# Hierarchically Structured Mobile Agents and their Migration

Ichiro Satoh*

Department of Information Sciences, Ochanomizu University
2-1-1 Otsuka Bunkyo-ku Tokyo 112-8610, Japan

**Abstract**

This paper presents a framework for mobile agents. The framework is unique among other mobile agent systems in structuring mobile agents hierarchically. The framework provides the notion of agent hierarchy, in which each mobile agent can be a container of other mobile agents. We construct a prototype implementation of this framework. It is built on the Java language and mobile agents are given as Java objects. The framework helps us to construct a mobile agent application that is large in scale and complex.

## 1 Introduction

Component-based software development technology is being used widely [13] and allows a distributed system to be easily constructed through the combination of existing subcomponents. On the other hand, many mobile agent systems have recently explored, but most of them do not provide any mechanism for structurally assembling one or more mobile agents as one mobile agent. This is because each mobile agent is basically designed as an isolated entity which always acts and migrates independently even though some systems provide the notions of places and inter-agent communication, which can couple mobile agents loosely. This is a serious limitation in the development of a large and complicated system based on mobile agents.

Therefore, we introduce a new mobile agent model inspired by the concept of mobile ambients studied by L.Cardelli and A.D.Gordon in [3]. Ambients are computational entities like mobile agents, but are characterized in allowing the movement of a group of one or more ambients, as opposed to the technique in existing mobile agents where single agents or individual objects migrate between computers. Each ambient can nest other ambients within itself and can migrate between ambients. The concept enables agents to be organized hierarchically and dynamically. Like mobile ambients, our model allows each mobile agent to be a container of other mobile agents inside itself, and to migrate to the other agents as well as in other computers. The model enables us to combine one or more mobile agents as a

*Email: ichiro@is.ocha.ac.jp

mobile agent, as component-based software development technology. It helps us to construct a mobile agent application that is large in scale and complex.

This paper consists of the following sections. In Section 2, we present basic ideas of our mobile agent model. Section 3 presents an implementation of the model, called MobileSpaces. Section 4 describes the current implementation status and present some examples of the model. Section 5 surveys related work in terms of mobile agent systems. In Section 6, we give some concluding remarks.

# 2 Basic Framework

This section introduces basic ideas of our mobile agent model.

## 2.1 Hierarchically Structured Mobile Agents

Our mobile agents are computational entities like other mobile agents. Once they are invoked, they will autonomously decide which locations they will visit and what instructions they will perform. When an agent migrates, not only the code of the agent but also its state can be transferred to the destination. Moreover, our mobile agent is characterized by the following concepts:

**Agent Hierarchy:** Each mobile agent can nest other mobile agents inside itself and has to be contained by one agent. Mobile agents are organized in a tree structure.

**Inter-agent Migration:** Each mobile agent can migrate between mobile agents as a whole with all its inner agents. That is, if a migrating agent includes other agents inside itself, all its inner agents have to be moved by causing the movement of the agent.

Figure 1 shows an example of an inter-agent migration in an agent hierarchy. When an agent contains other agents, we call the former agent *parent* and the latter agents *children* hereafter. Since this framework allows agents to be nested, a child agent can contain one or more agents in the same way. We call the agents which are nested by an agent, the *descendent* agents of the agent, and in reverse we call the agents which are nesting an agent, the *ancestral* agents of the agent. The agent hierarchy enables us to combine one or more mobile agents as a mobile agent, like in component-based software development technology [13]. It helps us to construct a mobile agent application that is large in scale and complicated.

It is worth mentioning differences between our model and other mobile agents models. The model is similar to the concept of mobile ambients. However, the concept is just a theoretical framework and thus must be changed and restricted in order to implement a practical mobile agent system. Also, each ambient is allowed to migrate itself to only its parent and its sibling agents. On the other hand, our model assumes that each agent can freely move into any agents in the agent hierarchy, except for itself nor its descendants, as long as the destination agent accepts the moving agent.

In Telescript [14]. the concept of places are introduced as collections of resources to visiting agents. Agents can access these resources as they enter the places which offer the resources. Similarly, in our model, each mobile agent is introduced as a collection of its

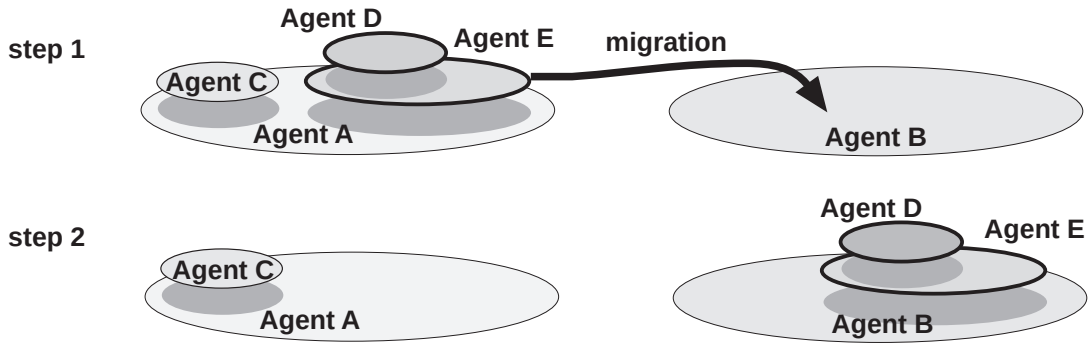


Figure 1: Agent Hierarchy and Inter-agent Migration

own resources and services for its child agents. That is, each parent agent is responsible for providing its own resources for its child agents. Different services can be provided in different agents. Therefore, when a mobile agent wants a service and resources, the agent can acquire the service by migrating to the agent providing the service.

- Each agent has direct control of its descendent agents. That is, an agent can instruct its descendent agents to move to other agents, serialize, and destroy them. Each agent can issue certain events to its descendent agents in order to press them to do something.
- In contrast, each agent has no direct control on its ancestral agents. Instead, each agent offers a collection of service methods for its children. A child agent can access its parent agent through the collections of their service methods. It can access those of its ancestral stationary agents, including the base agent, as well as its its parent agent.

Our system disallows any agent to access any services provided in their ancestral agents except for their parent agents and stationary agents. This restriction is a key idea of our mobile agent model and allows agents to properly capture computing resources. If an agent is assumed to access its ancestral mobile agent as well as its parent one, the agent cannot access any services provided by its grandparent agent any longer, as its parent agent moves from the grandparent agent to a new location.

# 3 Implementation

We developed an implementation of the hierarchically structured mobile agents presented in the previous section. It builds on the Java virtual machine and mobile agents are implemented in the Java language. Moreover, we have already constructed various mobile agent applications on the implementation, for example workflow management systems, chat systems, and distributed information search systems.[1]

[1]The distribution of the prototype implementation of the system is available from `http://islab.is.ocha.ac.jp`, but some of its documents are still written in Japanese.

## 3.1 The Runtime System

Hereafter, we describe some of the features in which our mobile agent system is unique among other systems below:

### Agent Hierarchy Management

The runtime system maintains an agent hierarchy, which is implemented as a tree structure where each node contains a mobile agent and its attributes. It alters the tree structure to migrate agents in the agent hierarchy, and can be abstracted as a stationary agent at the root node of the tree structure.

Agent migration in the same agent hierarchy can be viewed as just a transformation of the tree structure of the hierarchy. For example, an inter-agent migration is performed as follows: the system removes the subtree whose root node has the descendent agent from the whole hierarchy and then adds the subtree to the node which contains the destination agent, without altering the structure of the subtree. When an agent and its nested agents are moving, they can still be running.

### Agent Execution Management

Each agent can have one or more activities. The runtime system controls the activities of all agents with the Java thread mechanism. When an agent is transferred, serialized, or destroyed, the system stops and disposes of threads captured by the agent. Furthermore, the system maintains the life-cycle of agents: initialization, execution, suspension, and termination. When the state of an agent is changed, the system issues certain events to the agent and its descendent agents. These events can invoke specified methods defined in the agent.

### Agent Migration Over Network

When an agent is transferred over network, it has to be marshaled into a bit-stream and then unmarshalled from it later. The runtime system provides a mechanism for marshaling and unmarshaling for the states of agents. When an agent is marshaled, the runtime system propagates certain events to the agent and its descendent agents which are still running in order to press to stop. Also, it can automatically stop and serialize them after a given time period. The runtime system can transfer agents to the destination computer by means of an application-layered protocol for agent transmission whose mechanism is extended of the HTTP protocol over TCP/IP communication. However, in the current implementation of the system, agent migration over network is implemented in mobile agents instead of the runtime system. Therefore, we can easily change and adapt the mechanism of agent transmission.

**Remark** The current implementation of the system uses the Java object serialization package in order to marshal and unmarshal agents. The package does not support capturing the stack frames and program counter of threads. Consequently, our system cannot serialize the execution states of any thread objects. However, as discussed in [12], we believe that this limitation is not serious, because it has not prevented the development of various distributed applications based on mobile agents so far.

## 3.2 Mobile Agents

Every agent is an instance of a subclass of the base class for mobile agents, called `Agent`. The class consists of fundamental methods used to control the mobility and the life cycles of a mobile agent.

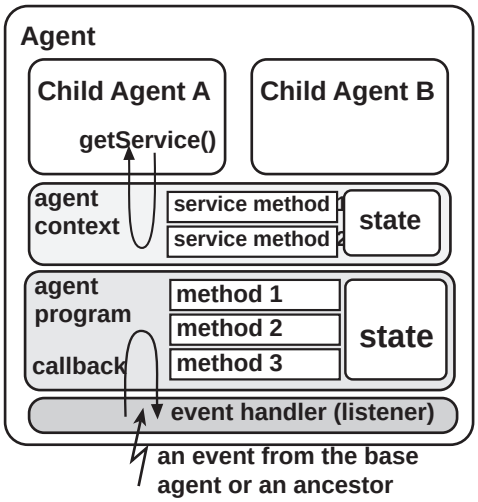


Figure 2: MobileSpaces Mobile Agent

```
 1: public class Agent {
 2:   // methods for registering listener objects to hook certain events
 3:   void addDefaultListener(DefaultEventListener listener){ ... }
 4:   void removeDefaultListener(DefaultEventListener listener){ ... }
 5:   ....
 6:   AgentURL getURL(){ ... }
 7:   Enumeration getChildren(){ ... }
 8:   ....
 9:   void getService(Message msg) throws NoSuchMethodException ... { ...}
10:   void dispatchEvent(AgentEvent evt) throws NoSuchEventException ... {...}
11:   void go(AgentURL dst) throws NoSuchAgentException ... { ...}
12:   ....
13: }
```

The `go(AgentURL dst)` method migrates itself and and its descendents to the destination agent specified as `dst`. An agent can call methods given in the context of its parent agent by calling the `getService()` method. The `dispatchEvent(AgentEvent evt)` method propagates an event specified as its argument to its When an agent is transferred or destroyed, our system does not automatically release all the resources, such as file, window, and socket, which are captured by the agent. Before or after the state of an agent changes, the system and its ancestral agents can propagate certain events to the agent, like event delegation event model in Aglets [6]. Therefore, each agent can have one or more listener objects in order to hook these events. We show a listener interface which defines fundamental methods invoked when agents are created, destroyed, serialized, and migrated to another agent and when visiting agents enter to and leave from them.

```
 1: interface DefaultEventListener extends AgentEventListener {
 2:                                  // timing of invocation
 3:   void create(AgentURL url);     // after creation at url
 4:   void destroy();                // before termination
 5:   void serialize();              // before serialization
 6:   void deserialize();            // after deserialization
 7:   void add(AgentURL child);      // after accepted a child
 8:   void remove(AgentURL child);   // before removed a child
 9:   void arrive(AgentURL dst);     // after arrived at the destination
10:   void leave(Agent dst);         // before moving to the destination
11:   ....
12: }
```

### 3.3 Performance

To evaluate the cost of agent migration, we examined a basic experiment of agent migration in two cases: agent migration in an agent hierarchy and agent migration between different computers. The former experiment is performed in a prototype implementation of the run-time system. In the latter experiment, agent migration is supported by transmitter agents allocated on two computers.

Table 1: the cost of agent migrations

| | time (in msec) |
|---|---|
| agent migration in an agent hierarchy | 3 |
| agent migration between two computers | 60 |

These results have been measured with two computers (Pentium II-450MHz with 128MB memory with MS-Windows98 and JDK 1.1.7) connected via 10BASE-T Ethernet. The first result includes the cost to check whether the visiting agent is permitted to enter the destination agent or not. The second result is the sum of the marshalling, opening TCP connection, transmission, and unmarshalling.

## 4 Related Work

Recently, there have been many mobile agent systems, for example Telescript [14], AgentTcl [5], Aglets [6], Mole [12], MOA [7], Voyager [8], AgentSpace [10] and so on. Like ours, most of them have been implemented in the Java language. However, to our knowledge, most existing mobile agent systems do not allow one or more mobile agents to be organized as a mobile agent, although some of the systems offer mechanisms for inter-agent communications in order to couple agents loosely. Several existing systems introduce the concepts of places in addition to mobile agents. Mobile agents can move from place to place in order to meet other agents and make use of the services provided by the places. However, places are stationary entities instead of mobile ones. Mole introduces the notion of agent groups in order to encourage coordination among mobile agents [2]. Mole's agent groups can consist of agents working together on a common task but are not mobile.

# 5 Conclusion

In this paper we have seen a way to design and implement a mobile agent system. The system is characterized by the concepts of agent hierarchy and inter-agent migration. It allows one or more mobile agents to be structurally and dynamically composed into a mobile agent. This helps us to construct a mobile agent application that is large in scale and complex. Although this paper addresses a model for mobile agents, we believe that the model is designed for constructing a runtime system whose facilities can be dynamically extensible and adaptable to its execution environments.[2] Also, the model provides a powerful framework for mobile agents running on mobile computers, because the model allows us to view the migration of a mobile computer as that of a hierarchical mobile agent.

Finally, we would like to point out some further issues. Our mobile agent system presented in this paper is originally inspired by the concept of mobile ambients studied in [3], but does not intend to implement the concept itself. We are interested in formalizing a new theoretical framework for our mobile agent model based on process calculus approaches as studied in [9, 11]. Although, security is essential in mobile agent computing, many security features are left open for our future work, such as authentication, and authorization of agents. Also, the current implementation lacks resource management. We are interested in introducing a mechanism for resource management based on our agent hierarchy.

# References


[1] K. Arnold and J. Gosling, The Java Programming Language, Addison-Wesley, 1996.

[2] J. Baumann and N. Radounklis, Agent Groups in Mobile Agent Systems, Conference on Distributed Applications and Interoperable Systems, 1997.

[3] L. Cardelli and A. D. Gordon, Mobile Ambients, Foundations of Software Science and Computational Structures, LNCS, Vol. 1378, pp. 140–155, 1998.

[4] General Magic, Inc. Introduction to the Odyssey, http://www.genmagic.com/agents, 1997.

[5] R. S. Gray, Agent Tcl: A Transportable Agent System, CIKM Workshop on Intelligent Information Agents, 1995.

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

[7] D. S. Milojicic, W. LaForge, and D. Chauhan, Mobile Objects and Agents (MOA), USENIX Conference on Object Oriented Technologies and Systems, April 1998.

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

[9] I. Satoh and M. Tokoro., Time and Asynchrony in Interactions among Distributed Objects, European Conference on Object Oriented Programming, LNCS 952, pp. 331–350, Springer-Verlag, 1995.


[2] We leave the details of the extensible and adaptable runtime system for hierarchical mobile agents to another paper.

[10] I. Satoh, AgentSpace, http://islab.is.ocha.ac.jp/agent/index.html, 1997.

[11] I. Satoh, A Mobile Agent-Based Framework for Active Networks, to appear in Proceedings of IEEE Systems, Man, and Cybernetics Conference (SMC'99), October, 1999.

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

[13] C.Szyperski, Component Software, Addison-Wesley, 1998.

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