An Architecture for Next Generation Mobile Agent Infrastructure

Ichiro Satoh*

Department of Information Sciences, Ochanomizu University†/
Japan Science and Technology Corporation

Abstract

This paper presents a portable and adaptive mobile agent system. The system consists of a core system and subcomponents like micro-kernel operating systems. The core system offers only the minimal and common facilities and the latter is implemented as a collection of mobile agents which offer various facilities independent on its execution environment. The system can dynamically evolve and extend its own facilities by moving agents which support the facilities, according to its mobile computing environment. An implementation of the mobile agent system is available.

1 Introduction

Over the past several years, there has been a lot of work in the area of mobile agents. Mobile agents are autonomous programs that can travel from computer to computer under their own control. They can provide a convenient, efficient, and robust framework for implementing distributed applications including mobile applications. Several mobile agent systems have been released over the last few years (for example [8, 9, 14, 15]).

Mobile agents have been used in the development of various networked applications. These applications often need to dynamically customize facilities provided by mobile agent systems, for example execution and migration of agents. For example, a mobile agent for electronic commerce needs to be transformed into an encrypted bit stream before transferring itself over a network. In mobile computing settings, network protocols for agent migration should tolerate network disconnection. However, existing mobile agent systems cannot dynamically adapt their facilities, including network processing to the requirements of visiting agents and changes in their surrounding environments. This is because such facilities in existing mobile agent systems are statically embedded inside their runtime systems. Moreover, most existing systems are explicitly and implicitly dependent on particular execution environments such as operating systems, hardware, and networks.

This paper addresses a portable and adaptable mobile agent system and describes the architecture and implementation of the system. The system is characterized in that it is composed of a collection of subcomponents implemented as mobile agents. Also, it is unique among existing mobile agent systems in having the ability to treat mobile agents as first-class objects [7], in the sense that mobile agents can be passed to and returned from other mobile agents as values. Therefore, our framework allows various operations for mobile agents, including network processing for agent migration, to be naturally constructed and performed by other mobile agents. For example, network protocols for agent migration are implemented by mobile agents. These mobile agent-based protocols can transmit other agents to the destinations of the agents in the most appropriate way.

This paper consists of the following sections. Section 2 states the basic ideas of the system and Section 3 presents presents an adaptive mobile agent system. Section 4 describes the current implementation status and Section 5 surveys related work and Section 6 gives some concluding remarks.

2 Background

A lot of mobile agent systems have been released nowadays, for example see Aglets [8], Mole [14], Telescript [15], and Voyager [9]. However, to our knowledge, no existing mobile agent systems can extend and change their own functions, unlike ours. In the literature on extensible operating systems and meta-level architecture, several researchers have explored frameworks to change the behavior of operating systems and applications according to their environments (for example, see [3, 4, 5, 16]). These systems can adapt themselves to their surrounding environ-

*E-mail: ichiro@is.ocha.ac.jp

†2-1-1 Otsuka Bunkyo-ku Tokyo 112-8610, Japan Tel: +81-3-5978-5388 Fax: +81-3-5978-5390

ments by means of special operations such as code migration or meta-level semantics. However, most of them lack any mechanism for the migration of running programs between computers.

In an earlier paper [11], the author presented two notions: agent hierarchy and inter-agent migration, which give a mechanism for the development of a large and complex mobile application by assembling mobile agents into a single mobile agent, like software component technology. On the other hand, this paper addresses that the concepts are available in the construction of an adaptable mobile agent system. We present an architecture for extensible and adaptive infrastructure for supporting mobile agents based on the two notions. Also, we introduce agent migration in an agent hierarchy as a unified mechanism for computation and adaptation in the infrastructure.

3 Architecture Overview

In our mobile agent system, mobile agents are computational entities like other mobile agents. When each agent migrates, not only the code of the agent but also its state can be transferred to the destination. Furthermore, our mobile agent system is characterized in that it can dynamically extend and adapt itself to the changes of its execution environments. the requirements of users. The system is built on two ideas. The first idea is to construct a mobile agent system according to a micro-kernel architecture as shown in several operating systems. That is, it consists of two parts: a core system and subcomponents. The former offers only minimal and common functions independent of the underlying environment. The latter is introduced as a collection of subcomponents outside the core system and provides the other functions. The second idea is to implement all subcomponents as mobile agents so that these subcomponents can be dynamically added to and removed from the system by migrating and replacing the corresponding agents. Therefore, the system can adapt itself to its execution environment and the requirements of its executing mobile agents. It introduces mobile agents as the only constituent of the system. This gives users/programmers a single unified perspective of the system and applications running on the system.

Therefore, we need a mechanism for dynamically and structurally combining mobile agents as software components. However, existing mobile agent systems unfortunately lack any mechanism for structurally assembling more than one mobile agent. This is because each mobile agent is basically designed as an isolated entity which always acts and migrates independently. Therefore, we introduce the following unique concepts. (1) *Agent Hierarchy:* Each mobile agent can be contained within one mobile agent. (2) *Inter-agent Migration:* Each mobile agent can migrate between mobile agents as a whole with all its inner agents.

These concepts are available in the development of not only a mobile agent-based application which is large in scale and complicated, but also an adaptable mobile agent system and its extendable application.

Mobile Agents as Service Providers: As mentioned previously, our mobile agent system is characterized by offering its own facilities through mobile agents. The concepts allow our mobile agent system to be constructed as a collection of mobile agents organized structurally. The system can customize its structure and its functions by migrating agents into it, while it is running. Accordingly, the system can dynamically change and evolve its facilities by migrating agents implementing the facilities. For example, while the system is running, it can add a new function to itself by migrating a new mobile agent which implements the function to the system. The system can be open to evolve and adapt itself to its execution environment and the requirements of visiting agents.

Agent Migration as Meta Mechanism: It is often argued that the advantage of agent migration lies in the reduction of communication costs in distributed computing settings. Although this argument is understandable, our system can make use of agent migration as a meta mechanism for changing and evolving a system consisting of one or more mobile agents. When an agent wants a service, it can access the service by migrating itself to the agent which provides the service. The semantics and properties of an agent are partially provided by its parent agent and these can be changed by moving to other agents. In this sense, a parent agent can be viewed as a meta interpreter of its children.

4 Architecture Details

Next, we will describe our method for using the MobileSpaces system to construct mobile compound documents.[1] The system can execute and migrate mobile agents that are incorporated with the two concepts presented in the previous section. It has been incorporated in Java Development Kit version 1.2 and can run on any computer that has a runtime compatible with this version.

[1]Details of the MobileSpaces mobile agent system can be found in our previous paper [11].

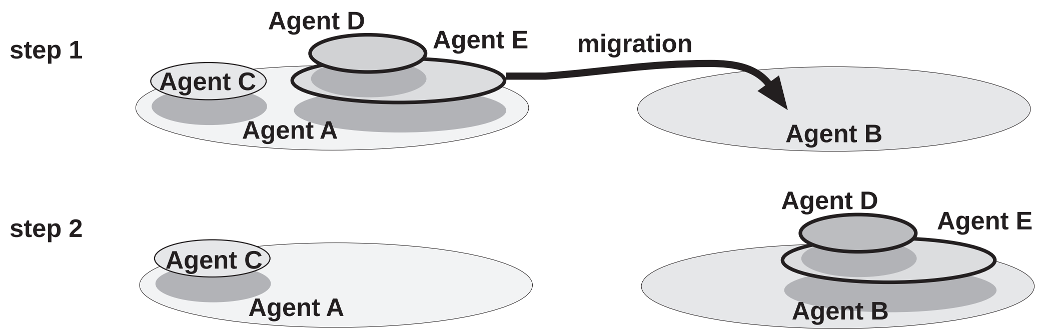


Figure 1: Agent Hierarchy and Inter-agent Migration

4.1 Core System

Our mobile agent runtime system is a platform for executing and migrating mobile agents. It is built on a Java virtual machine and mobile agents are given as Java objects [2]. The runtime system has the following functions:

Agent Hierarchy Management: The agent hierarchy is given as a tree structure in which each node contains a mobile agent and its attributes. The runtime system is assumed to be at the root node of the agent hierarchy. Agent migration in an agent hierarchy is performed just as a transformation of the tree structure of the hierarchy. In the runtime system, each agent has direct control of its inner agent. That is, a container agent can instruct its embedded agents to move to other agents or computers, serialize and destroy them. In contrast, each agent has no direct control over its container agent. Instead, each container can offer a collection of service methods which can be accessed by its embedded agents.

Agent Execution Management: The runtime system is at the root node of the agent hierarchy and can control all the agents in the agent hierarchy. Furthermore, it maintains the life-cycle of agents: initialization, execution, suspension, and termination. When the life-cycle state of an agent is changed, the runtime system issues events to invoke certain methods in the agent and its containing agents. Moreover, the runtime system enforces interoperation among mobile agents. The runtime system monitors the changes of agents and propagates certain events to the right agents. For example, when an agent is added to or removed from its parent agent, the system dispatches specified events to the two agents.

Agent Serialization: When an agent is transferred, it has to be marshaled into a bit-stream and then unmarshaled from it later. The core system provides a mechanism for marshaling and unmarshaling the states of agents. The current system uses the Java object serialization package for marshaling agents. The package does not support the capturing of stack frames of threads. Consequently, our system cannot serialize the execution states of any thread objects.[2]

4.2 Subcomponents

The core system supports only functions that are independent of the underlying environment, including agent migration in its agent hierarchy. Other functions, including agent migration between different computers, must be provided by subcomponents outside the core system. Each subcomponent is implemented as a mobile agent. Since our framework can treat mobile agents as first-class objects, mobile agents can handle and transfer other agents as data packets. More specifically, it has the following characteristics:

- Each subcomponent is designed to provide its service to its inner agents. When an agent wants a service, the agent migrates itself into a subcomponent that can provide the service in the same agent hierarchy and then the subcomponent automatically provides the service for the visiting agent.
- Such subcomponents can offer various services for mobile agents, for example migration between different computers, persistence, duplication, and higher level inter-agent communication. These subcomponents can be dynamically and autonomously deployed at the runtime systems by migrating the agents corresponding to them.

[2]This limitation is not serious in the development of real mobile agent-based applications, as discussed in [14].

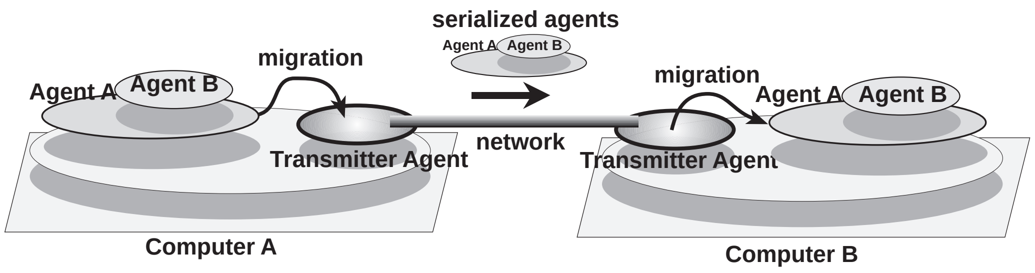


Figure 2: Transmitter mobile agents which establish channels between

4.2.1 Agent Migration between Computers

Agent migration between different computers is offered by subcomponents, called *transmitter* mobile agents, instead of the core system. Transmitter agents are allocated on hosts. Each transmitter agent can exchange its inner agents with each other through its favorite communication protocol (as shown in Figure 3). When a mobile agent is preparing for a trip, the agent migrates itself into an appropriate transmitter agent.

The transmitter suspends the moving agent (including its nesting agents) and then serializes its state, classes, and destination address into a proper form for its communication protocol. Next, it transfers the serialized agent to a transmitter agent on the destination side. The transmitter agent receives the data and then reconstructs an agent (including its nesting agents) according to the data. Each runtime system can be equipped with more than one transmitter agent in order to exchange agents through various communication protocols and networks. We have already implemented several transmitter agents which can transport their inner agents via several communication protocols such as TCP, UDP, and SMTP.

4.2.2 Routing Mechanisms for Agent Migration

Application-specific mobile agents often need to travel to multiple computers to perform their tasks. However, it is difficult to determine the itinerary at the time the agent is designed or instantiated. Therefore, we introduce two approaches for determining and managing the itinerary of agents. These approaches are built on transmitter agents running on computers and correspond to kinds of application-specific routing protocols.

Navigator Agent: The first approach offers a service provider, called a navigator agent, for conveying its inner agents over a network. Each navigator agent can be a container of other agents and can travel with them in accordance with a list of computers statically or algorithmically determined, or dynamically based on the agent's previous computations and the current environment. That is, a navigator agent can migrate itself to the next place as a whole with all its inner agents.

We developed a routing mechanism for managing a routing table consisting of computers to visit. Each navigator agent can maintain a list of computers to be visited and can provide methods to add and remove elements from this list. Whenever a navigator agent moves to a new place, the agent accesses a local SNMP agent in order to update its own routing table and then evaluates the table to determine what the next hop should be. The interaction between a navigator agent and its inner agents is based on an event-based communication. Upon arrival at a place, the navigator propagates certain events to its inner agents in order to instruct them to do something during a given time period. After the events have been processed by all the inner agents, the navigator continues with its itinerary.

Forwarder Agent: The second approach is based on a service provider, called a forwarder agent, for redirecting moving agents to new destinations. Each forwarder agent is a mobile agent and is designed to stay at computers and automatically transfer its inner agents to specified computers through appropriate transmitter agents. Consequently, a forwarder agent can be regarded as a programmable router for mobile agents.

A forwarder agent offers a mechanism to track the trails which a moving agent leaves behind. Just before an agent moves into another agent, the agent can leave a forwarder agent behind it. The forwarder inherits the old name of the moving agent and transfers its inner agent to the new location of the moving agent.[3] Therefore, when an agent wants to migrate to another agent, which was moved to somewhere,

[3]Each forwarder agent cannot transfer its inner agents to its original agent when the original agent is moved to other computers. Instead, it can request an appropriate transmitter to transfer them to the destination agents.

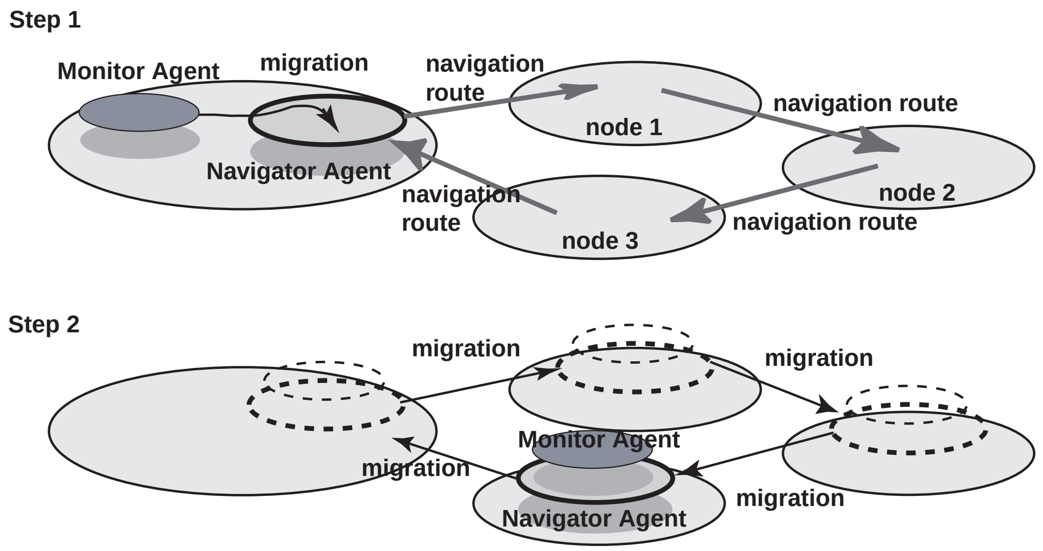


Figure 3: A navigator agent for traveling among three computers with its inner agents.

it can jump into the forwarder agent in return for the target agent. Then, the forwarder automatically transfers it to the target agent.

Several schemes to efficiently locate mobile agents have been explored in the literature of process/object migration in distributed operating systems. Our forwarder agents can easily support these schemes, including a smart directory service for finding suitable agents that can offer required services.

4.2.3 Storage Service

Although the core system can serialize the states of agents into a bit-stream, the way to store and restore such a bit-stream in secondary storage is often dependent on the underlying system, such as operating system and hardware. Therefore, we introduce storage agents which can store their inner agents on secondary storages in their favorite ways. When an agent is to be stored onto a disk, the agent migrates to a storage agent corresponding to the disk. The storage agent serializes and stores the states and codes of its visiting agents as a persistent data on the disk.

4.2.4 Agent Communication

Each agent can offer a meeting place for its inner agents via its context structure (mentioned later), and thus initially supports basic types of inter-agent communication, for example asynchronous message passing, synchronous method call, and future communication.[4] However, we need various inter-agent communications suitable for enriched interactions among agents; for example, multicast communication and higher-layer coordination protocols. Therefore, we have constructed a special agent for mediating among agents, like a facilitator of KQML [6]. The agent is equipped with a simple mechanism which gives its inner agents some useful services, for example maintaining a registry of agents, providing matchmaking between inner agents, and forwarding messages to appropriate agents.

4.2.5 Remarks

We have implemented a lot of mobile agent-based subcomponents for supporting various functions for agents, such as agent termination, agent duplication , and resource management in addition to the above functions. The system can be open to evolve and adapt its functions to the execution environment and the requirements of visiting agents by migrating and changing mobile agent-based subcomponents for supporting the functions.

4.3 Mobile Agent Program

Our mobile agents are programmable entities like other mobile agents. Each agent consists of three parts: body program, context objects, and inner agents. Every body program is an instance of a subclass of abstract class `Agent`.[5] This class defines fundamental callback methods invoked when

[4]The system does not offer any mechanism to communicate between agents over networks because this is done by the migration of a messenger agent.

[5]Some examples of mobile agent programs are given in the Appendix.

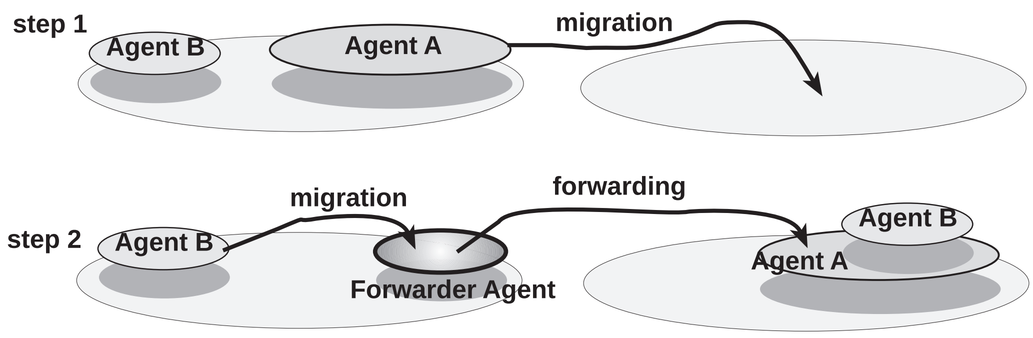


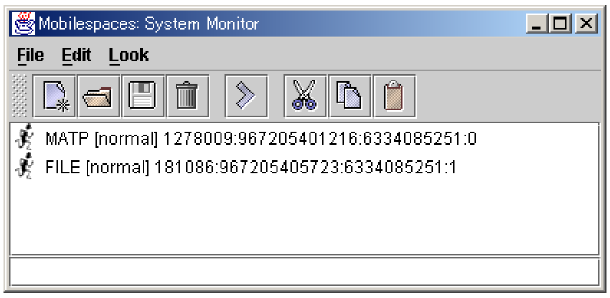
Figure 4: A forwarder agent which transfers its inner agents to other forwarder agents

the lifecycle of a mobile agent changes due to creation, suspension, marshaling, unmarshaling, or destruction etc, like the delegation event model in Aglets [8]. The class also provides a command for agent migration in an agent hierarchy, written as `go(AgentURL destination)`. When an agent performs the command, it migrates itself to the destination agent specified as the argument of the command. An inner agent cannot access any methods defined in its container agent. Instead, each container can be equipped with a context object which offers service methods in a subclass of the `Context` class, like the `AppletContext` class of Java's Applet. These methods can be indirectly accessed by its inner agents to get information about and interact with the environment such as their container, their sibling agents, and the underlying computer system. Each inner agent can invoke the public methods defined in the context of its container via several built-in application programming interfaces.

Each agent is associated with a resource limit that functions as a generalized Time-To-Live field. This limit is carried with the agent and decremented by nodes as resources are consumed when the agent arrives at a new place. Nodes can discard agents when their limit reaches zero. In order to restrict total resource bounds, when one agent creates another inside the network, the resources allocated to each created agent must be strictly less than those of the creating agent.

5 Current Status

Our mobile agent system has been implemented in the Java language (JDK1.1 or later version). The core system is constructed independently of the underlying system and can run on any computer with a 1.1-compatible Java runtime. We have tried to keep the implementation within the framework as much as possible.[6]. The current system provides graphical user interfaces for operating mobile agents with the MobileSpaces system as shown in Figure 5. These interfaces allows us to load and migrate mobile agents via fully drag-and-drop operations.



```
MATP [normal] 1278009:967205401216:6334085251:0
FILE [normal] 181086:967205405723:6334085251:1
```

Figure 5: The control window of the MobileSpaces system

Even though our implementation was not built for performance, we have conducted a basic experiment on agent migration. The cost of an agent migration in an agent hierarchy was measured to be 5 ms, including the cost to check whether the visiting agent is permitted to enter the destination agent or not. The cost of agent migration supported by transmitter agents allocated on two computers was measured to be 30 ms. A transmitter agent can communicate with another by using an application-level protocol for agent transmission whose mechanism is modeled on that of the HTTP protocol over TCP/IP communication. On the sender side, a transmitter agent serializes and transfers the codes and state of an agent (including its inner agents) to the transmitter on the receiver side and waits for an acknowledgment message. The second result is the sum of the marshaling, compression, opening TCP connection, transmission, acknowledgment, decompression, security and consistency verifications, and unmarshaling.

[6]An implementation of the mobile agent system, including its examples is available from `http://islab.is.ocha.ac.jp/`.

The moving agent is a simple navigator agent and consists of basic callback methods and contains two child agents. Its data size is about 3 Kbytes (zip-compressed).

Moreover, we have already implemented various applications of the MobileSpaces system in [13]. One of them is a compound document framework like OpenDoc [1] and Taligent [10]. Since our framework introduces software components as mobile agents, components, including documents, are active and mobile. In a previous paper [12], we constructed a formalization for hierarchical mobile agents in the MobileSpaces system.

6 Conclusion

This paper presented an adaptive mobile agent system which consists of subcomponents implemented as mobile agents. The system introduces agent migration as a meta mechanism for changing its functions, while it is running. That is, the system can dynamically change and evolve its functions by migrating agents that offer the functions. Mobile agent-based applications running on the system can enjoy the extensibility and adaptability of the system.

Finally, we would like to point out some further issues. Agents need a way of finding suitable agents which can offer their required service. We are implementing a directory service mechanism. Security is essential in mobile agent computing, but our current implementation still does not provide any reasonable level of security to make it safe to use mobile agent applications in the real world. Many security features are left open for our future work, such as authentication, and authorization of agents.

References

[1] Apple Computer Inc., OpenDoc: White Paper, Apple Computer Inc., 1994.

[2] K. Arnold and J. Gosling, The Java Programming Language, Addison-Wesley, 1998.

[3] G. Blaier, G. Coulson, P. Robin, and M. Papathomas, An Architecture for Next Generation Middleware, Proceedings of Middleware'2000, pp.191-206, Springer, 1998.

[4] B. N. Bershad, et al, Extensibility, Safety and Performance in the SPIN Operating System, Proceedings of Symposium on Operating Systems Principles, 1995.

[5] D. R. Engler, M. F. Kaashoek, and J. O. Toole, Exokernel: An Operating System Architecture for Application-level Resource Management, Proceedings of Symposium on Operating Systems Principles, 1995.

[6] T. Finin, Y. Labrou, and J. Mayfield, *KQML as An Agent Communication Language*, Software Agents, MIT Press, 1997.

[7] D. P. Friedman, M. Wand, and C. T. Haynes, Essentials of Programming Languages, MIT Press, 1992.

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

[9] ObjectSpace Inc, ObjectSpace Voyager Technical Overview, ObjectSpace, Inc. 1997.

[10] M. Potel and S. Cotter, Inside Taligent Technology, Addison-Wesley, 1995.

[11] I. Satoh, MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System, Proceedings of International Conference on Distributed Computing Systems (ICDCS'2000), pp.161–168, IEEE Computer Society, April, 2000.

[12] I. Satoh, A Formalism for Hierarchical Mobile Agents, Proceedings of Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'2000), pp.165–172, IEEE Computer Society, June, 2000.

[13] I. Satoh, MobiDoc: A Framework for Building Mobile Compound Documents from Hierarchical Mobile Agents, to appear in Proceedings of Symposium on Agent Systems and Applications / Symposium on Mobile Agents (ASA/MA'2000), LNCS, Springer, 2000.

[14] M. Strasser and J. Baumann, and F. Hole, Mole: A Java Based Mobile Agent System, Proceedings of ECOOP Workshop on Mobile Objects, 1996.

[15] J. E. White, Telescript Technology: Mobile Agents, General Magic, 1995.

[16] Y. Yokote, The Apertos Reflective Operating System: The Concept and its Implementation, Proceedings of OOPSLA'92, pp. 414–434, 1992.