *Proceedings of the 2004 ACM symposium on Applied computing (SAC2004)*, pages 1574–1579, 2004.

[2] Federico Bergenfi and Agosfino Poggi. Leap: a fipa platform for handheld and mobile devices, 2001.

[3] Gregory Biegel and Vinny Cahill. A framework for developing mobile, context-aware applications. In *Proceedings of 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom2004)*, pages 361–365, 2004.

[4] Bluetooth. http://www.bluetooth.com/.

[5] Jenna Burrell, Geri K. Gay, Kiyo Kubo, and Nick Farina. Context-aware computing: A test case. In *Proceedings of UbiComp 2002: Ubiquitous Computing: 4th International Conference*, pages 1–15, 2002.

[6] CE Powerline Communication Alliance (CEPCA). http://www.cepca.org/.

[7] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[8] Kai Chen and Klara Nahrstedt. Effective location-guided tree construction algorithms for small multicast in manet. In *Proceedings of IEEE INFOCOM2002*, 2002.

[9] Digital Living Network Alliance. http://www.dlna.org/.

[10] FIPA. Fipa2000 specifications. http://www.fipa.org.

[11] Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM*, 2001.

[12] Yang hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, 2000.

[13] Hiroshi Igaki, Masahide Nakamura, Haruaki Tamada, and Kenichi Matsumoto. Implementing integrated services of networked home appliances using service-oriented architecture. *IPSJ Journal*, 46(2):314–326, 2005. (in Japanese).

[14] ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, 1998.

[15] Young-Bae Ko and Nitin Vaidya. Geocasting in mobile ad hoc networks: Location-based multicast algorithms. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, pages 101–110, 1999.

[16] Satoshi Konno. http://www.cybergarage.org/net/upnp/java/index.html.

[17] Matsushita Electric Industrial Co., Ltd. http://national.jp/appliance/product/kurashi-net/ (in Japanese).

[18] MOTE. http://www.xbow.com/Products/wireless_Sensor_Networks.htm.

[19] Shinji Motegi, Kiyohito Yoshihara, Hiroki Horiuchi, and Sadao Obana. Mobile community formation mechanism in intelligent transportation systems (its). *IPSJ Journal*, 42(7):1840–1846, 2001. (in Japanese).

[20] Tatsuo Nakajima and Ichiro Satoh. Personal home server: Enabling personalized and seamless ubiquitous computing environments. In *Proceedings of 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom2004)*, pages 341–345, 2004.

[21] Tatsuo Nakajima and Ichiro Satoh. A software infrastructure for supporting spontaneous and personalized interaction in home computing environments. *Journal of Personal and Ubiquitous Computing (PUC)*, 10, 2006.

[22] Minoru Nakazawa and Shimmi Hattori. A proposal and its implementation of p2p system based on semantic and preference information. *IPSJ Journal*, 44(3):826–834, 2003. (in Japanese).

[23] Kouji Nishigaki, Keiichi Yasumoto, Naoki Shibata, Teruo Higashino, and Minoru Ito. Framework and rule-based language for facilitating contextaware computing using information appliances. In *Proceedings of the first International Workshop on Services and Infrastructure for the Ubiquitous and Mobile Internet (SIUMI'05)*, pages 345–351, 2005.

[24] Kouji Nishigaki, Keiichi Yasumoto, Takaaki Umedu, Teruo Higashino, and Minoru Ito. Middleware for cellular phones providing group formation based on context and group communication facility. *IPSJ Journal*, 45(12):2666–2677, 2004. (in Japanese).

[25] Kouji Nishigaki, Keiichi Yasumoto, Takaaki Umedu, Teruo Higashino, and Minoru Ito. Middleware providing group communication facility based on multi-way synchronization for cellular phone applications. In *Proceedings of the 4th International Workshop on Smart Appliances and Wearable Computing (IWSAWC2004)*, pages 434–437, 2004.

[26] Kouji Nishigaki, Keiichi Yasumoto, Takaaki Umedu, Teruo Higashino, and Minoru Ito. Demonstration of a cellular phone application based on context-aware group formation. In *Proceedings of the 7th International Conference on Mobile Data Management (MDM'06)*, 2006.

[27] Satoshi Nishiyama, Gen Hattori, Chihiro Ono, and Hiroki Horiuchi. Lightweight fipa compliant agent platform on java-enabled mobile phone for ubiquitous services. *IPSJ Journal*, 45(2):575–585, 2004.

[28] OMRON. jumon. http://www.e-jumon.com/.

[29] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. Almi: An application level multicast infrastructure. In *Proceedings of 3rd Usenix Symposium on Internet Technologies & Systems*, 2001.

[30] Stefan Poslad, Heimo Laamanen, Rainer Malaka, Achim Nick, Phil Buckle, and Alexaner Zipf. Crumpet: Creation of user- friendly mobile services personalised for tourism, 2001.

[31] Nirmalya Roy, Abhishek Roy, and Sajal K. Das. Context-aware resource management in multi-inhabitant smart homes: A nash h-learning based approach. In *Proceedings of 4th IEEE International Conference on Pervasive Computing and Communications (PerCom 2006)*, pages 148–158, 2006.

[32] Sun microsystems. http://www.sun.com/software/jini/.

[33] Hiroyuki Tarumi, Ken Morishita, Megumi Nakao, and Yahiko Kambayashi. Spacetag: An overlaid virtual system and its applications. In *Proceedings of 1999 International Conference on Multimedia Computing and Systems (ICMCS'99)*, volume 1, pages 207–212, 1999.

[34] Hiroyuki Tarumi, Kasumi Nishihara, Hirotoshi Hori, Kazuya Matsubara, Yuuki Mizukubo, Shouji Nishimoto, and Fusako Kusunoki. An application and evaluation of 3d virtual city service for mobile phones. *IPSJ Journal*, 47(1):41–50, 2006. (in Japanese).

[35] Tsutomu Terada, Masahiko Tsukamoto, Keisuke Hayakawa, Tomoki Yoshihisa, Yasue Kishino, Atsushi Kashitani, and Shojiro Nishio. Ubiquitous chip: Rule-based i/o control device for ubiquitous computing. In *Proceedings of Second International Conference on Pervasive Computing (Pervasive2004)*, pages 238–253, 2004.

[36] TinyOS. http://www.tinyos.net/.

[37] TOSHIBA. http://www3.toshiba.co.jp/feminity/index_j.html (in Japanese).

[38] TOSHIBA. picoplangent. http://www2.toshiba.co.jp/plangent/.

[39] TRON PROJECT. http://www.assoc.tron.org/jpn/tp.html (in Japanese).

[40] Takaaki Umedu, Keiichi Yasumoto, Akio Nakata, Hirozumi Yamaguchi, and Teruo Higashino. Middleware for synchronous group communication in wireless ad hoc networks. In *Proceedings of IASTED International Conference on Communications and Computer Networks (CCN2002)*, pages 48–53, 2002.

[41] UPnP Forum. http://www.upnp.org/.

[42] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer S.J. Pister. Smart dust: Communicating with a cubic-millimeter computer. *IEEE Computer Magazine*, pages 44–51, 2001.

[43] Mark Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):99–104, 1991.

[44] ZigBee Alliance. http://www.zigbee.org/.

List of Major Publications

Journal Papers

1. Kouji Nishigaki, Keiichi Yasumoto, Takaaki Umedu, Teruo Higashino and Minoru Ito: Middleware for Cellular Phones Providing Group Formation Based on Context and Group Communication Facility, *IPSJ Journal*, (in Japanese), Vol. 45, No. 12, pp. 2666-2677, December, 2004.

International Conference

1. Kouji Nishigaki, Keiichi Yasumoto, Takaaki Umedu, Teruo Higashino and Minoru Ito: Demonstration of a Cellular Phone Application based on Context-Aware Group Formation, *Proceedings of the 7th International Conference on Mobile Data Management(MDM'06)*, CD-ROM, May, 2006.

2. Kouji Nishigaki, Keiichi Yasumoto, Naoki Shibata, Teruo Higashino and Minoru Ito: Framework and Rule-based Language for Facilitating Context-aware Computing using Information Appliances, *Proceedings of the first International Workshop on Services and Infrastructure for the Ubiquitous and Mobile Internet(SIUMI'05)*, pp. 345–351, June, 2005.

3. Kouji Nishigaki, Keiichi Yasumoto, Takaaki Umedu, Teruo Higashino and Minoru Ito: Middleware Providing Group Communication Facility Based on Multi-way Synchronization for Cellular Phone Applications, *Proceedings of the 4th International Workshop on Smart Appliances and Wearable Computing(IWSAWC2004)*, pp. 434–437, March, 2004.

4. Kouji Nishigaki, Keiichi Yasumoto, Takaaki Umedu, Teruo Higashino and Minoru Ito: Middleware Providing Dynamic Group Communication Facility for Cellular Phone Applications, *Proceedings of the 2004 IEEE International Conference on Mobile Data Management(MDM2004)*, p. 170, January, 2004.

5. Hiroshi Nishikawa, Shinya Yamamoto, Morihiko Tamai, Kouji Nishigaki, Tomoya Kitani, Naoki Shibata, Keiichi Yasumoto and Minoru Ito: UbiREAL: Realistic Smartspace Simulator for Systematic Testing, *Proceedings of the 8th International Conference on Ubiquitous Computing(UbiComp2006)*, to appear, September, 2006.