The Third International Conference on Internet and Web Applications and Services

Persistency for Java-based Mobile Agent Systems

Shinichi Motomura
Information Media Center
Tottori University
4-101, Koyama-Minami
Tottori, JAPAN
Email: motomura@tottori-u.ac.jp

Takao Kawamura, Kazunori Sugahara
Faculty of Engineering
Tottori University
4-101, Koyama-Minami
Tottori, JAPAN
Email: {kawamura,sugahara}@ike.tottori-u.ac.jp

***Abstract*—In this paper, we present mobile agent systems supporting persistency. In order to develop the mobile agent systems, a mobile agent framework has to have functions to support persistence of agents and persistence of an agent runtime environment. Our developed Java-based mobile agent framework named Maglog is taken up an example of a mobile agent framework. Maglog consists of three basic components, which are agents, agent servers and fields. Agent server and fields are corresponding to an agent runtime environment. The effectiveness of persistency is confirmed through descriptions of two mobile agent systems: a distributed e-Learning system and a scheduling arrangement system.**

I. Introduction

Recently computing systems have been usually composed of computers connected by a network, and that is called a distributed system. Mobile agent technology is an important issue to construct distributed systems. An advantage of mobile agent technology is that overall system performance can be improved because mobile agents are moved from heavily-loaded to lightly-loaded computers. Moreover, mobile agent technology can improve performance by exploiting parallelism without the usual intricacies related to parallel programming. For example, it is searching for information in the Web. By using several copies of a mobile agent that move from site to site, improving of performance compared to using just a single program instance can be achieved. In addition to improve performance, mobile agent technology improves flexibility. Because a mobile agent moves between different computers, a computer can acquire code that is not on the computer when necessary. Therefore systems can be configured dynamically.

In order to develop mobile agent systems, a mobile agent framework is usually needed. A mobile agent framework consists of programming language, libraries and an agent runtime environment (hereafter referred to as ARE).

In a mobile agent system, a number of autonomous agents cooperate mutually and achieves given tasks. These agents are spread on some computers and migrate among these by computer networks. However, a number of agents concentrate on one computer according to the circumstances. For execution of these agents on one computer at same time, a large amount of resources such as mass memory capacity and a fast CPU are required. Consequently, under the limited resource circumstances, some agents on the computer should be swapped out to second storage such as a hard disk drive.

In order to realize swapping out agents, a mobile agent framework has to support persistence of agents. Many mobile agent frameworks have been studied, and several mobile agent frameworks, such as Aglets[1] and MobileSpaces[2], support persistence of agents. However, in those mobile agent frameworks, when an agent executes a program which describes the behavior of the agent, the agent can suspend but can not resume from the stopped point. Because these frameworks support only weak mobility, they cannot save execution states. In order to save execution states, a mobile agent framework must support strong mobility, which involves the transparent migration of an agent's execution state as well as its program and data. The persistence of agents that we require is described as follows.

- Suspend
  When an agent suspends, the agent and its suspended state are written in hard disks.
- Resume
  When the agent resumes, the agent restarts from the stopped point.

In general, it is necessary to consider suspending systems. Because a system must suspend processing when a power failure occurs or the operating system is patched up. Systems, such as the Web system, do not need time for processing in many cases, and so it is not much necessary to consider suspending systems. However, it is often that an agent needs comparatively long time for processing in mobile agent systems. Therefore, it is important to consider suspending systems. If a system which is developed by a mobile agent framework tries to suspend, not only agents but also an ARE has to support persistency. However, there have been no mobile agent frameworks which support persistence of an ARE.

Accordingly, we have examined that persistence of agents and an ARE has been introduced into a mobile agent framework which is implemented on the Java platform. Maglog[3] which is our proposed mobile agent framework is took up as an example of a strong mobility supported mobile agent framework. It is based on Prolog and is implemented by extending PrologCafé[4], which is a Prolog-to-Java source-to-source translator system.

In this paper, the design and implementation of persistency in Maglog are explained. Moreover, the usages of persistency

978-0-7695-3163-2/08 $25.00 © 2008 IEEE
DOI 10.1109/ICIW.2008.72

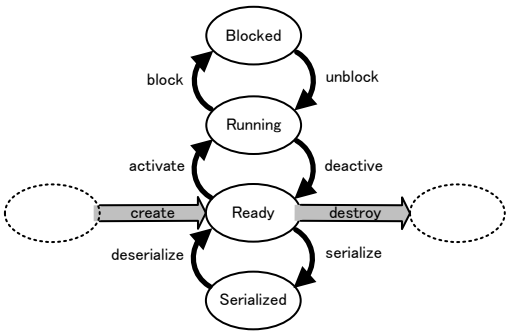

Fig. 1. Agent's life cycle.

in two distributed applications which are developed by Maglog are described.

II. Design

A mobile agent framework consists of agents and an ARE in general. Agents and an ARE consist of Java objects in mobile agent frameworks, which are implemented on the Java platform. An agent is an autonomous program and a thread on an ARE. The characteristic of an ARE is to provide the following functions for agents:

1) Management agents, such as creation and destruction,
2) Communication with other agents,
3) Migration to another computer in a network,
4) Accessing an operation system's functions, such as reading/writing files.

The remaining of this section will explain each of how to realize persistence of agents and how to realize persistence of an ARE.

A. Persistence of Agent

1) Suspending/Resuming Timing
   Figure 1 shows an agent's life cycle while an agent is destroyed after the agent is created. To begin with, an agent is created, that is, the objects of the agent are created. In this stage, the agent does not have a thread, so that the agent cannot execute procedures. The state of the agent is `Ready`. Next, in order to execute procedures which the agent has, a thread is assigned to the objects of the agent, and so the agent becomes `Running`. If the agent waits a message from other agents, the thread stops, and so the agent becomes `Blocked`. When the blocked agent receives the message, the thread runs again, and so the agent becomes `Running` again. If an agent suspends, the objects of the agent are serialized then are saved in files. When the agent resumes, the files are loaded, then the objects are deserialized. Agents run autonomously, so that agents can suspend when only agents are `Running`. There are the following situations when agents try to suspend.
   a) An agent executes a procedure to suspend the agent.
   b) An agent receives an offer of which other agents request to suspend the agent.
   c) An agent receives an order of which an agent server requests to suspend the agent.

   If an agent receives a suspending order from an agent server when the agent is `Blocked`, the agent becomes `Running` again then suspends. After the agent resumes, the agent is `Blocked` again.
   On the other hand, an agent is resumed by an agent server in the following situations.
   a) A user or another agent requests to resume the agent.
   b) An agent server decides to resume the agent. For example, if an agent server suspends agents which are under low load, the agent server will resume the agents later.
2) Required Mechanisms
   Agents consist of Java objects. In order to realize persistence of agents, Java objects are serialized. The general contract for serializing Java objects is specified in the Java Object Serialization Specification[5]. Object Serialization is a mechanism built into the core Java libraries for writing a graph of objects into a stream of data. However, Object Serialization is not enough. If no custom Object Serialization is used, objects of an agent may not be deserialized on a host where the agent was not created. Because the class descriptions of the objects may not be in the host. Therefore, Object Serialization is customized with dynamic class loaders. A dynamic class loader contains bytecodes of Java classes of an agent and loads the Java classes if necessary. Details of the custom Object Serialization and the dynamic class loader are described in another paper[6].
3) Steps of Suspending/Resuming
   Figure 2 shows an overview of persistency mechanisms which is implemented in an ARE. AgentController manages agent's life cycle in Fig. 1. When an agent executes a procedure to suspend, the agent calls a function in the ARE on which it is running, the following steps are performed.
   Step1. AgentController receives the call then dispatches the agent to AgentSerializer, which implements the custom Object Serialization.
   Step2. AgentController stops the thread of the agent.
   Step3. AgentSerializer retrieves the dynamic class loader object from the agent. The dynamic class loader object is serialized then is written in a file.
   Step4. AgentSerializer serializes objects of the agent then writes in a file.
   Step5. In order to be able to resume, AgentController knows the suspended agent's information, such as an agent's identifier and about above files.

   When an ARE resumes an agent, the following steps are performed.
   Step1. AgentController gets the dynamic class loader

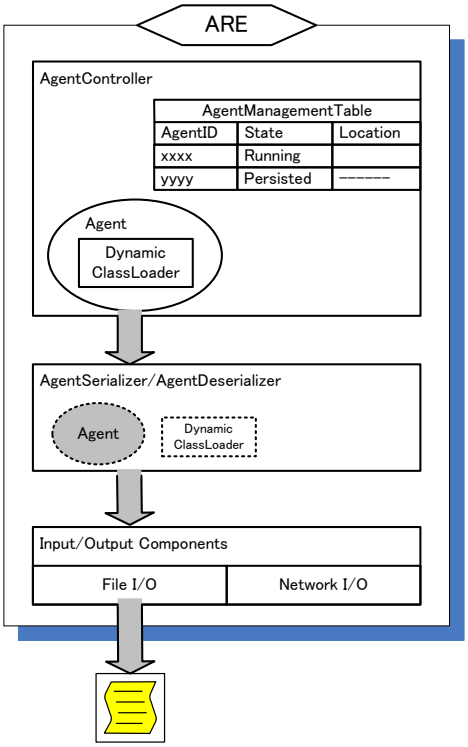

Fig. 2. An overview of agent's persistency mechanisms.

object which is deserialized by AgentDeserializer.

Step2. AgentController gets objects of the agent which are deserialized by AgentDeserializer with the dynamic class loader object.

Step3. AgentController runs a thread of the agent.

B. Persistence of ARE

An ARE is suspended when a user or an agent requests to suspend. Although an ARE also consists of Java objects, a whole of it cannot be serialized. For example, I/O components in Fig. 2 to provide network communication and accessing files cannot be serialized. AgentSerializer and AgentDeserializer are not necessary to be serialized. AgentManagementTable in AgentController and the other some components have to be serialized. When an agent server is suspended, the following steps are performed.

Step1. AgentController gives a suspending order to all agents then waits for until the all agents complete suspending.

Step2. The ARE serializes components which have to be serialized then writes them in files.

Step3. The ARE shuts down.

When a user wants to run an ARE again, the ARE is resumed. The resuming processes are performed as follows.

Step1. A user starts up the ARE with the parameter to resume.

Step2. The ARE reads the files then deserializes the components.

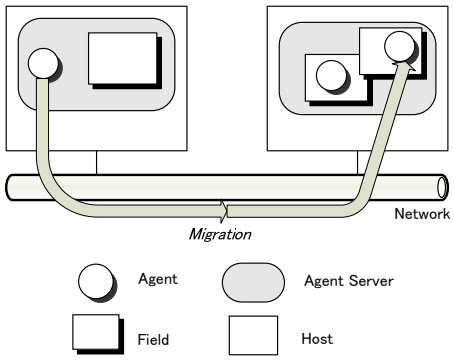

Fig. 3. Overview of a mobile agent system built using Maglog.

Step3. The ARE resumes, then AgentController resumes all suspended agents.

III. Implementation

We have implemented persistence of agents and AREs in our mobile agent framework named Maglog.

A. Overview of Maglog

Maglog is a mobile agent framework based on Prolog and is implemented on the Java platform. Figure 3 shows an overview of a mobile agent system built by Maglog. Maglog consists of three basic components, which are agents, agent servers and fields. An agent server and fields provide functions of as an ARE.

1) Agent
An agent has the following functions:
   a) Execution of a program that describes the behavior of the agent,
   b) Execution of procedures stored in a field where the agent is currently located,
   c) Communication with other agents through a field,
   d) Creation of agents and fields,
   e) Migration to another host in a network.

   An agent has Prolog interpreter because the agent resumes from its stopped point. If an agent is running on JVM directly without Prolog interpreter, when the agent cannot resume from the stopped point. Because Java API does not provide methods to access program counter and stack in order to access a thread's execution state. In contrast, Maglog has the WAM[7] as Prolog interpreter which is an abstract machine tailored to Prolog.

2) Agent Server
An agent server is a runtime environment that provides required functions for agents. For example, an agent server provides a migration function. When an agent migrates from host-A to host-B, the agent server on host-A suspends the agent's execution and transports the agent to host-B. After that, the agent server on host-B resumes execution of the agent. An agent server also manages

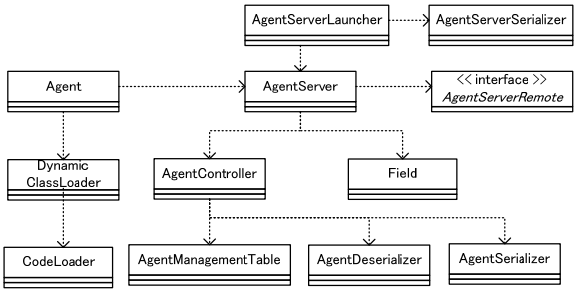

Fig. 4. A UML diagram which is an overview of classes of Maglog

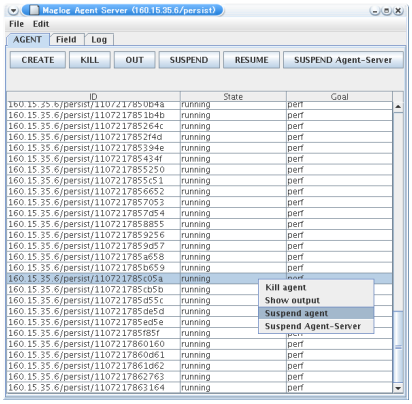

Fig. 5. A screen-shot of the user interface program when a user tries to suspend an agent.

fields and provides functions that enable an agent to utilize them.

3) Field
A field is an object managed by an agent server to hold Prolog clauses. An agent communicates with other agents indirectly through fields. An agent can communicate with other agents not only asynchronously but also synchronously.

B. Persistence of Agent

Figure 4 shows a UML diagram which is an overview of classes of Maglog. ClassLoader and CodeLoader are corresponded to DynamicClassLoader. AgentScheduler is corresponded to AgentController. AgentServerRemote provides network functions which are using Java RMI and XML-RPC.

An agent can suspend by executing the predicate, which name is *suspend*. When an agent executes *suspend* predicate, objects of the agent are dispatched to AgentSerializer via AgentServer. AgentSerializer writes serialized ClassLoader object and serialized CodeLoader objects in a file, which name is generated using the agent's identifier. The file is called SuspendedClassLoaderFile. Then AgentSerializer writes serialized Agent object in a file, which name is generated using the agent's identifier. The file is called SuspendedAgentFile.

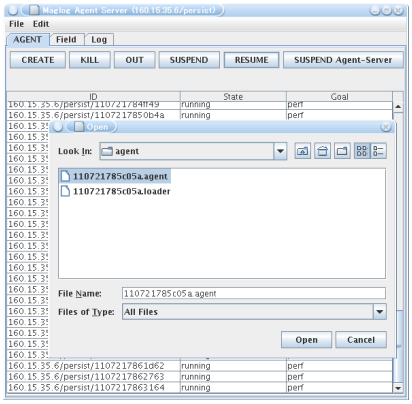

Fig. 6. A screen-shot of the user interface program when a user tries to resume an agent.

Finally, AgentScheduler stops the thread of the agent.

A user can suspend an agent and can resume the agent by a user interface program for manipulation of agent servers. Figure 5 shows a screen-shot of the user interface program when a user tries to suspend an agent. If the thread of the agent waits or sleeps, the thread is woken up by AgentScheduler using the Java notify method, then objects of the agent are serialized.

Figure 6 shows a screen-shot of the user interface program when a user tries to resume an agent. The user chooses the SuspendedAgentFile. AgentDeserializer reads the file then reads the SuspendedClassLoaderFile. AgentDeserializer deserializes objects of the agent using above read files then dispatches deserialized objects to AgentScheduler. The thread of the agent is started by AgentScheduler.

C. Persistence of Agent Server and Field

A user can suspend an agent server by the user interface program and can resume the agent server with a command-line option, which is `--resume`. When a user tries to suspend an agent server, the user clicks the *SUSPEND Agent-Server* button in Fig. 5. First, AgentScheduler serializes objects of all agents by AgentSerializer. Secondly, objects of Field and objects of Property are serialized by AgentServerSerializer then are written in a file. The file is called SuspendedAgentServerFile. Finally, the agent server shuts down. When a user starts the agent server again with the command-line option, AgentServerDeserializer reads the file then deserializes the objects of Field and objects of Property. AgentScheduler deserializes objects of all serialized agents by AgentDeserializer, then threads of the agents are started.

IV. Experiments

This section presents experimental results. Table I shows the experimental conditions. In one experiment, we examine the elapsed time and size of the generated files when an agent is suspended. Subsequently, we examine the elapsed time when the agent resumes. The agent's class file size

is 1.2KB. Table II shows the elapsed times of suspending and resuming. Table III shows SuspendedAgentFile size and SuspendedClassLoaderFile size.

In the other experiment, we examine the elapsed time of suspending and SuspendedAgentServerFile size when an agent server is suspended. The 100 agents are running on the agent server. Subsequently, we examine the elapsed time when the agent server resumes. The agent's class file size is 1.2KB. Table IV shows the elapsed times of suspending and resuming. The SuspendedAgentServerFile size is 69KB.

TABLE I
THE EXPERIMENTAL CONDITIONS.

CPU	Intel Xeon 3GHz
Memory	1GB
OS	TurboLinux 10 Desktop
JRE	1.4.2

TABLE II
THE ELAPSED TIME OF SUSPENDING AN AGENT AND RESUMING THE AGENT.

Suspending time	Resuming time
280 msec	154 msec

The agent's class file size is 1.2KB.

TABLE III
THE SUSPENDEDAGENTFILE SIZE AND THE SUSPENDEDCLASSLOADERFILE SIZE.

SuspendedAgentFile	SuspendedClassLoaderFile
66.8 KB	2.2 KB

The agent's class file size is 1.2KB.

TABLE IV
THE ELAPSED TIME OF SUSPENDING AN AGENT SERVER AND RESUMING THE AGENT SERVER.

Suspending time	Resuming time
36,175 msec	10,426 msec

Each agent's class file size is 1.2KB.

V. APPLICATION

We have developed two distributed applications using Maglog. In these applications, the performance is improved using persistence of agents. Additionally, persistence of an agent server is used to suspend a whole of the systems. In this section, we describe circumstantially how persistency is used in these applications.

A. Distributed e-Learning system

Figure 7 illustrates the overview of our distributed e-Learning system[8], [9]. This e-Learning system has two distinguishing features. Firstly, it is based on P2P architecture for scalability and robustness. Secondly, each exercise in the system is not only data but an agent so that it can mark user's answers, tell the correct answers, and show some extra information without human instruction. Maglog plays an important role to realize the both features.

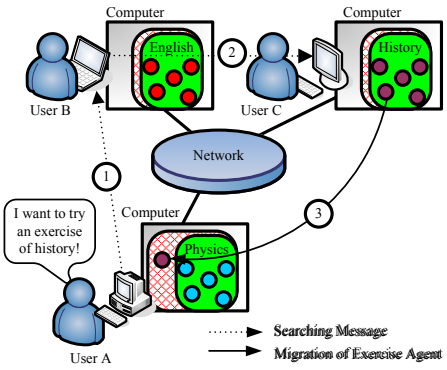

Fig. 7. Overview of the distributed e-Learning system.

While a user uses this e-Learning system, his/her computer is a part of the system. Namely, it receives some number of agents in them from another computer when it joins the system and has responsibility to send appropriate exercises to requesting computers. When the system begins, one initial computer has all agents in the system. When another computer joins the system, it is received some number of agents from the initial computer. The agents are distributed amon all computers in the system according as computers join the system or leave the system.

In this system, one exercise is corresponding to one agent. When there is a lot of exercise, many agents exist. These agents are spread on computers which are joined in the system. However, if few computers join the system, a number of agents concentrate on one computer. As a result, a large amount of resources on the computer is consumed. As the solution, the agents which are not requested from users are swapped out to a hard disk.

When a computer joins the system, it is received some agents from another computer. When a lot of agents which are swapped out to a hard disk try to migrate, its resuming consumes a large amount of resources and time. To improve a performance, Maglog supports that agents can be transferred without resuming.

We examined comparison of agent's migration time between with resuming and without resuming. One agent of which file size was 332 KB migrated between two PCs 100 times, and the average times were examined. The average time of with resuming is 1,750 msec. The average time of without resuming is 590 msec. Migration of an agent without resuming is about three times speed of with resuming.

B. Meeting scheduling system

Our meeting scheduling system[10], [11] establishes and arranges meeting schedule without human negotiations. Once a convener convenes a meeting through the system, an agent

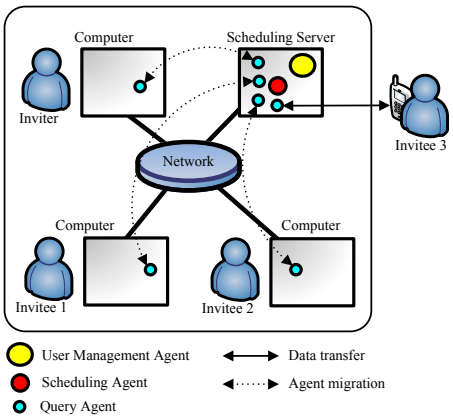

Fig. 8. Overview of the meeting scheduling system.

moves around the members of the meeting and negotiates with them automatically.

Figure 8 illustrates the overview of the meeting scheduling system. As shown in Fig. 8, there is a scheduling server which runs always as long as the proposed system provides the meeting scheduling service. A user management agent which stands on the scheduling server manages user information that include login name, password, IP address of the user's computer, and online/offline status. Login name and password are static while IP address and online/offline status are updated dynamically when a user logins to the system. A user can use any computer on the network because his login name is tied up with the IP address of his computer dynamically. A scheduling agent corresponding to a meeting creates query agents for each participant of the meeting. Each query agent asks one participant's schedule concurrently. The scheduling agent does the rest of the work for meeting scheduling.

In this system, when all participants' computer stops during a meeting scheduling, all scheduling data disappear. Such situation may occur to take time for a scheduling, for example, a blackout occurs. As the solution, this system uses persistence of agents and persistence of an agent server. When all participants' computer tries to stop, firstly all agents migrate to the scheduling server. Secondly, all agents on the scheduling server suspend, then the agent server on the scheduling server suspends.

VI. Conclusion

We have implemented persistence of agents, agent servers and fields in Maglog for the following two purposes. One is for suspending agents. The other is for suspending a system which is developed by using Maglog. The first purpose is used to improve performance in our e-Learning system. In our meeting scheduling system, the second purpose is used to suspend a whole of the system.

In order to realize persistence of agents, Maglog supports customized Object Serialization and the dynamic class loading mechanism. Agent servers do not consist of only serialized components, and so agent server's persistency steps which serialized components are serialized and an agent server shuts down are defined.

Maglog supports that suspending agents can be transferred without resuming, however the agents have restrictions to resume on a migrated computer. For example, if a suspending agent which was in a field is transferred to another computer without resuming, the suspending agent cannot resume on the computer. Because the field is not on the transferred computer. As the solution, it will be added in the future that a condition of a suspending agent is updated.

References

[1] D. B. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley, 1998.

[2] I. Satoh, "Mobilespaces: A framework for building adaptive distributed applications using a hierarchical mobile agent system," in *Proc. IEEE International Conference on Distributed Computing Systems*. IEEE Press, April 2000, pp. 161–168.

[3] S. Motomura, T. Kawamura, and K. Sugahara, "Logic-based mobile agent framework with a concept of "field"," *IPSJ Journal*, vol. 47, no. 4, pp. 1230–1238, 4 2006.

[4] M. Banbara and N. Tamura, "Translating a linear logic programming language into Java," in *Proc. ICLP'99 Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages*, M.Carro, I.Dutra *et al.*, Eds., December 1999, pp. 19–39.

[5] Sun Microsystems, "The Java^{TM} Object Serialization Specification," http://java.sun.com/j2se/1.5.0/docs/guide/serialization/.

[6] S. Motomura, T. Kawamura, and K. Sugahara, "Combination of xml-rpc and mobile agent technologies," in *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems*, 11 2006, pp. 552–557, dallas, Texas, USA.

[7] H. Aït-Kaci, "Warren's Abstract Machine A Tutorial Reconstruction," 2 1999.

[8] T. Kawamura and K. Sugahara, "A mobile agent-based p2p e-learning system," *IPSJ Journal*, vol. 46, no. 1, pp. 222–225, 1 2005.

[9] T. Kawamura, S. Kinoshita, S. Motomura, and K. Sugahara, "Backup and recovery mechanism for a distributed e-learning system," in *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems*, 11 2006, pp. 546–551, dallas, Texas, USA.

[10] S. Motomura, K. Kagemoto, T. Kawamura, and K. Sugahara, "Meeting arrangement system based on mobile agent technologies," *IPSJ Journal*, vol. 46, no. 12, pp. 3123–3126, 12 2005.

[11] T. Kawamura, S. Motomura, K. Kagemoto, and K. Sugahara, "Meeting arrangement system based on mobile agent technology," in *Proceedings of the 2nd International Conference on Web Information Systems and Technologies*, 4 2006, pp. 117–120, setubal, Portugal.