A Hierarchical Model of Mobile Agents and Its Multimedia Applications

Ichiro Satoh

Department of Information Sciences, Ochanomizu University*/
Japan Science and Technology Corporation
E-mail: ichiro@is.ocha.ac.jp

Abstract

This paper presents a new framework for constructing networked applications, including multimedia ones. The framework is based on a hierarchical mobile agent system for allowing more than one mobile agent to be dynamically assembled into a single mobile agent. Networked applications constructed in the framework can be implemented as a collection of mobile agents and can dynamically change and evolve their own functions by migrating agents that offer the functions. To demonstrate how to exploit our framework we construct extensible multimedia applications based on the framework.

1 Introduction

Component-based software development technology is being used [11] as a powerful approach for the development of distributed systems and multimedia systems. The technology can combine subcomponents into an application or a large-scale component. On the other hand, mobile agents are self-contained entities like software components and can travel from computer to computer under their own control. Mobile agent technology has received a rapidly growing attention over the last few years due to its salient properties.

However, existing mobile agent systems 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. Although several systems provide the notion of places and inter-agent communication, they can couple mobile agents *loosely* and thus cannot migrate a group of mobile agents to another computer as whole. This is a serious limitation in the development of a mobile agent-based application, which is large in scale and complicated.

In a an earlier paper [9], we proposed the notion of hierarchical mobile agents and presented a mobile agent system whose functions can be dynamically extended and adapted to its execution environment. However, unfortunately we did not present that mobile agent-based applications based on the concept can naturally inherit the extensibility and adaptability of the system.

Therefore, this paper presents a framework for constructing new dynamically adaptable networked multimedia applications based on the concept of the agent hierarchy. Our approach can naturally introduce mobile agents as mobile software components and can easily construct a large-scale mobile application as a compound mobile agent, which contains mobile agents supporting its subcomponents. Therefore, our approach is a powerful framework for the development of a distributed application, in particular a large-scale and adaptable multimedia application.

This paper consists of the following sections. Section 2 surveys related work. In Section 3, we present the basic ideas of the system presented in this paper. Section 4 presents a mobile agent system called *MobileSpaces* and the current implementation status of the system. In Section 5, we show some examples of multimedia applications constructed in the framework and in Section 6 we give some concluding remarks.

2 Background

A lot of mobile agent systems have been released over the last few years. Telescript [14] was the first commercial implementation of the mobile agent paradigm; AgentTcl [4] is a mobile agent system based on an extended Tcl interpreter that executes Tcl agents. Like ours, most of these systems have been implemented in the Java language, for example, see Aglets [6], Mole [8] and Voyager [7].

To our knowledge, no existing mobile agent systems, including mobile object systems, are based on the concept of agent hierarchy proposed in this paper. Mole introduces the notion of agent groups in order to encourage coordination among mobile agents [3]. Mole's agent groups can consist

*2-1-1 Otsuka Bunkyo-ku Tokyo 112-8610, Japan

of agents working together on a common task, but they are not mobile. Also, Telescript, Odyssey, and MOA introduce the concept of places in addition to mobile agents. Places are agents which can contain mobile agents and places inside them, but they are not mobile. Our mobile agent system, on the other hand, allows one or more mobile agents to be dynamically organized into a single mobile agent, and thus we do not have to distinguish between mobile agents and places. Therefore, a distributed application, in particular a mobile application, that is large in scale and complex can be easily constructed by combining more than one agent.

Recently, several researchers have have explored active networks to dynamically deploy programs for handling network protocols at intermediate and end nodes by using mobile-code techniques. (for example see [12, 13, 15]). Most active networks have been designed to dispatch miniature programs to remote hosts when communication methods need to be changed. They thus cannot deal with networked applications, for example, server programs and client programs, including user interfaces. There have been a few works to incorporate mobile agent technology with the active network technology (for example, see [5]), but the purpose of these works are to apply mobile agents to network management and control.

3 Basic Framework

We intend our framework to provide a practical infrastructure for constructing distributed applications, in particular networked multimedia applications.

The framework has to be designed to dynamically deploy programs for handling protocols at each per communication session and each multimedia content. Also, a multimedia application is often given as a large-scale and complex program. Therefore, the framework can combine subcomponents into an application. A networked application is often expected to be used in various networks, and thus it should be constructed independently of its underlying hardware, communication network, and operating system and can be extendable and adaptable to its execution environments.

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. To solve the above requirements in the construction of networked multimedia applications, our mobile agent model introduces 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*. We call all the agents which are nested by an agent, the *descendent* agents of the agent, and in reverse we call all the agents which are nesting an agent, the *ancestral* agents of the agent. The hierarchy enables us to combine one or more mobile agents as a mobile agent, like in component-based software development technology [11]. It helps us to construct a mobile agent application that is large in scale and complicated.

Moreover, 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 framework can make use of agent migration as a meta mechanism for changing and evolving an application consisting of one or more mobile agents. Such an application can extend and change itself by migrating and replacing these agents. That is, each parent agent gives its own services and resources to its children. Therefore, when a mobile agent wants different services, the agent can acquire those services by migrating to the agent providing those services. Moreover, our mobile agent system itself is based on the framework. Therefore, we can easily add a new function to a system by migrating an agent offering the function to the system while the system is running. The system is extensible in the sense that it can dynamically change and adapt itself to its environment and the requirements of its executing mobile agents.

4 The MobileSpaces Mobile Agent System

This section presents a mobile agent system named *MobileSpaces*. The system can execute and migrate mobile agents that are incorporated with the framework presented in the previous section. Moreover, the architecture of the system is characterized in being based on the framework.

It is built on the Java virtual machine and mobile agents are given as Java objects [2]. The structure of the system is similar to a micro-kernel architecture as shown in several operating systems. That is, it consists of two parts: a core system and subcomponents as shown in Figure 2. 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. Each subcomponent is

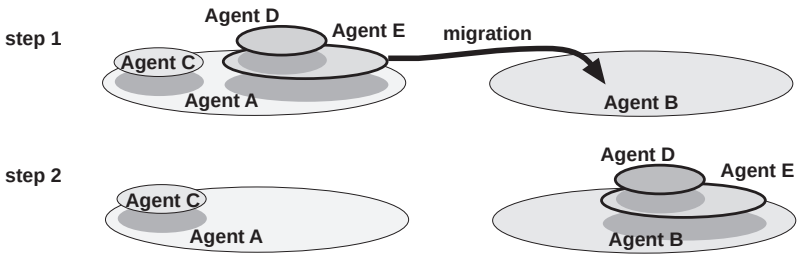

Figure 1: Agent Hierarchy and Inter-agent Migration

implemented as mobile agents so that these subcomponents can be dynamically added to and removed from the system by migrating and replacing the corresponding agents.

4.1 The Runtime System

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

Agent Hierarchy Management

Each agent hierarchy is given as a tree structure in which each node contains a mobile agent and its attributes. Agent migration in an agent hierarchy is performed as just a transformation of the tree structure of the hierarchy. Since each agent hierarchy is basically maintained inside a computer, when an agent is moved in the same agent hierarchy, it and its descendent agents can still be running. The runtime system 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. Consequently, agents can be viewed as the only constituent of our mobile agent system.

Agent migration in an agent hierarchy can be protected by a security manager. A destination agent can judge whether it accepts a new visitor or not beforehand, whereas a visiting agent can know the available methods provided by the destination agent by using the class introspector mechanism of JDK 1.1.

Agent Execution Management

The runtime system can control all the agents in its agent hierarchy. 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. Moreover, each agent can directly invoke all the public methods of its descendent agents.[1]

[1]The current implementation of MobileSpaces permits a parent agent to obtain references to the Java objects corresponding to its descendants.

In contrast, each agent has no direct control over its ancestral agents. Instead, each agent can have a collection of service methods which can be accessed by its children, instead of its descendant. A child agent can invoke the service methods provided by its parent under the control of the parent. In addition, each agent can access the service methods provided by its ancestral stationary agents, including the base agent.

We imposed the restriction that a mobile agent may not access any services supported by ancestral agents other than their parent and stationary agents. This restriction is key idea for allowing successful migration to occur. If it were not imposed, then migrating an agent could mean that the descendants of that agent might suddenly find they could no longer access services upon which they relied.

Each agent can have one or more activities which are implemented by using the Java thread library. Furthermore, the runtime system 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 certain events to the agent and its descendent agents. The system can impose specified time constraints on all method invocations between agents in order to avoid to be blocked forever.

Agent Migration Over Network

When an agent is transferred over network, it has to be marshaled into a bit-stream and then unmarshaled from it later. Agent migration between different computers is offered by subcomponents, called Transmitter agents, instead of the runtime system itself, because the system is made as small as possible for the sake of portability and offers only the minimal facilities independent of the execution environments and agent migration over network is dependent on its underling network. 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 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.

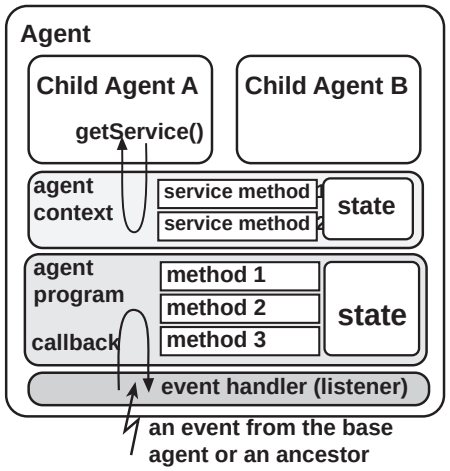

Figure 2: MobileSpaces Mobile Agent

```
 1: public class Agent {
 2:   // methods for registering listener
 3:   // objects to hook certain events
 4:   void addDefaultListener(
 5:     DefaultEventListener listener){...}
 6:   void removeDefaultListener(
 7:     DefaultEventListener listener){...}
 8:   ....
 9:   AgentURL getURL(){ ... }
10:   Enumeration getChildren(){ ... }
11:   ....
12:   void getService(Message msg) throws
13:     NoSuchMethodException ... { ...}
14:   void dispatchEvent(AgentEvent evt)
15:     throws NoSuchEventException {...}
16:   void go(AgentURL dst) throws
17:     NoSuchAgentException ... { ...}
18:   ....
19: }
```

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
 2:   extends AgentEventListener {
 3:  // invoked after creation at url
 4:    void create(AgentURL url);
 5:  // invoked before termination
 6:    void destroy();
 7:  // invoked before serialization
 8:  void serialize();
 9:  // invoked after deserialization
10:    void deserialize();
11:  // invoked after accepted a child
12:    void add(AgentURL child);
13:  // invoked before removed a child
14:    void remove(AgentURL child);
15:  // invoked after arrived at the destination
16:    void arrive(AgentURL dst);
17:  // invoked before moving to the destination
18:    void leave(Agent dst);
19:  ....
20:}
```

4.3 Implementation and Performance

The MobileSpaces 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.[2]

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 runtime system. In the latter experiment, agent migration is supported by transmitter agents allocated on two computers.

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

Table 1: The cost of agent migrations (msec)

	time
agent migration in an agent hierarchy	5
agent migration between two computers	35

These results have been measured with two computers (Pentium II-450MHz with 128MB memory with MS-Windows98 and JDK 1.1.8) 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 marshaling, compression, opening TCP connection, transmission, uncompression, and unmarshaling. The size of the moving agent is about 10KB.

5 Examples

This section presents two examples in order to demonstrate the utility of our framework in the construction of networked multimedia applications.

5.1 A mobile agent-based E-mail System

One of the most important examples is applications based on the concept of compound documents like OpenDoc developed by Apple Computer and IBM [1]. Our agent hierarchy allows compound documents given as mobile agents to be dynamically composed into a compound document, while traditional mobile agents are isolated programs and thus cannot support any compound documents.

We have constructed an electronic mail system where each letter is a mobile agent incorporated with the framework presented in this paper. Therefore, each letter can contain more than one mobile agent-based component: some text, graphics, and animations on the document of the letter as shown in Figure 3 and 4. Users can edit these inner components written in arbitrary data formats, because they are mobile agents and thus can include programs to edit their own contents. For example, to edit the text, simply click on it, and its editor program is invoked. The letter agent can autonomously deliver itself and its inner components to the destination. The receiver can read all the contents of the arriving letter, because the letter is a mobile agent that contains its components to view the contents.

5.2 A Video-On-Demand System

We try to develop a dynamically adaptable video-on-demand system based on the mobile agent system presented in this paper. A typical video-on-demand system consists of

Figure 3: Window of the Compound Letter Agent

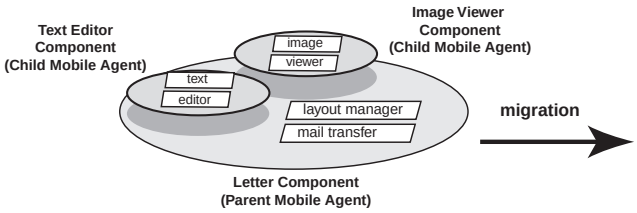

Figure 4: Structure of the Compound Letter Agent

a server and its clients. The former is given as a file-sharing server for allowing multimedia contents to be stored and transmitted and the latter receives the transmitted multimedia contents from the server and supports user interfaces to show us them.

The transmission of multimedia contents should be performed through protocols appropriate to the multimedia contents. Therefore, we need a mechanism to automatically and dynamically change only the way of transmitting multimedia contents, without changing the other facilities, including user interfaces.

Our video-on-demand system consists of a server agent and client agents that are built on the MobileSpaces runtime system. The server agent is given as a stationary container of child mobile agents, called *sender* agents, which can send multimedia contents through the protocols appropriate to the contents. Also, agent migration is introduced as a basic mechanism for obtaining and changing functions of the server agent. That is, when a video server is required to make use of a new protocol, it can extend the protocol to itself by migrating the agents, which offer the protocol.

Like the server agent, each client agent is a mobile agent that supports user interfaces which are preferred by its user and can be equipped with more than one agent, called *receiver* agents, which can receive multimedia contents from networks. Before transmitting multimedia contents, these receiver agents arrive at the client agent and then pass their received multimedia contents to the client agent through a common interface between client agents and receiver agents. Therefore, the client agent can dynamically extend

and change protocols for receiving multimedia contents and keep its other facilities including user interfaces.

A program for receiving multimedia contents is deployed at a client agent as follows:

(1) The server agent loads two child mobile agent, called *sender* agent and *receiver* agent, for implementing a protocol for receiving multimedia contents. The former includes a program for sending the multimedia contents and the latter for receiving them.

(2) The client agent sets the parameters of the child agents for a communication session through the protocol.

(3) The client agent migrates the receiver mobile agent to the client agent between which the communication session will be established.

(4) After arriving at the client agent, the receiver notifies the sender agent its arrival and then receives the multimedia contents from the sender agent.

This framework can also facilitate short-lived protocols because mobile agents have explicitly survival periods. They are automatically destroyed and discarded from the client nodes when their survival periods elapse.

In our current implementation of the video-on-demand system, our server agent is given as a container of server agents and client agents. Each sender agent is given as a HTTP server for dispatching various files, including video contents encoded in MPEG-1 and QuickTime. On the other hand, each receiver agent is a mobile agent, which can receive MPEG-1 files in a manner of HTTP and then a viewer agent embedded in the client agent views the files by means of Sun's Java Media Framework.

6 Conclusion

We presented a new mobile agent system which can support the two concepts, *agent hierarchy* and *inter-agent migration* presented in Section 2. The implementation of our system has paid as much attention to keep obeying the two concepts as possible. The system allows more than one mobile agent to be dynamically assembled into a single mobile agent. Furthermore, our system is characterized in its extensibility and adaptability, and mobile agent-based applications running on the system can naturally inherit these features. This advantage is especially useful in the construction of networked multimedia applications running on the system, because functions of multimedia applications must often be changed according to the multimedia contents which the applications process.

Finally, we would like to point out further issues. Security is essential in mobile agent computing. The current implementation of the system relies on the JDK 1.1 security manager and provides a simple mechanism for authentication of agents. However, many security features are left open for the next release. Moreover, we formalized a process calculus for reasoning about our hierarchical mobile agents [10]. We are interested in specifying and verifying multimedia applications of the system by using the calculus.

References

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

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

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

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

[5] A. Karmouch, Mobile Software Agents for Telecommunications, IEEE Communication Magazine, vol. 36 no. 7, 1998.

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

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

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

[9] I. Satoh, MobileSpaces: A Framework for Building Adaptive Distributed Applications using a Hierarchical Mobile Agent System, to appear in proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS'2000), IEEE Press, 2000.

[10] I. Satoh, A Formalism for Hierarchical Mobile Agents, to appear in proceedings of Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'2000), IEEE Press, June, 2000.

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

[12] D. L. Tennenhouse et al., A Survey of Active Network Research, IEEE Communication Magazine, vol. 35, no. 1, 1997.

[13] D. J. Wetherall, J. V. Guttag, and D. L.Tennenhouse, ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols, in Proceedings of International Conference on Open Architectures and Network Programming, April 1998.

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

[15] Y. Yemini, and S. da Silva, Towards Programmable Networks in Proceedings of FIP/IEEE International Workshop on Distributed Systems, October, 1996.