Application-Specific Routing for Mobile Agents

Ichiro Satoh*
National Institute of Informatics /
Japan Science and Technology Corporation
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
Tel: +81-3-4212-2546 Fax: +81-3-3556-1916

Abstract

An approach for building application-specific protocols for migrating mobile agents over a network is presented. The approach has two key ideas. One is to introduce mobile agents as first-class objects. The other is to build and deploy network protocols for agent migration as mobile agents, which allows network protocols for agent migration to be naturally implemented within mobile agents and then dynamically deployed at network nodes by migrating the agents that carry them. A prototype implementation was built on a hierarchical mobile agent system, and several practical protocols for agent migration were designed and implemented. These protocols can customize network processing for agent migration. This paper also demonstrates the utility of this framework by describing mobile agent-based routing protocols for migrating agents.

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 [7, 8, 12, 14]). Mobile agents have been used in the development of various networked applications. These applications often need application-specific network processing for agent migration. For example, a typical application of mobile agent technology is a task for monitoring the system, in which an agent travels to multiple nodes in a network to observe the components locally. The itinerary of a monitoring agent seriously affects the achievement and efficiency of their tasks. Also a mobile agent may be able to roam over multiple hosts or may have to return to its home server after each hop, instead of proceeding to another destination. However, such an itinerary is statically embedded inside the agent and is often designed dependently on the topology of a particular network. Therefore, it is nearly impossible for a mobile agent to dynamically configure its itinerary in response to changes in its network environments and its goals and be reused in another network.

This paper addresses the dynamic customization of network processing for agent migration, rather than for data transmission. I describe a new framework for dynamically deploying and changing network protocols for agent migration. My framework is characterized by two key ideas. The first is to apply active network technology to a network infrastructure for mobile agents. The second is to construct network protocols for agent migration within the agents themselves. That is, my mobile agent-based protocols can transmit mobile agents as first-class objects to their destinations. Also, the dynamic deployment of the mobile agent-based protocols can be naturally and easily performed by the migration of the agents that support them. Therefore, my framework allows network processing for mobile agents to be adapted to the requirements of visiting agents and to changes in the environment. The framework is built on a hierarchical mobile agent system called *MobileSpaces* [9]. This system can hierarchically organize more than one mobile agent and introduce mobile agents as service providers for other mobile agents. The notion of hierarchical mobile agents in MobileSpaces also allows active networks to be constructed based on a layered architecture, in which current active networks are often designed for application to particular layers. I also design several novel protocols. Although these are designed for agent migration in MobileSpaces, they can be easily applied to other active network frameworks.

This paper is organized as follows. Section 2 surveys related work, and Section 3 explains my approach to customizable network processing for agent migration. Section 4 briefly reviews MobileSpaces. Section 5 presents sev-

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

eral mobile agent-based protocols running on the system, and Section 6 shows the usability of my framework based on three real-world examples. Section 7 is a summary and mentions future issues.

2 Basic Framework

The framework presented in this paper provides a self-configuring infrastructure for mobile agents. It can deploy and configure network protocols for agent migration according to the requirements of visiting agents and changes in the network environment. This section outlines the overall architecture of the framework and describes the basic idea of network protocols based on the framework.

2.1 Hierarchical Mobile Agents

Mobile agents are autonomous programs, which can travel between different computers. My 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, my framework is built on MobileSpaces [9], which is characterized by two novel concepts: **agent hierarchy** and **inter-agent migration**. The former means that one mobile agent can be contained within another mobile agent. That is, mobile agents are organized in a tree structure. The latter means that each mobile agent can migrate to other mobile agents as a whole, with all its inner agents. Each agent can freely move into any agent in the same agent hierarchy except into itself or its inner agents, as long as the destination agent accepts it. A container agent is responsible for automatically offering its own services and resources to its inner agents and can subordinate its inner agents. Therefore, an agent can directly instruct its inner agents to move to another location. My protocols for agent migration transmit other mobile agents as first-class objects [4], in the sense that mobile agents can be passed to and returned from other mobile agents as values. A container agent is still mobile and also can migrate its inner agents to their destinations. Therefore, mobile agents are introduced as the only constituent of my network architecture and my network protocols for agent migration can be implemented within mobile agents. Such protocols can be dynamically and easily changed by migrating agents that implement the protocols to nodes.

2.2 Application-Specific Routing for Agent Migration

The achievement and efficiency of a moving agent depends on the routing of its migration. My framework allows routing protocols for agent migration to be performed by mobile agents. The protocols are classified into two approaches, in contrast with data transmission. The first approach provides a function similar to that of routers. I introduce the notion of forwarder agents, which are responsible for forwarding their inner agents. These agents stay at nodes and redirect other agents to new destinations through suitable data communication among multiple nodes. They hold tables describing part of the structure of the network. The second approach is similar to the notion of an active packet (also called a programmable capsule) in active network technology. Existing mobile agents can move from one node to another under their own control in the same way that active packets can define their own routing. I propose a navigating agent, which migrates itself and its inner agents to their destinations. Such a navigating agent can be designed to suit the topology of a particular network. Since the two approaches can hide the description of an agent's itinerary from its behavior, mobile agents become independent of the network structure and the modularity and reusability of application-specific mobile agents are promoted.

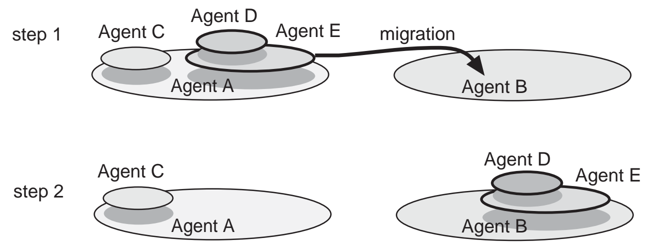

Figure 1. Agent hierarchy and inter-agent migration.

3 MobileSpaces

Here I briefly review MobileSpaces, which provides an infrastructure for building and executing mobile agents for network processing in addition to mobile agent based-applications. It supports mobile agents incorporating the notions of agent hierarchy and inter-agent migration presented in the previous section.

3.1 Runtime System

Each runtime system running on a computer can be regarded as an active node in active networking technology and corresponds to the root node of an agent hierarchy. It offers only three facilities.

Agent Hierarchy Management: The agent hierarchy is maintained as a tree structure in which each node contains a mobile agent and its attributes. Agent migration in an agent

hierarchy is performed simply as a transformation of the tree structure of the hierarchy. A container agent is introduced as a service provider for its inner agents. Each agent offers a collection of service methods that can be accessed by its inner agents. Each agent is active but subordinate to its container agent. That is, a container agent can instruct its inner agents to move to other agents or computers, marshal them, and terminate them.

Agent Execution Management: Each agent can have more than one active thread under the control of the system. The core system maintains the life-cycle state of agents. When the life-cycle state of an agent is changed, for example creation, termination, or migration, the core system issues events to invoke certain methods in the agent and its containing agents. The core system can explicitly limit the length of an agent's visit and the number of visiting agents. When the time limit of a staying agent is reached, it can automatically terminate the agent. This limitation offers a mechanism for caching network protocols.

Agent Serialization and Security Management: The runtime system provides a facility for marshaling agents into bit streams and unmarshaling them later. The current implementation of the system uses the Java object serialization package for marshaling the states of agents, so agents are transmitted based on the notion of weak mobility [5]. The runtime system itself is designed to be independent of the environment, including the network infrastructure, so network processing for agent migration between neighboring computers, which corresponds to data-link layered protocols, can be performed by special mobile agents. Moreover, the current implementation has a built-in mechanism for transmitting agents over the network by using an extension of the HTTP running on TCP/IP. The runtime system verifies whether a marshaled agent is valid or not to protect the system against invalid or malicious agents, by means of Java's security mechanism.

3.2 Mobile Agent Program

Our mobile agents are programmable entities like other mobile agents. Each agent consists of three parts: a body program, context objects, and inner agents, as shown in Fig. 2. The body program is an instance of a subclass of abstract class `Agent`.[1] This class defines fundamental callback methods invoked when the life-cycle of a mobile agent changes due to creation, suspension, marshaling, unmarshaling, destruction etc., like the delegation event model in Aglets [7]. It 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 by the argument of the command. An inner agent cannot access any methods defined in its container agent. Instead, each container is equipped with a context object that 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 to interact with the environment, such as with 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.

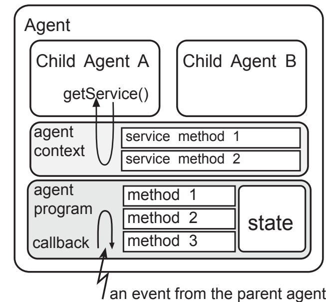

Figure 2. Hierarchical mobile agent.

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 is decremented by nodes as resources are consumed when the agent arrives at a new place. Nodes can discard agents when their limit reaches zero. To restrict the total resource bounds, when one agent creates another inside the network, the resources allocated to the created agent must be strictly less than those of the creating agent.

4 Application-Specific Protocols for Agent Migration

Since our framework can treat mobile agents as first-class objects, various types of network processing for mobile agents can be implemented as special mobile agents, called service agents, running on the MobileSpaces runtime system and can also handle and transfer other agents as data packets.

- Each mobile agent is designed to provide its service to its inner mobile agents. When a mobile agent is preparing for a trip over a network, the agent migrates itself into a service agent that is suitable for providing appropriate network processing in the same agent hierarchy and then the service agent automatically transfers the visiting agent or migrates itself to its destination, or

[1] Examples of mobile agent programs are given in the Appendix.

delegates other service agents in the same agent hierarchy.

- A runtime system permits one service to be provided by one or more service agents. That is, different network protocols for agent migration can be supported by different service agents. Also, an agent can dynamically select a suitable service agent in the current execution environment and move itself to the selected agent to access its required service.

- Since network protocols are performed by mobile agents, the protocols can be dynamically and autonomously deployed at nodes by migrating the corresponding agents to the nodes.

Hereafter, we present two basic protocols for agent migration. Since these protocols are given as abstract classes in the Java language, we can easily define further application-specific protocols by extending the basic protocols.

4.1 Protocol Distribution

Given a dynamic network infrastructure, a mechanism is needed for propagating mobile agents that support protocols to where they are needed. The current implementation of our framework provides the following three mechanisms: (1) mobile agent-based protocols autonomously migrate to nodes at which the protocols may be needed and remain at the nodes in a decentralized manner; (2) mobile agent-based protocols are passively deployed at nodes that may require them by using forwarder agents prior to using the protocols as distributors of protocols; and (3) moving agents can carry mobile agent-based protocols inside themselves and deploy the protocols at nodes that the agents traverse. This mechanism can improve performance in the expected common case of agent migration, i.e., a sequence of agents that follow the same path and require the same processing. All the mechanisms are managed by mobile agents, instead of by the runtime system. As a result, the deployment of transmitter agents needs to be performed by other transmitter agents.

4.2 Routing Mechanisms for Agent Migration

Application-specific mobile agents often need to travel to multiple nodes 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 nodes and correspond to kinds of application-specific routing protocols.

4.2.1 Navigator Agent Approach

The first approach offers a service provider agent, called a *navigator*, for conveying inner agents over a network, as shown in Figure 3. Each navigator agent is a container of other agents and travels with them in accordance with a list of nodes 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. Each navigator has a routing mechanism for managing a routing table consisting of nodes the navigator agent needs to visit. It maintains a list of nodes to be visited and provides methods for dynamically adding and removing elements from this list. Whenever a navigator agent moves to a new place, it 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 event-based communication. Upon arrival at a place, the navigator propagates certain events to its inner agents, instructing them to do something during a given time period. After the events have been processed by the inner agents, the navigator continues with its itinerary.

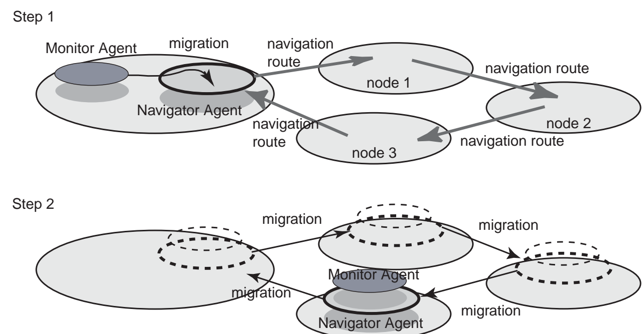

Figure 3. Navigator agent for traveling among nodes with its inner agent.

4.2.2 Forwarder Agent Approach

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 nodes and automatically transfer its inner agents to specified nodes through appropriate transmitter agents as shown in Figure 4. Consequently, a forwarder agent can be regarded as a programmable router for mobile agents.

The use of forwarder agents allows various routing schemes used in wired and wireless networks to be eas-

ily performed and evaluated. Such forwarder agents are dynamically deployed at nodes and coordinate with each other to redirect moving agents to their destinations. That is, when an agent requests a forwarder agent to migrate to its destination, the forwarder agent makes an effort to transfer the moving agent to the destination. However, if the destination is not reachable, it tries to transfer the moving agent to another forwarder agent running on an intermediate node as near to the destination as possible. Each forwarder agent will repeat the entire process in the same way until the agent arrives at the destination.

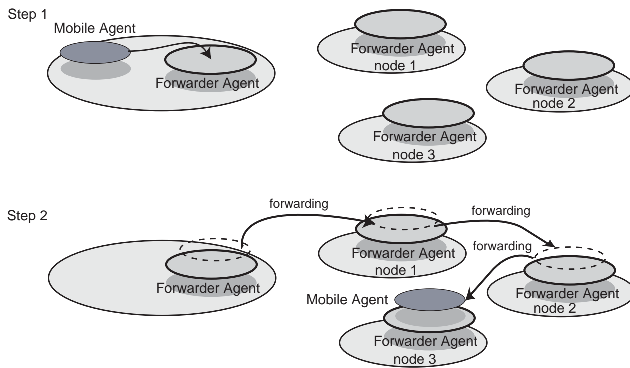

Figure 4. Routing agents for forwarding agent the next nodes.

4.3 Current Status

MobileSpaces 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 system. I provided several useful libraries for constructing network protocols within mobile agents. I have also developed various mobile agent-based protocols, for example transmitter agents for TCP, UDP, and SMTP, forwarder agents, and navigator agents for traveling among multiple active nodes, as presented in this paper. These protocols were written in Java and tested using MobileSpaces.

My current implementation was not built for performance. However, in order to compared two routing protocols, the forwarder agent approach and the navigator agent approach I measured the per-hop latency in microseconds and the throughput of a single node in agents per second in a network, which consisted of PCs (Intel Pentium III-500 MHz with Windows2000 and JDK 1.3) connected by 100Mbps Ethernet via a switching hub. The per-hop latency of migrating agents using simple forwarder agents running on the computers was 38 ms per hop and the throughput was 9.2 agents per second. The forwarder agent determines the computer that their inner agents will visit at the next hop; its routing tables are maintained by periodically polling the routing table of the SNMP agent. In contrast, the per-hop latency of migrating agents using a simple navigator agent running on the computers was 42 msec per hop and the throughput was 8.3 agents per second. The navigator agent migrated itself and its inner agents to the nodes sequentially. In both cases, I migrated minimal-size agents, which consist of only the common callback methods invoked at changes in their life-cycle states by the runtime system, by using a built-in mechanism based on the PUT method of HTTP. The cost of an agent migration between two neighboring computers was measured to be 31 ms and the cost of an agent migration in an agent hierarchy was measured to be 5 ms, including the cost for checking whether the visiting agent was permitted to enter the destination agent. This results from the cost of agent hierarchy management in MobileSpaces and thus is basically dependent on the speed of the CPU.

I compared the two routing approaches. In this preliminary experiment, the former was better than the latter, because the navigator agent needs to migrate not only the target agent but also itself. Also, when both approaches migrated more than one mobile agent in a network, the congestion of each computer was occasionally unbalanced, because my agent-based protocols are performed asynchronously. All the above experiences were measured in a trial without any optimization of performance and thus I could not strictly evaluate them yet. However, the overhead of my mobile agent-based protocols in the latency of each agent migration is reasonable for a high-level prototype of application-specific protocols for agent migration, instead of data communication. The throughput of each agent migration is limited by the security mechanism of the MobileSpaces system rather than by the protocols. We believe that the current throughputs are fast enough for the deployment of mobile agent-based applications.

5 Examples

This section describes two practical examples of our framework in order to demonstrate how it can be exploited.

5.1 Agent Migration for Network Management

As mentioned, a typical application of mobile agents is as a monitoring system for network management. A discussion of the suitability of mobile agents in network management can be found in [3, 6]. Using navigator agents presented in the previous section, we constructed a system for monitoring a set of equipment located at nodes in a network and reacting to certain behavioral patterns. A monitoring agent collects the network traffic load by accessing SNMP data

from the management information base. However, it has no mechanism for its own itinerary and thus is not dependent on a particular network. On the other hand, a navigator agent is responsible for periodically traveling among nodes in a network. It can be designed for navigating in a particular network. And it can guide monitoring agents inside itself through its itinerary over the network.

When a monitoring agent is preparing to monitor a network, it enters a navigator agent designed for that network. The navigator then generates an efficient travel plan for visiting certain nodes in the network. Next, it migrates itself and the monitoring agent to those nodes sequentially. When it arrives at each destination, it dispatches certain events to its inner agents at specific timings. A navigator agent can handle exceptions such as inactive hosts on behalf of monitoring agents while trying to migrate itself and its inner agent to new destinations. When the agent has to travel over a network more than one time, it can reflect the result of its previous itinerary, such as reachable nodes and arrival timings, in the next itinerary.

5.2 Disconnection-Tolerant Agent Migration

Mobile agent technology has the potential to mask disconnections in mobile computing. This is because once a mobile agent has completely transferred to a new location, it can continue its execution there, even if the new location is disconnected from the source location. However, the technology often cannot solve network failures during the process of agent migration. That is, agents can be migrated from the source to the destination, when all the links from the source to the destination are established at the same time. However, mobile computers do not have a permanent connection to a network and are often disconnected for long periods of time. When a mobile agent on a mobile computer wants to move to another mobile computer through a local-area network, both computers must be connected to the network at the same time.

To overcome this problem, we introduce *relay* agents, which are an extension of the forwarder agent with the notion of store-and-forward migration, as shown in Figure 5. This notion is similar to the process of transmitting electronic mail using SMTP. When an agent requests a relay agent on the source node to migrate to its destination, the relay agent makes an effort to transmit the moving agent to the destination through transmitter agents. If the destination is not reachable, the relay agent automatically stores the moving agent in its queue and then periodically tries to transmit the waiting agent to either the destination or a reachable intermediate node as close to the destination as possible. The relay agent to which the moving agent is transferred will repeat the process in the same way until the agent arrives at the destination. When the next node on a route to the destination is disconnected, the agent is stored in the current place until the node is reconnected. When a mobile computer is attached to a network, its relay agent multicasts a message to relay agents on the other connected computers. After receiving a reply message from relay agents at the destinations of agents stored in its queue, it tries to transfer those agents to their destinations.

6 Related Work

Many mobile agent systems have been released over the last few years, for example, Aglets [7], Mole [12], Telescript [14], and Voyager [8]. To my knowledge, none can extend and adapt their functions to the requirements of their visiting agents and applications while they are running, whereas this framework can. Although mobile agents need to be used in heterogeneous environments, for example, mobile computers, information appliances, and wireless networks, existing systems explicitly and implicitly assume a particular network infrastructure.

An application-specific mobile agent must make a network-dependent itinerary in order to travel to multiple hosts and perform its task at those hosts. Most existing mobile agent systems assume that each mobile agent embeds such an itinerary inside itself. However, it is difficult to determine the itinerary at the time the agent is designed or instantiated because the network topology cannot always be known. Therefore, such a mobile agent cannot be used in another network. To overcome this problem, Aglets introduces the notion of traveling patterns [1], like design patterns studied in software engineering. This notion allows us to design application-specific itineraries independent of the logical behaviors of mobile agents. However, the itinerary patterns must to be statically and manually embedded in their mobile agents. Consequently, the agents cannot dynamically change their itineraries and thus cannot travel beyond familiar networks. MobileSpaces can dynamically adapt itself to changes to dynamic environments. However, my previous papers lacked any approach for building application-specific protocols for agent migration, although it presented the preliminary notion of mobile agent-based channels for agent migration between neighboring computers, called transmitter agents, which corresponds to a data-link layered protocol.

My framework is similar to active network technologies [13] and also makes several contributions to the technologies. Mobile agent technology has been used for prototype implementations of active networks. For example there have been many attempts to apply mobile agent technology to the development of active networks [2, 6], since mobile agents can be regarded as a special case of mobile code technology, which is the basis of most existing active network technologies. In contrast, the goal of this paper is to apply active network technology to mobile agent technol-

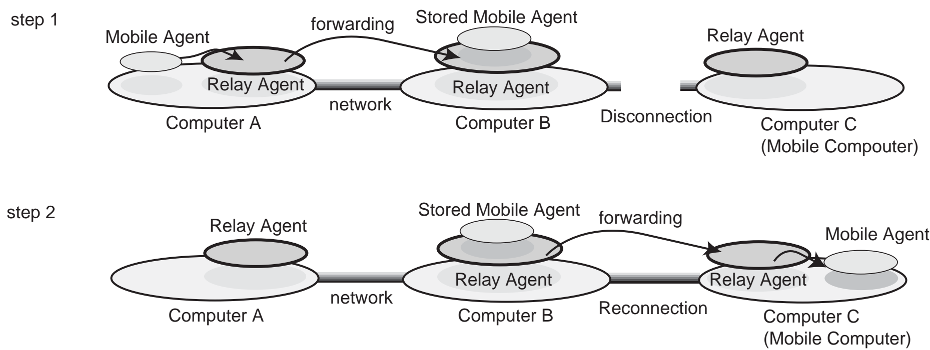

Figure 5. Relay agent for disconnection-tolerant network.

ogy. In fact, my framework can introduce mobile agents as first-class objects so it allows existing network protocols based on the two approaches to be easily implemented by mobile agents.

7 Conclusion

I have presented a framework for building application-specific routing for mobile agents. In this framework, network protocols for agent migration can be naturally implemented within mobile agents and thus can be dynamically added to and removed from the system by migrating the corresponding agents, according to the requirements of visiting agents and changes in the environment. The framework is also significant and practical as a powerful testbed for building and evaluating various algorithms for mobile entities over a network. My prototype implementation built on a Java-based mobile agent system, called MobileSpaces, allowed us to experiment with the construction and deployment of these protocols.

Finally, I would like to mention further issues. My early performance measurements indicate that the performance of my mobile agent-based protocols is reasonable for a high-level prototype and fast enough for experimenting with application-specific protocols for agent migration. However, the performance of the current implementation is not yet satisfactory and thus further measurements and optimizations are needed. I have constructed various mobile agent-based applications, such as workflow management, CSCW, distributed information retrieval, active networks, and compound documents [10]. I are interested in developing mobile agent-based system for further particular applications.

References

[1] Y. Aridor, and D.B. Lange, “Agent Design Patterns: Elements of Agent Application Design”, in Proc. *Second International Conference on Autonomous Agents (Agents ’98)*, ACM Press, 1998, pp. 108-115.

[2] C. Baumer, and T. Magedanz, “The Grasshopper Mobile Agent Platform Enabling Short-Term Active Broadband Intelligent Network Implementation”, in Proc. of *Intenral Working Conference on Active Networks*, LNCS, Vol.1653, Springer, 1999. pp.109–116.

[3] A. Bieszczad, B. Pagurek, and T. White, “Mobile Agents for Network Management”, *IEEE Communications Surveys*, Vol. 1, No. 1, Fourth Quarter 1998.

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

[5] A. Fuggetta, G. P. Picco, and G. Vigna, “Understanding Code Mobility”, *IEEE Transactions on Software Engineering*, 24(5), 1998. pp. 352-361.

[6] A. Karmouch, “Mobile Software Agents for Telecommunications”, *IEEE Communication Magazine*, vol. 36 no. 7, 1998.

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

[8] ObjectSpace Inc, “*ObjectSpace Voyager Technical Overview*”, ObjectSpace, Inc. 1997.

[9] I. Satoh, “MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System”, in Proc. *International Conference on Distributed Computing Systems (ICDCS’2000)*, IEEE Computer Society, April, 2000, pp.161-168.

[10] I. Satoh, “MobiDoc: A Framework for Building Mobile Compound Documents from Hierarchical Mobile Agents”, in Proc. Symposium on Agent Systems and Applications / Symposium on Mobile Agents (ASA/MA’2000), Lecture Notes in Computer Science, Vol.1882, Springer, 2000, pp.113-125.

[11] I. Satoh, “Adaptive Protocols for Agent Migration”, in Proc. *IEEE International Conference on Distributed Computing Systems (ICDCS’2001)*, IEEE Computer Society, 2001, pp.711-714.

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

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

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

Appendix: Agent Programs

Suppose an agent migrates between two nodes by using transmitter agents as described in Section 6. The following code fragment is the `TCPTransmitter` class that defines simple transmitter agents on these nodes. `TCPTransmitter` agents can exchange agents with each other via their own communication protocol. Since these `TCPTransmitter` agents are mobile agents, we can create and allocate them on nodes dynamically.

```
 1: public class TCPTransmitter extends
 2: TransmitterAgent implements AgentEventListener {
 3:   public TCPTransmitter() {
 4:     // registering itself as a listener
 5:     addAgentListener(this);
 6:     // offering a context
 7:     addChildrenContext(new BaseContext());
 8:     // registering itself as one of transmitters
 9:     registryAs("transmitter");
10:   }
11:   // invoked when an agent arriving
12:   public void add(AgentEvent evt) {
13:     Message msg = new Message("serialize");
14:     msg.setArg(evt.getSourceURL());
15:     // serializing the arriving agent
16:     byte[] data = (byte[])getService(msg);
17:     // dst specifies the original destination
18:     AgentURL dst = url.getTarget();
19:     // transmitting the serialized agent to dst
20:     send_agent(data, dst);
21:   }
22:   void send_agent(byte[] data, AgentURL dst) {
23:     // sending the serialized agent (data)
24:     // to the destination (dst)
25:     ...
26:   }
27:   void receive_agent(byte[] data, AgentURL dst){
28:     // invoked at receiving data
29:     // for a remote Transfer
30:     Message msg = new Message("deserialize");
31:      // data is a serialized agent
32:     msg.setArg(data);
33:     // dst specifies the destination agent
34:     msg.setArg(dst);
35:     // deserializing data at dst
36:     AgentURL url = (byte[])getService(msg);
37:     ...
38:   }
39:   ...
40: }
```

Our system has an event mechanism based on the delegation-based event model introduced in the Abstract Window Toolkit of JDK 1.1 or later, so each agent must be informed of lifecycle state changes so that they can release various resources, such as files, windows, and sockets, which are captured by the agent. To hook these events, each agent can have one or more listener objects. A listener object implements a specific listener interface extended from the generic `AgentEventListener` interface, which defines callback methods that should be invoked by the core system before or after the lifecycle state of the agent changes. For example, the `create()` method is invoked after creation, the `destroy()` method is invoked before termination, the `add()` method is invoked after accepting an inner agent, the `remove()` method is invoked before removing an inner agent, the `arrive()` method is invoked after arriving at the destination, and the `remove()` method is invoked before moving to the destination. The following code fragment defines the `SimpleNavigator` class, which is a simple implementation of the navigator agent presented in Section 5.

```
 1: public class SimpleNavigator extends
 2: NavigatorAgent implements AgentEventListener {
 3:   Vector route;
 4:   int i = 0;
 5:   public SimpleNavigator() {
 6:     // offering a context
 7:     addChildrenContext(new BaseContext());
 8:     // registering itself as a listener
 9:     addDefaultListener(this);
10:     // making a list structure
11:     route = new Vector();
12:     // the 1st destination
13:     route.addElement("first.place.com");
14:     // the 2nd destination
15:     route.addElement("second.place.com");
16:     // the 3rd destination
17:     route.addElement("third.place.com");
18:   }
19:   // invoked after creation
20:   public void create(AgentEvent evt) {...}
21:   // invoked after arriving
22:   public void arrive(AgentEvent evt) {
23:   // making a message named "do"
24:     Message = new Message("do");
25:     // its argument is the current address
26:     msg.setArg(evt.getCurrentURL());
27:     // invoking the method of its inner agents
28:     dispatch(msg);
29:     System.out.println("moving to the next place");
30:     // trying to move to the next place
31:     moveToNextHop();
32:   }
33:   // invoked before migration
34:   public void leave(AgentURL url) {...}
35:   // invoked after accepting an agent
36:   public void add(AgentEvent evt) {
37:     moveToNextHop();
38:   }
39:   // invoked before removing an agent
40:   public void remove(AgentEvent evt) {...}
41:   private void moveToNextHop() {
42:     // i-th element of the route list
43:     String host = route.elementAt(i++);
44:     try {
45:     // requesting a tranmitter to migrate to "host"
46:       go(new AgentURL("transmitter://"+host));
47:     } catch (MalformedURLException e) {...}
48:   }
49:   ...
50: }
```