

# A Multi-Agent Approach to a Distributed Schedule Management System

●Yuji Wada   ●Masatoshi Shiouchi   ●Yuji Takada

(Manuscript received June 11, 1997)

**More and more people are engaging in cooperative team activities over worldwide networks such as the Internet. The multi-agent technology is a framework suitable for constructing distributed applications to support these activities. Systems based on this framework have the necessary autonomy, independency, flexibility, and extensibility. We have focussed on the distributed schedule management, which is a typical application of the multi-agent technology. We have developed a system, IntelliDiary, based on this framework to show how the framework is used and how the characteristics of the framework enhance the functions of distributed applications.**

## 1. Introduction

As the popularity of worldwide networks such as the Internet increases, there are more and more opportunities for people to collaborate with each other over networks. Using communication through such networks, people can work as a team to complete projects even when they do not see each other face to face. For example, some people are now engaging in temporal team activities such as *virtual corporations* and *virtual enterprises*. In these activities, people make a temporary team and cooperate with each other for a common purpose. The members of a team can be members of different sections or even competing companies. As computers and communication through networks become faster, new types of collaboration over networks are appearing. To enable teamwork over networks to be conducted more smoothly and effectively, various types of groupware are needed to support collaboration over networks.

The *client-server technology* can be applied to construct such kinds of groupware. The client-server framework consists of a server which offers services and clients which request and use the services of the server. Servers and clients are not equal to each other in terms of functions. Using this framework it is quite simple to design and implement a system over networks. However, it is insufficiently flexible for applications over

practical networks having many servers, for example, the Internet. To enable these servers to collaborate, a single entity is required to flexibly change its behavior depending on its situation so that it behaves as a server in some cases and as a client in other cases. Application systems over networks require such a flexible feature since each application system simultaneously offers its own services and requests services offered by other systems. To achieve this feature, the client-server technology is not sufficient.

The *multi-agent technology* provides a framework which enables a system to behave as a server or a client according to the situation. In this framework, a distributed system is regarded as a collection of units called *agents*. Agents are autonomous and have their own objectives. Services of a system are realized by collaboration among its agents to achieve their own objectives. Agents offer services and request other agents to perform tasks. The *autonomy* of the agents makes them independent of each other; they have their own objectives and local states. With this feature, an agent can refuse to offer its services if the requests from other agents are likely to be impossible or do not fit its objectives. Agents do not simply send messages and requests to each other but *negotiate* with each other through communication to achieve their goals.

However, despite these features of the multi-agent framework, it is not so clear how the framework can be used to construct a distributed system. In our current work, we have focused on *distributed schedule management*, which is a typical application of the multi-agent framework. In this paper, we describe the benefits of the multi-agent technology and how the technology is applied to design and implement our **IntelliDiary** system. Finally, we describe which kinds of services are provided and how these services are realized in our system.

## 2. Distributed Schedule Management and Issues Regarding the Client-Server Technology

Distributed schedule management involves arranging group schedules such as group meetings among a group of persons whose schedules are managed in a distributed manner. Arranging group schedules means finding mutually unoccupied times and dates among attendees of the schedules.

It is described in Ref.1) that most scheduling systems over networks which are commercially available are based on the client-server framework. Such systems perform centralized management of schedule data or a centralized control to arrange group schedules. A server is a system which performs this type of centralized management for an entire system. A server manages the schedules of all users and information about who can use its schedule management service. Therefore, clients do not have to perform complicated tasks to manage schedules. Clients communicate only with their server to access services. In other words, clients form bridges between users and their server to access the schedule management service. Without a server, clients cannot offer any service by themselves. Even for manipulating personal schedules of a user, his client has to ask its server to access the schedules. This style is very convenient when a system is working on a local area network (LAN) such as an intranet or pri-

ivate network inside a company. As almost all services are offered by a single server, the overall system works fine as long as its server is maintained properly. To use a distributed schedule management service, all the user needs to do is to start up their clients to access a server.

However, in a client-server framework, clients always require their server. This is a drawback when a system works over a wide area network (WAN) such as the Internet. As a WAN spreads worldwide, communication over the network is often slow and unreliable. Under such environment, clients cannot always find their server and their responses to their users tend to be slow because of the slow and unreliable communication between clients and their server. Moreover, all clients communicate with a single server. As a result, servers tend to be heavily loaded and the communication between clients and their server becomes a bottleneck of the overall system.

*Privacy* of schedules is a primary issue of schedule management systems. If systems cannot properly control the privacy of users' schedules, users will never use the systems. The client-server framework assumes a server which controls the behavior of the whole system in a centralized manner. In other words, how privacy is maintained in the system mainly depends on the server. The privacy of users' schedules is controlled by managers of the server, and users should accept these managers. This is not such a critical problem if a system is for intra-company use. Recently, however, inter-company team activities such as virtual corporations and virtual enterprises are becoming popular. To effectively support such kinds of inter-company activities using networks, the members of a team can have an online meeting over networks and schedules for their project can be managed over networks. To achieve this, schedule management and arrangement services are required. However, the client-server framework is not suited to managing schedules that belong to personnel of different companies. If a server is introduced to manage schedules, the

participants will hesitate to register their entire schedules because they could contain important information that must not be leaked to people outside their own company. Thus, group schedules such as group meetings cannot be arranged properly with a system which has a server.

Another important feature is *scalability* of the system. To allow users to create group schedules with any other user, a system should be able to dynamically coordinate schedules among arbitrary users on demand. Scalability is an important feature for a distributed schedule management system to dynamically coordinate and arrange schedules among arbitrary users. Moreover, the topology and scale of worldwide networks are constantly changing. As a result, it is impossible to know exactly certain status information, for example, the total number of users, communication delay times, and reachability between communicating nodes. Every minute, new computers are connected to networks and new users begin to communicate through them. At the same time, computers are disconnected from networks and users stop using networks. Users of such networks cannot always expect to communicate with other computers and other users, and systems working over the networks must work independently of other systems. In the client-server framework, communication with a server is indispensable for clients to offer any service. When new users begin to use a system, its server should be reconfigured to allow the new users to use the system. This makes the client-server framework difficult to apply to dynamic networks. A distributed schedule management system is needed to flexibly adapt to dynamics such as changes in the number of users and work by itself without requiring the services of other systems.

There are many services available over networks, for example, ticket reservation, hotel reservation, flight schedule information. After registering business trip schedules, some users want to access these information and reservation services. To offer these services, a schedule manage-

ment system should be able to collaborate with other kinds of systems. Otherwise, the system itself should offer such services. In the client-server framework, a system offers only the same services as its server does. Users cannot access services which are not offered by the server. In the case where a server does not offer a service and cannot collaborate with other servers, the server should be modified to support the service. Generally speaking, modifying a server takes some amount of time and affects many of the users who access the server. Furthermore, such kinds of new information and reservation services are going to be offered over networks every moment. To effectively access these services, a system must have high *extensibility* to enable easy adaptation to changes of cooperation with other systems.

### 3. The Multi-Agent Technology

#### 3.1 Features of the Multi-Agent Technology

The multi-agent technology<sup>2),3)</sup> is a framework suitable for constructing distributed systems over worldwide networks such as the Internet. In this framework, a system is decomposed into units, and services of the system are realized by the behavior of each unit and interactions among units. Each unit is called an agent. An agent has its own objectives and acts to complete its objectives autonomously and independently of other agents. An agent offers its services to its user and to other agents, and at the same time, the agent uses services of other agents. When an agent finds that cooperation with other agents is necessary to complete its objectives, it looks for other agents to cooperate with. Agents do not simply collaborate, but negotiate with each other. The agents that are asked for help can decide whether or not to support the asking agent, depending on their situations.

The notion of an agent is not fixed yet. Therefore, an agent can have many features depending on the situation in which it is used. We focus mainly on autonomy, independency, and cooperability. With autonomy and independency, systems based

on the multi-agent framework do not require the centralized management and control necessary in the client-server framework. Moreover, autonomy and independency enhance extensibility of the system and the privacy of the information managed by a system. Cooperability enables the systems to dynamically construct temporary relationships with other agent systems for collaboration.

A system with the multi-agent framework consists of many kinds of autonomous agents which perform various tasks to complete their objectives and offer their services. Each agent works independently without being aware of the existence of other collaborating agents. In this framework, centralized management of data and shared information among agents are not necessary at all. Agents need to be aware of the existence of another agents only when they request the services of that agent. If they find other agents which are cooperative, they can access the services of those other agents. Even if they cannot find other agents, agents offer their own services which do not require collaboration with other agents. When starting cooperation, no agent and no system intercede with agents and no information is shared among agents. Any agent can coordinate collaboration among agents, and the collaboration is performed through communication between agents involved.

An agent can protect its local states and information from other agents. Each agent manages its local data independently of other agents, and this local data are manipulated only by the owner agent. The only way to access information maintained inside other agents is through communication among agents. Even if an agent is asked to cooperate by other agents, the asked agent autonomously decides whether or not it will help the other agents based on the intentions of its user. When an agent receives a request to access information it has, the requested agent can reject the request if its user does not allow the requesting agent to access the information. In this way, agents can autonomously control accessibility to

their own local states and information by sieving received requests.

An agent dynamically establishes temporary relationships to cooperate with other agents. Basically an agent performs its tasks autonomously and independently. When collaboration with others is found to be necessary to achieve a task, an agent looks for cooperative agents by sending messages. If cooperators are found, the agent establishes cooperative relationships with them. Agents are scalable; that is, they can dynamically decide which agents they will collaborate with and temporarily establish cooperative relationships with those agents on demand. New users can easily begin to use a system by starting up their own agents, and existing users can easily stop using a system by terminating their own agents. There is no difference between previously existing agents and newly joined agents, and no agent is adversely affected by an increase or a decrease in the numbers of users. That is, the whole system is scalable. Scalability is an important requirement for a system which works over worldwide networks.

Since agents are quite independent, each agent can modify its services without significantly affecting other agents. In principle, each agent acts by itself and communicates with other agents on demand. If an agent modifies its behavior to extend its ability, the effect of the modification is localized into communication parts of other agents. That is, agents communicating with the modified agent have only to modify the behavior of their message handling parts to use the modified services. Each agent can also add new services and even new agents can be introduced into the system in the same way. These effects are also localized into the communication parts of other agents for the new services. Autonomy and independency provide high extensibility of the system itself.

### 3.2 Application to Distributed Schedule Management

Agent-based schedule management systems have already been reported in Refs.4) and 5). How-

ever, they assume centralized management of users' schedules. Centralized management has the same problem with privacy of schedules as the client-server framework. Most systems based on the multi-agent framework can control privacy.<sup>1),(6)-(8)</sup> They arrange meeting schedules in cooperation with the agents of the attendees. However, they do not discuss about extensibility and cooperability with other agent systems. We show that the multi-agent framework can enhance these features with autonomy, independency, and cooperability of agents.

When the multi-agent technology is applied to construct a distributed schedule management system, the system is designed to consist of many kinds of agents. In such a system, an agent manages the schedules of its user autonomously and independently of other users' agents. Agents do not share any information between themselves. Unlike the client-server framework, the agents do not need a server which centrally controls the whole system and offers almost all of the services of the system. Instead, each agent manages the schedules of its user and communicates with other agents to collaborate, for example, when group schedules are arranged. Group schedules are schedules of events to which many users are expected to attend. To arrange such schedules, an agent must communicate with the agents of the attendees to negotiate whether the schedules are to be registered. The agent which starts the registration of a group schedule takes the initiative in negotiating for the registration. The initiating agent arranges the schedule in collaboration with the attendees' agents through communication.

Since the schedules of a user are managed by his agent and every agent can be an initiating agent to arrange group schedules, there is no agent and no system which always behaves as a server in the client-server framework, where the server offers services to manage personal and group schedules in a centralized manner. Autonomy and independency of agents enable a system to work without such a server.

An agent protects the schedules of its user from being referred to, registered, deleted, or modified by other agents directly. When a user accesses the schedules of another user, his agent sends a request to the agent of the accessed user. The requested agent checks who sent the request and judges whether the requesting user is allowed to refer to its schedules based on the intentions, preferences, or instructions of its user. In other words, users can easily control the privacy of their schedules with their responsibilities. This also means that users are personally responsible if their agents allow an access by a person to whom the user does not wish to permit access. Again, no server is required for the overall system and each agent works as an independent schedule management system. It is not necessary for all agents to be executed on the same machine, and there is no server and no information shared among agents. With the multi-agent framework, each user can control the privacy of his schedules independently of other users.

An agent dynamically finds agents with which to collaborate to refer to the schedules of their users or to arrange group schedules with their users. When a user asks its agent to perform a task which requires collaboration with other agents such as arranging group schedules, the agent establishes cooperative relationships with the agents on demand. Such relationships are only temporary, and an agent creates new relationships with the appropriate agents whenever a new request to manipulate schedules is processed. Agents have such scalability as to flexibly establish cooperative relationships with appropriate agents on demand. The number of agents is not mentioned in the discussion above since this does not matter in the multi-agent framework. A system with the multi-agent framework does not assume a static configuration of agents. An agent dynamically decides which agents to communicate with for collaboration according to requests of its user. Agents enable a system to flexibly adapt to changes in the number of agents.

New features are easily introduced into a system based on the multi-agent framework. There are two ways to do this: extending existing agents or adding new agents. Because of the independency of agents, the effects of extending existing agents to offer new features and adding new agents are localized into the communication parts of agents which use the new features. The addition of new agents enables, for example, collaboration with other kinds of systems. As there are many services available over networks, a system working over such networks is likely to be requested to extend its features to use those services. In the case of a distributed schedule management system, the system will be extended to support a service for accessing information such as cinema information, flight information, or hotel information. To support such services, intermediary agents are introduced into the system to make bridges to other systems. Intermediator agents enable agents and other systems to communicate with each other by converting protocols and message formats and forwarding the converted messages. When an agent accesses a service of a system, it sends a requesting message in its format to its intermediary agent. The intermediary agent converts the format of the received message to the one understood by the accessed system, and then forwards the converted message to the system. When receiving the reply, the intermediary agent converts the format of the reply to the one understood by the original sender, and then forwards the converted message to the sender. In this way, a system can collaborate with other systems. From the point of view of such a system, an intermediary agent for another system behaves the same as any other agent in the system, which send and receive messages in the format of the system and offer services accessible through communication. From the point of view of the other system, an intermediary agent for it behaves as a client which accesses its services by sending and receiving messages in its format. Intermediator agents make bridges between a system and other

systems in a simple and smart way. Thus, a system with the multi-agent framework can have high extensibility and can easily cooperate with other systems.

The multi-agent technology has some drawbacks compared with the client-server technology. In the client-server technology, all services of a system are provided by its server. Users are requested to install and start up their clients to access the services of the server. Services are successfully accessed as long as the server is properly maintained. Therefore, users do not have to maintain the system. Moreover, clients communicate only with their server. They do not communicate with each other since the server does everything for its clients. In the multi-agent technology, the services of a system are provided by each agent. Users must install and start up their agents to utilize the services. Users must maintain their agents to successfully access their services. Since agents are autonomous and independent, communication is indispensable for collaboration between agents. Consequently, agents require more communication than in the client-server framework when cooperation among agents is necessary.

**Table 1** summarizes a comparison of the multi-agent framework and the client-server framework.

Table 1. Comparison of the multi-agent framework and the client-server framework

	Client-Server	Multi-Agent
Centralized or distributed	Centralized	Distributed
Independency	No	Yes
Privacy	Fair	Good
Scalability	Low	High
Extensibility	Fair	High
Maintenance cost	Low	Medium
Communication complexity	Low	Medium

## 4. A Distributed Schedule Management System: IntelliDiary

### 4.1 Overview of IntelliDiary

One of the main purposes of our research is to effectively support cooperative activities over local and wide area networks.<sup>9)</sup> To achieve this, we first chose an example of distributed schedule management over such networks. This is a typical example of applying the multi-agent technology and evaluating its usefulness. **IntelliDiary** is a distributed schedule management system which is designed and implemented as an agent based on the multi-agent technology. With the framework, our system achieves privacy of schedules, scalability, and extensibility.

Each user starts up his IntelliDiary agent, and schedules of a user are managed by his agents. Each agent serves only its own user. If a user permits, his agent collaborates with the agents of other users to refer to other users' schedules or to arrange group schedules.

### 4.2 How IntelliDiary Works

Our system can be accessed with a WWW browser such as Netscape. The initial screen shows a calendar of the current month (see **Fig. 1**).

The “\*” mark indicates that there are sched-

ules on the day. If a mouse is clicked on the “Schedule List” button, a list of the schedules of the month are displayed. By clicking on a day with a “\*” mark, the schedules of the day are displayed.

**Figure 2** shows a schedule for a meeting at 13:00 in Fukuoka that is displayed when 22nd with a “\*” mark is clicked on the screen shown in **Fig. 1**. The user can edit or remove the schedule.

To arrange group schedules, the user can check his current schedules and those of the users who are expected to attend the scheduled events. The schedules of other users can be accessed by clicking on the Other Users button and specifying their names. In **Fig. 3**, two users “demo@bishop” and “wada” are specified. When the “Go!” button is clicked, the calendars of the month for the specified users are displayed in the same way as in **Fig. 1**. To make it easy to compare schedules of multiple users and find available time periods among them, their schedules for the day are aligned as shown in **Fig. 4**. After finding the free time of the attendees, a group schedule is created by specifying the names of all attendees as shown in **Fig. 5**. A schedule contains the following information: the date, start and end time, location of the event, attendees, subject, notes, and privacy. The privacy information specifies who is

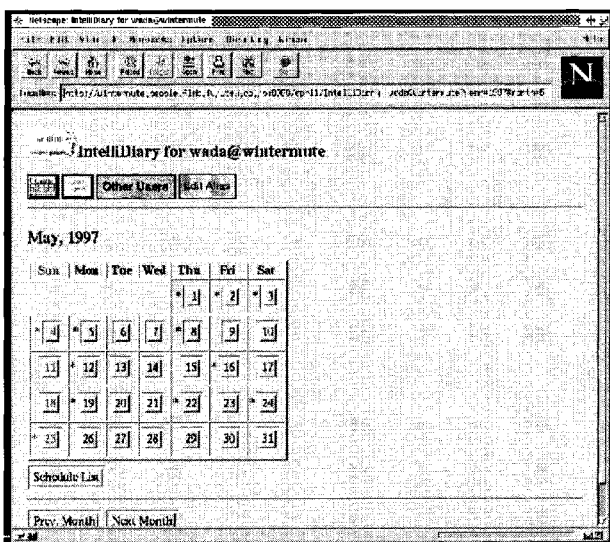


Fig.1– Calendar of the current month.

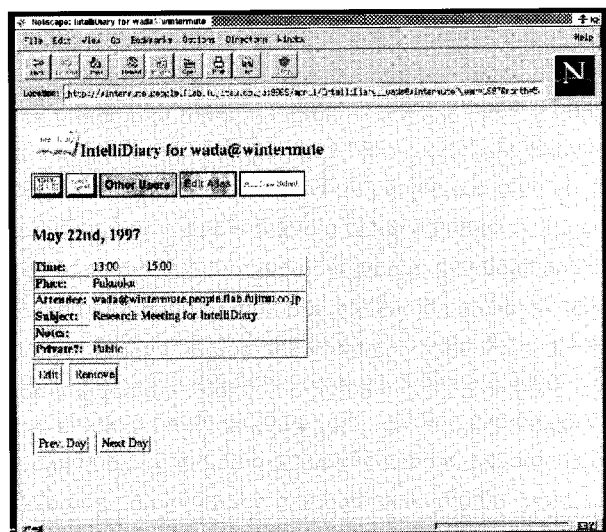


Fig.2– A schedule of a day.

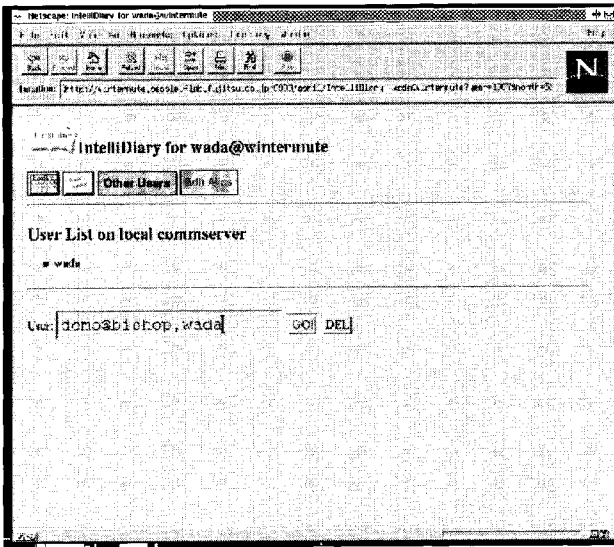


Fig.3- Specifying other users.

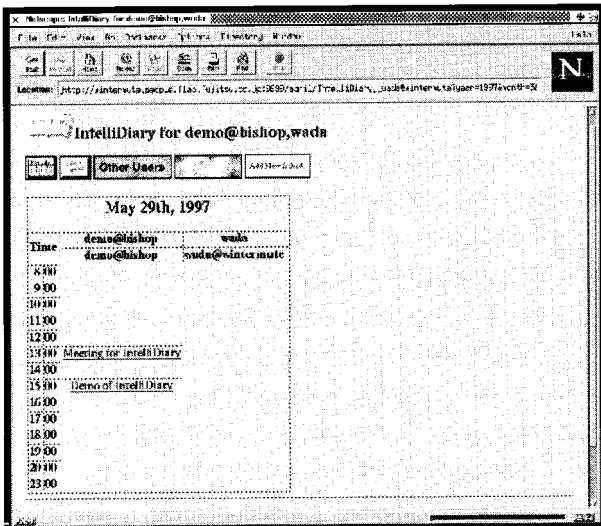


Fig.4- Day schedules of multiple users.

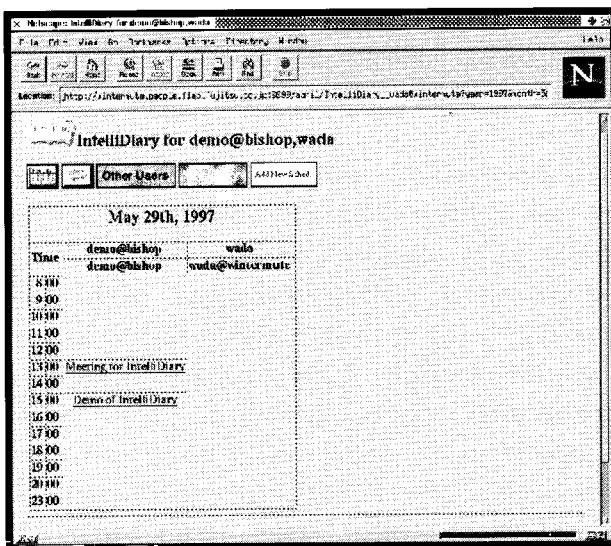


Fig.5- Form of a schedule.

allowed to access the schedule. In Fig. 5, “Private” and “Public” can be selected. “Private” means that the schedule is private and no one but the owner can access it. “Public” means that the schedule is accessible by everyone.

If all attendees to a schedule can register the schedule, it is registered to the **IntelliDiarys** of the attendees. If the schedule causes double booking against existing schedules, it is rejected. Then, our system proposes available time periods and dates of the attendees. **Figure 6** shows the screen displayed when a double booking is detected.

When accessing to other users’ schedules, a user specifies whose schedules he is going to access to. To specify a user, the name of his **IntelliDiary** agent is used. If a user is “foo”, then his agent can be specified with the name of “foo”. As there are many people on networks, different users could have the same name. To avoid this problem, our system adopts the *E-mail* style specification of users. If a user is “foo” and his schedules are managed on a computer named “bar.some.domain”, his agent is specified with the name “foo@bar.some.domain”. Inside our system, an agent name is prefixed with “IntelliDiary\_”, so the agent name of a user “foo@bar.some.domain” is “IntelliDiary\_foo@bar.some.domain”. However,

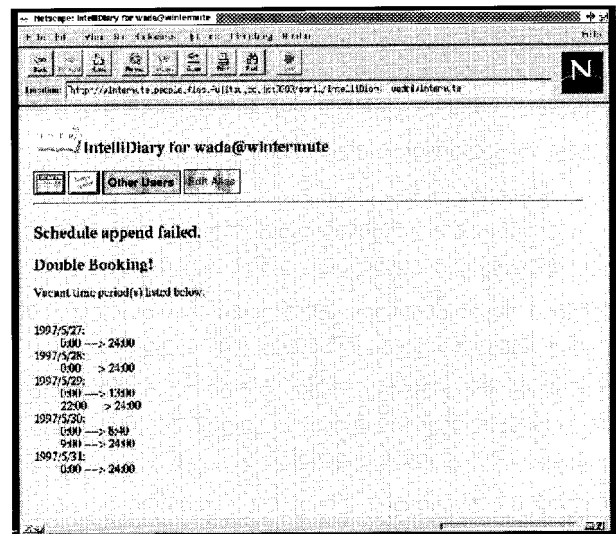


Fig.6- Available time periods and dates.

users are requested to specify only the names of users with whom to interact. When the names of users are specified, our system automatically transforms them into the names of the agents with which to collaborate. To make it easy to specify long names or repeatedly specify the same groups of users, our system has the same alias facility that is used in the *E-mail* system. Users can define and use their own aliases for the names of other users.

**IntelliDiarys** exchange messages in the *KQML*<sup>10</sup> format. Basically, *KQML* defines a protocol for communication and a format of messages: what kinds of messages are to be used for requests, what kind of information should be included in messages, how information is arranged in messages, and so on. All agents understand *KQML* messages and communicate with each other in such a uniform manner. When new agents are introduced into our system, they can communicate with existing agents if they understand *KQML* messages. The uniformity in communication enhances the simplicity of communication among agents and the extensibility of our **IntelliDiary** system.

### 4.3 Features of IntelliDiary

**IntelliDiary** has many features to manage users' schedules over networks. We describe how privacy of schedules is managed and how group schedules are arranged while maintaining privacy. Collaboration with other agent systems is also described. Finally, we explain the alias facility for user names.

#### - Privacy

Privacy of schedules is controlled by *Schedule Manager* according to the intentions of its user. Figure 5 shows the screen used to register new schedule events. The last attribute "Private?" specifies the privacy of the schedule. If "Private" is selected, the schedule is exported only to its owner. If "Public" is specified, the schedule is published to all users. When a user refers to other users' schedules or when group schedules are arranged, accesses to other users' schedules are nec-

essary. There is a trade-off between convenient facilities and privacy of schedule. Our system allows its user to specify which schedules are private and which are public. The contents of a private schedule such as when it starts and ends, location, attendees, and purpose, are exported only to its owner. As a default policy of our system, other users are allowed to obtain information such as whether a new schedule has been successfully registered into the schedules of a user and when a user is free. With such information, **IntelliDiary** arranges group schedules and finds available time periods and dates of users.

#### - Group schedules

Group schedules are schedules which have more than one attendee. Users create a group schedule by specifying its attendees as the "Attendee" attribute shown in Fig. 5. To register a group schedule, the **IntelliDiarys** of all attendees cooperate with each other. The **IntelliDiary** which is trying to register a group schedule coordinates the registration. If an **IntelliDiary** of an attendee causes double booking, the schedule is not registered. In this case, the initiating agent asks all of the attendees' agents to get the times and dates near to the originally planned time slot for which the attendees have no schedule.

Such collaboration among **IntelliDiary** agents is dynamically achieved when a group schedule is going to be registered. At first, a user specifies some users as attendees to a group schedule. His agent coordinates arrangement of the schedule by sending messages to the attendees and asking for the collaboration. From the point of view of the coordinating agent, it establishes temporal relationships between the attendees' agents. Each relationship is similar to the relationship in the client-server framework. In this case, the coordinating agent behaves as a server which offers an arrangement service of group schedules and the agents of its attendees are clients which utilize the service. In our system, every agent can be a coordinator of a group schedule. That is, every user can create a group

schedule. Moreover, attendees to a group schedule are dynamically specified when the group schedules are registered. Therefore, all **IntelliDiary** agents are designed to behave as a server and a client and to establish client-server relationships among agents on demand.

- Collaboration with other agent systems

With our system, users can access the services of *SAGE*.<sup>11)</sup> *SAGE* is an agent oriented system which allows access to public information such as flight schedules, hotel reservation information, train ticket information, and so on. Since this information is useful for planning schedules, collaboration with *SAGE* enables **IntelliDiary** to offer more convenient schedule management services. Also, through cooperation with *DUET*,<sup>12)</sup> the current location of a user can be detected. *DUET* offers an agent based support for personal communication under various situations. It senses current information about a user, such as the user's location, communication capabilities, and so on. Based on this information, *DUET* decides how to communicate among users. By using the current location information of a user offered by *DUET*, **IntelliDiary** offers a smart navigation service<sup>13)</sup> which, for example, automatically notifies the user about schedules that the user has sufficient time to travel to.

Collaboration with these systems is achieved by introducing an agent called an *intermediator*. *Intermediators* of *SAGE* and *DUET* enable our system to communicate with these systems. *Intermediators* converts protocols and message formats properly. *Intermediators* understand messages in the format used in our system. When an *intermediator* for a system communicates with agents in our system, it acts on behalf of that system and provides access to its services. An *intermediator* for a system also behaves as a client which accesses the services of the system through communication with messages in the format of the system. When **IntelliDiary** cooperates with new systems, our system can be extended by introducing *intermediators* for them.

- Aliases for user names

When specifying other users, **IntelliDiary** requires a name such as "foo@bar.some.domain", which means **IntelliDiary** of "foo" is on a machine named "bar.some.domain". To make it easier to specify a user, the user can define a simple alias for users and then use the alias instead of the full name of the users. An alias can also be defined with names of users and of aliases themselves.

#### 4.4 Configuration of IntelliDiary

Each user starts up his own **IntelliDiary** to maintain schedules, and **IntelliDiaries** collaborate with each other to access other users' schedules or to arrange group schedules. From a macroscopic point of view, each **IntelliDiary** is an agent which constructs a distributed schedule management system.

Personal schedules are managed by each **IntelliDiary** agent, and group schedules are arranged in cooperation with those of the attendees.

**IntelliDiary** consists of many agents which have their own objectives and offer various functions to each other. The services of each **IntelliDiary** are realized through a combination of the behavior of the agents and interactions among them. **Figure 7** shows the internal configuration

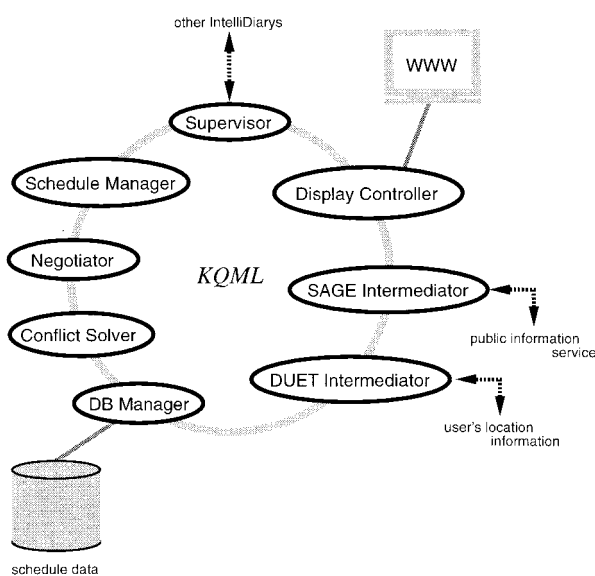


Fig.7- Internal configuration of IntelliDiary.

of our system and the agents which play main roles in our system.

- *Supervisor* agent

*Supervisor* behaves as an interface to other **IntelliDiary**. All incoming messages from other **IntelliDiarys** are received by this agent. When *Supervisor* receives a message, it interprets its contents, creates new messages according to the contents, and sends the new messages to the appropriate agents in the system. All incoming messages are delivered to *Supervisor* since only its name is exported to other **IntelliDiarys** in order to be used for communication. That is, messages to "IntelliDiary\_foo" are delivered to the *Supervisor* of foo's **IntelliDiary** and there is no other name which can be used to communicate with agents of foo's system.

- *Schedule Manager* agent

*Schedule Manager* manages schedule data by using *Negotiator*, *Conflict Solver*, and *DB Manager*. All requests which need to access schedule data, such as for referring to, creating, deleting, or modifying schedules, are sent to *Schedule Manager*. To cooperate with other users, users are requested to export information such as whether new schedules have been successfully registered and the time periods and dates when the user is available. Unless information is exported to other users, it is impossible to properly arrange group schedules. However, users hesitate to publish all of their information since there can be private schedules maintained with our system. There is therefore a trade-off between the convenience of group schedules and the need for privacy of schedules. In our system, users can specify which schedules are private or public. As a default policy, **IntelliDiary** exports information such as whether new schedules cause double booking and when its user is free. *Schedule Manager* checks who is allowed to access its schedules based on information included in each schedule.

- *Negotiator* agent

Registering new schedules can cause double booking; that is, more than one schedule occupies

the same time period. If a new schedule is found to cause double booking, it is not registered and **IntelliDiary** notifies its user of the double booking and the times and dates when the attendees will be free. *Negotiator* checks whether new schedules will cause double booking with existing schedules. To check the schedules of its own user, *Negotiator* asks *DB Manager* to access the schedule data it manages. If a new schedule is a group schedule, *Negotiator* of the user who is going to register the schedules takes the initiative in negotiating with the attendees' **IntelliDiarys**. To check the schedules of other users, *Negotiator* sends messages to the attendees' systems to arrange group schedules through collaboration. As mentioned above, these messages are received via the *Supervisors* of the attendees' systems. If the system detects double booking against the schedules being registered, the schedules is not registered and *Conflict Solver* calculates the available time periods and dates of the attendees.

The negotiation process is not complicated. First, an agent registering a new schedule checks the schedules of its user to see whether the schedule will cause double booking. Then, if the schedule has multiple attendees, the agent asks the other attendees' agents to perform similar checks.

- *Conflict Solver* agent

**IntelliDiary** itself does not resolve double booking. When double booking is detected, our system proposes available time periods and dates around the originally planned date. To resolve double booking, one or more schedules need to be canceled or moved to other time periods and dates. In general, it is not so easy to decide which schedules a user will allow to be cancelled or moved. Moreover, moving schedules can cause other double booking. If there are free times and dates and attendees agree with the schedule being moved, double booking is easily resolved. Otherwise, double booking presents many difficulties.

*Conflict Solver* finds the available time periods and dates of attendees in collaboration with their **IntelliDiarys**. The collaboration is done in

the same way as when registering new group schedules. The *Conflict Solver* of a user who tries to register a rejected schedule takes the initiative in cooperation with the attendees' **IntelliDiarys**. The initiating agent finds the times and dates by accessing the schedules of its user that are managed by *DB Manager*. Then, the agent asks the attendees' agents for the attendees' free time slots. The attendees' agents do not tell contents of their schedules to maintain privacy of their users' schedules but tell free times and dates of their users as a default policy. After collecting this information, the initiating agent finds the time slots for which all the attendees are free.

Our system adopts a simple method to find available time periods and dates. First, an initiating agent asks agents of attendees to find out when their users are free. Then, it collects the results from these agents. Then, using the results, the initiator finds the time slots for which none of the attendees have any booking.

- *DB Manager* agent

*DB Manager* maintains its schedule data using a database engine. *DB Manager* encapsulates the database engine it uses. There can be differences between database engines, for example, they may use different protocols and operations to manipulate data. This is not a problem however since *DB Manager* encapsulates the database engine it is using by providing the agents of its system with a uniform interface so they can manipulate schedule data without being aware of a concrete database engine used.

- *Display Controller* agent

**IntelliDiary** is expected to be used in various kinds of environments, for example, it can be used on a powerful workstation with sophisticated graphics or on a portable terminal with a simple character display. To adapt to these differences, the user tells *Display Manager* which type of terminal is used. Then, *Display Controller* adjusts the way information from **IntelliDiary** is displayed according to the information from the user. In the current version of **IntelliDiary**, three types

of user interfaces are provided. The first is a character based display for terminals with only simple graphics abilities. Schedules are displayed in the text format using new lines, tabs, symbol characters, and so on. The second is a WWW browser based display for users who use **IntelliDiary** services via networks. WWW browsers are getting common as navigation tools over networks and many services are already accessed with such tools. The last is an X window based display for computers running the X Window system.

- *Intermediator* agent

*SAGE Intermediator* and *DUET Intermediator* are *Intermediator* agents which make bridges to *SAGE* and *DUET* so that their services can be accessed.

#### 4.5 Future Directions

To construct a distributed system, there are many issues to be solved, for example, privacy control for the information maintained by the system, flexibility and scalability, and extensibility for easy collaboration with other systems. To solve these issues using the multi-agent framework, we focused on a typical application of the framework and constructed a distributed schedule management system called **IntelliDiary** based on the framework. As has been described in previous sections, **IntelliDiary** solves these issues using the framework. As a result, the configuration of our system is suitable for a WAN such as the Internet. There are various possibilities to extend our system and some of them are being investigated.

- Mobile environment

As a portable PC with networks through a portable handy phone is going to be widely used these days, cooperative activities under mobile environment are desired to be supported by distributed tools like **IntelliDiary**. A key characteristic of the mobile environment is network availability. Under mobile environment, the agents working on the computers are sometimes unable to communicate with other agents since mobile computers can become disconnected from networks.

Therefore, to adapt to a mobile environment, **IntelliDiary** has a dual configuration.<sup>13)</sup> As shown in **Fig. 8**, *Ego* and *Alter Ego* play main roles in adapting to the mobile environment. *Ego* is an agent with all the functions of **IntelliDiary** and must be run on a machine which is permanently connected to a network. In most cases, *Ego* behaves as an **IntelliDiary** agent which manages the schedules of its user. *Alter Ego* is an agent with the functions required for **IntelliDiary** to manage the schedules of its user and works on machines which can be disconnected from networks. *Alter Ego* has a subset of the functions of its *Ego* and a copy of the schedule data maintained by its *Ego*. When *Alter Ego* cannot communicate with its *Ego*, *Alter Ego* behaves as an **IntelliDiary** agent and modifications to schedule data are saved in *Alter Ego*. When *Alter Ego* finds its *Ego* on a network, it sends the agent the saved history of modifications. In this way, consistency between the schedule data of *Ego* and *Alter Ego* is maintained.

- Navigation

**IntelliDiary** notifies the user about a schedule event when the start time of the event is approaching.<sup>14),15)</sup> Our system uses the current location of its user to give the notification in a smart way. In collaboration with *DUET*, the current location of its user is known. With this information and the destination where the next schedule is

requested to be performed, the time required to move from the current location to the destination is estimated. Then, based on the estimated time, **IntelliDiary** gives the timely notification to the user so the user can arrive on time.

The current location of a user is also used to check whether schedule is feasible. For example, when new schedules are created, our system checks the current time, the current location of its user, and the start times of the schedules. If there is no time to reach the locations of the schedules, a warning message such as “It is impossible to complete the schedule” is given.

- Event information management

The framework of **IntelliDiary** can be used to manage various kinds of information that has time and date attributes. One example of such management is event information management. For instance, cinemas have their schedules such as what movies are screened and when they are released. By registering such information of each cinema to **IntelliDiary**, the screening plans of cinemas can also be managed with our system. An event map of cinemas, which is a collection of information of cinemas, can be dynamically created in cooperation with **IntelliDiarys** of cinemas. **IntelliDiary** collects cinema information in cooperation with other **IntelliDiarys** in the same way it collects users’ schedules. The only difference is in the contents of the information collected.

- Role information management

The alias facility can be extended to manage role information, which identifies users in terms of their roles in organizations, for example, worker, project leader, manager, and so on. These public names are regarded as aliases of the person. In that sense, agent can have several names according to their services. If one agent offers several services, then the agent can be specified with any of several names, each of which represents its corresponding service. That is, each service has its suitable name and these names are aliases of the agent which offers the services. Role information management makes it possible to spec-

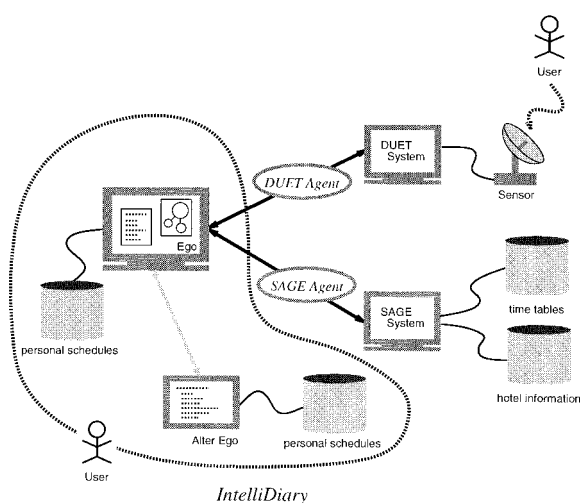


Fig.8– Overall configuration of IntelliDiary.

ify agent with names that reflect the situations in which the agents are used without needing to know their real names.

- IntelliTeam project

We are currently working on a project called *IntelliTeam*.<sup>9)</sup> The main aim of this project is to help people efficiently cooperate as a team over networks. To achieve this, we are designing and implementing some groupware tools working over networks based on the multi-agent framework. *IntelliTeam* includes a schedule management tool **IntelliDiary**, a work-flow management tool, a location information management tool, and so on. These tools are designed to cooperatively work over a WAN such as the Internet.

In this project, **IntelliDiary** assists in managing personal schedules and arranging group schedules over networks to support cooperative activities over networks. With our system the users manage their own schedules independently of others and arrange group schedules by collaborating with the systems of other users. To achieve more effective support for cooperative activities, **IntelliDiary** should be able to collaborate with other tools of our project more smoothly and offer convenient combinations of services.

## 5. Concluding Remarks

In this paper, we discussed the multi-agent technology and how the technology can be applied to design and implement a distributed system. The multi-agent technology is suitable as the base of a distributed system. Autonomy and independence enable agents to control the privacy of information maintained in the agents and enhances scalability and extensibility of systems with the multi-agent framework. However, how to construct such a system with the technology has not been sufficiently explored. We showed how the framework can be used to design and implement a distributed system using an example of a distributed schedule management system called **IntelliDiary**. We also showed that the framework is suitable for constructing systems over a WAN

such as the Internet.

The number of people who use computers and networks is growing rapidly, and cooperative activities over networks are getting more important. To enhance utilization of the networks and support these activities over the networks, we should continue to work on the topics addressed in this paper and find which kinds of services are needed and how they can be easily provided using the multi-agent technology.

## References:

- 1) Haynes, T., Sen, S., Arora, N., and Nadella, R.: An Automated Meeting Scheduling System that utilizes User Preferences. The First International Conference on Autonomous Agents, Feb. 1997, pp. 308-315.
- 2) Wooldridge, M. and Jennings, N.: Intelligent Agents - Theories, Architectures, and Languages, Vol. 890 of Lecture Notes in Artificial Intelligence. Springer-Verlag, 1995.
- 3) Wooldridge, M. and Jennings, N.: Intelligent Agents II., Vol. 1037 of Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996.
- 4) Jennings, N. and Jackson, A.: Agent-based Meeting Scheduling: A Design and Implementation. Electric Letters, The Institute of Electric Engineering, **31**, 5, pp.350-352 (1995).
- 5) Sandip, S. and Durfee, E.: The Role of Commitment in Cooperative Negotiation. International Journal of Intelligent Cooperative Information System, **3**, 1, pp.67-81 (1994).
- 6) Sen, S. and Durfee, E.: A Formal Analysis of Communication and Commitment in Distributed Meeting Scheduling. The 11th International Workshop on Distributed Artificial Intelligence, 1992, pp.333-344.
- 7) Sen, S. and Durfee, F.: PashaII-Personal Assistant for Scheduling Appointments. The First International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology, Apr. 1996, pp.523-542.

- 8) Belakhdar, O. and Ayel, J.: Meeting Scheduling: an Application for Protocols Driven Cooperation. The First International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology, Apr. 1996, pp.25-44.
- 9) McCabe, F. G.: IntelliTeam - managing projects in the 21st century. FUJITSU Sci. Tech. J., **32**, 2, pp.224-237 (1996).
- 10) Finin, T. et al.: Specification of the KQML Agent-Communication Language, DRAFT Paper, 1993.
- 11) Masuda, R. and Maruyama, F.: Ontology for Database Access. AAAI 1997 Spring Symposium Series, Mar. 1997.
- 12) Iida, I., Nishigaya, T., and Murakami, K.: DUET: An Agent-Based Personal Communication Network. IEEE Communications Magazine, **33**, 11, pp.44-49 (1995).
- 13) Wada, Y., Kawamura, A., McCabe, F.G., Shiouchi, M., Teramoto, Y., and Takada, Y.: An Agent Oriented Schedule Management System - **IntelliDiary**. The First International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology, Apr. 1996, pp.655-667.
- 14) Takada, Y., Mohri, T., Ichiki, H., Shiouchi, M., Wada, Y., and McCabe, F.C.: A Multi Agent Model for Schedule Navigation with Location Information. The First International Conference on Autonomous Agents, Feb. 1997, pp.532-533.
- 15) Mohri, T., McCabe, F.G., Wada, Y., and Takada, Y.: Using Location Information to Guide Scheduling. The Second International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology, Apr. 1997, pp.532-533.



**Yuji Wada** received the B.S. and M.S. degrees in Information and Computer Science from Osaka University in 1990 and 1992, respectively. He has been a member of Fujitsu Laboratories Ltd. since 1992. His current research interests are the multi-agent technology and distributed applications. He is a member of the IPSJ and JSSST.

E-mail : wada@flab.fujitsu.co.jp



**Yuji Takada** received the B.A. and M.A. degrees from the Department of Behavioral Science in 1983 and 1985, respectively and the Dr. Eng. degree from the Department of Information Engineering, Hokkaido University in 1993. Since 1985, he has been with Fujitsu Laboratories Ltd. In 1994 he was a visiting researcher at the Department of Computing, Imperial College, U.K. His current research interests include multi-

agent systems, distributed computing, groupware, and machine learning. He is a member of EATCS, IPSJ, JSAI.

E-mail : yuji@flab.fujitsu.co.jp



**Masatoshi Shiouchi** received the B.E. and M.E. degrees in Information and Computer Sciences from Kyushu University, Fukuoka, Japan in 1985 and 1987, respectively.

He joined Fujitsu Laboratories Ltd. in 1987 and has been engaged in research and development of machine translation systems, natural language processing, and intuitive search systems. His current research interest is

multi-agent systems. He is a member of the IPSJ.

E-mail : shiouchi@flab.fujitsu.co.jp