Middleware Providing Dynamic Group Communication Facility for Cellular Phone Applications

Kouji Nishigaki† Keiichi Yasumoto† Takaaki Umedu‡ Teruo Higashino‡ Minoru Ito†

† Graduate School of Information Science, Nara Institute of Science and Technology
{koji-ni,yasumoto,ito}@is.aist-nara.ac.jp
‡ Graduate School of Information Science and Technology, Osaka University
{umedu,higashino}@ist.osaka-u.ac.jp

Abstract In this paper, we propose a middleware library for efficiently developing distributed cooperative applications consisting of a large number of cellular phones with Java executors. The proposed middleware provides (1) a dynamic group formation mechanism depending on users’ geographical 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 the current cellular phones do not support direct communication among multiple user programs. Usable resources are also restricted. Therefore, in the proposed middleware, most of users’ programs are executed on their servers as agents. Message exchange among users for group formation and group communication is implemented as inter-process communication on their servers. Only the user-interface parts (UI parts) are executed on the cellular phones where each agent and its UI part communicate with each other via Servlet on the server that uses HTTP when the communication is required. As an index for forming a group with common subjects, 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 hundreds of cellular phones can be easily developed using the middleware, and their group communication performance is reasonable for practical use.

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. Recently, 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 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 [13], we 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 [4].

In this paper, by extending our existing middleware, we propose a new middleware library, which provides a dynamic group formation mechanism and a group communication mechanism by multi-way synchronization to user programs on cellular phones. In our existing middleware, a group is formed step by step by advertising for a member in a radio range and by receiving its responses. However, 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); (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 narrow the user domain.

In order to solve the above three problems (1) – (3), the proposed middleware make the most of users’ programs running on their servers as agents where message exchange implementing 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 that uses HTTP when the communication is required. 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.

The proposed middleware has been implemented using JDK1.4.1, Tomcat4.1.24 and WTK1.0.4_01. From some experiments, we have confirmed that group applications consisting of a few hundreds of cellular phones can be easily developed using the middleware, and their group communication performance is reasonable for practical use.

1.1 Related Work

Recently there is a lot of research work for efficient group communication in P2P environments using application level multicast [5, 6, 8]. 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 [2, 10]. 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.

On the other hand, as a middleware library for mobile terminals, a multi-agent based platform called FIPA has been developed [3]. Based on FIPA, some light-weight platforms such as LEAP [1] and Crumpet [9] have been developed so that cooperative applications can be efficiently executed on mobile terminals with restricted resources such as handheld PCs and cellular phones. KDDI has also developed a FIPA based agent platform which runs on its cellular phones with Java executors [7]. 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. On the other hand, our approach has facilities to (1) dynamically form a group consisting of a lot of cellular phone users based on given subjects and users’ locations and (2) perform powerful group communication among group members using the multi-way synchronization mechanism. From this point of view, the two approaches have a big difference.

In [11], 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.

2 Proposed Middleware for Cellular Phones

2.1 Our Existing Middleware for Wireless Environment

Our existing middleware [13] provides a dynamic group formation mechanism for multiple agents (processes) running on mobile terminals in wireless networks. In the middleware, agents running on mobile terminals can use two basic communication primitives to form a group: Advertise(ch_list) which advertises a group member; and Participate(ch_list) which requires to participate in a group. Here, ch_list is the list of channel names shared among members in the same group, and the value of ch_list of Advertise is assigned to that of Participate when a group is formed.

For example, in Fig. 1, suppose that agents A1 and A2 have executed Advertise(\{g\}) and Participate(\{h\}) primitives, respectively. When A1 and A2 are in their common radio range, the channel g is established and shared between A1 and A2 as shown in Fig. 1(1). Moreover, suppose that A2 executes Advertise(\{g\}) primitive and another agent A3 in A2’s radio range executes Participate(\{f\}) primitive. As a result, the channel g is shared among A1, A2 and A3 (Fig. 1(2)). As shown in the lower part of Fig. 1, a logical tree called the synchronization tree (corresponding to the syntax tree of LOTOS language[4]) spanning group members is constructed, where an ID is attached to each tree node (each ID is produced when a new member joins to the group by executing a pair of Advertise and Participate primitives). In order to leave from the group and to release the channel, Disc(ID) primitive is used (Fig. 1 (3a), (3b)). When leaving causes the separation of the tree, the tree is re-constructed according to the semantics, which we proposed in [12].
We call the set of agents sharing the same channels as the *agent group*. In the agent group, group members can communicate with each other using a group communication mechanism called multi-way synchronization [4].

### 2.2 Proposed Middleware

The main purpose of this paper is to propose a new model to specify cooperative cellular phone applications consisting of a large number of users, and develop a middleware library based on the model.

Unlike the wireless networks which our existing middleware supposes, cellular phones can communicate with any hosts (servers) in the Internet through a base station. There is no notion about radio range, in other words, the radio range corresponds to whole the Internet. Moreover, in order to form a group efficiently, we should be able to use geographical location information (e.g., obtained from GPS) and to use the relevance between users’ concerns.

From the above discussion, we propose a new model that supports group formation and group communication mechanisms where both location information and users’ concerns can be specified as conditions to form a group.

**Context Distance** We have defined the relevance between given two keyword lists $A$ and $B$ as the following function called the *context distance*.

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

Here, $|X|$ 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 $B$. When there is no common keywords between $A$ and $B$, the distance will be 1. For example, when $A = \{\text{kids wear}, \text{jeans}\}, B = \{\text{jeans}\}$, the context distance $CDist(A, B) = 0.5$. For the context distance, we can adopt existing text-mining techniques and other related techniques, however, these are out of scope of this paper.

**Communication Primitives of Our Middleware** From the reasons in Sect. 3, in the proposed middleware, we implement each user program as two separate Java programs: the agent program running on a server and the user interface program running on a cellular phone. Here, we will explain primitives used in agent programs.

Each agent program is implemented as a sub-class of class *Agent*. In class *Agent*, two methods *Advertise(ch_list, val_list, cond)* and *Participate(ch_list, val_list, cond)* are defined for the group formation. These are extensions of our existing middleware. Method *Advertise* and method *Participate* are executed with parameters *ch_list, val_list*, and *cond* for the request for advertising group members and the request to join the group, respectively. In *val_list*, we can specify various information such as the location information, the keyword list, and so on. The values in *val_list* of an agent can be referred by another agent when evaluating condition *cond* to form a group. In *cond*, we can use $#k$ to refer $k$’th value in *val_list* of the peer agent.

For example, suppose that two agents $A1$ and $A2$ execute the following (1) and (2), respectively.

**Figure 1: Channel Establishment and Disconnection in Existing Middleware**


![Figure 2: Multi-cast communication and exclusive control](image)

\[
\text{Advertise}\{a, b\}, (\langle \text{Loc1}, \{\text{kidswear, jeans}\} \rangle, \text{GDist}(\langle \text{Loc1}, \#1 \rangle \leq 0.3 \\
\quad \text{and} \ \text{CDist}(\{\text{kidswear, jeans}\}, \#2 \rangle \leq 0.5))
\]

\[
\text{Participate}\{a, b\}, (\langle \text{Loc2}, \{\text{kidswear}\} \rangle, \text{GDist}(\langle \text{Loc2}, \#1 \rangle \leq 0.5 \\
\quad \text{and} \ \text{CDist}(\{\text{kidswear}\}, \#2 \rangle \leq 0.6))
\]

If the geographical distance (denoted by \(\text{GDist}(x, y)\)) between \(\langle \text{Loc1} \rangle\) and \(\langle \text{Loc2} \rangle\) is equal to or less than 0.3 (Km), all conditions hold. Then, \(A1\) and \(A2\) form a group where channels \(\{a, b\}\) are shared.

The relationship among member agents in a group is represented by a synchronization tree, similarly to our existing middleware. Each agent knows the ID attached to the corresponding node of the tree generated when the pair \(\text{Advertise/Participate}\) is executed. With method \(\text{Disc(ID)}\), each agent can release the channel and leave from the group. When method \(\text{Disc}\) is executed, exception \(\text{DiscException}\) is notified to the other agents of the group.

### 2.3 Group Communication by Multi-way Synchronization

Hereafter, the input/output from/to each channel is called the event. \(g?x[p(x)]\) denotes an input event on channel \(g\) which receives the value from channel \(g\) and assigns it to variable \(x\) only if predicate \(p(x)\) holds (the event is not executed when \(p(x)\) does not hold). \(g!f(x)\) denotes an output event on channel \(g\) which outputs the value of expression \(f(x)\) to channel \(g\). In our model, member agents in the agent group communicate based on synchronous communication.

For example, assume that four agents \(A1, ..., and\ A4\) form a group whose synchronization tree is \((\langle A1||[g]|A2||[g]\rangle\ A3||[g, h]|A4)\). In this case, when \(A1, A2, A3\) and \(A4\) are ready to execute events \(g!0\), \(g?x[x \leq 3]\), \(g?y\) and \(g?z[z > 0]\), respectively, the tuple of events is executed according to the semantics of multi-way synchronization, and as a result, the output value “0” of \(g!0\) is assigned to the input variables \(x, y\) and \(z\) (here, the variable types of \(x, y\) and \(z\) must match the type of “0” and \(x \leq 3\) and \(z > 0\) must hold). Since channel \(g\) behaves as a data bus shared among these agents, multi-cast data transmission to all member agents is realized as shown in Fig. 2. Note that any data including a binary data block can be transmitted by multi-way synchronization.

On the other hand, when \(A1, A2, A3\) and \(A4\) execute \(h?u, h?v, h?t,\) and \(h!1\), either \(A1, A2\) or \(A3\) can execute the event on channel \(h\). For example, “1” is assigned to variable \(v\) when \(A2\) is selected to synchronize with \(A4\). Since channel \(h\) behaves as a switching bus between the sub agent group (consisting of \(A1, A2\) and \(A3\)) and \(A4\) as shown in Fig. 2, we can easily implement a mutual exclusion mechanism in accessing a common resource among competitors (agents).

Moreover, if each of \(A1, A2, A3\) and \(A4\) is ready to execute two events on channels \(g\) and \(h\) shown above as a choice, either the tuple on channel \(g\) or that on \(h\) must be selected nondeterministically.

**Primitives for Group Communication** In our middleware, we have defined method \textit{Synchronize} for multiple agents to communicate with each other through shared channels by multi-way synchronization. Method \textit{Synchronize} has a parameter corresponding to executable events. So, first an array of class \textit{Event} is prepared to store executable events, and then method \textit{Synchronize} is executed. Each event contains the channel name, input/output parameters, and the execution condition consisting of those parameters and constants. According to the semantics of LOTOS [4], one tuple of events is selected in an agent group, and returned as the instance of class \textit{Event}. The returned instance contains the list of synchronizing agents and the values assigned to input parameters as a result of synchronization. So, each agent can obtain the values from other agents in the agent group.
3 Basic Policy to Implement Middleware

When implementing each primitive of the proposed middleware of Sect. 2 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, in order to solve the above problems (1) and (2), 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.

![Diagram](image)

Figure 3: Outline of Middleware Architecture Using Servlet

3.1 Middleware Architecture Using Servlet

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 (Fig. 3(1)) and the other is an agent program executed on a server (Fig. 3(5)). 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 thread started by Servlet (Fig. 3(3)). And then, the agent and the UI communicate with each other by pseudo RMI (simulated by HTTP) through Servlet (Fig. 3(2)). Using the extension of our existing middleware (Fig. 3(7)), each agent performs group communication by multi-way synchronization, as shown in (Fig. 3(6)).

3.2 Load Distribution Using Two or More Servers

When a large number of users may join the same application system, we must consider the scalability issue, that is, it is not a realistic solution to load to a single server all the agents corresponding to all cellular phone users. So, in the proposed middleware, we adopt a simple load distribution mechanism using two or more servers. In the mechanism, when each user starts an application, the corresponding agent is loaded to one of the servers. In order to perform group communication at high speed, each agent is reloaded after the group formation so that all agents in the same group are executed on the same server.

As shown in Fig. 4, group formation over two or more servers is performed as follows: (1) An agent (A1) on a server (S1) which executes Advertise broadcasts a group member advertisement message (including a keyword list, user location, etc.) to all the agents in the user ID list which S1 manages. Simultaneously, the message is also sent to other servers (S2, S3, ..., Sn). Then, each server Si broadcasts the received message to all the agents of the user ID list which Si manages. (2) If an agent (A2) which executes Participate receives the message from A1, it judges whether the specified conditions (the context distance is less than the specified value, and so on) hold. When A2’s condition holds, A2 returns to A1 a participation request message (including a keyword list, user location, etc.). (3) A1 judges its condition with the values in the received message. If A1’s condition holds, A1 sends back to A2 an "ack" to permit A2’s participation to the
specified group. (4) When \( A_2 \) receives the "ack", \( A_2 \) moves to the same server as \( A_1 \) if \( A_2 \) is executed on the different server, and establishes the channel for group communication.

4 Implementation of Middleware

4.1 Implementation of Servlet

We have implemented class \textit{LotosMServlet} that invokes and manages the agents. This class also acts as mediator between the agent and the user interface.

\textbf{Implementation of Methods executeApp/executeMethod} We have implemented method \textit{executeApp} in class \textit{LotosMServlet}. Method \textit{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 \textit{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 \textit{LotosMServlet}.

We have implemented method \textit{executeMethod} to access the agents from each user's UI. Method \textit{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 \textit{executeMethod} picks up the agent identified by the ID and invokes the target method by using the reflection functions.

\textbf{Implementation of Communication Primitives} In principle, we would like to use the communication primitives of existing middleware such as \textit{Advertise} or \textit{Participate}. The existing middleware provides these primitives using under layer communication primitives modeled as (1) broadcast of messages for all the nodes in the radio range (for \textit{Advertise}) and (2) transmission of messages to specific nodes (for \textit{Participate}). However, in the proposed middleware, there is no concept of the radio range, we must have modified the under layer functions such that the agents invoked by method \textit{executeApp} are managed using a list of user IDs and the advertisement messages sent by method \textit{Advertise} are transmitted to all the agents in the list.

The construction of groups is executed as shown in Fig. 5. (1) The agent executing method \textit{Advertise} broadcasts advertisement messages (that includes keywords, location of the nodes, and so on) to all the agents whose IDs are listed in the server. (2) If the agent that is executing method \textit{Participate} receives the messages and the specified conditions (e.g., the context distance must be within the specified length) are satisfied, the agent will send back a participation request message. (3) The advertising agent will test the conditions and send back an "ack" message if the conditions are satisfied.

Our middleware constructs a synchronization tree for the group formed by method \textit{Advertise/Participate} as shown in Fig. 6. In the figure, \([g, h]\) \(A_2\) shows that agent \( A_2 \) must check the condition of synchronization (here we suppose that the advertising agent will become a responsible node).

\textbf{Implementation of Multi-way Synchronization} The agents can synchronize with other agents in the group by using
method \textit{Synchronize} provided by our middleware. Since the synchronization tree shown in Fig. 6 is equivalent to the behavior expression of LOTOS\cite{4}, we can derive what events can be executed synchronously among agents by evaluating synchronization condition at each operator node in the tree from the leaf to the root node based on the technique in \cite{14}. Each intermediate responsible node of the synchronization tree receives the list of executable events from the child nodes and evaluates executable combination of these events based on the specified synchronization relation. Then the responsible node will make the list of executable events on the sub tree that the node is its root node and send them to the parent node. By executing this process sequentially from the leaf nodes to the root node, all the executable combinations of executable events in the synchronization tree are listed up and the root node select one of them. The selected event is sent to all the nodes through the tree.

The existing middleware executes the above decision process by transmitting event information through TCP. In the proposed middleware, we have implemented these functions in two different ways; based on TCP communication and based on shared variables among threads. We have compared these two implementations in Sect. 6.

\textbf{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 request is sent to the Servlet like in Fig. 7(1). The request contain the parameters consisting of the name of the agent to invoke and the user ID. The Servlet will invoke method \texttt{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. 7(2). The parameter includes the name of the agent, the user ID, the name of method and the string that consists of comma separated list of the parameters for method invocation. The Servlet will invoke specified method and send back the result to the UI as an HTTP response.

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.

4.2 Snapshots when Executing Middleware by Cellular Phone Emulator

We have implemented a simple application with the proposed middleware where each cellular phone user can advertise for group members with a keyword, or participate in a group which was already advertised for by another user, and can multicast/receive a data to all members/from a user of the group. Here, we have used cellular phone emulators (ez-JADE for A3012CA) which implement most of the functions of the current cellular phone on the market (CASIO A3012CA). We have executed four emulators in parallel on one PC as shown in Fig. 8. For the terminal in the rightmost side, we set the different keyword from the other three terminals. We let the leftmost terminal sends the group advertisement request (Fig. 8), and also let the remaining terminals execute participation request (Fig. 9). Consequently, the group was formed by three terminals which set the same keyword (Fig. 10). Since the rightmost terminal set the different keyword, it could not join the group. Next, the leftmost terminal set the data and multicast it to the group members. Consequently, the data was transmitted to all of the terminals in the group (Fig. 11).
5 Example of Cellular Phone Application

Let us suppose the following situation at which users want to obtain or exchange information with other persons using the cellular phone.

**Typical situation in the flea market place** A cellular phone user (Koji) is in a flea market place. Since he was looking for some old clothes, he would like to obtain any information about old clothes from other persons in the market place. So, he set up a simple conference system (i.e., chat system) where only persons who are in the market place and are interested in “old clothes” can join the system and share information by chat writings. While executing the system, someone suddenly called him. So, he changed the system to the observer mode. After finishing the talk, he changed the system to the participation mode. Then, he saw in new chat writings “Rare Levis 501 Jeans are on sale at 150USD” which was written 3 minutes ago. From the location information attached to the chat writing, he could soon get to the shop of the message, and finally obtain the jeans, which he was looking for a long time. If there is no observer mode, he may have failed to hear the information, thus failed to buy the jeans.

The conference system explained above can be easily developed with the proposed middleware as shown in Fig. 12.

The system has two behavior modes: the participation mode (Fig. 13(1)) and the observer mode (Fig. 13(2)). The observer mode can easily be implemented by letting the agent on the server do not communicate with the UI on the corresponding cellular phone until switched to the participation mode. In this case, the agent only receives and stores chat writings transferred by multi-way synchronization from other members.

The whole system consists of one host user (called promoter) and an arbitrary number (n) of guest users (called participants). The promoter specifies as parameters the subject (given by a keyword list) of the conference and the condition about the geographical range and the maximum context distance which restrict the participants, and executes method Advertise with the parameters (i.e., broadcasts a group member advertisement message to all potential users listed in the server(s)). Similarly, each participant specifies as parameters his/her geographical location and the preferred subject (a list of keywords) and executes method Participate with the parameters. If the context distance between two keyword lists of the promoter and if the participant is within the specified value, and the participant’s location satisfies the condition given by the promoter, the participant becomes a group member and can participate in the conference. During the conference session, the promoter advertises for new members periodically by executing method Advertise.

---

*1Here, we assume that during the observer mode, the system automatically records chat writings so that they are transferred to the cellular phone when the system resumes.
When a new user participate in the conference, two channels, $g$ for control and $h$ for data transmission, are shared. Each participant can leave from the conference by executing method `Disc` (Fig. 12(3)).

We suppose that each agent is distinguished by the unique $ID$ and simultaneously at most one user can get into a chat session during which he/she can write sentences in real-time in the conference. When a user wants to get into a chat session in the conference, he/she tries to send a request `Talk` (Fig. 12(4)). When the user wants to listen a message in the conference, he/she tries to send a `Listen` request (Fig. 12(5)). These requests are set to an array of class `Event` and method `Synchronize` with the array is executed. A request `Talk` is described as choice of the output event (denoted by $g!ID$) of its $ID$ to channel $g$, and the input event (denoted by $g?id$) of $ID$ of the speaker from $g$. A `Listen` request is described as event $g?id$ which inputs a speaker’s $ID$ from $g$. The possible combination of events executed by multi-way synchronization on $g$ is only the case that only one user selects a request `Talk` and the other users selects `Listen` requests. However, when all users try to selects request `Talk`, the possible combinations will be $n$. In multi-way synchronization, since $ID$ of only one person is selected non-deterministically, finally one user is selected and gets into a chat session. The chat writings during a chat session are transmitted from the agent of the selected user to the other member agents through gate $h$ (Fig. 12(6)). Finally, the writings are transmitted from those agents to the corresponding users’ cellular phones and displayed on their screens (Fig. 12(7)).

A `Suspend` request is transmitted to leave from the conference temporally (Fig. 12(8)), and a `Resume` request is transmitted to resume (Fig. 12(9)). In this case, messages from other users are received and stored by the agent so that the messages are sent to the cellular phone after resuming.

6 Experimental Results

In this section, we carry out some experiments to investigate the performance of the proposed middleware: (1) overhead to calculate conditions to form a group, (2) performance of multi-way synchronization, and (3) performance of pseudo RMI from a cellular phone. For (2), we also investigate how much the performance is improved compared with our existing middleware by executing all agents on the same server. In the experiments, a PC with Athlon2200+, 512MB memory, Tomcat 4.1.24 (servlet container), and JDK1.4.1 were used as a server, and Mozilla on PC was used as a UI.

For the above (2), we have measured the performance of synchronous execution of the event by multi-way synchronization by $n$ members of the agent group when $n = 25$ and $n = 100$. We show the experimental result in Table 1 where TCP and the shared variable is used for exchanges of event information between agents to calculate executable tuples of events based on the synchronization tree (see Fig. 6). With TCP communication, when the number of agents is 100, the tuple of synchronous events by multi-way synchronization can be executed about once per second. From Table 1, we see that communication with TCP between agents becomes a bottleneck when implementing our existing middleware on Servlet as is.

On the other hand, our new implementation of the multi-way synchronization mechanism uses shared variables for
Figure 8: Advertising

Figure 9: Participating

Figure 10: A Group is Formed

Figure 11: Result of Multicasting

Figure 12: Example of Application (Electronic Conference System)
data exchange among agents. In this case, as shown in Table 1, even when the number of agents is 100, about 39 tuples of events can be executed by multi-way synchronization for every second. From the result, we think that using the proposed middleware, we can implement cellular phone applications including frequent user interactions with sufficiently high performance.

For the above (1), we have measured the time to calculate the context distance and compare it with the specified value for agent groups. Here, we have used \( a = 1, 2, 4 \) as the number of elements in each keyword list. Here, shared variables were used for exchanges of messages among agents. The result is shown in Table 2. From the result, even when the number of potential users registered in a server is 1000, the calculation and comparison of the context distance was within 1 second. We think this result is reasonable for practical use. However, when the number of potential users on a server becomes tens of thousands scale, it is necessary to adopt the load distribution mechanism, which was explained in Sect. 3.2.

Finally, for the above (3), we have measured the performance of our pseudo RMI that invokes an agent and a method of the agent from each cellular phone. For this experiment, we have developed a MIDlet as a UI and executed it in a real cellular phone (Hitachi, A5303H whose processor is SH-mobile). Here, dual PentiumIII 1GHz, memory 768M, Debian GNU/Linux 3.0, Apache 1.3.26, Java Servlet 2.0, and gjj (GNU libgcj) version 0.0.7 were used as the server. We have measured the time interval between the time when the result was received and the time when the cellular phone requested either an agent invocation or a method invocation via HTTP. Here, since the data size of a request message and an execution result are small enough, the measurement result can be considered as an overhead for the agent invocation and the method invocation without communication overhead. The experimental result is shown in Table 3. The overheads of the remote agent invocation and the remote method invocation from the UI was about 1330 msec and 1280 msec, respectively (these are average values when executing 10 times). From the result, we think that our middleware has practically sufficient performance for applications where the UI and the corresponding agent do not communicate so frequently.
7 Conclusions

In this paper, 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.

Part of the future work, we plan to evaluate the performance of the middleware using practical applications executed on real cellular phones. Moreover, we plan to extend the middleware to support various kinds of hosts such as PDAs with wireless LAN I/F and PCs in the Internet as well as cellular phones at the same time.

References


