Building Reusable Mobile Agents for Network Management

Ichiro Satoh *Member, IEEE,*

***Abstract*— Mobile agents can migrate among nodes to perform a set of management tasks at each of the visited nodes. Existing mobile agent-based network management systems often assume that their mobile agents are designed to work in particular networks to raise the efficiency of agent migration among multiple nodes. Unfortunately, such mobile agents cannot be reused in different networks. This paper proposes a framework where a mobile agent for network management is composed of two kinds of software components, a itinerary part and a behavioral logic part. Both components are implemented as mobile agents. The former is a carrier designed for particular networks, and it can efficiently navigate other mobile agents among nodes in its target network. The latter defines management tasks performed at each node independently of any local network. This framework allows a mobile agent for network management to be reused in various networks without being modified. A prototype implementation of this framework and its application were built on a Java-based mobile agent system.**

***Index Terms*— mobile agent, network management, reusability, active network**

I. Introduction

Mobile agent technology provides a solution to the flexible management of telecommunication systems. Mobile agents can locally observe and control equipments at each node by migrating among the nodes. Mobile agent-based network management has several advantages in comparison with traditional approaches, such as the client/server one. For example, they can reduce network traffic and easily support disconnected operation. Moreover, the dynamic deployment and configuration of new or existing functionalities into a network system are extremely important tasks especially as they potentially allow outdated systems to be updated in an efficient manner. Adopting the mobile agent technology eliminates the need for the administrator to constantly monitor many network management activities, e.g., installation and upgrading of software and periodic auditing of the network. There have been several attempts to apply this technology to network management tasks.

However, there have been serious problems associated with the development of mobile agent-based applications, in addition to security problems. Such applications are required to migrate their agents among all specified nodes efficiently to perform their own tasks at each of the visited nodes, because the itinerary of an agent greatly affects its achievement and efficiency. However, it is often difficult to dynamically generate an efficient itinerary among multiple nodes, without having any knowledge of the network. Even if a smart agent can create an efficient itinerary based on its previous processing and the current environment, such an agent is not always be appropriate, because both the cost of discovering such an itinerary and the size of its program tend to be large.

Therefore, most existing mobile agent-based applications explicitly and implicitly assume that their mobile agents are statically designed for their target networks for greater efficiency of agent migration over the networks. However, an agent optimized for particular networks cannot be reused in other networks. This results in an inevitable trade-off between the performance and reusability of a mobile agent. Furthermore, this problem becomes more serious when mobile agents are used for network management. This is because network management systems must often handle networks that may have some malfunctions and whose topology may not be exactly unknown. Consequently, it is almost impossible for each mobile agent to efficiently migrate among nodes in such networks. This is one of the reasons why there have not been many attempts to use mobile agent technology in the domain of network management, although the technology can be used in this domain.

This paper addresses several problems, including the problem of a trade-off between the performance and reusability of a mobile agent in the development of mobile agent-based network management systems and proposes a new framework for building mobile agent-based network management systems in order to solve these problems. The framework allows us to build efficient and scalable mobile agents for network management without losing their reusability. Like other mobile agent-based network management systems, the framework uses mobile agents to implement network management functionalities, but it allows each mobile agent to be designed independently of any network and dynamically change its itinerary without modifying its application-specific behaviors. That is, when a mobile agent arrives at an unknown sub-network, it can dynamically obtain an itinerary statically designed for the visited sub-network and thus can efficiently migrate among the nodes on the sub-network. The current implementation of the framework is built on a Java-based mobile agent system, called MobileSpaces [15], which is unique among existing systems because it hierarchically organizes multiple mobile agents.

This paper is organized as follows: Section 2 discusses the advantages of mobile agent-based network management and the actual problems associated with it. Section 3 presents the basic ideas of the framework described in this paper. Section 4 presents the design and implementation of the framework. Section 5 describes practical applications of the framework and discusses its usefulness. Section 6 reviews some related works, and Section 7 makes some concluding remarks.

II. Background

To manage a network system, we sometimes need to locally observe and control components on multiple nodes in the system. Existing network management systems essentially use the client/server mechanism for their functionalities. Such systems often suffer from poor scalability due to an increase in the amount of communication and the number of failures in nodes and channels. In contrast, mobile agent technology can be used for a variety of network management functions. Discussion on the advantages of mobile agents in network management can be found in [2], [10]. However, in addition to the trade-off problem mentioned in the previous section, there have been several other problems with obtaining these advantages.

a) Reduction in network traffic and information retrieval:: Network nodes, including gateways, databases, and sensors, often record a large amount of data. In the traditional approach all the data recorded in remote nodes must be frequently transmitted to a central management systems. In contrast, since the transmission of a mobile agent to data sources creates less traffic than the transmission of the data, mobile agent technology substantially reduces the network bandwidth for the collection and filtering of data from remote networks. However, to take this advantage of this feature, each mobile agent must be small, so it is usually designed for particular networks and tasks. This is because even if a general-purpose and adaptive mobile agent could be constructed, it may not work well, because the execution cost of such a sophisticated agent could be very high.

b) On-demand distribution of software:: In the client/server approach, static multiple servers require duplication of functionality at every node, which often has only limited resources, such as CPU power and memory. In contrast, a mobile agent resides only on one node at a time while other nodes do not run an agent if they do not need to. Such an agent carries a management function to the node. Therefore, it is not always necessary for every node to have software for network management. However, if an agent does not know all the nodes to which it must distribute software, it is difficult for the agent to detect and reach all the nodes on the fly.

c) Automation and fault tolerance:: Each mobile agent is a self-contained and autonomous entity, so it can be controlled in a decentralized manner and perform its management tasks independently of its source node. Consequently, the technology can relieve the administrator from the need to continuously monitor some network management activities. Furthermore, mobile agents can survive if moved closer to resources, or away from partially failed nodes. However, this often requires the central manager to have knowledge about the network, because it is difficult for each agent to visit all the required nodes and move away from malfunctioning nodes to another node.

d) Direct manipulation:: A mobile agent is locally executed on the node it is visiting, and it can easily discover the types and functions of devices on this node to directly control the devices. This is helpful in network management, in particular in detecting and removing device failures. However, we cannot take full advantage of this feature because of a security mechanism. Although there has been a lot of efforts to solve the security problems of mobile agent technology, most existing security mechanisms cannot be used for mobile agents in network management, because they are designed to restrict low-level procedure calls while such agents often need to directly access low-level resources.

III. Approach

The goal of this paper is to provide a framework for building and operating mobile agents capable of autonomously traveling among nodes on multiple sub-networks to perform their management tasks at each node they visit.

To solve the above mentioned problems, the framework introduces two types of mobile agents: *task* agents and *navigator* agents, as shown in Figure 1. The idea was inspired by a tour bus going around the sights of a town. A task agent corresponds to a tourist, who takes a tour-bus to visit sights in an unfamiliar town. A navigator agent corresponds to a tour-bus guide, who guides several tourists among the places of the town.

- The *Navigator* agent does not have any application-specific tasks. Instead, it carries task agents and is designed for a particular sub-network. It must be familiar with the topology of its target sub-network. Thus, it can efficiently guide one or more task agents to their multiple destinations in the sub-network.
- The *Task* agent is an application-specific agent that performs its management task at each of the nodes it visits. It can travel from sub-network to sub-network, but may be unfamiliar with the sub-networks it visits.

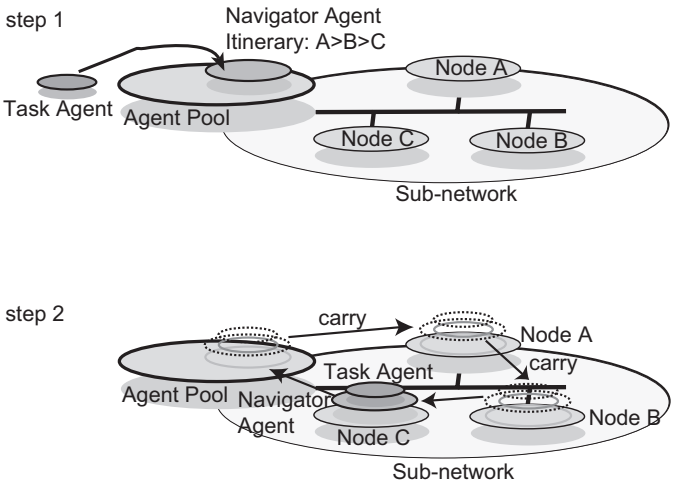

Fig. 1. Navigator agents and task agents

When a task agent arrives at an unknown sub-network, it enters an idle navigator agent that knows the current network well. Then, the selected navigator agent carries the visiting task agent to the nodes that the task agent wants to visit. Each navigator agent is defined and managed by its network and can explicitly limit the nodes to which it can carry task agents.

This framework also provides a mechanism for allowing a task agent to select a navigator agent suitable for the current network. The mechanism, called Agent Pool, stores idle agents in a manner similar to that in a bus-terminal or a taxi stand, as shown in Figure 2. Each sub-network has multiple agent places for storing navigator agents specific to the sub-network and each navigator agent is designed to return to its place soon after achieving its navigation task to wait for the next task. Each task agent is responsible for traveling among the agent pools of its destination sub-networks, where each navigator agent is responsible for navigating its inner agents among the nodes in its sub-network. Therefore, to travel among some of the nodes on a sub-network, a task agent migrates to the agent pool at the sub-network and asks a navigator agent stored in the pool to carry it among the nodes.

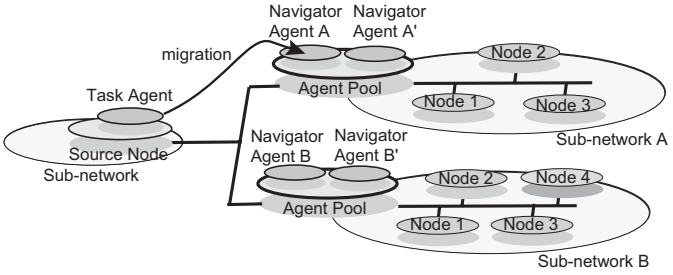

Fig. 2. Agent Pools

We should explain why our hierarchical agent model is needed in the development of network management. There may be other approaches in addition to our approach. One of the approaches is to build an omniscient agent, which can travel over all of the sub-networks, but the size and execution cost of such an agent tends to be large as mentioned previously. Another approach is to dynamically incorporate an application-specific agent with a knowledge component for determining and managing the itinerary of the agent. However, such an agent is not self-contained and may not be able to migrate over a network under its own control. Also, the distribution of knowledge of the sub-network must be limited to the sub-network for reasons of security.

Our framework introduces such a knowledge component as a navigator agent, which is a container comprised of more than one task agent. To visit nodes on a sub-network, task agents must be carried by a navigator agent that is authorized by the sub-network and has its own itinerary. Therefore, each task agent does not have to be modified and can remain autonomous and self-contained, even while it is contained in a navigator agent traveling over the sub-network. Since the knowledge of the topology of the sub-network is kept inside the navigator agent, the task agent does not access to and cannot have such knowledge, unlike the two approaches discussed above. When a network management task consists of multiple mobile agents, a navigator agent can carry these agents as a whole. Moreover, all nodes do not have the capability of authenticating their visiting arbitrary agents. Since each agent pool can authenticate its visiting task agents on behalf of its sub-network before the task agents are contained and carried by a navigator agent, each node can thus accept only authorized navigator agents.

IV. Design and Implementation

Before describing the framework presented in this paper, we describe the MobileSpaces mobile agent system that provides the infrastructure for this framework. Based on this system, we then explain how we envisage the construction of task and navigator agents.

A. MobileSpaces: A Hierarchical Mobile Agent System

This framework consists of navigator agents, task agents, and agent pools. These agents are implemented as mobile agents in MobileSpaces.

Hierarchical Mobile Agents in MobileSpaces: Mobile agents in MobileSpaces are programmable entities like other mobile agents. They are capable of conserving their state while on the move and their itineraries can include multiple network nodes. Furthermore, MobileSpaces provides each mobile agent with two novel concepts: *agent hierarchy* and *inter-agent migration*. The former means that another mobile agent can be contained within one mobile agent. 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. Therefore, an agent can contain other mobile agents inside it as shown in Figure 3.

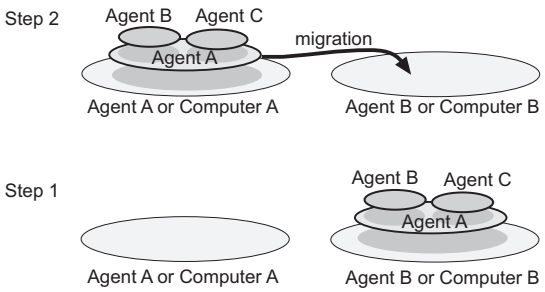

Fig. 3. Agent hierarchy and inter-agent migration

Each agent has direct control of all its inner agents and thus can instruct them to move to other locations and can destroy them. In contrast, each agent has no direct control over its container agents. Instead, each agent can have a set of service methods, which can be accessed by its containers. Each agent has a globally unique name and can have more than one active thread under the control of the runtime system.

MobileSpaces Runtime System: Each runtime system is a platform for executing and migrating agents. It is built on a Java virtual machine, and mobile agents are Java objects. Each runtime system can subordinate all the agents inside it, and the system maintains the life-cycle state of the 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 the agents it contains. The runtime system provides a mechanism

for marshaling and unmarshaling agents.[1] When an agent is marshaled, the runtime system propagates certain events to the agent and its inner agents that are still running to instruct them to stop. It also can automatically stop and serialize them after a given time period. The runtime system can transfer agents to the destination computer over TCP/IP connection.

B. Navigator Agent

Each navigator agent is a container of one or more task agents and is responsible for guiding them to nodes in the network it covers. That is, it travels with its inner agents in accordance with its itinerary, which is statically or algorithmically determined, or dynamically based on the agent's previous computations and the current environment. This framework provides abstract classes in the Java language and navigator agents can be defined by extending these classes. A typical navigator agent has a routing mechanism for managing its own routing table, which consists of all the nodes on its target network, and can dynamically add and remove elements from the table. After it has achieved its navigation task, the navigator agent goes back to the agent pool of the sub-network that it covers, and advertises the list of reachable nodes to the pool. It then waits for the arrival of other task agents. The interaction between a navigator agent and the task agents inside it is based on event-based communication introduced in the Abstract Window Toolkit of JDK 1.1. A navigator agent invokes certain methods of its task agents, whenever it arrives at one of the destinations. The navigator agent executes its built-in method, `go(AgentURL url)`, in order to migrate itself and its task agents to the next destination specified as `url`, after they have performed their tasks. Each navigator agent can explicitly limit the length of the execution period of its incoming task agents after arriving at each destination. When the time limit of a task agent inside it expires, it automatically terminates the task agent.

```
public class NavigatorAgent extends Agent {
  // advertising its possible destinations
  void register(HostSet h)
    throws IllegalAccessException ... { ... }
  // getting the address of the current host
  AgentURL getURL() { ... }
  // moving to the host specified as url
  void go(AgentURL url)
    throws NoSuchHostException ... { ... }
  // sending an event to all its inner agents
  void dispatchEvent(AgentEvent e)
    throws NoSuchEventException ... { ... }
  ....
}
```

C. Task Agent

Each task agent is a mobile agent that defines its management tasks at each of the nodes in accordance with its management criterion. It travels among the agent pools of its target sub-networks. When arriving at an agent pool, a task agent gives the pool a list of the names or types of the nodes at which it needs to perform its tasks by invoking the `setNodes()` method and then the pool recommends to the agent a suitable navigator that fits the description on the list or conditions. To hook events invoked by its container agent and the runtime system, each task agent can have one or more listener objects. One of the most basic listener interfaces, `TaskAgentEventListener`, is shown as follows:

```
interface TaskAgentEventListener
extends AgentEventListener {
  // after creation at url
  void create(AgentURL url);
  // before termination
  void destroy();
  // before serialization
  void serialize();
  // after deserialization
  void deserialize();
  // after arrived at one of the destinations
  void arrive(AgentURL from);
  // before moving to one of the destinations
  void leave(AgentURL to);
  // before traveling among the destinations
  void departure(AgentURL to);
  // after traveling among the destinations
  void finish();
  ....
}
```

When a task agent arrives at an agent pool, it is allocated to a navigator agent by the pool and then the `departure` method defined in the task agent is invoked with the first destination. Upon arrival at a node, the navigator agent invokes the `arrive` method of its task agent to instruct it to do something during a given time period at the node. After receiving a certain event from all the task agents or after the period has elapsed, the navigator agent invokes the `leave` method with the address of the next node and then moves itself and its task agents to the destination according to its itinerary. After it has traveled among all the required nodes, the navigator agent invokes its `finish` method. For reasons of security, all agents must be authenticated by the agent pool of the sub-network and then carried by a navigator agent managed by the agent pool of the sub-network, since a sub-network may explicitly prohibit any task agent from visiting its nodes. Therefore, a task agent alone cannot migrate to the nodes, even if it has been authenticated and knows the addresses of its target nodes in the sub-network. This appears to imply that each task agent needs to know the location of the agent pools of its target sub-networks, but in fact the framework can provide task agents that have no knowledge about the location of an agent pools because navigator agents can carry them among the to agent pools.

D. Agent Pool

When a task agent arrives at a sub-network, if it knows the topology of the sub-network, it travels over the sub-network according to its own itinerary. Otherwise it migrates itself to an agent pool of the sub-network to find a suitable navigator agent. Each agent pool is a stationary container of several

[1]The current implementation of the system uses the Java object serialization package provided by JDK to marshal and unmarshal agents. The package does not support capturing the stack frames or a program counter of threads. Consequently, our system cannot serialize the execution states of any thread objects.

navigator agents and is responsible for managing one or more sub-networks. It maintains inside itself a repository of idle navigators standing by for a chance to navigate. It can save such idle navigators with their states on its secondary storage by using Java's serialization mechanism. When it receives a request from a visiting task agent via the `setNodes` method, it detects one of the most suitable navigator agents from the repository. The selection mechanism of the current implementation compares the reachable nodes of all the navigators stored in the pool and the list of the nodes that the task agent must visit. That is, each agent pool selects a navigator agent whose reachable nodes include the nodes that the task agent must visit. If more than one navigator agent that satisfies the conditions, the pool selects the navigator with the fewest reachable nodes. Since mobile agents in MobileSpaces can duplicate themselves, the selected navigator agent can create its own copy on the agent pool before leaving the pool to handle a new task agent visited while the agent travels among nodes. This framework provides Java-based abstract classes that allow us to easily define advanced agents by extending the classes.

E. Current Status

The system is implemented as a collection of mobile agents on MobileSpaces and it can be run on any computer with a JDK 1.2-compatible Java runtime system that can migrate agents over a network using a TCP-based agent migration protocol. The current implementation of this framework was not built for performance, but a basic agent migration experiment was done using eight computers (Pentium III-800 MHz with Windows2000 and JDK 1.4.1) and an agent pool (Pentium4-1.6GHz with Windows2000 and JDK 1.4.1) connected with a 100-Mbps Ethernet.

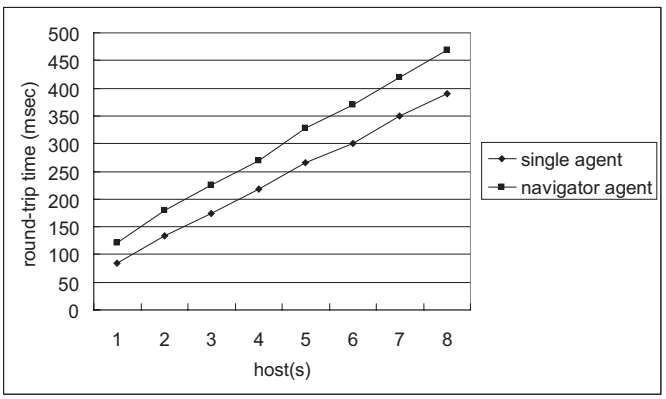

Fig. 4. Performance of agent migration

Figure 4 shows the basic performance of agent migration over a network when a single agent or a navigator agent containing a task agent leaves from an agent pool and visits more than one nodes and then comes back to the pool again. Each round-trip time is an average time lag between the departure time and the arrival time at an agent pool when an agent tours among the specified number of nodes. The former agent is a simple implementation of the `TaskAgentEventListener` interface presented in Section IV and can control its own itinerary. The agent corresponds to a null RPC and the data size is about 2.4 Kbytes (zip-compressed). The latter agent is a simple navigator agent, which has a static itinerary list of more than one node carries its inner agents to the nodes sequentially by incorporating them inside itself. The result shows that the overhead of the hierarchical structure of this framework is less than two percent, so the latency costs in the above table are basically dependent on the MobileSpaces runtime system. In this experiment, the runtime systems on the nodes exchange agents with each other through a simple TCP-based agent migration protocol. The marshaled agent consists of its serialized state, its code, and its attributes such as a name and capability, and it is packed and compressed into a bit stream. Therefore, each time in Figure 4 is the sum of delays in the marshaling of the agent, zip-based compression, the opening of a TCP connection, transmission, security verification, decompression, and the unmarshaling of the agent.

V. Application to Network Management

To explain the utility of the framework, we illustrate an application of the framework. The application is a network management system for a Grid-based computational environment consisting of three sub-networks and each of the sub-networks has from four to eight processor elements distributed geographically.[2] The purpose of the management system is to monitor certain network and computational resources at nodes. The system deploys agent pools at one node of each sub-network and offers several task agents and navigator agents. For example, a task agent that monitors network traffic loads is designed to perform its task at each node that it visits as shown in Figure 5. Although the system itself is independent of any network management protocols, we constructed a task agent that can access SNMP data from a small stationary agent situated at its visiting node. The stationary agent allows that visiting task agent to access the MIB of its node via interagent communication. Since the task agent can contain codes to perform both information retrieval and filtering, it can carry only relevant information. Also, the system has three other task agents for monitoring computational resources at the processor nodes. They are designed to collect information on the use of CPU, memory, and disks by incorporating performance monitoring systems at the nodes. The system also provides several navigator agents having different itineraries. The agents are statically optimized for the topology of their target sub-networks so that they can efficiently travel among the nodes in the sub-networks.

As mentioned previously, the goal of the framework is to construct reusable mobile agents for network management. Actually, our early experience with this system proves that the framework presented in this paper enables each task agent to be built independently of any sub-network. The task agents presented in this section contain no specific knowledge about

[2]The Grid environment is small in scale because it is implemented as a testbed for developing middleware and applications for Grid computing rather than a computational infrastructure.

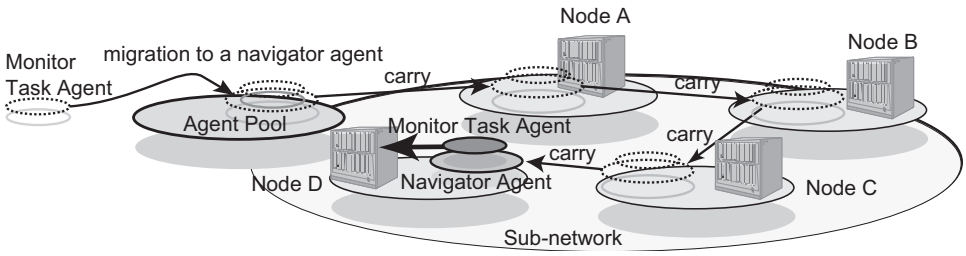

Fig. 5. Mobile agent-based management system

sub-networks and are designed to perform their management tasks when their navigator agents invoke certain methods. Therefore, they do not have to change their own programs when they are reused in different sub-networks. Moreover, the experience tells that the programs of the task agents are relatively simple because they leave their itinerary control to their navigator agents. Actually, the total size of a navigator agent containing one of the task agents is about 5 KB (zip-compressed) and it is only 20 percent greater than the size of a self-contained task agent that can control its own itinerary. This is a small increase in size if we take into account the amount of data such agents can collect from the nodes.

Unfortunately, there have not been any unified algorithms for navigating agents in arbitrary networks because the complexity of such algorithms is large. Instead, the current implementation of the framework provides three basic types of navigator agents, whose itineraries are based on migration patterns described in [11], [14] and optimized to the target sub-networks, and their minor derivations. (1) Navigator agents of the first type are designed for traveling sequentially around the destination nodes to perform tasks at each node. (2) Navigator agents of the second type travel among nodes in a star-shaped route. That is, they go back and forth between destination nodes and a given base node and performs their tasks in the destination nodes. (3) Navigator agents of the third type generate as many copies of themselves as the number of nodes that they must monitor before migrating to the nodes. After that, each copy moves to the node and accesses the resources, and then goes back to the source node. Each copy reports to the leader agent among the copies and then disappears. These types define as Java subclasses of the `NavigatorAgent` class and enable us to easily define other navigator agents whose routes are more complex by extending the three types' Java classes. For example, the system currently offers navigator agents that travel among specified nodes at intervals required by their task agents. We should discuss the size of a navigator agent. If each navigator agent of the three types migrates along an itinerary defined as a static list of hosts, the size of the agent is smaller than 4KB (zip-compressed), where the size of the minimal agent in MobileSpace is about 2.4KB (zip-compressed) and the size of a navigator agent supporting all the three patterns is larger than 6KB (zip-compressed).

We have obtained a preliminary measurement of the cost of migrating a navigator agent over a sub-network of the Grid system. Note that the system is just a prototype implementation, hence it is not optimized for efficient agent migration. Actually, the total cost of network management depends on application-specific tasks performed at nodes rather than agent migration. After receiving a task agent at the agent pool of the sub-network, the navigator agent travels straightly around four nodes and then returns to the agent pool of the sub-network, where the nodes and the pool are Pentium III-600 MHz computers connected using a 100-Mbps Ethernet. The itinerary of the navigator agent is statically defined and corresponds to five hops. The round-trip time of the agent is about 480 msec and the cost of detecting a navigator agent in an agent pool is less than 10 msec. After a navigator agent leaves from the agent pool, it can travel among nodes under its own control. As a result, the network management system is operated in a fully decentralized manner. Also, the result of the basic experiment presented in the previous section tells that the performance of our framework is scalable in the number of nodes. Hence, we can naturally expect the system to still to be scalable even when applying it to a larger Grid environment. Moreover, our experience tells us that our navigator agents are useful in the resource management of Grid-based computational environments. This is because they can provide a mechanism for the deployment of computational tasks at remote nodes in a decentralized manner. To perform a variety of applications efficiently, a Grid-based environment must support multiple policies for task deployment. This framework allows such policies to be naturally defined as navigator agents.

A. Discussion

The remainder of this section describes how the framework presented in this paper solves the problems discussed in Sections I and II.

- *Reusability and Performance:* This framework enables each navigator agent to be optimized for particular networks independently of any application-specific logic. Therefore, the agent can efficiently guide various task agents among nodes in the networks. On the other hand, each task agent has its application-specific tasks, which are designed to be performed at each of the visited nodes regardless of the sub-network. It needs to know the location of the agent pools of its target sub-networks but does not have to know the topology of the networks. By dynamically changing to a navigator agent suitable for its current network, a task agent can efficiently migrate among nodes in various networks to perform its task, without modifying its own program.

- *Simplification of Agents:* The framework enables both navigator and task agents to be small and simple, because navigator agents can be designed for particular networks and thus do not have to offer any adaptive mechanisms for handling various networks, which would make the programs of the agents large and complex. On the other hand, task agents leave their itineraries to corresponding navigator agents only when they know the location of the agent pools of their destinations.
- *Network-dependent Migration:* Since each navigator agent is optimized for a particular network, it can statically have knowledge about the networks. After achieving its current task, it returns to the given agent pool and stands by for the next navigation without any initialization. Consequently, when it detects changes in the network environment, such as malfunction in nodes, network disconnection, or network topology changes, it keeps the changes in its state and reflect them in its next navigation in a heuristic manner.
- *Limitation of Reachable Nodes:* Each navigator agent can limit the migration range of task agents. This is because each navigator agent can explicitly define its own reachable nodes and each node accepts only authorized navigator agents. Consequently, when a task agent is carried by a navigator agent whose reachable nodes are limited, it can travel only among the reachable nodes of the navigator agent. Moreover, each agent pool can authenticate its visiting task agents on behalf of its subnetwork. This is helpful in network management systems whose nodes may have limited CPU power and memory.

VI. Related Work

Many mobile agent systems have been developed over the last few years, for example, Aglets [11] and Telescript [20]. There have been several attempts to develop mobile agent-based network management, for example see [3], [2], [8], [13], [14]. Typically, a mobile agent for network management must visit multiple hosts to perform its task, so the itinerary of such an agent can affect its success and efficiency. However, most of these studies often assume that the agents are designed for particular networks, because it is difficult for the agents to dynamically make their itineraries to visit all the specified nodes in their target networks, which may be incomplete or lack any global perspective. Several studies attempted to build smart mobile agents that can dynamically learn the topology of networks, (see, for example [12]). However, most of these studies explicitly and implicitly assume to be performed on only simulated networks. Even if the studies could be performed in a real system, the costs of generating efficient routes tend to be large and thus they are not always suitable in mobile agent-based network management systems.

Some solutions to this problem have been found outside the domain of network management. For example, ADK [9] 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 the travel itinerary and performer components responsible for executing one or more management tasks at each node. Aglets [11] 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 an application-specific itinerary for an agent independent of the agent's logical behavior, but the itinerary parts must be statically and manually embedded in the agent. Consequently, this agent, unlike ours, cannot dynamically change its itinerary and cannot travel beyond its familiar networks.

Like this framework, the ambient calculus [4] allows mobile agents (called ambients in the calculus) to contain other agents and to move as a whole with all its subcomponents. A mobile ambient, which may carry other ambient, must migrate along a hierarchy of stationary ambients corresponding to the logical structure of sub-networks, whereas itineraries of real mobile agents may be complicated. For example, mobile agents for network management are often required to directly transmitted to other sub-networks independently of the logical structure of sub-network, because target networks have some malfunctions and disconnections. To enable an ambient to migrate to sub-networks independently of the logical structure of sub-networks, we must change its whole semantics. The Seal calculus [19] is similar to the mobile ambients and ours in its expressiveness of hierarchical structure of mobile agents, but its main purpose is to reason about the security mechanism of mobile agents. The Polis language [6] is a theoretical framework for specifying and analyzing mobile entities, including mobile codes and mobile agents, which can contain other entities inside them. However, it is not executable and needs a kind of shared memory over remote nodes, whereas our framework can operate reusable mobile agents for network management in a decentralized manner.

We described an approach to building configurable protocols for agent migration in other papers [16], [18]. While that approach customizes network processing for agent migration embedded in a mobile agent runtime system, the approach presented in this paper can change network-dependent routings embedded in a mobile agent according to the topology of the current network. Our previous paper [17] presented a preliminary of this framework, but it did not present relationships between task agents and navigator agents, including the both agents' programming models unlike this paper. Also, this paper evaluates the utility of the framework with a practical application of the framework.

VII. Conclusion and Future Work

This paper presented a new approach to building mobile agents for network management. The key idea is to build a mobile agent from two subcomponents: a navigator agent and a task agent. The former is designed for its target networks and thus can efficiently carry multiple task agents among hosts in the networks. The latter defines a set of management tasks to be performed at each of the host to be visited. This framework also provides a mechanism for storing idle navigator agents. When a task arrives at an unknown network, it finds a navigator agent for the network and enters the navigator

agent to migrate to nodes in the network. As a result, each task agent can be reused in different networks. A prototype implementation of the framework built on a Java-based mobile agent system, called MobileSpaces, allowed us to experiment with mobile agent-based network management based on this framework. We believe that using this framework, we can easily build mobile agents for network management without any limitation on the reusability of application-specific agents or the agent migration efficiency.

Finally, we would like to mention some future research directions. The framework presented in this paper is designed to a general-purpose framework. To prove the utility of the framework, we need to apply the framework to various network management systems. The current implementation relies on a JDK 1.1 security manager and provides an authentication mechanism for navigator agents; however many other security problems are left open for our future work. The performance of the current implementation is not yet satisfactory, so further measurements and optimization are needed.

ACKNOWLEDGMENT

We are grateful to the anonymous reviewers.

REFERENCES

[1] Y. Aridor, and D.B. Lange, "Agent Design Patterns: Elements of Agent Application Design," in Proc. 2nd ACM Int. Conf. Autonomous Agents, 1998, pp. 108-115.
[2] A. Bieszczad, B. Pagurek, and T. White, "Mobile Agents for Network Management," IEEE Commun. Surveys, vol. 1, no. 1, 1998.
[3] C. Bohoris, G. Pavlou, and H. Cruickshank, "Using Mobile Agents for Network Performance Management," in Proc. IEEE/IFIP Network Operations and Management Symp., April, 2000, pp. 637-652.
[4] L. Cardelli and A. D. Gordon, "Mobile Ambients," in Proc. Foundations of Software Science and Computational Structures, LNCS, vol. 1378, Springer 1998, pp. 140-155.
[5] J. Case, M. Fedor, M. Schoffstall, and J. Davin:, "A Simple Network Management Protocol (SNMP)," RFC 1157, 1990.
[6] P. Ciancarini, F. Franzè and C. Mascolo, "Using a Coordination Language to Specify and Analyze Systems Containing Mobile Components," ACM Trans. Software Engineering and Methodology, vol. 9, no. 2, pp. 167-198, 2000.
[7] I. Foster and C. Kesselman (eds.), "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann, 1999.
[8] D. Gavalas, D. Greenwood, M. Ghanbari, and M. O'Mahony, "An Infrastructure for Distributed and Dynamic Network Management based on Mobile Agent Technology," in Proc. Conf. Communications, 1999, pp. 1362-1366.
[9] T. Gschwind, M. Feridun, and S. Pleisch, "ADK: Building Mobile Agents for Network and System Management from Reusable Components," in Proc. Symp. Agent Systems and Applications / Symp. Mobile Agents, 1999, pp.13-21.
[10] A. Karmouch, "Mobile Software Agents for Telecommunications," IEEE Commun.n Mag., vol. 36, no. 7, 1998.
[11] B. D. Lange and M. Oshima, "Programming and Deploying Java Mobile Agents with Aglets," Addison-Wesley, 1998.
[12] N. Minar, K. H. Kramer, P. Maes, "Cooperating Mobile Agents for Dynamic Network Routing," in Software Agents for Future Communication Systems, Springer, 1999, pp.287-304.
[13] A. Puliafito and O. Tomarchio, "Advanced Network Management Functionalities through the use of Mobile Software Agents," in Proc. Workshop on Intelligent Agents for Telecommunication Applications, LNCS, vol. 1699, Springer, Aug. 1999, pp.33-45,
[14] A. Sahai and C. Morin, "Mobile Agents for Managing Networks: The MAGENTA Perspective," in Software Agents for Future Communication Systems, Springer, 1999, pp. 358-380.
[15] I. Satoh, "MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System," in Proc. Int. Conf. Distributed Computing Systems IEEE Computer Society, April, 2000, pp. 161-168.
[16] I. Satoh, "Network Processing of Mobile Agents, by Mobile Agents, for Mobile Agents," in Proc. Workshop on Mobile Agents for Telecommunication Applications, LNCS, vol. 2146, Springer, 2001, pp. 81-92.
[17] I. Satoh, "A Framework for Building Reusable Mobile Agents for Network Management," in Proc. IEEE/IFIP Network Operations and Managements Symp., IEEE Communication Society, April, 2002, pp. 51-64.
[18] I. Satoh, "Configurable Network Processing for Mobile Agents on the Internet," Cluster Computing, vol. 6, no.4 Kluwer, Oct. 2003.
[19] J. Vitek, "Seal: A Framework for Secure Mobile Computations," in Proc. Workshop on Internet Programming Languages, LNCS, vol. 1686, Springer 1998, pp. 47-77.
[20] J. E. White, "Telescript Technology: Mobile Agents," General Magic, 1995.

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.