Mobile Agents

Ichiro Satoh*

Abstract

Mobile agent technology has been promoted as an emerging technology that makes it much easier to design, implement, and maintain distributed systems, including cloud computing and sensor networks. It does not provides an infrastructure for only executing autonomous agents but also migrating them between computers. This chapter discusses the potential uses of mobile agents in distributed systems, lists their potential advantages and disadvantages. It describes technologies for executing, migrating, and implementing mobile agents. It present several practical and potential applications of mobile agents in smart environments in addition to distributed systems.

1 INTRODUCTION

Mobile agents are autonomous programs that can travel from computer to computer in a network, at times and to places of their own choosing. The state of the running program is saved, by being transmitted to the destination. The program is resumed at the destination continuing its processing with the saved state. They can provide a convenient, efficient, and robust framework for implementing distributed applications and smart environments for several reasons, including improvements to the latency and bandwidth of client-server applications and reducing vulnerability to network disconnection. In fact, mobile agents have several advantages in the development of various services in smart environments in addition to distributed applications.

- **Reduced communication costs:** Distributed computing needs interactions between different computers through a network. The latency and network traffic of interactions often seriously affect the quality and coordination of two programs running on different computers. As we can see from Figure 1, if one of the programs is a mobile agent, it can migrate to the computer the other is running on communicate with it locally. That is, mobile agent technology enables remote communications to operate as local communications.

- **Asynchronous execution** After migrating to the destination-side computer, a mobile agent does not have to interact with its source-side computer. Therefore, even when the source can be shut down or the network between the destination and source can be disconnected, the agent can continue processing at the destination. This is useful within unstable communications, including wireless communication, in smart environments.

*National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan, Tel: +81-3-4212-2546, Fax: +81-3-3556-1916, E-mail: ichiro@nii.ac.jp

- **Direct manipulation** A mobile agent is locally executed on the computer it is visiting. It can directly access and control the equipment for the computer as long as the computer allows it to do so. This is helpful in network management, in particular in detecting and removing device failures. Installing a mobile agent close to a real-time system may prevent delays caused by network congestion.

- **Dynamic-deployment of software** Mobile agents are useful as a mechanism for the deployment of software, because they can decide their destinations and their code and data can be dynamically deployed there, only while they are needed. This is useful in smart environments, because they consist of computers whose computational resources are limited.

- **Easy-development of distributed applications** Most distributed applications consist of at least two programs, i.e., a client-side program and a server side program and often spare codes for communications, including exceptional handling. However, since a mobile agent itself can carry its information to another computer, we can only write a single program to define distributed computing. A mobile agent program does not have to define communications with other computers. Therefore, we can easily modify standalone programs as mobile agent programs.

As we can see from Figure 2, mobile agents can save themselves through persistent storage, duplicate themselves, and migrate themselves to other computers under their own control so that they can support various types of processing in distributed systems.

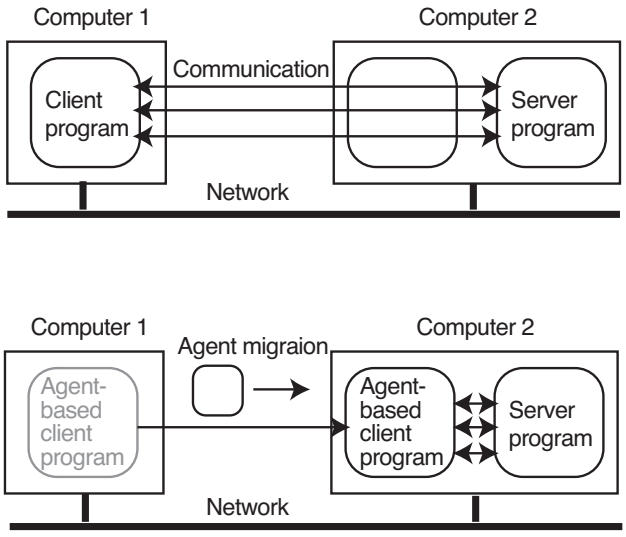

Figure 1: Reduced communication

Although not all applications for distributed systems will need mobile agents, there are many other applications that will find mobile agents the most effective technique for implementing all or part of their tasks. Mobile agent technology can be treated as a type of software agent technology, but it is not always required to offer intelligent capabilities, e.g., reactive, pro-active, and social behaviors that are features of existing software agent technologies. This is because these capabilities tend to be large in terms of scale and processing, and no mobile agent should consume

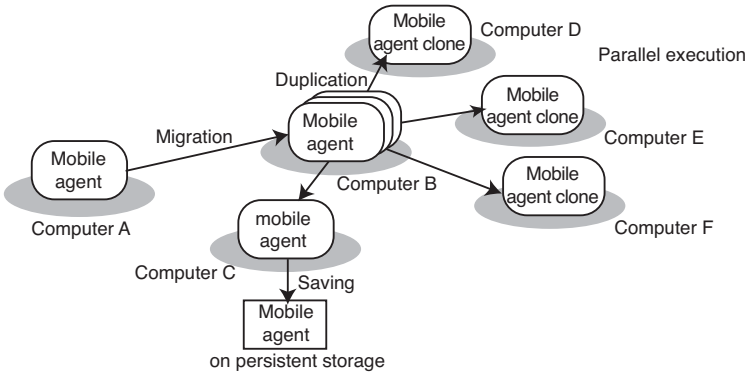

Figure 2: Functions of mobile agents in distributed system

excessive computational resources, such as processors, memory, files, and networks, at its destinations. Also, the technology is just an implementation approach of distributed systems rather than intelligent systems.

1.1 Mobility and Distribution

Fuggetta, et al [4] provided a description of mobile software paradigms for distributed applications. These are classified as client/server (CS), remote evaluation (REV), code on demand (COD), and mobile agent (MA) approaches. By decompiling distributed applications into code, data, and execution, most distributed executions can be modeled as primitives of these approaches as we can see from Figure 3.

- The client-server approach is widely used in traditional and modern distributed systems (Figure 3 a)). The code, data, and execution remain fixed at computer A. Computer B requests a service from the server with some data arguments of the request. The code and remaining data to provide the service are resident within computer B. As a response, computer B provide the service requested by accessing computational resources provided in it. Computer B returns the results of the execution to computer A.

- The remote evaluation approach assumes that the code to perform the execution is stored at computer A (Figure 3 b)). Both the code and data are sent to computer B. As a response, computer B executes the code and data by accessing computational resources, including data, provided in them. An additional interaction returns the results from computer B to computer A.

- The code-on-demand approach is an inversion of the remote evaluation approach (3 c)). The code and data are stored at computer A and execution is done at computer B. Computer A fetches code and data from computer B and then executes the code with its local data as well as the imported data. An example of this is Java applets, which are Java codes that web-browsers download from remote HTTP servers to execute locally.

- The mobile agent approach assume that the code and data are initially hosted by computer A (Figure 3 d)). Computer A migrates the data and code it need to computer B. After it

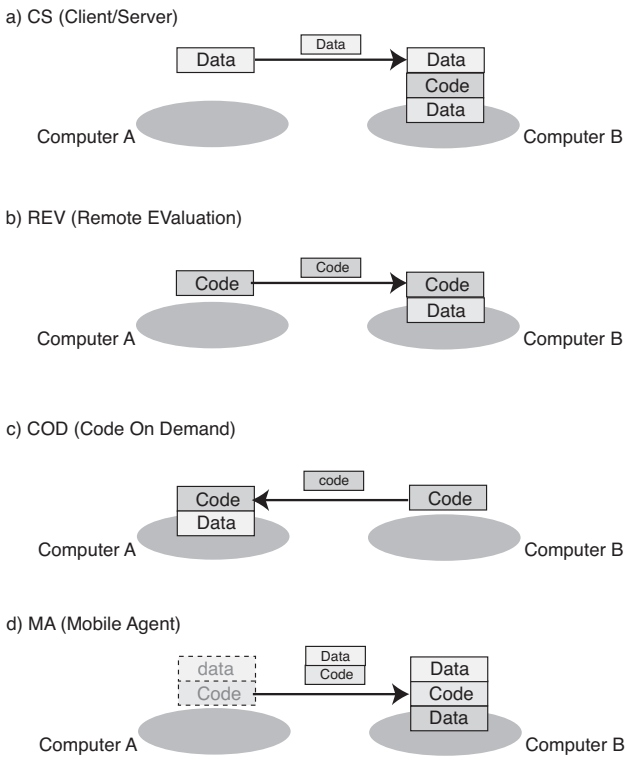

Figure 3: Client/server, remote evaluation, code on demand, and mobile agent

has moved to computer B, the code is executed with the data and the resources available on computer B.

2 MOBILE AGENT PLATFORM

Mobile agent platforms consist of two parts: mobile agents and runtime systems. The former defines the behavior of software agents. The latter are called agent platforms, agent systems, and agent servers, and support their execution and migration. The same architecture exists on all computers at which agents are reachable. That is, each mobile agent runs within a runtime systems on its current computer. When an agent requests the current runtime system to migrate itself, the runtime system can migrate the agent to a runtime system on the destination computer, carrying its state and code with it. Each runtime system itself runs on top of the operating system as a middleware. It provides interpreters or virtual machines for executing agent programs, or the system themselves are provided on top of virtual machines, e.g., the Java virtual machine (JVM).

2.1 Remote procedure call

Agent migration is similar to RPC (Remote Procedure Calling) or RMI (Remote Method Invocation). RPC enables a client program to call a procedure for server programs running in separate processes, generally in different computers from the client [2]. RMI is an extension of local

method invocation that allows an object to invoke the methods of the object on a remote computer. RPC or RMI can pass arguments to a procedure or method of a program on the server and receives a return value from the server. The mechanism for passing arguments and results between two computers through RPC or RMI correspond to that for agent migration between two computers. Figure 4 shows flow for the basic mechanism of RPC between two computers.

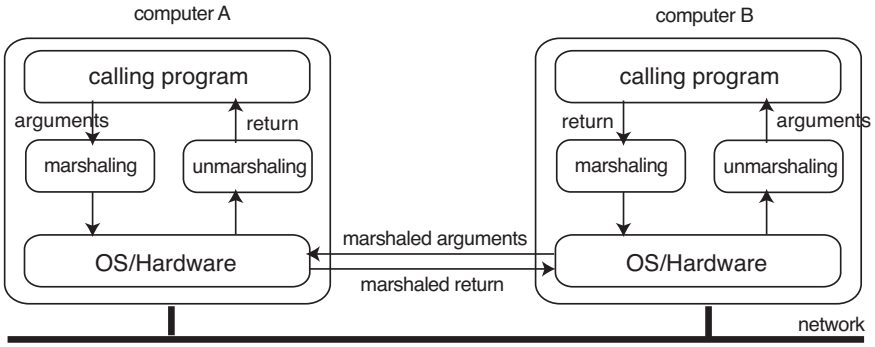

Figure 4: Remote procedure call between two computers

2.1.1 Agent marshaling

Data items, e.g., objects and values, in a running program cannot be directly transmitted over a network. They must be transformed into external data representation, e.g., a binary form or text form, before migrating them (Figure 5). Marshaling is the process of collecting data items and assembling them into a form suitable for transmission in a message. Unmarshaling is the process of disassembling them on arrival to produce an equivalent collection of data items at the destination.[1] The marshaling and unmarshaling processes are carried out by runtime systems in mobile agent systems. The runtime system at the left (at sender-side computer) of Figure 6 marshals an agent to transmit it to a destination through a communication channel or message and then the runtime system at the right (at receiver-side computer) of Figure 6 receives the data and unmarshals the agent.

2.1.2 Agent migration

Figure 6 shows the basic mechanism for agent migration between two computers.

Step.1 The runtime system on the sender-side computer suspends the execution of the agent.

Step.2 It marshals the agent into a bit-chunk that can be transmitted over a network.

Step.3 It transmits the chunk to the destination computer through the underlying network protocol.

[1]Note that marshaling and serialization are often used without any distinction between them. The latter is a process of flattening and converting an object, including its referring objects, into a sequence of bytes to be sent across network or saved on a disk.

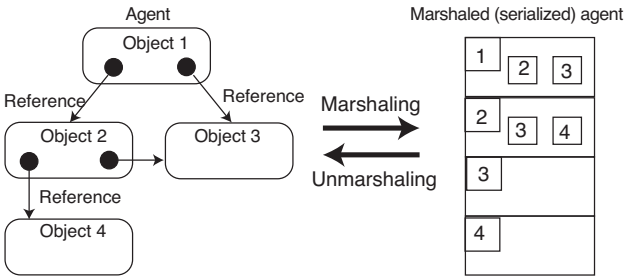

Figure 5: Marshaling agent

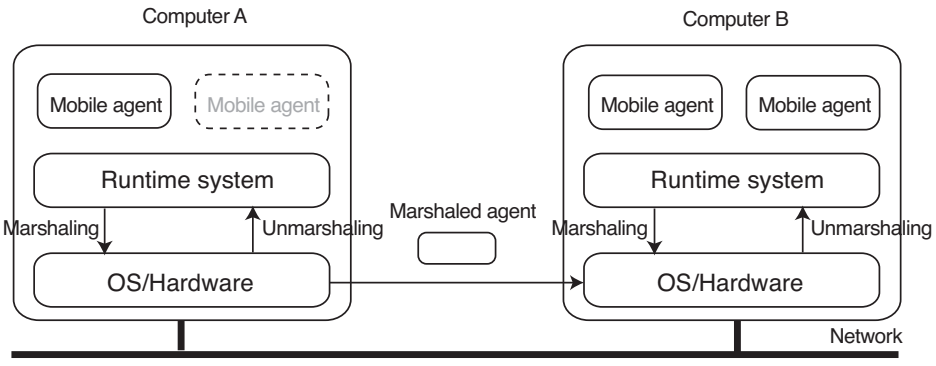

Figure 6: Agent migration between two computers

Step.4 The runtime system on the receiver-side computer receives the chunk.

Step.5 It unmarshals the chunk into the agent and resumes the agent.

Most existing mobile agent systems use TCP channels, SMTP, or HTTP as their underlying communication protocols. Mobile agents themselves are separated from the underlying communication protocols.

2.1.3 Strong migration vs. weak migration

The state of execution is migrated with the code so that computation can be resumed at the destination. According to the amount of detail captured in the state, we can classify agent migration into two types: strong and weak.

- **Strong migration:** is the ability of an agent to migrate over a network, carrying the code and execution state, where the state includes the program counter, saved processor registers, and local variables, which correspond to variables allocated in the stack frame of the agent's memory space, global variables. These correspond to variables allocated in the heap frame. The agent is suspended, marshaled, transmitted, unmarshaled and then restarted at the exact position where it was previously suspended on the destination node without loss of data or execution state.

- **Weak migration:** is the ability of an agent to migrate over a network, carrying the code and partial execution state, where the state is variables in the heap frame, e.g., instance variables in object oriented programs, instead of its program counter and local variables declared in methods or functions. The agent is moved to and restarted on the destination with its global variables. The runtime system may explicitly invoke specified agent methods.

Strong migration can cover weak migration, but it is a minority. This is because the execution state of an agent tends to be large and the marshaling and transmitting of the state over a network need heavy processing. Moreover, like the latter, the former cannot migrate agents that access the computational resources only available in current computers, e.g., input-and-output equipment and networks. The former unfortunately has no significant advantages in the development and operation of real distributed applications as discussed by Srasser et al. [22].

The program code for an agent needs to be available at the destination where the agent is running. The code must to be deployed at the source at the time of creation and at the destination to which it moves. Therefore, existing runtime systems offer a facility for statically deploying program code that is needed to execute the agent, for loading the program code on demand, or for transferring the program code along with the agent.

2.2 Mobile agent languages

Since mobile agents are programming entities, programming languages for defining mobile agents are needed. There has been a huge number of programming languages, but all of these are not available for mobile agents. Programming languages for mobile agents must support the following functions. They should enable programs to be marshaled into data and vice versa. They should

Table 1: Functions available in agents

command	parameters	function
`go`	destination address, agent-identifier	agent migration
`terminate`	agent-identifier	agent termination
`duplicate`	agent-identifier	agent duplication
`identify`	agent-type	identification
`lookup`	agent-type, runtime system address	discovery of available agents
`communicate`	agent-identifier	inter-agent communication

also download code from remote computers and link it at run-time. A few researchers have provided newly designed languages for defining mobile agents, e.g., Telescript [24], and most current mobile agent systems use existing general-purpose programming languages that can satisfy the above requirements, e.g., Java [1]. Telescript provides primitives for defining mobile agents, e.g, `go` operation, and enables a thread running on an interpreter to migrate to another computer. The Java language itself offers no support for the migration of executing code, but offers dynamic class loading, a programmable class loader, and a language-level marshaling mechanism, where these can be directly exploited to enable code mobility. Creating distributed systems based on mobile agents is a relatively easy paradigm because most existing mobile agents are object oriented programs, e.g., Java, and can be developed by using rapid application development (RAD) environments.

Distributed systems are characterized by heterogeneity in hardware architectures and operating systems. To achieve heterogeneity, the state and code of an agent need to be saved in a platform-independent representation. Hidden differences between platforms is provided at the language level, by using intermediate byte code representation in Java or by relying on scripting languages, such as Python and Ruby. Therefore, Java-based mobile agents are executed on Java virtual machines. The costs of running agents in a Java virtual machine on a device are decreasing by using just-in-compiler technologies.

2.3 Agent execution management

The runtime system manages execution and monitoring of all agents on a computer. It allows several hundred agents to be present at any one time on a computer. It also provide these agents with an execution environment and executes them independently of one another. It manages the life-cycle of its agents, e.g., creation, termination, and migration.

Each agent program can access basic functions provided by its runtime system by invoking APIs (Table 1). The agent uses the `go` command to migrate from one computer to another with the destination system address (and its target agent's identifier) and does not need to concern itself with any other details of migration. Instead, the runtime system supports the migration of the agent. It stops the agent's execution and then marshals the agent's data items to the destination via the underlying communication protocol, e.g., TCP channel, HTTP (hyper text transfer protocol), and SMTP (simple mail transfer protocol). The agent is unpacked and reconstituted on the destination.

2.4 Inter-agent communication

Mobile agents can interact with other agents residing within the same computer or with agents on remote computers as other multi-agents. Existing mobile agent systems provide various inter-agent communication mechanisms, e.g., method invocation, publish/subscribe-based event passing, and stream-based communications.

2.5 Locating mobile agents

Since mobile agents can autonomously travel from computer to computer, a mechanism for tracking the location of agents is needed by the users to control their agents and for agents to communicate with other agents. Several mobile agent systems provide such mechanisms, which can be classified into three schemes:

- A name server multicasts query messages about the location of an agent the to computers and receives a reply message from a computer hosting the agent (Figure 7 (a)).
- An agent registers its current location at a predefined name server whenever it arrives at another computer (Figure 7 (b)).
- An agent leaves a footprint specifying its destination at its current computer whenever it migrates to another computer to track the trails of the agent (Figure 7 (c)).

In many cases, locating agents is application specific. For example, the first scheme is suitable for an agent moving within a local region. It is not suitable for agents visiting distant nodes. The second scheme is suitable for an agent migrating within a far away region; in the case of a large number of nodes, registering nodes are organized hierarchically. However, it is not suitable for a large number of migrations. The third scheme is suitable for a small number of migrations; it is not appropriate for long chains.

2.6 Security

Security is one of the most important issues with mobile agent systems. Most security issues in mobile agents are common to existing computer security problems in communication and the downloading of software. In addition, many researchers have explored mechanisms to enhance security with mobile agent systems. There are two problems in mobile agent security: the protection of hosts from malicious mobile agents and the protection of mobile agents from malicious hosts. It is difficult to verify with complete certainty whether an incoming agent is malicious or not. However, there are two solutions to protecting hosts from malicious mobile agents. The first is to provide access-control mechanisms, e.g., Java's security manager. They explicitly specify the permission of agents and restrict any agent behaviors that are beyond their permissions. The second is to provide authentication mechanisms by using digital signatures or authentication systems. They explicitly permit runtime systems to only receive agents that have been authenticated, have been sent from authenticated computers, or that have originated from authenticated computers.

There have been no general solutions to the second problem, because it is impossible to keep agent private from runtime systems executing the agent. However, (non-malicious) runtime systems can authenticate the destinations of their agents, to check whether these are non-malicious,

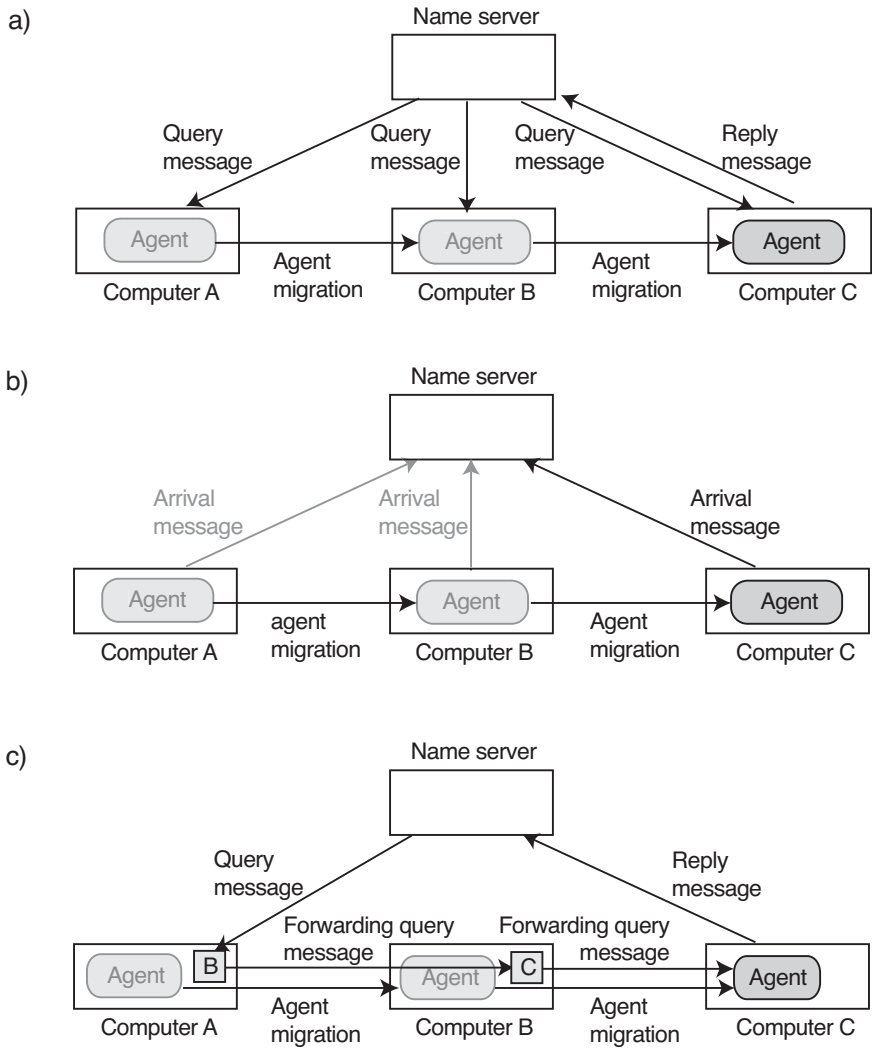

Figure 7: Discovery for migrating agents

before they migrate the agents to these destinations. While strong security features would not immediately make mobile agents appealing, the absence of security would certainly make mobile agents unattractive and unpractical.

2.7 Remarks

Several technologies have been presented for enabling software to migrate between computers, e.g., mobile code, process-migration, and mobile objects. Mobile agents differ from mobile codes, e.g., downloadable applets, in that mobile codes can maintain the states of running programs. As a result, they must start their initial states after they have been deployed at remote computers.

One of the most important differences between mobile agents and traditional techniques, e.g., process-migration or mobile objects is in their acceptable levels of mobility-transparency. Introducing too much transparency can adversely affect other characteristics, such as complexity, or the scope of modifications made to the underlying environment. For example, a solution allowing the migration of processes or objects at any time in response to a request from any other object would require significant changes to the underlying environment, e.g., balancing the processor load and escaping from a shutdown computer, whereas mobile agents can move where and when they choose, typically through a `go` statement. Similarly, solutions that insist on continuous communication and name resolution could be achieved for naming and open channel handling, but they would incur significant complexity in communication support and the naming model. Process-migration and mobile object technologies require fully transparent solutions at the operating system level to minimize complexity. For example, processes and objects still continue to access the computational resources, e.g., file systems, database systems, and channels, that they accessed at their source-side computers, even after they have moved. With a reasonable choice of transparency-requirements, mobile agents can access computational resources provided in current computers after mobile agents have moved to their destinations. Although mobile agents are similar to mobile objects at the programming-level, they contain threads and they are therefore active and can act autonomously, whereas most mobile objects are implemented as passive entities.

3 MOBILE AGENT APPLICATIONS

Many researchers have stated that there are no killer applications for mobile agent technology [7], because almost everything you can do with MAs can be done with more traditional technologies. However, mobile agents make it easier, faster, and more effective to develop, manage, and execute distributed applications than other technologies. We describe typical applications of mobile agents as follows:

3.1 Remote information retrieval

This is one of the most traditional applications of mobile agents. If all information were stored in relational databases, a client could send a message containing SQL commands to database servers. However, given that most of the world's data is in fact maintained in free text files on different computers, remote searching and filtering require the ability to open, read, and filter files. Since mobile agents can perform most of their tasks locally at the destination, Client can send its agents

to database servers so that they locally perform a sequence of query or update tasks on the servers. Communications between the client and server can be minimized, i.e., the migration of a search agent to the server and the migration of an agent to the client. Since agents contain program codes for filtering information that is of interest to their users from databases, they only need to carry wanted information back to the client to reduce communication traffic. Furthermore, agents can migrate among multiple database servers to retrieve and gather the interesting data from the servers. They can also determine the destinations based on information they have acquired from the database servers that they have thus far visited.

3.2 Network management

Mobile agent technology provides a solution to the flexible management of network systems. Mobile agents can locally observe and control equipment at each node by migrating among nodes. Mobile agent-based network management has several advantages in comparison with traditional approaches, such as the client/server one.

- As code is very often smaller than the data it processes, the transmission of mobile agents to sources of data creates less traffic than transferring the data itself. Deploying a mobile agent close to the network nodes that we want to monitor and control prevents delays caused by network congestion.
- Since a mobile agent is locally executed on the node it is visiting, it can easily access the functions of devices on this node.
- 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.
- Network management systems must often handle networks that may have various malfunctions and disconnections and whose exact topology may not be known. Since mobile agents are autonomous entities, they may be able to detect proper destinations or routings on such networks.

Adopting mobile agent technology eliminates the need for administrators to constantly monitor many network management activities, e.g., the installation and upgrading of software and periodic network auditing. There have been several attempts to apply this technology to network management tasks. Karmouch presented typical mobile agent approaches to network management [8]. Satoh proposed a framework for building and operating agent itineraries for network management systems [14, 17] and constructed domain-specific languages for describing agent migration for network management [19].

3.3 Cloud computing

Load-balancing is a legacy application of process migration and mobile agent technologies. In a distributed system, e.g., a grid or cloud computing system, computers tend to be numerous and their computational loads are different. Computers may also be dynamically added to or removed from the system. Tasks should be dynamically deployed at computers which loads light rather

than those lose with heavy loads. Since mobile agents can migrate to other computers, tasks that are implemented as mobile agents can be relocated at suitable computers whose processors can execute the tasks. This is practical in implementing massively multi agent systems that must operate a huge number of agents, which tend to be dynamically created or which terminate, on a distributed system that consists of heterogeneous computers.

3.4 Mobile computing

Mobile agents use the capabilities and resources of remote servers to process their tasks. When a user wants to do tasks beyond the capabilities of his or her computers, the agents that perform the tasks can migrate to and be executed at a remote server. Mobile agents can also mask temporal disconnections in networks. Mobile computers are not always connected to networks, because their wired networks are disconnected before they are moved to other locations or wireless networks become unstable or non-available due to deteriorating radio conditions or are not uncovered by the area at all. A stable connection is only requested at the beginning to send the agent, and to take the agent back at the end of the task, but this is not requested during the execution of the whole application execution. Several researchers have explored mechanisms for migrating agents through unstable networks [3, 6, 16]. When a mobile agent requests a runtime system to migrate itself, the system tries to transmit the moving agent to the destination. If the destination cannot be reached, the system automatically stores the moving agent in a queue and then periodically tries to transmit the waiting agent to either the destination or another runtime system on a reachable intermediate node as close to the destination as possible. These relay runtime systems repeat the process until the agent arrives at its destination.

3.5 Software testing

Mobile agents are useful in the development of software as well as the operation of software in distributed and mobile computing settings. An example of these applications is testing methodology for software running on mobile computers, called *Flying Emulator* [15, 18]. Wireless LANs or 4G-networks incorporate wireless LAN technologies, and mobile terminals can access the services provided by LANs, as well as global network services. Therefore, software running on mobile terminals may depend on not only its application-logic but also on services within the LANs that the terminals are connected to. Effective software constructed to run on mobile terminals for 4G wireless networks and wireless LANs must be tested in all networks to which the terminal could be moved and then connected to. Like existing approaches, this provides software-based emulators for mobile terminals for software designed to run on the terminals. It also constructs the emulators as mobile agents that can travel between computers. As we can see from Figure 8, these emulators can carry target software to networks that the terminals are connected to and allow it to access services. These services are provided by the networks in the same way as if the software had been carried by and executed on terminals connected to the networks.

3.6 Active networking

There are two approaches to implementing active networks (for example, see [23]). The active packet approach replaces destination addresses in the packets of existing architectures with minia-

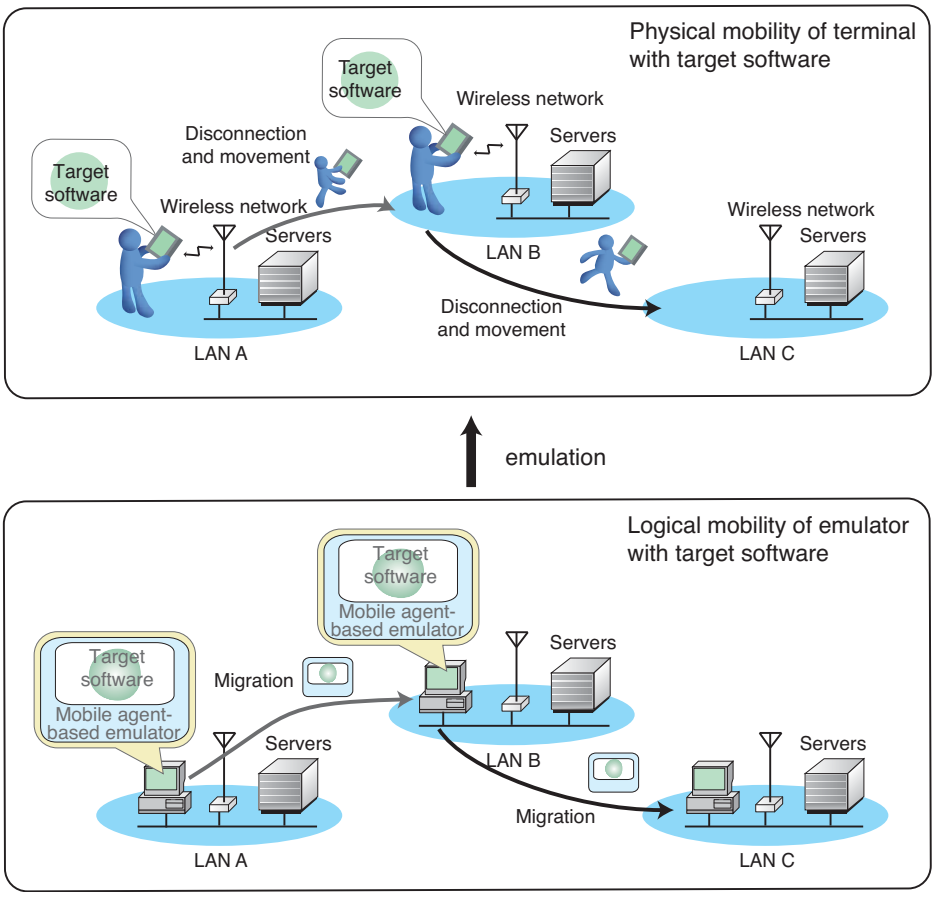

Figure 8: Corelation between the movement of target mobile computer and migration of mobile agent-based emulator

ture programs that are interpreted at nodes on arrival. The active node approach enables new protocols to be dynamically deployed at intermediate and end nodes using mobile code techniques. Mobile agents are very similar to active networks, because a mobile agent can be regarded as a specific type of active packet, and an agent platform in traditional networks can be regarded as a specific type of active node. There have been a few attempts to incorporate mobile agent technology with active network technology (for example, see [8]). Of these, the MobileSpaces system [16] provides a mobile agent-based framework for integrating the both approaches. The framework enables us to implement network processing of mobile agents with mobile agent-based components, where the components are still mobile agents so that they can be dynamically deployed at computers to customize network processing.

3.7 Active documents

Mobile code technology is widely used in plug-in modules for rich internet applications (RIA) in web-browsers, e.g., Java Applet and Macromedia Flash. Such modules provide us with interactive user experiences because their virtual machines, e.g., Java virtual machines and Flash players, can locally execute and render them across multiple platforms and browsers without having to com-

municate with remote servers. However, it is not easy to save their results on local computers or remote servers, and to resume them with the previous results later, since their code can be transported but not their state. Mobile agents solve this problem and provide a next-generation RIA. Mobile agent-based modules for RIA can naturally carry both their code and state at client computers. For example, MobiDoc [11] is a mobile agent-based framework for building mobile compound documents where a compound document can be dynamically composed of mobile agent-based components, which view or edit their contents, e.g., text, images, and movies. It can migrate itself over a network as a whole, with all its embedded components. Each component is self-contained in the sense that it maintains its content and program code for viewing and modifying the content inside it, and multiple components can be combined into an active and mobile document.

4 AMBIENT COMPUTING

Ambient computing is one of the most important applications of mobile agents and mobile agents is useful in building and operating ambient computing environments.

4.1 Mobile agent-based middleware for Ambient computing

Ambient computing environments, which consist of computers often have limited resources, such as restricted levels of CPU power and amounts of memory. Mobile agents can help to conserve these limited resources, since each agent only needs to be present at the computer when the computer needs the services provided by that agent. Ambient computing environments are used as social infrastructures, which must support massive users and consist of numerous devices, in building and cities in future. The scale and complexity of large-scale ambient computing environments are beyond our ability to manage these using traditional approaches, such as those that are centralized and top-down. On the other hand, mobile agents can support a non-centralized approach for ambient computing environment.

4.2 Context-aware Mobile Agent

Several research projects also attempted to migrate agents between computers according to changes in the real world. The SpatialAgent framework [12, 13] provides a bridge between the movement of physical entities, e.g., people and things, and the movement of mobile agents to support and annotate the entities using location-tracking systems, e.g., RFID technology. It binds physical entities with mobile agents and navigate agents to stationary or mobile computers near the locations of the entities and places to which the agents are attached, even after their locations have changed. Figure 9 (a) shows that a moving entity carrying an RF-tagged agent host and a space containing a place-bound RF-tag and RF reader. When the reader detects the presence of the RFID tag that is bound to the agent host, the framework instructs the agents attached to the tagged place to migrate to the visiting agent host to offer location-dependent services of for that place. Figure 9 (b) shows that an RF-tagged agent host and an RF reader have been allocated. When an RF-tagged moving entity enters the coverage area of the reader, the framework instructs the agents attached

to the entity to migrate to the agent host within the same coverage area to offer entity-dependent services to the entity.

One of the most typical application of mobile agent technology in ambient computing is *follow-me* application [5], which tracks the current location of the user and allows him/her to access his/her applications at the nearest computer as he/she moves around in the building. Previous studies of such approaches deploy only the screenshots of applications at appropriate computers. Mobile agent technology can migrate not only the user interfaces of applications but also the applications themselves to computers as shown in Figure 10. To provide context-aware services according to contextual information in the real world, we need a model about the real world, called world model or location model. Satoh propose a mobile agent-based location model as a location-based service discovery system, because it did not maintain only spatial relationships between physical spaces and entities but also the locations of location-based services and computing devices within the model [20].

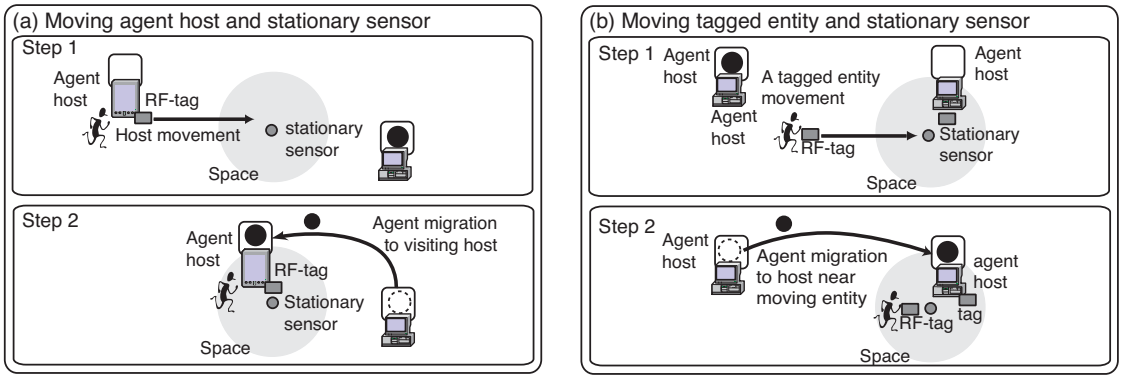

Figure 9: Linkages between physical and logical worlds

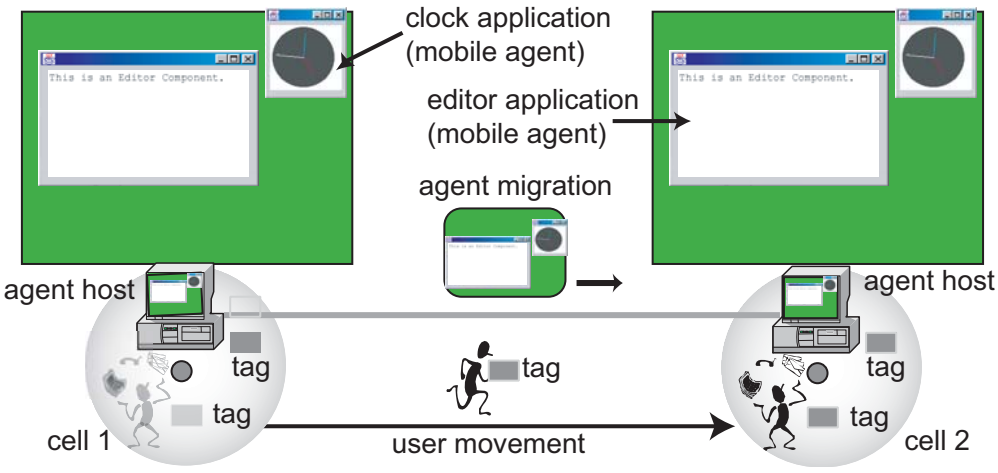

Figure 10: Follow-Me Desktop Applications

4.3 Personal-assistant agents

Agents are often used to assist users in the real world as well as cyberspaces. Their assistance should be provided in personalized form according to their users' requirements. Mobile agents can maintain user preferences and experiences inside them and migrate at at nearby computers to interact with their users without communication delay between the users and agents. Satoh proposed a framework for managing and operating agent-based user-guides in museums. Visitors move from exhibit to exhibit in a museum. Therefore, when they move to another exhibit, their agents should be deployed at computing devices close to their destination exhibits. That is, the virtual agents could accompany their visitors and annotate exhibits in front of the visitors in the real-world on behalf of museum guides or curators (Figure 11). Agents as visitor guides can record and maintain user preferences and experiences because they are active and have state inside them. Therefore, they can provide users with assistant-services according to the users' previous behaviors. They navigate visitors to the next exhibits according to the previous exhibits that they watched in addition to their routes.

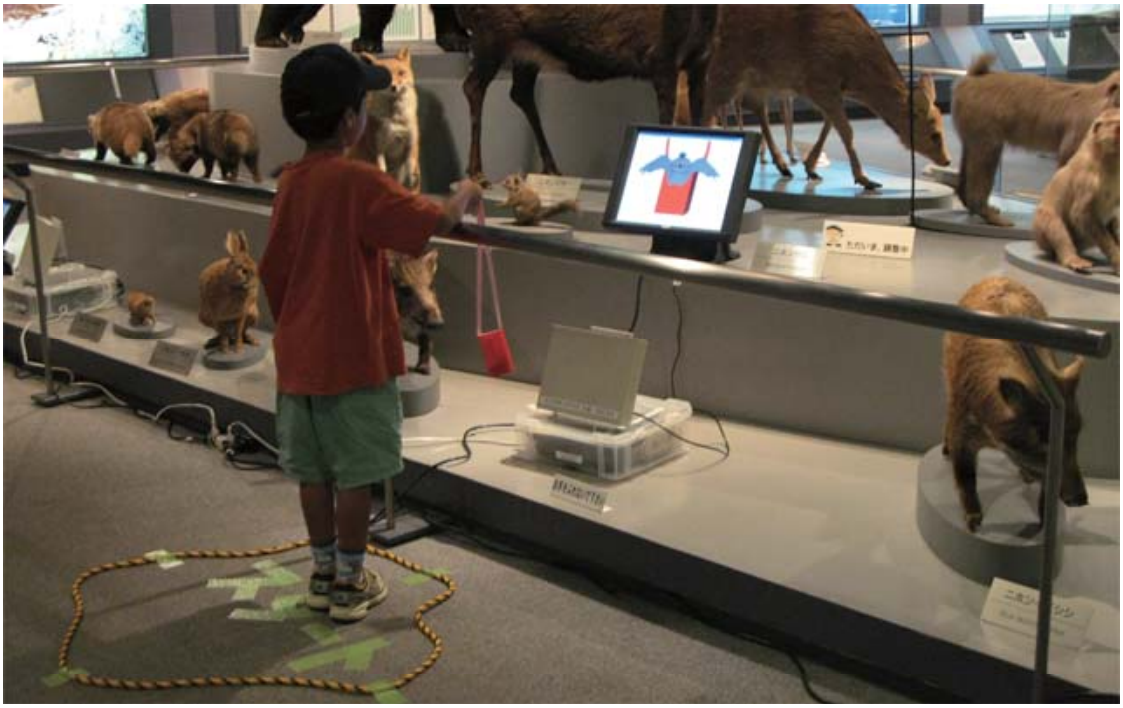

Figure 11: User assistant agent at Museum of Nature and Human Activities in Hyogo.

Each agent is attached to at most one visitor and maintains his/her preference information and programs that provide annotation and navigation to him/her. To enable agents to be easily developed and configured without any professional administrators, we divided each agent into three parts: The first is a user-preference part to maintain and record information about visitors, e.g., their knowledge, interests, routes, name, and the durations and times they spend at exhibits they visit. The second is an annotation part to define a task for playing annotations about exhibits or interacting with visitors. The third is a navigation part to define a task for navigating visitors to their destinations. The third part in the experiment provides four type built-in navigations

corresponding to typical behaviors of visitors in museums outlined in Figure 12. The *navigation* service instructs users to move to at least one specified destination spot. *selection* service enables users to explicitly or implicitly select one spot or route from one or more spots or routes close to their current spots by moving to the selected spot or one spot along the selected route. The *termination* service informs users that they have arrived at the final destination spot. The *warning* informs users that they had missed their destination exhibit or their routes.

As application-specific services could be defined and encapsulated within the agents, we were able to easily change the services provided by modifying the corresponding agents while the entire system was running. More than two different visitor-guide services could also be simultaneously supported for visitors. Even while visitors were participating, curators with no knowledge of context-aware systems were able to configure the annotative content by doing drag-and-drop manipulations using the GUI-based configuration system. Such dynamic configuration is useful, because museums need to continuously provide and configure services for visitors.

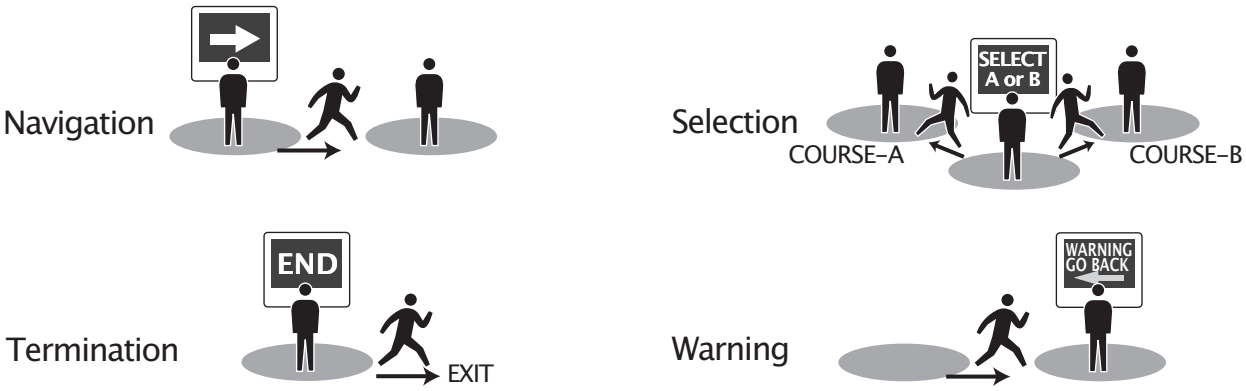

Figure 12: User-navigation patterns.

5 CONCLUSION

Mobile agents are just an implementation technique used in the development and operation of distributed systems, including smart environments, as other software agents, including multi-agents, are themselves not goals but tools for modeling and managing our societies and systems. Therefore, the future of mobile agents may not be specifically as mobile agents. They will be used as essential technologies in distributed computing or ambient computing, even though they will not be called mobile agents. In fact, although monolithic mobile agent systems were developed in the past decade to illustrate the concepts of mobile agents, recent several mobile agent systems have been developed based on several slightly different semantics for mobile agents.

Ambient computing environments become social infrastructures in building and cities. Mobile agents are useful for building and operating such large-scale ambient computing environments, which consist of massive users and numerous devices.

References

[1] K. Arnold, and J. Gosling, The Java Programming Language, Addison-Wesley 1998.

[2] A. Birrel and B. Nelson, Implementing remote procedure calls, ACM Transactions on Computer Systems, vol. 2, no.1, February 1984.

[3] J. Cao, X. Feng, J. Lu, and S. K. Das, Mailbox-Based Scheme for Designing Mobile Agent Communication Protocols, IEEE Computer, pp.54-60, vol. 35, no.9, 2002.

[4] A. Fuggetta, G. P. Picco, and G. Vigna Understanding Code Mobility IEEE Transactions on Software Engineering archive Vol. 24, No. 5, May 1998.

[5] Harter A, Hopper A, Steggeles P, Ward A, Webster P. The Anatomy of a Context-Aware Application. Proceedings of Conference on Mobile Computing and Networking (MOBICOM'99); ACM Press; 1999; 59-68.

[6] D. Kotz, R. S. Gray, S. Nog, D. Rus, S. Chawla, and G. Cybenko, Mobile Agents for Mobile Computing. in D. Milojicic, F. Douglis, and R. Wheeler (ed), Mobility, Mobile Agents and Process Migration, Addison Wesley and ACM Press, 1999.

[7] Dejan Milojicic, Mobile agent applications, IEEE Concurrency, vol. 7, no.4, pp.80-90, July-September 1999

[8] V. A. Pham, A Karmouch, Mobile Software Agents: An Overview, IEEE Communications Magazine, vol. 36 no. 7, pp.26-37, July 1998,

[9] I. Satoh, MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System, Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS'2000), pp.161-168, April 2000.

[10] I. Satoh, MobiDoc: A Framework for Building Mobile Compound Documents from Hierarchical Mobile Agents, Proceedings of International Symposium on Agent Systems and Applications/International Symposium on Mobile Agents (ASA/MA2000), pp.113-125, Lecture Notes in Computer Science (LNCS), vol. 1882, Springer, September 2000.

[11] I. Satoh, MobiDoc: A Mobile Agent-based Framework for Compound Documents, Informatica, vol.25, no. 4, pp.493-500, December 2001.

[12] I. Satoh, Physical Mobility and Logical Mobility in Ubiquitous Computing Environments, Proceedings of 6th International Conference on Mobile Agents (MA'2002), Lecture Notes in Computer Science (LNCS), vol. 2535, pp.186-202, Springer, October 2002

[13] I. Satoh, SpatialAgents: Integrating User Mobility and Program Mobility in Ubiquitous Computing Environments, Wireless Communications and Mobile Computing, vol.3, no.4, pp.411-423, John Wiley, June 2003.

[14] I. Satoh, Building Reusable Mobile Agents for Network Management, IEEE Transactions on Systems, Man and Cybernetics, vol.33, no. 3, part-C, pp.350-357, August 2003.

[15] I. Satoh, A Testing Framework for Mobile Computing Software, IEEE Transactions on Software Engineering, vol. 29, no. 12, pp.1112-1121, December 2003.

[16] I. Satoh, Configurable Network Processing for Mobile Agents on the Internet, Cluster Computing, vol. 7, no.1, pp.73-83, Kluwer, January 2004.

[17] I. Satoh, Selection of Mobile Agents, Proceedings of 24th IEEE International Conference on Distributed Computing Systems (ICDCS'2004), pp.484-493, IEEE Computer Society, March 2004.

[18] I. Satoh, Software Testing for Wireless Mobile Computing, IEEE Wireless Communications, vol. 11, no. 5, pp.58-64, IEEE Communication Society, October 2004.

[19] I. Satoh, Building and Selecting Mobile Agents for Network Management, Journal of Network and Systems Management, vol.14, no.1, pp.147-169, Springer, 2006.

[20] I. Satoh, A Location Model for Smart Environment, Pervasive and Mobile Computing, vol.3, no.2, pp.158-179, Elsevier, 2007.

[21] I. Satoh, Context-aware Agents to Guide Visitors in Museums, in Proceedings of 8th International Conference Intelligent Virtual Agents (IVA'08), Lecture Notes in Artificial Intelligence (LNAI), vol.5208, pp.441-455, September 2008.

[22] Strasser, M., Baumann, J. and Hole, F.: Mole: A Java Based Mobile Agent System, Proceedings of Workshop on Mobile Object Systems, Lecture Notes in Computer Science (LNCS), Vol. 1222, Springer, 1997.

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

[24] J. E. White, Telescript Technology: Mobile Agents, in Software Agents, Bradshaw, J. (ed.), MIT Press, 1997.