Flying Emulator: Rapid Building and Testing of Networked Applications for Mobile Computers

Ichiro Satoh

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

Abstract. This paper presents a mobile-agent framework for building and testing mobile computing applications. When a portable computing device is moved into and attached to a new network, the proper functioning of an application running on the device often depends on the resources and services provided locally in the current network. To solve this problem, this framework provides an application-level emulator of portable computing devices. Since the emulator is constructed as a mobile agent, it can carry target applications across networks on behalf of a device, and it allows the applications to connect to local servers in its current network in the same way as if they were moved with and executed on the device itself. This paper also demonstrates the utility of this framework by describing the development of typical location-dependent applications in mobile computing settings.

1 Introduction

Advances in networking technology have produced a shift in the nature of mobile computing systems and applications. A new class of mobile computing has enabled portable devices to link up with servers in networks to access information from them and delegate heavy tasks to them. Another new class is context-sensitive devices that have a distinct awareness of their locations and current network environments.

The focus of current research, however, is on the design of network and system infrastructure and context-aware applications for mobile computing. As a result, the tasks of building and testing applications have received little attention so far. This is a serious impediment to the growth of mobile computing, because the development of software for mobile computing devices is very difficult due to the limited computational resources of these devices. Furthermore, the tasks are often tedious and susceptible to errors, because changes in network connectivity and location may lead to sudden and unpredictable changes in contextual information. That is, a change in network and location may imply movement away from the servers currently in use, toward new ones. For example, a handheld device with a short-range radio link, such as Bluetooth, carried across the floors of an office building may have access to different resources, such as printers and directory information for visitors, on each floor. Therefore, to construct a correct application, the developer must test it in the environments of all the networks

that the device might be connected to. However, it is almost impossible for the developer to actually carry a portable computing device to another location and connect it to networks in that location. In fact, nobody wants to go up and down stairs carrying a portable device simply to check whether or not it can successfully print out data at networked printers on its current floor.

This paper presents a new framework for building and testing networked applications for mobile computing. This framework, called *Flying Emulator*, addresses the development of networked applications running on mobile computing devices that can be connected to servers through wired or short-range wireless networks. The key idea of the framework is to introduce a mobile agent-based emulator of a mobile device. The emulator performs application-transparent emulation of its target device for applications written in the Java language. Furthermore, since the emulator is implemented as a mobile agent, it can carry its applications to remote networks according to patterns of physical mobility and test the applications in the environments of those networks. Also, the framework provides a platform for building mobile computing applications from a collection of Java-based software components and allowing such an application to be executed on its target portable device without being modified or recompiled.

The remainder of this paper is organized as follows. Section 2 surveys related work and Section 3 explains a framework for building and testing mobile computing applications. Section 4 briefly reviews my mobile agent system and then presents the design and implementation of the framework. Section 5 demonstrates the usability of the framework through two real-world examples. Section 6 offers conclusions and suggestions for further work.

2 Background

There are two different notions of mobility: logical and physical. Physical mobility entails the movement and reconnection of mobile computing devices among networks, while logical mobility involves software, such as mobile code and mobile agents, that migrates among different servers and may use different sets of services on each of them (for example see [5, 9, 14]).

One of the most typical problems in physical mobility is that the environment of a mobile entity can vary dynamically as the entity moves from one network to another. A lot of research has been proposed to either transparently mask variations in mobility at the network or system level or adapt to the current environment at the application level [1, 12, 13]. Nevertheless, current work on these approaches focuses on a location-transparent infrastructure for applications and location-aware applications themselves. Accordingly, the task of building and testing applications has received only limited attention.

Among the recent works, a few researchers have explored emulators of portable computing devices [4, 10]. However, those approaches were designed to emulate some limited resources of target devices on standard workstations and networks, whereas the approach presented in this paper is designed to emulate the mobility itself of portable devices. In fact, it is very difficult for an emulator running on a standalone computer to simulate the whole context that its target device can interact with through networks.

An extreme approach is to actually carry portable devices and attach them to local networks in the current location, but this is extremely cumbersome troublesome for the developer. Another extreme approach enables an application to run on a local workstation and link up with remote servers through networks to access particular resources and services provided by remote networks; for example, the InfoPad project at Berkeley [10] and the network emulator of Lancaster University [4]. However, accomplishing this in a responsive and reliable manner is difficult, and the emulators cannot remotely access all the services and resources that are available only within the local networks because of security protection. Moreover, the approach is inappropriate since the network traffic increases when the amount of the data exchanged between the emulator and the remote servers is large. This is a serious problem in testing monitoring applications for gathering a large quantity data from network nodes or sensors.

Logical mobility is just a new design tool for the developers of distributed applications, while physical mobility results from new requirements for distributed applications. As discussed in [14], these two mobilities have been almost unrelated so far, despite their similarities. Although many mobile agent systems have been released over the last few years, few researchers have introduced the logical mobility approach, including mobile agent approach, as a mechanism for extending and adapting context-sensitive applications to changes in their environments by migrating agents or codes to the applications and servers, for example [8, 11]. These approaches were designed as infrastructures for executing context-aware applications, so the researchers did not intend to test such applications. On the other hand, the framework presented in this paper is unique among existing research on both physical and logical mobility, because it introduces logical mobility as a methodology for building and testing applications in physical mobility.

3 Approach

The goal of this paper is to present a framework for building and testing mobile computing applications. An important target application of this framework is a network-dependent application, in the sense that it is designed to run on a portable computing device and may often access servers on local networks in the device's current location either through wired networks such as Ethernet or short-range wireless networks, such as IEEE802.11b or Bluetooth.[1] As the device moves across networks, the environment may change. That is, some new servers become available, whereas others may no longer be relevant. Such an application must be tested in all the network environments that the device could be moved into and attached to. Furthermore, most portable computing devices, including personal digital assistants and mobile phones, support few debugging and profiling aids since they are kept simple to reduce power consumption and weight.

To solve these problems, this framework introduces a mobile agent-based emulator of a portable computing device for use with applications. The key idea is to emulate

[1] Wireless networking technology permits continuous access to services and resources through the land-based network, even when device's locations change. On the other hand, my goal is to build and test applications running on a mobile computing device which may be connected to local networks in the current location.

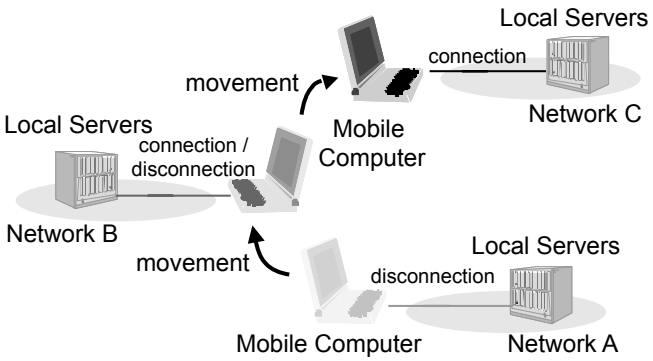

Fig. 1. Physical mobility of a portable device.

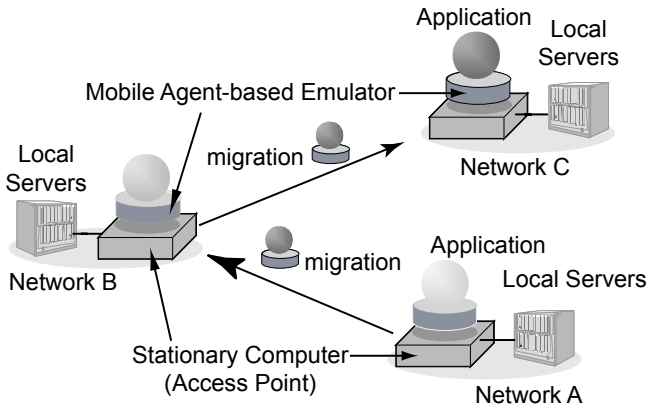

Fig. 2. Emulation of physical mobility by logical mobility.

the physical mobility of a portable computing device by using the logical mobility of the targeted applications (see Figs. 1 and 2). The emulator supports applications with not only the internal environment of its own target portable device, but also the external environment, such as resources and servers provided in the current network. To accomplish this, the framework satisfies the following requirements:

- Like other computer emulators, this framework performs application-level emulation of its target portable device to support applications by incorporating a Java virtual machine.
- Depending on the movement patterns of its target portable device, the mobile agent-based emulator can carry applications on behalf of the device to networks that the device may be moved into and connected to.

– The emulator allows us to test and debug applications with services and resources provided through its current network as if the applications were being executed on the target device when attached to the network.
– All applications tested successfully in the emulator can still be performed in the same way without being modified or recompiled them.

Each mobile agent is just a logical entity and thus must be executed on a computer. Therefore, this framework assumes that each of the networks into which the device may be moved and attached to has more than one special stationary host, called an access point host, which offers a runtime system for mobile agents. Each access point host is a runtime environment for allowing applications running in a visiting emulator to connect to local servers in its network. That is, the physical movement of a portable computing device from one network and attachment to another network is simulated by the logical mobility of a mobile agent-based emulator with the target applications from an access point computer in the source network to another access point computer in the destination network. As a result, each emulator is a mobile agent, and thus it can basically carry not only the code but also the states of its applications to the destination, so the carried applications can basically continue their processes after arriving at another host as if they were moved with its targeted device.

In this framework applications are written in JDK 1.1 or 1.2-compatible Java language, including Personal Java. However, some typical units of the Java language, such as Java applications and Java applets, are not always appropriate in developing software in mobile computing settings. This is because these units are essentially designed to run in a static context and lack any unified mechanism for reporting contextual changes. Instead, this framework introduces mobile agent-based software components for building context-sensitive applications.[2] Another key idea of this framework is to introduce a hierarchical mobile agent system, called MobileSpaces [15], as infrastructure for the framework. This system is characterized by allowing multiple mobile agents to be dynamically assembled into a single mobile agent. Therefore, it enables us to construct an application as a collection of mobile agents, like software component technology [18]. Furthermore, such an application can be extensible and adaptable in the sense that it can dynamically customize its structure and functions to environmental changes in its context by having mobile agent-based components migrated to it.

Since the framework itself and applications are written in the Java language, the target portable devices must support this language. Moreover, while the framework does not require any custom hardware, its current implementation requires its target devices to offer TCP/IP communication over wired or wireless networks.

4 The Flying Emulator Framework

This section presents the prototype implementation of this mobile agent-based framework, called *Flying Emulator*, which consists of the following four parts:

[2] In fact, most Java Applets and Java Beans can be easily translated into mobile agents in the MobileSpaces. Moreover, I implemented an adapter for executing Java Applets and Java Beans within this mobile agent-based components, but it is not compatible with all kinds of Applets and Java Beans.

Mobile Agent-based Emulator: A mobile agent capable of carrying the target applications to specified access point hosts on remote networks on behalf of a target portable device.

Application Runtime System: Middleware, which runs on a portable device, to support the execution of mobile agent-based applications.

Access Point Runtime System: A runtime system, which is allocated in each network, to allow the applications carried by an emulator to connect with various servers running on the network.

Remote Control Server: A graphical front-end to the whole system, which allows us to monitor and operate the moving emulator and its applications by remotely displaying their user interfaces on its screen.

The above parts are constructed independently of the underlying system and thus can run on any computer with a JDK 1.1 or 1.2-compatible Java virtual machine, including Personal Java.

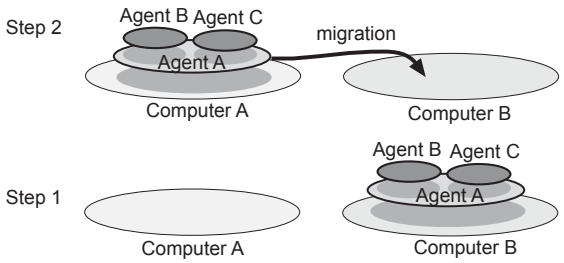

Fig. 3. Migration of hierarchical mobile agents.

4.1 MobileSpaces: A Mobile Agent System

Like other existing mobile agent systems, MobileSpaces can offer mobile agents as computational entities that can travel over networks under their own control. Furthermore, the system is characterized by the notion of hierarchical mobile agents. That is, the system allows multiple mobile agents to be dynamically combined into a single mobile agent. Fig. 3 shows hierarchical mobile agents and their migration. Each agent can directly access the services and resources offered by its inner agents and it is responsible for providing its own services and resources to the inner agents. This concept is applicable in constructing the mobile agent-based emulators presented in this paper, although it was initially introduced for constructing large and complex applications by assembling multiple mobile agents in distributed computing settings.

The MobileSpaces system is built on a Java virtual machine, and mobile agents are provided as Java objects. When an agent is transferred over a network, the runtime system stores the state and code of the agent, including mobile agent-based applications,

in a bitstream defined by Java's JAR file format that can support digital signatures for authentication. The MobileSpaces runtime system supports a built-in mechanism for transmitting the bitstream over networks by using an extension of the HTTP protocol. Almost all intranets have firewalls that prevent users from opening a direct socket connection to a node across administrative boundaries. Since the mechanism is based on a technique called HTTP tunneling, it enables agents to be sent outside a firewall as HTTP POST requests and responses to be retrieved as HTTP responses.

4.2 Mobile Agent-based Emulator

The mission of the mobile agent-based emulator is to carry and test applications designed to run on its target portable device. Each mobile agent-based emulator is just a hierarchical mobile agent of the MobileSpaces system. Since every application is provided as a collection of mobile agent-based components, the emulator can naturally contain more than one mobile agent-based application inside itself and can migrate itself and its inner applications to another place. Since such contained applications are still mobile agents, both the applications running on an emulator and the applications running on the portable device are mobile agents of the MobileSpaces system and can thus be executed in the same runtime environment. Actually, this framework basically offers a common runtime system to both its target devices and access point hosts, in order to minimize differences between them as much as possible. In addition, the Java virtual machine can actually shield applications from most features of the hardware and operating system of target portable devices. Fig. 4 illustrates the correlation between the physical mobility of a running device and the logical mobility of an emulator of the device. As a result, the emulator is dedicated to emulating the movement of its target device.

Emulation of Physical Mobility Each emulator can have its own itinerary as a list of hosts corresponding to the physical movement pattern of its target mobile device. The list is a sequence of the tuples of the network address of the destination, the length of stay, and the name of the method to be invoked upon arrival. An emulator can interpret its own itinerary and then migrate itself to the next destination. Such an itinerary can be dynamically changed by the emulator itself or statically defined by the user through the graphical user interface as shown in Fig. 5. Moreover, the developer can interactively control the movement of the emulator through the remote control server.

When a portable computing device moves in physical space, it may be still running. On the other hand, the emulator cannot be migrated over networks as long as its inner applications are running, because they must be suspended and marshaled into a bitstream before being transferred to the destination. To solve this problem, the framework divides the life-cycle state of each application into three phases: networked running, isolated running, and suspended. In the networked running state, the application is running in its emulator on an access point host and is allowed to link up with servers on the network. Upon disconnection, the application enters the isolated running state. In this state, it is still running in its emulator on an access point host but is prohibited from communicating with any servers on the network. The suspended state means that

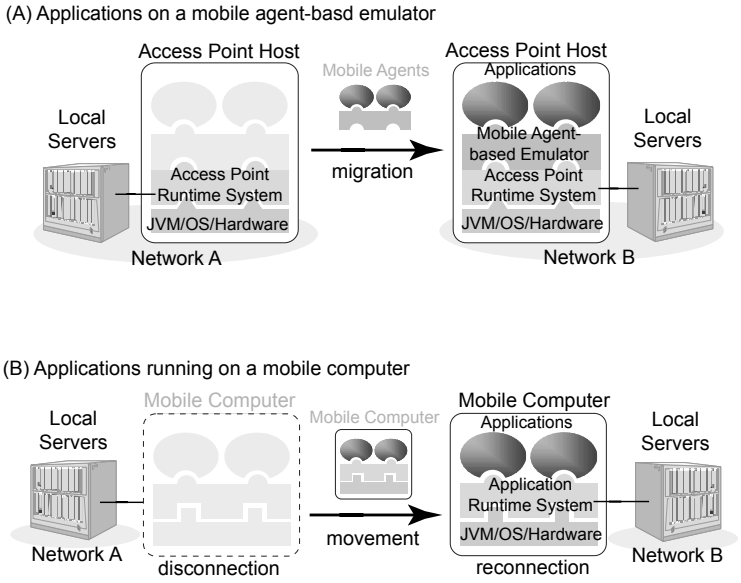

Fig. 4. Emulation of the movement of a mobile computer by migrating a mobile agent-based emulator.

the emulator stops its inner applications while keeping their execution states. This state corresponds to that of a device that is sleeping to save battery life and avoid the risk of accidental damage while moving.

For example, the movement of a suspended and disconnected device corresponds to the suspended state. The movement of a running and then disconnected device is simulated by the combination of the isolated running state on the source or destination host for a specified duration and the suspended state only while migrating. Each emulator maintains the life-cycle states of its inner applications. When the life-cycle state of an application is changed, the emulator dispatches certain events to the application as mentioned in the Appendix.

The Java virtual machine can marshal the heap blocks of a program into a bitstream, but not its stack frames when migrating them, so it is impossible for a thread object to migrate from one virtual machine to another while preserving its execution state. [3] Instead, these events enable an application that has one or more activities using the Java thread library to explicitly stop and store them before migrating over networks.

[3] Several researchers have explored mechanisms for migrating all the execution states of Java objects, including threads and stack frames. However, these existing mechanisms are still premature for my goal, because they cannot transfer most computational resources and do not often coexist with essential optimization techniques for the Java language.

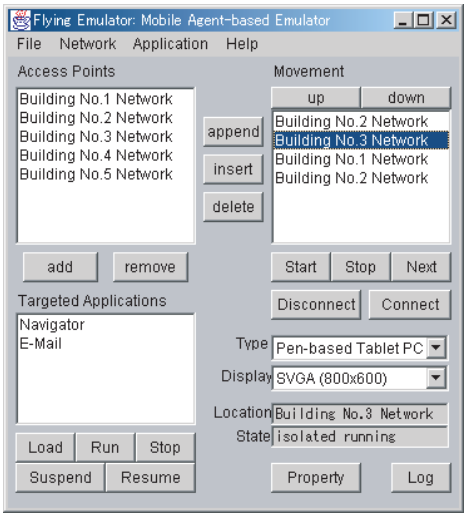

Fig. 5. User interface of a mobile agent-based emulator.

Emulation of Portable Computing Devices The Java VM supports instruction-level emulation of target portable devices and each emulator permits its inner applications to have access to the standard classes commonly supported by the Java virtual machine as long as the target device offers them. In addition, each emulator offers its inner applications the particular resources of the target devices. The current implementation of this framework supports emulators for two kinds of portable computing devices: standard notebook PCs and pen-based tablet PCs running Windows or Linux. Also, the emulators support several typical resources of portable computing devices; for example, file storage and user interfaces such as displays, keyboards, and mouse-based pointing devices.

File Storage: Each emulator can maintain a database to store files. Each file can be stored in the database as a pair consisting of its file/directory path name pattern and its content. Each emulator provides basic primitives for file operation, such as creation, reading, writing, and deletion and also allows a user to insert files into it through its graphical user interface.

Network: When anchored at an access point host, each emulator can directly inherit most network resources from the host, such as `java.net` and `java.rmi` packages. In the current implementation, a moving emulator cannot have its own network identifier, such as IP address and port number. However, this is not a serious problem because most applications on a portable device are provided as client-side programs, rather than server-side ones, as discussed in [7].

User Interface: The user interfaces of most handheld computers are limited by their screen size, color, and resolution, and they may be not equipped with traditional input

devices such as a keyboard and mouse. Each emulator can explicitly constraint only the size and color of the user interface available from its inner applications by using a set of classes for visible content for the MobileSpaces system, called MobiDoc [16]. As mentioned later, our framework furthermore enables the developer to view and operate the user interfaces of applications in an emulator on the screen of its local computer, even when the emulator is deployed at remote hosts.

4.3 Application Runtime System

Like other mobile agents, each mobile agent in the MobileSpaces system must be executed on runtime systems, i.e., an agent platform, that can create, execute, transfer, and terminate agents. Since applications designed for running on the device are implemented as mobile agents, this framework needs to offer a runtime system to each target portable device. Each runtime system maintains the execution of applications. Moreover, to make applications aware of environmental changes, each runtime system monitors the environment of the device, including characteristics such as network connectivity and location. Since this framework introduces the publish-subscribe event model used in the Abstract Window Toolkit of JDK 1.1 or later, the system notifies interested applications by invoking certain of their methods when detecting changes. Furthermore, it provides a collection of service methods to allow applications to have access to the device and its external environment, without any particular knowledge of the operating system and hardware of its target device.

You might wonder whether a mobile agent system is too large to run on portable devices. However, the MobileSpaces runtime system is characterized by its adaptability and its structure can thus easily be customized to be as small as possible by removing additional functions, including agent migration over networks. Also, the performance of applications running on the minimal runtime system is almost equal to that of the corresponding applications executed directly on the Java virtual machine.

4.4 Access Point Runtime System

Each access point host is treated as a peep-hole of the resources and services provided in its network from the applications in a visiting emulator. This framework assumes that more than one access point host is allocated in each network to which the target portable device may be attached. Each access point is constructed based on the common runtime system, which can be used for the target portable devices, and runs on a standard workstation without any custom hardware.

Many applications have their own graphical user interfaces (GUIs). To test such applications, the framework should offer a mechanism for letting these GUIs be remotely viewed and operated on the screen of the remote control server, instead of on the screen of their current hosts. The mechanism is built on the Remote Abstract Window Toolkit (RAWT) developed by IBM [6]. This toolkit allows Java programs that run on a remote host to display GUI data on a local host and receive GUI data from it. Each access point host can perform the toolkit, thus allowing all the windows of applications in a visiting emulator to be displayed on the screen of the control server and be operated using the

keyboard and mouse of the server. Therefore, none of the access point hosts has to offer any graphics services.

4.5 Remote Control Server

This server is a control entity responsible for managing the whole system. It can run on a standard workstation that supports the Java language. It can always track the locations of all the emulators, because each access point host sends certain messages to the control server whenever a moving emulator arrives or leaves. Moreover, the server acts as a graphical front end for the system and thus allows the developer to freely instruct moving emulators to migrate to another locations or terminate, through its own GUIs. Moreover, by incorporating with a server of the RAWT toolkit, it enables us to view and operate the GUIs of target applications on the behalf of their moving emulators. Also, it can monitor the status of all the access point hosts by periodically multicasting query messages to them.

5 Experience

To illustrate the utility of the framework, this section briefly describes my experience in building and testing two typical networked applications designed to run on portable computing devices.

Development of a Location-aware Printing Service System By using the framework presented in this paper, I developed a directory service system for printers in an office building at Ochanomizu University. Each floor of the building is equipped with its own Ethernet-based sub-network and one or more printers, which are of various types and managed by different operating systems. Suppose that a user wants to print out data stored in a portable computer by using a printer on the current floor. This system offers a directory server to the sub-network of each floor. In the current implementation, each server is implemented based on the Jini system [2] and is responsible for discovering printers in only its sub-network. After a portable device attaches to the sub-network of the current floor, the server allocated in the sub-network can automatically advertise its printers to the visiting device. To construct a client-side application for the system, the developer needs to carry the device, attach it to the sub-network of each floor, and then check whether or not it can successfully access every printer on the current floor. We have developed such an application by using the framework. While it was impossible to measure the framework's benefit in a quantitative manner, it did enable a developer to test such applications in the environments of all the floors, without going to each floor. I experimentally compared my system with one of the most conventional testing approach that runs the applications locally and allows them to access remote printing services through proxies for the services. Unlike my approach, the conventional approach can offer only particular services, but not all the services that are locally provided within the sub-network.

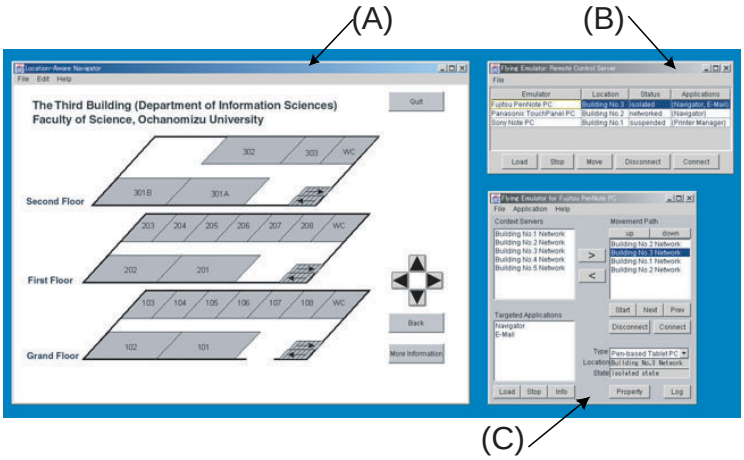

Fig. 6. Screenshot of the remote control server when the user navigation system runs on the mobile agent-based emulator.

Development of a User Navigation System This example illustrates the development of a location-dependent information system for assisting visitors to Ochanomizu University, like [1, 3]. The current implementation of the system provides each visitor with a pen-based tablet PC, which can obtain various information from servers allocated on the sub-network of the current location through an HTTP-based protocol via IEEE802.11b wireless networks. Each building has one or more ranges of wireless networks. When moving from building to building, the tablet PC changes the displayed directory and map to match the user's location. This framework could successfully test the system. That is, I constructed a mobile agent-based emulator for the tablet PC. The emulator can migrate a viewer application designed to run on the tablet PC to the sub-network of another building and enable the application to access the local database of the building and display suitable contents. Since the emulator can travel under its own control, it can exactly simulate the mobility of each visitor in testing such a user navigation system. By using the RAWT toolkit, this framework allows a content creator to view location-dependent information, which should be displayed on the tablet PC on the screen of his/her stationary computer as shown in Fig. 6. Fig. 6 (A) shows a window of the viewer application tested in the emulator. Fig. 6 (B) shows a user interface of the control server for monitoring several emulators and Fig. 6 (C) shows a window of an emulator for controling itself and its applications. Fig. 7 shows the target tablet PC (Fujitsu PenNote Model T3 with Windows98) running the viewer application. As illustrated in Fig. 6 (A) and Fig. 7, both the application running on the emulator and the application running on the target device can have the same presentation of navigation information in the same location. That is, the tested application can be performed in the target device in the same way as if it were executed in the emulator. Furthermore, this example shows that the framework can provide a powerful methodology not only

for testing applications for portable computers but also for creating location-dependent contents.

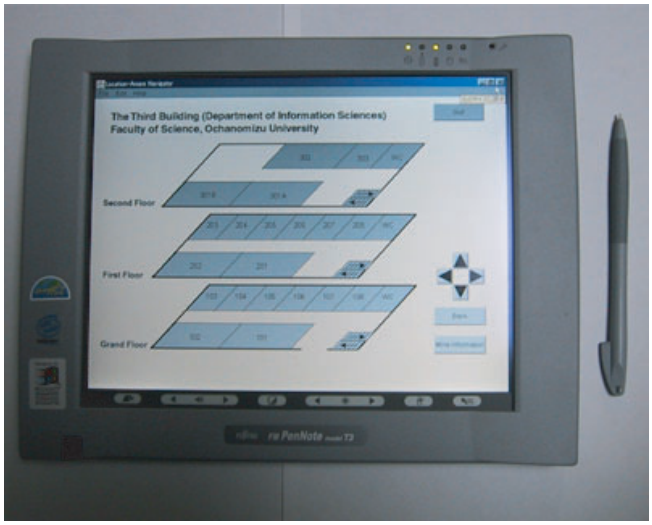

Fig. 7. User navigation system running on a pen-based tablet PC.

6 Conclusion

I have presented a framework for building and testing networked applications for mobile computing. It was inspired by the lack of methodologies for developing context-aware applications in mobile computing settings. IT aims to emulate the physical mobility of portable computing devices by the logical mobility of applications designed to run on the devices. I designed and implemented a mobile agent-based emulator of portable computing devices. Each emulator can perform an application-level emulation of its target device. Since it is provided as a mobile agent in the MobileSpaces system, it can carry and test applications designed to run on its target portable device in the same way as if they were moved with and executed on the device. My early experience with the prototype implementation of this framework strongly suggested that the framework can greatly reduce the time needed to develop networked applications in mobile computing settings. I also believe that the framework is a novel and useful application area of mobile agents and thus provides a significant contribution to mobile agent technology.

Finally, I would like to point out further issues to be resolved. The current implementation of the framework relies on the JDK 1.1 security manager. Although my framework should be used just as a development tool, I plan to design another scheme to perform security and access control. This framework does not support any disconnection operation or addressing scheme for mobile devices. These issues are left open for

future work. Also, the current implementation supports two kinds of portable computing devices: notebook PCs and pen-based tablet PCs. However, the framework can basically support mobile agent-based emulators of any devices having JDK 1.1 or a later version, including Personal Java. I plan to support other devices, including personal digital assistants and information appliances. I presented a mechanism for dynamically customizing routing schemes for mobile agents in another paper [17] and am interested in applying the mechanism to the routing of my mobile agent-based emulator.

Acknowledgments

I would like to thank the anonymous reviewers for their making a lot of significant comments on an earlier version of this paper.

References

1. G.D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, “Cyberguide: A Mobile Context-Aware Tour Guide”. ACM Wireless Networks 3, pp.421–433. 1997.
2. K. Arnold, A. Wollrath, R. Scheifler, and J.Waldo, “The Jini Specification”. Addison-Wesley, 1999.
3. K. Cheverst, N. Davis, K. Mitchell, and A. Friday, “Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project”, Proceedings of ACM/IEEE Conference on Mobile Computing and Networking (MOBICOM’2000), pp.20–31, 2000.
4. N. Davies, G. S. Blair, K. Cheverst, and A. Friday, “A Network Emulator to Support the Development of Adaptive Applications”, Proceedings of USENIX Symposium on Mobile and Location Independent Computing, USENIX, 1995.
5. A. Fuggetta, G. P. Picco, and G. Vigna, “Understanding Code Mobility”, IEEE Transactions on Software Engineering, 24(5), 1998.
6. International Business Machines Corporation, “Remote Abstract Window Toolkit for Java”, http://www.alphaworks.ibm.com/, 1998.
7. J. Jing, “Client-Server Computing in Mobile Environments”, ACM Computing Survey.
8. K. Kangas and J. Roning, “Using Code Mobility to Create Ubiquitous and Active Augmented Reality in Mobile Computing”, ACM/IEEE Conference on Mobile Computing and Networking (MOBICOM’99), pp.48–58, 1999.
9. B. D. Lange and M. Oshima, “Programming and Deploying Java Mobile Agents with Aglets”, Addison-Wesley, 1998.
10. M. Le, F. Burghardt, and J. Rabaey, “Software Architecture of the Infopad System”, Workshop on Mobile and Wireless Information Systems. 1994.
11. N. Minar, M. Gray, O. Roup, R. Krikorian, and P. Maes, “Hive: Distributed agents for networking things”, Proceedings of Symposium on Agent Systems and Applications / Symposium on Mobile Agents (ASA/MA’99), IEEE Computer Society, 2000.
12. B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, K. R. Walker, “Agile Application-Aware Adaptation for Mobility”, Proceedings of ACM Symposium on Operating System Principles, 1997.
13. C. Perkins, “IP Mobility Support”, Internet Request For Comments RFC 2002, 1996.
14. G. Roman, G. Pietro, and A. L. Murphy, “A Software Engineering Perspective on Mobility”, in The Future of Software Engineering (A. Finkelstein eds.), pp.241–258, IEEE Computer Society, 2000.

15. I. Satoh, “MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System”, Proceedings of International Conference on Distributed Computing Systems (ICDCS’2000), pp.161-168, IEEE Computer Society, April, 2000.
16. I. Satoh, “MobiDoc: A Framework for Building Mobile Compound Documents from Hierarchical Mobile Agents”, Proceedings of Symposium on Agent Systems and Applications / Symposium on Mobile Agents (ASA/MA’2000), Lecture Notes in Computer Science, Vol.1882, pp.113-125, Springer, 2000.
17. I. Satoh, “Network Processing of Mobile Agents, by Mobile Agents, for Mobile Agents”, Proceedings of Workshop on Mobile Agents for Telecommunication Applications (MATA’2001), LNCS, pp.81-92, Springer, 2001.
18. C. Szyperski, “Component Software”, Addison-Wesley, 1998.

Appendix: Application Programs

As mentioned previously, each application is constructed as a collection of mobile agent-based components. Each component and each emulator is defined as an instance of a subclass of the abstract class `Agent`, which consists of some fundamental methods for mobility and inter-agent communication.

```
 1: public class Agent {
 2:   // methods for registering a collection of
 3:   // service methods for its inner agents
 4:   void addChildrenContext(Context c){...}
 5:   void removeChildrenContext(Context c){...}
 6:   // methods for registering listener objects
 7:   // to hook certain events
 8:   void addListener(AgentEventListener l){...}
 9:   void removeListener
10:     (AgentEventListener l){...}
11:   ....
12:   Service getService(Message msg)
13:     throws NoSuchSeviceException ... { ... }
14:   ....
15:   void send(AgentURL url, Message msg)
16:     throws NoSuchAgentException ... { ... }
17:   Object call(AgentURL url, Message msg)
18:     throws NoSuchAgentException ... { ... }
19:   void go(AgentURL url1, AgentURL url2)
20:     throws NoSuchAgentException ... { ... }
21:   ....
22: }
```

Each agent can call public methods defined in its emulator by calling the `getService()` method with an instance of the `Message` class that can specify the message type, arbitrary objects as arguments, and deadline time for timeout exceptions. The `send()` and `call()` methods correspond to the asynchronous invocation and method invocation of the agent specified as `url`, respectively. Hereafter, I will describe some

extensions of the agent program of the MobileSpaces system to run on portable computing devices. The `go(AgentURL url1, AgentURL url2)` method instructs the agent specified as `url1` to move to the destination specified as `url2`.

```
 1: interface MobilityListener
 2:   extends AgentEventListener {
 3:   void create(AgentURL url); // after creation at url
 4:   void leave(URL src); // before migration from src
 5:   void arrive(URL dst); // after arrived at dst
 6:   void suspend(); // before suspending
 7:   void resume(); // after resumed
 8:   void destroy(); // before termination
 9:   ....
10: }
```

As mentioned previously, each emulator (and its current context servers) can issue specified events to notify its applications of changes in their life-cycle states. Like Aglets [9], to hook these events, each application can implement a listener interface, `MobiltyListener`, which defines callback methods invoked by the emulator and the runtime system at certain times. For example, suppose that a mobile agent-based emulator is just about to migrate from its current host to another host. An application contained in the emulator is notified by the following process:

1. The `leave(URL src)` method of the application is invoked along with the name of the current network to handle the disconnection from the network, and then the application is prohibited from connecting to any servers.
2. Next, the `suspend()` method of the application is invoked to instruct it to do something doing the suspension, and then it is marshaled into a bitstream.
3. The emulator moves to the destination as a whole with all its inner applications.
4. After the application resumes, its `resume()` method is invoked to do something.
5. The `arrive(URL dst)` method is invoked along with the name of the new current network to handle the reconnection to the network.