Vol. 44 No. 3 IPSJ Journal Mar. 2003

Regular Paper

Network Protocols for Mobile Agents by Mobile Agents

Ichiro Satoh†

This paper presents a framework for building network protocols for migrating mobile agents over a network. The framework allows network protocols for agent migration to be naturally implemented within mobile agents and to be constructed in a hierarchy as most data transmission protocols are. These protocols are given as mobile agents and they can transmit other mobile agents to remote hosts as first-class objects. Since they can be dynamically deployed at remote hosts by migrating the agents that carry them, these protocols can dynamically and flexibly customize network processing for agent migration according to the requirements of respective visiting agents and changes in the environments. A prototype implementation was built on a Java-based mobile agent system, and several practical protocols for agent migration were designed and implemented. The framework can make major contributions to mobile agent technology for telecommunication systems.

1. Introduction

Mobile agent technology is an emerging technology that makes it much easier to design, implement, and maintain telecommunication systems. Although this technology has been expected to be used in a variety of applications in distributed system settings, there have few attempts to apply it there. The reason for this is a mismatch in network processing, in addition to the problem of protecting against malicious agents or malicious hosts. Many applications often require application-specific network processing for migrating agents over a network. For example, a typical application of the technology is network management, where an agent travels to multiple nodes in a network to observe and access the components locally. The itinerary of such a monitoring agent seriously affects the achievement of its tasks and the efficiency with which they are accomplished. Moreover, a mobile agent for electronic commerce may have to be transformed into an encrypted bit stream before it can transfer itself over a network. However, existing mobile agent systems assume particular network infrastructures and cannot dynamically change their own network processing, because it is statically embedded in them.

The goal of this paper is to propose a framework for building configurable network protocols for agents migration over a network and to describe several practical network protocols based on the framework. Our framework is based on 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, our mobile-agent-based protocols can transmit mobile agents as first-class objects to their destinations. Also, the protocols can be dynamically and easily deployed by the migration of the agents that support these protocols. The framework can provide a useful testbed for implementing and evaluating different types of network processing for mobile agents.

Our framework makes several contributions to network technologies, particularly active networking approaches. It can simplify and advance the development of network protocols. This is because it introduces mobile agents as the only constituents of a self-configurable network system, so that users/programmers can naturally define both network protocols and packets in a single unified perspective of the system and can easily and dynamically deploy the protocols at remote nodes by migrating the corresponding agents. It also allows active networks to be constructed on the basis of a layered architecture, in which current active networks are often designed within particular individual protocol layers. The framework enables us to build and test active network services across more than one layer.

In this paper we describe the design goals of our framework (Section 2), briefly review our mobile agent system, which is a basis of the framework (Section 3), present several mobile-agent-based protocols for agent migration (Section 4), show several examples of the framework (Section 5), survey related work (Section

† National Institute of Informatics/Japan Science and Technology Corporation

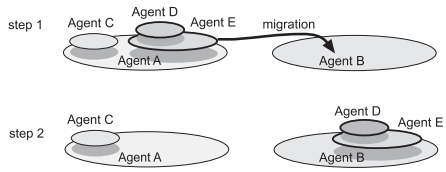

Fig. 1 Agent hierarchy and inter-agent migration.

6), and show some conclusions and describe research directions (Section 7).

2. Approach

This paper addresses the building of configurable network protocols for agents migration. For a discussion of a description of the configuration mechanism itself of mobile agent-based components for operating mobile agents, including agent migration over a network, readers are referred to a previous paper [9].

2.1 Mobile Agents as First-Class Objects

Mobile agents are autonomous programs that can travel between different computers. In the framework presented in this paper, mobile agents are computational entities like other mobile agents. When an agent migrates, not only the code of the agent but also its state can be transferred to the destination. The framework is built on a mobile agent system, called MobileSpaces, presented in a previous paper [9]. The system 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, as long as the destination agent accepts it, as shown in **Fig. 1**. A container agent is responsible for automatically offering its own services and resources to its inner agents, and it can subordinate its inner agents. Therefore, an agent can transmit its inner agents to another location as first-class objects [4], in the sense that mobile agents can be passed to and returned from other mobile agents as values. As a result, network protocols for agent migration can be implemented within mobile agents, so can be replaced by migrating the corresponding the agents.

2.2 Layered Protocols for Agent Migration

Most protocols for data transmission, such as the OSI model.☆, are often arranged in a hierarchy of layers. Each layer presents an interface to the layers above it and extends services provided by the layer below it. The hierarchical structure of mobile agents enables network protocols for agent migration to be organized hierarchically. That is, each agent hierarchy consisting of mobile agent-based protocols can be viewed as a protocol stack for agent migration, as shown in **Fig. 2**, and agent migration in an agent hierarchy is introduced as a basic mechanism for accessing services provided by the underlying layer. Mobile agent-based protocols in the bottom layer correspond to data-link layered protocols in the OSI model. They are responsible for establishing point-to-point channels for agent migration between neighboring computers. The middle layer corresponds to the network layer of the OSI model and offer routing protocols for agent migration. The protocols transmit mobile agents beyond the channels between directly connected nodes. The framework enables routing protocols for agent migration to be performed by mobile agents.

3. MobileSpaces: An Extensible Mobile Agent System

This section briefly reviews MobileSpaces, which provides, in addition to mobile agent-based applications, an infrastructure for building and executing mobile agents for network processing. MobileSpaces is built on a Java virtual machine and mobile agents are given as Java objects. Its architecture is designed based on a micro-kernel architecture and consists of two parts: a core system and higher-level components. The former offers only minimal and common functions, independent of the underlying environment. The latter is a collection of higher-level components outside the core system that provide other functions, including agent migration over a network, which may depend on the surrounding environment.

3.1 Core System

Each core system is made as small as possible for portability. It has only three functions:

Agent Hierarchy Management:

Each core system corresponds to the root node of an agent hierarchy, which is maintained as a tree structure in which each node contains

☆ This framework does not involve OSI protocols. The OSI model is mentioned here merely to provide a frame of reference for readers acquainted with the model.

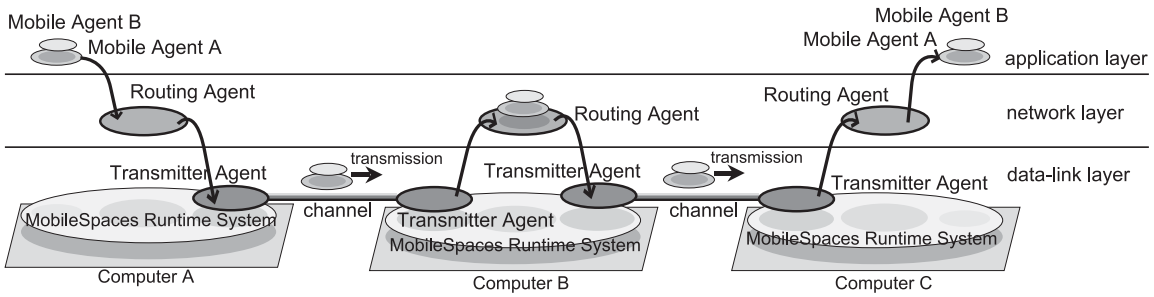

Fig. 2 Architecture of mobile-agent-based protocols for agent migration.

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.

Agent Execution Management:

Each agent can have more than one active thread under the control of the core system. The core system maintains the life-cycle state of agents. When the life-cycle state of an agent is changed for example, at creation, termination, or migration the core system issues certain events to invoke certain methods in the agent and its containing agents.

Agent Serialization and Security Management:

The core system has a function for marshaling agents into bit streams and unmarshaling them later. The current implementation of the system uses a Java object serialization package for marshaling the states of agents, so agents are transmitted is accordance with the notion of weak mobility [5]. The core 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

Each mobile agent consists of three parts: a body program, context objects, and inner agents. The body program is an instance of a subclass of abstract class `Agent`. 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 in the same agent hierarchy. An inner agent cannot access any methods defined in its container agent, including the core system. Instead, each container is equipped with a context object that offers service methods in a subclass of the `Context` class, such as the `AppletContext` class of Java's Applet. These methods can be indirectly accessed by the inner agents of a container to get information about and interact with the environment, including the container, sibling agents, and the underlying computer system.

4. Mobile-Agent-Based Protocols for Agent Migration

Since this 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 core system of MobileSpaces. These service agents are hierarchically organized as a protocol stack.

- Each service agent is designed to provide its service to its inner mobile agents. Therefore, each service agent in a lower layer can be viewed as a service provider for agents in an upper layer. The movement of an agent to a service agent in a lower layer in the same agent hierarchy corresponds to the process of applying the network service of the service agent to the moving agent.
- Each runtime system permits one service to be provided by one or more service agents. That is, different network protocols can be supported by different service agents. Moving agents or upper-layer protocols can dynamically select a suitable agent for their requirements and migrate their inner agents to the selected agent.
- Since service agents for performing protocols are still mobile, the protocols can be dynamically deployed at hosts by migrating the agents to the hosts.

Next, we present several 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 these basic protocols.

4.1 Point-to-Point Channels for Agent Migration

Our framework enables point-to-point agent migration to be provided by mobile agents, called *transmitters*, instead of by the core system. Transmitter agents correspond to a data-link layer or a network layer and are responsible for establishing point-to-point channels for agent migration between the source host and destination host through a (single-hop or multiple-hop) data transmission infrastructure, such as TCP/IP. They conceal the variety in the underlying network infrastructure and exchange their inner agents with coexisting agents running to remote computers through their preferred communication protocols. Furthermore, transmitter agents are implemented as mobile agents so that they can be dynamically added to and removed from the system by migrating and replacing the corresponding agents, to keep up with changes in the network environment. After an agent arrives at a transmitter agent from the upper layer, the arriving agent indicates its final destination. The transmitter suspends the arriving agent (including its inner agents), then requests the core system to serialize the state and code of the arriving agent. Next, it sends the serialized agent to a coexisting transmitter agent located at the destination. The transmitter agent at the destination receives the data and then reconstructs the agent (including its inner agents) and migrates it to the destination or to specified agents that offer upper-layer protocols.

Since each runtime system can be equipped with more than one transmitter agent, upper-layer protocols can dynamically select a suitable agent in their requirements and migrate their inner agents to the selected transmitter agents. We have already implemented several transmitter agents based on data communication protocols widely used on the Internet, such as TCP, HTTP, and SMTP. Authentication services normally available in secure communications infrastructure include this functionality. Therefore, we implemented transmitter agents, which can exchange agents with each other through Secure Socket Layer (SSL), one of the most popular secure communication protocols in the Internet. We provide a virtual class in Java that can be specialized to create transmitter agents for various protocols. Therefore, we can easily build point-to-point channels on other secure communication protocols for data transmission.

4.2 Application-Specific Routing for Agent Migration

A mobile agent often has to visit multiple hosts to perform its task, so it has to create an application-specific and network-dependent itinerary. On the other hand, channels between transmitter agents support point-to-point agent migration. Therefore, we need mechanisms to migrate an application-specific mobile agent among multiple hosts so it can perform its tasks. However, it is difficult to determine the itinerary at the time the agent is designed or instantiated. In addition, even if an agent were optimized for a particular network, it might not be reusable in another one. Therefore, we introduce two approaches for determining and managing the itinerary of agents, which are built on transmitter agents running on hosts.

4.2.1 Forwarder Agent Approach

The first approach provides a function similar to that of routers. We introduce a service provider, called a *forwarder* agent, for redirecting moving agents to new destinations. Each forwarder agent stays at a host. When receiving agents, it redirects the agents to their destinations through point-to-point channels established among multiple nodes, as shown in **Fig. 3**. Each forwarder agent holds a table describing part of the network structure, and can be dynamically deployed at nodes and coordinate with other forwarder agents to redirect moving agents to their destinations. However, if the destinations are not reachable, it tries to transfer the agents to other forwarder agents running on intermediate nodes as near the destinations as possible. Each forwarder agent will

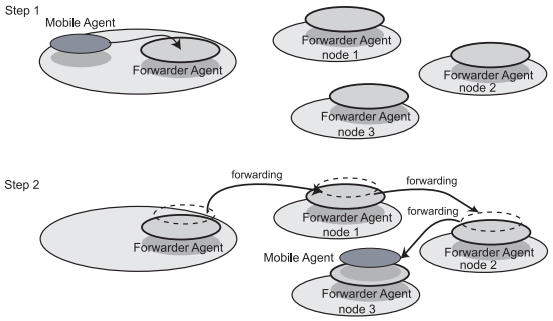

Fig. 3 Routing agents for forwarding another agent to the next nodes.

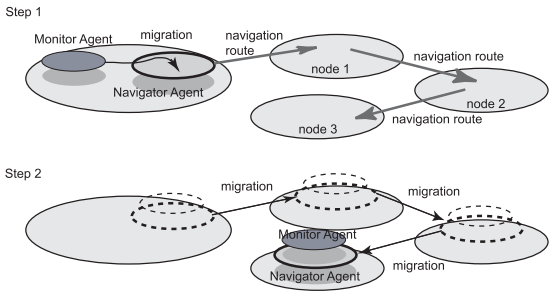

Fig. 4 Navigator agent for traveling among nodes with its inner agent.

repeat the entire process in the same way until it arrives at the destination.

4.2.2 Navigator Agent Approach

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, just as active packets can define their own routing. We propose a service provider, called a *navigator*, for conveying inner agents over a network, as shown in **Fig. 4**. 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 that 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.

4.2.3 Discussion

The interaction between a forwarder or navigator agent and its inner agents is based on event-based communication. Upon receiving agents, a forwarder propagates certain events to its visiting agents instructing them to do something during a given time period. After the events have been processed by the inner agents, the forwarder navigator transmits another forwarder agent running at the next host. Upon arriving at a place, a navigator propagates certain events to its inner agents. After the events have been processed by the inner agents, the forwarder or navigator continues on its itinerary. 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 enhanced.

Our forwarder and navigator agents are useful in the authentication of mobile agents. This is because each forwarder can limit the forwarding range of its inner agents and receive only those agents moved by authorized forwarder agents. Therefore, each node can explicitly reject any agents from unauthorized hosts. Each navigator agent can define its own reachable nodes and each node can accept only authorized navigator agents. That is, when an agent is moved by a navigator agent whose reachable nodes are limited, it can travel only among the reachable nodes of the navigator agent.

4.3 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 there 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 basically managed by mobile agents, instead of by the runtime system and thus can be customized easily.

4.4 Current Status

The framework presented in this paper and its mobile agent-based protocols were implemented on MobileSpaces in the Java language. They can be run on any computer with a JDK 1.2-compatible Java runtime system. The framework provides several useful libraries for constructing network protocols within mobile agents. Several mobile agent-based protocols were developed, in addition to the protocols presented in the next section. They include agents for establishing channels through TCP, HTTP, and SMTP, forwarder and navigator agents for traveling among multiple computers according to their own static routing tables and

Table 1 Performance of agent migration.

protocol	Latency (msec)	Throughput (agents/sec)
transmitter agent	25	7.2
forwarder agent	38	6.0
navigator agent	42	5.6

SNMP agents at each computer. The current implementation of this framework was not built for performance. However, in order to compare two routing protocols, the forwarder agent protocol and the navigator agent protocol, we measured the per-hop latency and the throughput of a single node in agents per second in a network consisting of ten PCs (Intel Pentium III-600 MHz with Windows 2000 and JDK 1.3) connected by 100-Mbps Ethernet via a switching hub.

Table 1 shows the basic performance of agent migration over a network. We measured the latency through two computers and the throughput of a single node in agents per second. In both cases, we migrated minimal agents, which consist only of the common callback methods invoked at the changes of their life-cycle states by the core system[☆]. They correspond to a null RPC and their data size is about 2.5 Kbytes (zip-compressed).

The first result shows the cost of agent migration between two simple transmitter agents, whose code is shown in the Appendix. Each transmitter agent can exchange the code and state of its inner agent with every other agent through an application-level protocol for agent transmission over a TCP channel. The marshaled agent consists of its serialized state, its code, and its attributes, such as name and capability, and is packed and compressed into a bit-stream. The latency shows the sum of the marshaling, zip-based compression, TCP connection opening, transmission, security verifications, decompression, and unmarshaling. Next, we measured the cost of sequentially relaying a minimal agent from forwarder agent to forwarder agent while running eight computers. The second result shows the average of one-hop transmission based on the forwarder agents. Each forwarder agent determines the computers that their inner agents will visit at their next hops, according to their own routing tables maintained by periodically polling the routing table of the SNMP agent, and then delegates a transmitter agent to dispatch their inner agents to the computers. The third result shows the cost of one-hop agent migration by using a simple navigator agent which has an itinerary list of eight computers and migrates itself and its inner agent to the computers sequentially by combining with the above transmitter agents. We present the code of the navigator agents in the Appendix.

We believe that the latency of agent migration in our framework is reasonable for a high-level prototypes of adaptive protocols for agent migration, instead of data communication. However, we anticipate further reductions in the latency costs, because the costs in the above table are basically dependent on the protocols rather than the MobileSpaces system and all the protocols used in these experiments were made as simple as possible, since our intention was to show the basic costs of our framework. The results of throughput measurements are limited by the MobileSpaces system and the underlying operating system.[☆☆]. Also, when these approaches migrate more than one mobile agent over a network, the congestion of each computer is occasionally unbalanced at some computers, because our agent-based protocols are performed asynchronously. The current throughput is fast enough for migrating application-specific agents. We compare two routing protocols: the forwarder agent approach and the navigator agent approach. In this experiment we found that the former was better, because the latter needs to transfer two mobile agents to the destination. Although the former needs more agent migration in agent hierarchies than the latter, the overhead of agent migration in a hierarchy is less than a few milliseconds.

Some readers may want to correlate the transmitter agent with protocols in the data-link layer of the OSI and the forwarder and navigator agents with protocols in the network layer of the model. The performance of our mobile agent-based protocols is inferior to that of existing protocol implementations for data transmission, including OSI-based protocols. However, the goal of the framework is to

☆ The moving agent is a simple implementation of the `DefaultEventListener` interface presented in Ref. 9).

☆☆ The performance of the current implementation is dependent on the cost of thread creation in the underlying operating system.

provide an infrastructure for dynamically and easily customizing network processing for agent migration and the performance of the protocols is sufficient for the migration of application-specific agents, which are programmable entities rather than passive data.

5. Examples

This section describes three practical examples of this framework to demonstrate how it can be used.

5.1 Example: Locating Mobile Agents

When an agent wants to interact with another agent, it must know the current location of the target agent. Therefore, we need a mechanism for tracking a moving agent. An extension of our forwarder agent offers such a mechanism shown in **Fig. 5**. Just before an agent moves into another agent, it creates and leaves a forwarder agent behind. The forwarder agent inherits the name of the moving agent and transfers any subsequently visiting agent to the new location of the moving agent. Therefore, when an agent wants to migrate to another agent that has moved elsewhere, it can migrate into the forwarder agent instead of the target agent. The forwarder agent then automatically transfers it to the current location of the target agent. Several schemes for efficiently locating mobile agents have been explored in the field of process/object migration in distributed operating systems. Our forwarder agents can easily support most of these schemes, because they are programmable entities and can flexibly negotiate with each other through data transmission protocols such as TCP/IP.

5.2 Agent Migration in Mobile Computing

Mobile agent technology has the potential to mask disconnections in some cases. This is because, once a mobile agent is completely transferred to a new location, the agent can continue its execution at the new location, even when the new location is disconnected from the source location. However, the technology often cannot solve network failures in 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.

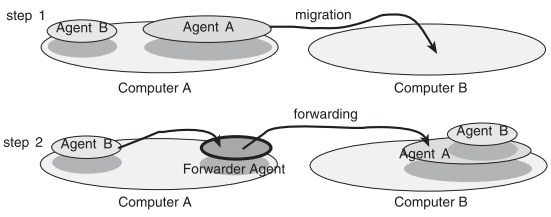

Fig. 5 Forwarder agents for locating moving agents.

To overcome this problem, *relay* agents are constructed by extending the forwarder agent approach to the notion of store-and-forward migration, as shown in **Fig. 6**. This notion is similar to the process of transmitting electronic mail by using SMTP. When an agent requests a relay agent on the source host 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 host 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 host on the route to the destination is disconnected, the moving agent is stored in its current place until the host is reconnected. When a mobile computer is attached to a network, its relay agent multicasts a message to relay agents on other connected computers. After receiving a reply message from the relay agents at the destinations of agents stored in its queue, the relay agent tries to transfer those agents to their destinations.

5.3 Example: Encrypting Mobile Agents

For security reasons, an agent should be encrypted before migrating itself over the Inter-

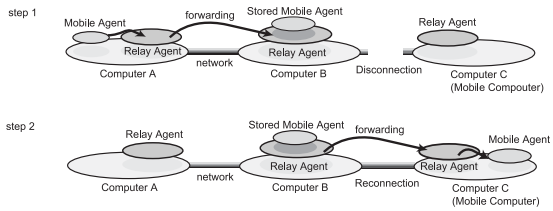

Fig. 6 Relay agent for tolerant network disconnection.

net. Since each navigator agent can treat its inner agents as first-class data, it can transform them and then carry them to their destinations. Our current implementation provides a special navigator agent, called a safe-box agent, which can be viewed as a cash transport truck. Each agent is a container of other agents and has a secret-key based cryptographic procedure inside itself. When an agent visits a safe-box agent the agent receives another agent, the safe-box agent automatically serializes and encrypts the visiting agent under a secret key. Next, it migrates itself to the destination as a whole with all its inner agents and its cryptographic procedure except for its secret keys. After arriving at the destination, the safe-box agent still keeps its inner agents inside it. When the destination provides its privacy to the visiting safe-box agent, the agent decrypts its inner agents. We can implement safe-box agents independently of any cryptographic algorithms, because the algorithms should be selected according to the requirements of the application. Each safe-box agent has an interface for hiding the differences between secret-key based cryptographic algorithms and can support any algorithms that can satisfy the interface. A nonstandardized cryptographic algorithm can be embedded into a safe-box agent without losing any interoperability, because the agent can carry a procedure for the algorithm and perform it at both the source host and destination host.

6. Related Work

Many mobile agent systems have been developed over the last few years, for example, Aglets [7], Telescript [14], and Voyager [8]. To our knowledge, none of them can dynamically extend and adapt their network processing for agent migration to the characteristics of current networks and the requirements of respective visiting agents, although mobile agents must be used in heterogeneous and dynamic network environments, such as in personal mobile communication, wireless networks, and active networks. This is because their agent migration protocols are statically embedded inside their systems.

The framework presented in this paper changes network processing by combining it with active network technology [13]. There have been many attempts to apply mobile agent technology to the development of active networks [2),3)] because mobile agents can be considered a special case in mobile code technology, which is the basis of existing active network technologies. For example, the Grasshopper system offers an active network platform consisting of stationary and mobile agents as service entities for telecommunication. In contrast, the framework presented in this paper applies active network technology to mobile agent technology. Existing active network approaches intend to customize network service in individual layers, such as network layer and transport layer in the OSI model, instead of across multiple layers. On the other hand, our framework can naturally model the layer hierarchy of network protocols by using the notion of hierarchical mobile agents in the MobileSpaces system. For example, an outer agent can be viewed as a packet or service provider in a lower layer and its inner agents as packets in an upper layer. Therefore, the framework provides an infrastructure for building and testing network protocols across more than one protocol layer.

In the literature of meta-level and self-reflective architecture, there have been many reported attempts to customize processing rather mobile agent technology. However, their customization mechanisms are often so complex that it is difficult to construct them and make them accessible and secure. We need to construct a simple and natural approach to configuring and adapting network processing for agent migration. To satisfy this requirement, the approach presented in this paper uses agent migration as a mechanism for deploying and managing agent migration protocols, because agent migration is one of the most essential mechanisms in mobile agent computing.

There have been few approaches to building configurable protocols for agent migration rather than data transmission. A mobile agent, which visits multiple hosts to perform its task, must have an application-specific itinerary. For example, a mobile agent may roam over more than one host without making any detours or may have to return to its home host after each hop instead of proceeding to another destination. Also, a network-dependent itinerary is often needed for a mobile agent to travel to multiple hosts efficiently. However, it is difficult to determine such an itinerary at the time the agent is designed or instantiated, because the network topology cannot always be known. Moreover, even if the itinerary of a mobile

agent were optimized for a particular network to travel to multiple hosts efficiently, it might not be reusable in another network. To overcome this problem, ADK [6] separates the travel itinerary of an agent from its behavior by building a mobile agent from a set of component categories: navigational components responsible for a travel itinerary and performer components responsible for executing one or more management tasks on each node. Aglets [7] introduces the notion of an itinerary pattern, which is similar to design patterns in software engineering, to shift the responsibility for navigation from an application-specific agent to a framework library described in [1]. Both approaches allow us to design the application-specific itinerary for an agent independent of the logical behavior of the agent, but the itinerary parts must be statically and manually embedded in the agent. Consequently, the agent cannot dynamically change its itinerary and cannot travel beyond its familiar networks.

We described a portable and extensible mobile agent system, MobileSpaces, in our previous paper [9]. The system serves as the basis for the framework presented in this paper. It can dynamically adapt its functions and structures to changes in the environments, but its goal is to provide a general platform for executing and migrating distributed applications. In our previous paper [10], we presented a architecture for building several agent migration protocols. That architecture is hierarchically organized like the notion of the protocol stack in existing data transmission protocols. While the goal of the previous paper was to propose the architecture and a few protocols available on the Internet, the goal of this paper is to propose various mobile-agent-based protocols for agent migration, in addition to the architecture. Hence, this paper describes several protocols that are not presented in the previous paper. Furthermore, in other previous papers [11),12)], we presented a mobile agent-based approach for network management. Although the approach is based on the notion of navigator agents presented in this paper, it is specific to network management, and the previous papers do not present other agent migration protocols, such as point-to-point channel agents and forwarder agents, studied in this paper.

7. Conclusion

This paper was described a framework for building configurable network protocols for agent migration. The framework provides a layered architecture for network protocols for migrating agents and allows these protocols to be naturally implemented by mobile agents. Therefore, network processing for mobile agents can be dynamically added to and removed from remote hosts by migrating corresponding agents. To demonstrate the utility of the framework, we developed several mobile agent-based protocols, such as point-to-point channels among neighboring hosts, and application-specific routing protocols for migrating agents among multiple nodes. A prototype implementation of the framework built on a Java-based mobile agent system called MobileSpaces was carried out. The framework can greatly simplify the development of active network technology [13]. This is because mobile agents are introduced as the only constituent of this framework and thus algorithms and protocols for active networks can be constructed and reused through a single programmable abstraction for composition and refinement of mobile agents. We believe that the framework can provide an a testbed for building and testing agent migration protocols.

Finally, we would like to mention further issues. Our early performance measurements indicate that the performance of protocols for a high-level prototype is fast enough for experimenting with application-specific protocols. However, the performance of the current implementation is not yet satisfactory, and we plan to improve it. We are interested in developing various agent migration protocols, in addition to the examples presented in this paper. Our protocols are not always dependent on our framework and can thus be applied to other active network infrastructure.

References

1) Aridor, Y. and Lange, D.: Agent Design Patterns: Elements of Agent Application Design, *Proc. Second International Conference on Autonomous Agents* (*Agents'98*), ACM Press, pp.108–115 (1998).
2) Bäumer, C. and Magedanz, T.: The Grasshopper Mobile Agent Platform Enabling Short-Term Active Broadband Intelligent Network Implementation, *Proc. Working Conference on Active Networks*, LNCS, Vol.1653, pp.109–116, Springer (1999).
3) Busse, I., Covaci, S. and Leichsenring, A.: Au-

tonomy and Decentralization in Active Networks: A Case Study for Mobile Agents, *Proc. Working Conference on Active Networks*, LNCS, Vol.1653, pp.165–179, Springer (1999).
4) Friedman, D.P., Wand, M. and Haynes, C.T.: *Essentials of Programming Languages*, MIT Press (1992).
5) Fuggetta, A., Picco, G.P. and Vigna, G.: Understanding Code Mobility, *IEEE Trans. Softw. Eng.*, Vol.24, No.5 (1998).
6) Gschwind, T., Feridun, M. and Pleisch, S.: ADK: Building Mobile Agents for Network and System Management from Reusable Components, *Proc. Symposium on Agent Systems and Applications/Symposium on Mobile Agents* (*ASA/MA'99*), pp.13–21, IEEE Computer Society (1999).
7) Lange, B.D. and Oshima, M.: *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley (1998).
8) ObjectSpace Inc: *ObjectSpace Voyager Technical Overview*, ObjectSpace, Inc. (1997).
9) Satoh, I.: MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System, *Proc. International Conference on Distributed Computing Systems* (*ICDCS'2000*), pp.161–168, IEEE Computer Society (2000).
10) Satoh, I.: Dynamic Configuration of Agent Migration Protocols for the Internet, *Proc. International Symposium on Applications and the Internet* (*SAINT 2002*), IEEE Computer Society, pp.119–126 (2002).
11) Satoh, I.: A Framework for Building Reusable Mobile Agents for Network Management, *Proc. Network Operations and Managements Symposium* (*NOMS 2002*), pp.51–64, IEEE Communication Society (2002).
12) Satoh, I.: Reusable Mobile Agents for Managing Networks, to be submitted to *Information Processing Society of Japan Journal* (conditionally accepted) (2002).
13) Tennenhouse, D.L., et al.,: A Survey of Active Network Research, *IEEE Communication Magazine*, Vol.35, No.1 (1997).
14) White, J.E.: 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.

```
class TCPTransmitter extends TransmitterAgent
implements AgentEventListener {
  public TCPTransmitter() {
    // registering itself as a listener
    addAgentListener(this);
    // registering itself as transmitter
    registryAs("transmitter");
  }
  // invoked when an agent arriving
  public void add(AgentEvent evt) {
    // serializing the arriving agent
    Message msg = new Message("serialize");
    msg.setArg(evt.getSourceURL());
    byte[] data = (byte[])getService(msg);
    // transmitting the serialized agent to
    // a transmitter agent on its destination
    send_agent(data, url.getTarget());
  }
  void send_agent(byte[] data, AgentURL dst) {
    // sending the serialized agent (data)
    // to the destination (dst)
    ...
  }
  void receive_agent(byte[] data, AgentURL dst) {
    // deserializing data as an agent at dst
    Message msg = new Message("deserialize");
    msg.setArg(data); msg.setArg(dst);
    AgentURL url = (byte[])getService(msg);
    ...
  }
  ...
}
```

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 it 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 im-

plementation of the navigator agent presented in Section 5.

```
class SimpleNavigator extends NavigatorAgent
implements AgentEventListener {
  Vector route = null; int i = 0;
  public SimpleNavigator() {
    // registering itself as a listener
    addDefaultListener(this);
    // making an itinerary to three nodes
    route = new Vector();
    route.addElement("first.place.com");
    route.addElement("second.place.com");
    route.addElement("third.place.com");
 }
 // invoked after arriving
 public void arrive(AgentEvent evt) {
   // invoking a callback method
   // of its inner agents
   Message = new Message("doYourTask");
   msg.setArg(evt.getCurrentURL());
   dispatch(msg);
   // moving to the next place
   moveToNextHop();
 }
 ...
}
```

(Received July 1, 2002)
(Accepted October 7, 2002)

Ichiro Satoh

Ichiro Satoh received his B.E., M.E, and Ph.D. degrees in Computer Science from Keio University, Japan in 1996. From 1996 to 1997, he was a research associate in the Department of Information Sciences, Ochanomizu University, Japan and from 1998 to 2000 was an associate professor in the same department. Since 2001, he has been an associate professor in National Institute of Informatics, Japan. His current research interests include distributed and mobile computing. He received IPSJ paper award, IPSJ Yamashita SIG research award, and JSSST Takahashi research award. He is a member of six learned societies, including ACM and IEEE.