A Testing Framework for Mobile Computing Software

Ichiro Satoh, *Member*, *IEEE Computer Society*

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

Index Terms—Testing, mobile computing, mobile agent, network-dependent application.

1 Introduction

Advances in wireless networking technology have produced a shift in the nature of mobile computing systems and applications. A new class of mobile computing has enabled mobile devices to link up with servers through wireless networks such as IEEE 802.11b and Bluetooth to access information from these and delegate heavy tasks to them. Another new class is context-sensitive devices that have a distinct awareness of their current networks.

The focus of current research, however, is on the design of network and system infrastructures and applications for mobile computing. As a result, the task of testing software has attracted little attention so far. This is a serious obstacle in 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 locations 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 computing device with a short-range radio link, such as IEEE 802.11b or 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 efficient application software, 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 mobile 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 mobile 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 testing applications for network-enabled mobile computing. This framework, called the *Flying Emulator*, addresses the development of application-level-software running on mobile computing devices that can be connected to servers through 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 application software written in the Java language. Furthermore, since the emulator is implemented as a mobile agent, it can carry its software to remote networks according to patterns of physical mobility and test the software in the environments of those networks. It also allows the target software successfully tested in the emulator to be executed on its target mobile device without having to be modified or recompiled.

The remainder of this paper is organized as follows: Section 2 surveys related work and Section 3 explains the framework for building and testing mobile computing applications. Section 4 briefly reviews our 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 discusses some future issues and Section 7 provides a summary.

2 Related Work

There are two different notions of mobility: logical and physical. Physical mobility entails the movement and reconnection of mobile computing devices between networks, while logical mobility involves software, such as mobile code and mobile agents, that migrates between different servers and may use different sets of services on each of them, e.g., [5], [9], [16]. A typical problem in

• *The author is with the National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ju, Tokyo 101-8430, Japan.*
E-mail: ichiro@nii.ac.jp.

Manuscript received 31 Dec. 2002; revised 6 July 2003; accepted 5 Aug. 2003.
Recommended for acceptance by M. Oivo and M. Morisio.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 118786.

physical mobility is that the environment of a mobile entity can vary dynamically as it 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 this to the current environment at the application-level [1], [14], [15]. Nevertheless, current work on these approaches has focused on a location-transparent infrastructure for the applications and location-aware applications themselves. Accordingly, the task of building and testing applications has gained only limited attention.

Moreover, a typical mobile computing device has a less powerful processor with less memory and a limited user interface with a clamped keyboard and a small screen. It is therefore difficult to build and debug the software for it within the device itself. A popular and practical solution to this problem is to offer a software-based emulator for the target device. Actually, some mobile computing devices, such as palm-sized PDAs and smart mobile phones, have their own software-based emulators, which are designed to run on workstations and simulate the application-level execution environments of the devices. These emulators have been widely used in the development of software for such devices.

Cooperation between mobile computing devices and servers within a domestic or office network may be indispensable because it complements various missing features in mobile devices. As a result, the correctness of software running on the devices not only depends on its internal execution environment, but also the external environments provided by the network that it connects to. On the other hand, mobile computing devices may be disconnected from the current network and then reconnected to another network. As a device moves across subnetworks, or joins or leaves a subnetwork, some new servers become available from the software running on it or it may no longer be able to access previous servers. Such software must be tested in all the network environments that the device could possibly be moved or attached to.

However, existing emulators are not always available for the development of network-dependent software in the sense that the software may have access to servers on current subnetworks. This is because they were designed to emulate some limited target device resources and it is almost impossible for an emulator running on a standalone computer to simulate the whole context that its target device interacts with through networks. The best way to solve this problem is for the developer to actually carry a workstation running an emulator of the target device (or the device itself) to run software and to attach it to subnetworks in the current location. However, this is extremely laborious for the developer and consequently should only be resorted to in the final phase of software development.

Another approach enables software to run on a local workstation and link up with remote servers through networks to access particular resources and services provided by remote networks, e.g., 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 only available within the subnetworks because of security, such as firewalls. Moreover, the approach is inappropriate in testing software using service discovery protocols. Since a mobile computing environment is dynamic, we require zero user configuration and administration. To solve this problem, several middleware systems, such as Jini [2] and Universal Plug and Play (UPnP) [11], have often been used to manage devices. These middleware systems use multicast communications to discover their management servers and devices, where multicast-based messages may only be transmitted to the hosts within specified subnetworks. Therefore, the target software to run on ubiquitous computing devices must be tested within the subnetworks that the devices can be connected to.

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 [16], these two mobilities have been almost unrelated thus far, despite their similarities. Although many mobile agent systems have been released over the last few years, few researchers have introduced logical mobility, including mobile agent technology, 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, e.g., [8], [12]. These approaches were designed as infrastructures for executing context-aware applications, so the researchers did not intend to test such applications. The framework presented in this paper, however, is unique in terms of both physical and logical mobility because it introduces logical mobility as a methodology for building and testing applications in physical mobility. We described an early prototype implementation of this framework in a previous paper [19], but the implementation only supported specified classes for I/O and network operations and was thus not available in the testing software on ubiquitous computers managed through multicast protocols. The previous paper also lacked explanations on target application software.

3 Approach

Wireless networking technology permits continuous access to services and resources through the land-based networks, even when a device's location changes. Our goal, on the other hand, is to build and test an application running on a mobile computing device which may be connected to local networks in the current location. The aim of this paper is to present a framework for building and testing mobile computing applications. An important target of this framework is a network-dependent application, in the sense that it is designed to run on a mobile computing device and may often access servers on local networks in the device's current location through short-range wireless networks, such as IEEE802.11b or Bluetooth. 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 mobile computing devices, including personal digital assistants and mobile

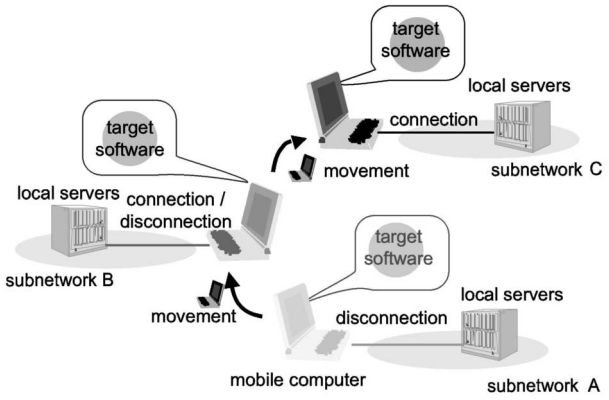

Fig. 1. Physical mobility of mobile device.

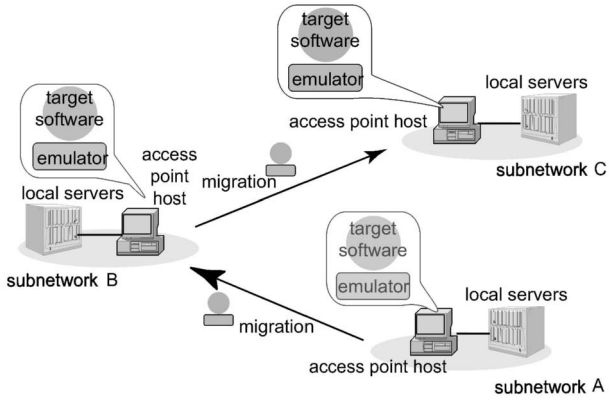

Fig. 2. Emulation of physical mobility through logical mobility.

phones, support few debugging and profiling aids since they have been kept simple to reduce power consumption and weight.

To solve these problems, this framework introduces a mobile agent-based emulator of a mobile computing device for use with target application software. The key idea behind the framework is the emulation of the physical mobility of a ubiquitous or mobile computing device by using the software's logical mobility, which has been designed to run on the device over various networks. Physical mobility entails the movement and reconnection of mobile computing devices between subnetworks (see Fig. 1), while logical mobility involves software, such as mobile codes and mobile agents, that migrates between hosts on the subnetworks (see Fig. 2). The emulator supports applications with not only the internal environment of its own target mobile device, but also the external environment, such as the resources and servers provided in the current network. The framework satisfies the following requirements to accomplish this.

- Like other computer emulators, this framework performs application-level emulation of its target mobile device to support applications by incorporating a Java virtual machine.
- Depending on the movement patterns of its target mobile 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 successfully tested in the emulator can still be performed in the same way without being modified or recompiled.

Each mobile agent is just a logical entity and must thus be executed on a computer. Therefore, this framework assumes that each of the subnetworks to which the device may be moved and attached has more than one special stationary host, called an *access point host*, which offers a runtime system for executing and migrating mobile agent-based emulators. 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 mobile computing device from one network and attachment to another 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 can thus 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 had been moved with its targeted device.

This framework assumes that every target application software must be written in JDK 1.1 or 1.2-compatible Java language, including Personal Java. However, some typical units of Java language, such as Java applications and Java applets, are not always appropriate in developing application 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 runtime changes, such as network disconnection and suspension. Instead, this framework introduces an application as a collection of mobile agent-based components, which are designed to be aware of such changes. The current implementation is also built on a hierarchical mobile agent system, called MobileSpaces [17]. The system is characterized by allowing a mobile agent to carry other mobile agents to another location. Therefore, our mobile agent-based emulator, which contains more than one target application, is still a mobile agent and can travel and, thus, migrate between subnetworks. Nevertheless, most Java Applets and Java Beans can easily be translated into mobile agents in the MobileSpaces. Actually, we implemented an adapter for executing Java components, such as Java Applets and Java Beans within these mobile agent-based components, but it is not compatible with all kinds of Applets and Java Beans because some existing Applets and Java Beans manage their threads and input and output devices in a deprecative manner.

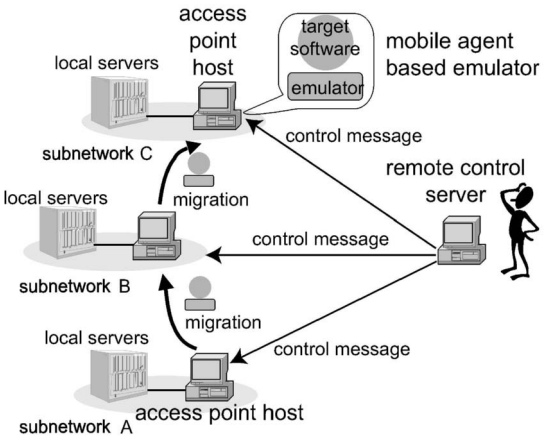

Fig. 3. Architecture of framework.

4 The Flying Emulator Framework

This section presents the prototype implementation of this mobile agent-based framework, called *Flying Emulator*. As we can see from Fig. 3, the framework has the following three components:

- **Mobile agent-based emulator:** A mobile agent capable of carrying the target application to specified access-point hosts on remote networks on behalf of a target mobile device.
- **Application runtime system:** Middleware, which runs on a mobile device, to support the execution of mobile agent-based applications.
- **Remote control server:** A graphical front-end to the whole system, which allows us to monitor and operate the moving emulator and its target application by remotely displaying its graphical user-interfaces on its screen.

In addition to the above, we provided a runtime system to run on a mobile computing device and support the execution of the tested application. As the framework is constructed independently of the underlying system, it can run on any computer with a JDK 1.1 or 1.2-compatible Java virtual machine, including Personal Java, and the MobileSpaces system.

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. 4 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.

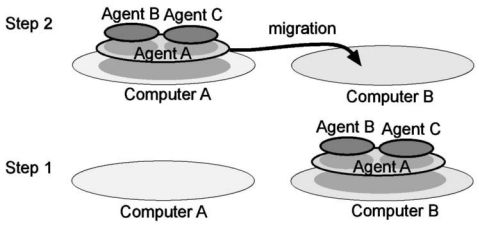

Fig. 4. Migration of hierarchical mobile agents.

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 mobile device. Each mobile agent-based emulator is just a hierarchical mobile agent of the MobileSpaces system. The framework assumes that every target 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 mobile 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, to minimize differences between them as much as possible. In addition, the Java virtual machine can actually shield the target application from most features of the hardware and operating system of the target mobile devices. Fig. 5 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.

4.2.1 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 a graphical user interface displayed on the remote control server. Moreover, the developer can interactively control the movement of the emulator through the remote control server.

When a mobile computing device moves in physical space, it may be still running. However, 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 the following three phases:

- **Networked running state**: A target application is running in its emulator on an access-point host and is allowed to link up with servers on the network.

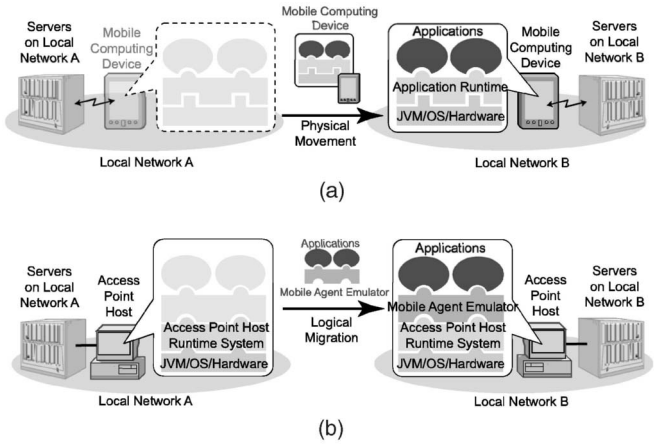

Fig. 5. Emulation of the movement of mobile computer by migrating mobile agent-based emulator. (a) Applications running on a mobile computer. (b) Applications on a mobile agent-based emulator

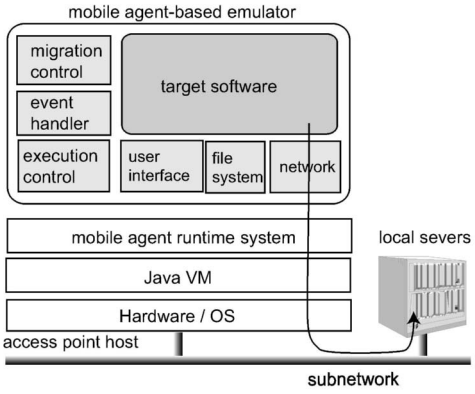

Fig. 6. Mobile agent-based emulator running on access-point host.

This state corresponds to that of a device that is running and connecting to a network in the current location.

- **Isolated running state**: The application is still running in its emulator on an access-point host but is prohibited from communicating with any servers on the network. On disconnection, the application enters an isolated running state.
- **Suspended state**: The suspended state means that the emulator stops its inner applications while maintaining their execution states. This state corresponds to that of a device that is sleeping to save battery life and it avoids 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 explained in the appendix. By using Java's object serialization package, the MobileSpaces 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.[1] 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.

4.2.2 Emulation of Mobile Computing Devices

The Java VM supports instruction-level emulation of target mobile 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 mobile computing devices: standard notebook PCs and pen-based tablet PCs running Windows or Linux. Also, the emulators support several typical resources for mobile computing devices; e.g., file storage and user interfaces such as displays, keyboards, and mouse-based pointing devices. Fig. 6 has the structure of a mobile agent-based emulator running an access-point host.

1. *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.
2. *Network*: While anchored at an access-point host, each emulator allows its inner application to directly access most network resources from the host, such as `java.net` and `java.rmi` packages. Thus, such a carried application can directly establish TCP channels and exchange UDP packets. Since it is deployed and executed within the domain of the current subnetwork, it also can receive multicast packets such as Jini's and UPnP's management messages that are available in the domain. In the current implementation, a moving emulator cannot have its own network identifier, such as an IP address and port number. However, this is not a serious problem because most applications on a mobile device are provided as client-side programs, rather than server-side ones, as discussed in [7].
3. *Input and Output Port*: Each emulator can permit its target application software to be Java's communication APIs (Java COMM), if they are provided on the device that the emulator runs on. Furthermore, the framework offers a mechanism that allows its target software to have access to equipment running on remote computers via serial ports. The mechanism consists of proxies whose interfaces are compatible

1. 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 to attain our goal because they cannot transfer most computational resources and do not often coexist with essential optimization techniques for the Java language.

with Java's communication APIs and which can forward the port's signals between the emulator and the remote-control server through TCP/IP channels. In almost all Intranet situations, a firewall prevents users from opening a direct socket connection to a node across administrative boundaries.

4. *User Interface*: The user interfaces of most mobile computers are limited by their screen size, color, and resolution, and they may be not equipped with traditional input devices such as keyboards and mice. Each emulator can explicitly constrain 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 Mobi-Doc [18]. As will be discussed, our framework 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.

Our framework enables the whole user interface of a device, including the graphical user interface of target applications, to be displayed on the screen of the remote control server and operated from the standard input devices of the server, such as the keyboard and mouse. This mechanism is constructed on the Remote Abstract Window Toolkit (RAWT) developed by IBM [6]. This toolkit enables Java programs that run on a remote host to display GUI data on a local host and receive GUI data from it. The toolkit can be incorporated into each access-point host, thus enabling all the application windows in a visiting emulator to be displayed on the screen of the remote control server and operated through the server's keyboard and mouse. Therefore, the developer can always test his/her target applications, including their GUIs, within a desktop computing environment and the access-point hosts do not have to offer any graphics services or user-input devices. The current implementation of the framework supports emulators for three kinds of computing devices: standard notebook PCs, pen-based tablet PCs, and palm-sized PDAs.

4.3 Access-Point Host

We assume that more than one access-point host would be allocated in each network, to which the wireless device may be attached. As previously mentioned, the framework is built on the MobileSpaces mobile agent system. Each access-point host is a server or workstation offering a MobileSpaces runtime system for executing the mobile agent-based emulator and migrating it to another access-point host. The host should provide Java VM (JDK 1.2 or later version), but does not need any custom hardware. When an agent is transferred over a network, the runtime system stores the state and codes of the agent, including its software, in a bitstream defined by Java's JAR file format, which 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. In almost all intranets, there is a firewall that prevents users from opening a direct socket connection to a node across administrative boundaries. Since this mechanism is based on a technique called HTTP tunneling, emulators can be sent outside the firewall as HTTP POST requests and responses can be retrieved as HTTP responses. Also, each access point host support IBM's Remote AWT to emulate the GUIs of the target software.

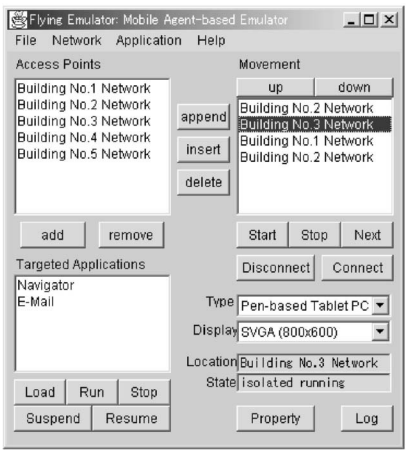

Fig. 7. User interface of mobile agent-based emulator.

4.4 Remote Control Server

This server is a control entity responsible for managing the whole system. It can run on a standard workstation that supports Java. It can always track the locations of all the emulators because each access-point host sends certain messages to the control server whenever the moving emulators arrive or leave. 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 other locations and terminate, through its own graphical user interface (Fig. 7). Moreover, incorporating with a RAWT toolkit's server, enables us to view and operate the graphical user interfaces of targeted applications on behalf of their moving emulators. It also can monitor the status of all access-point hosts by periodically multicasting query messages to them.

4.5 Application Runtime System on Mobile Computing Devices

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.

The reader 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 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 directly executed on the Java virtual machine.

4.6 Security

Security is essential in mobile agent computing. The framework is not serious in comparison with other mobile agent applications because it is used in the process of software development. Nevertheless, it can directly inherit the security mechanism of the underlying mobile agent system. To protect against the arrival of malicious mobile agent-based emulators from agent hosts, the MobileSpaces system supports a Kerberos-based authentication mechanism for agent migration [21]. It authenticates users without exposing their passwords on the network and generates secret encryption keys that can selectively be shared between mutually suspicious parties. The Java virtual machine can also explicitly restrict agents so that they can only access specified resources to protect hosts from malicious agents.

5 Experience

To illustrate the utility of the framework, this section describes our experience in testing two typical network-dependent applications designed to run on mobile computing devices.

5.1 Testing User Navigation System

This example illustrates the development of a location-dependent information system for assisting visitors to some buildings of Ochanomizu University like [1], [3] and then compares this framework with other emulator-based approaches presented in Section 2. The current implementation of the system provides each visitor with a wireless-LAN enabled tablet PC, which can obtain various information from servers allocated on the subnetwork of the current location through an HTTP-based protocol via IEEE 802.11b wireless networks. Each building has a subnetwork with more than one server, called *location information server*, and is covered by the coverage area of a wireless access-point connected to the servers without any overlap in the ranges of different buildings. Each location information server periodically multicasts an advertising message to notify its own network address to visiting tablet PCs within its subnetwork through UDP multicast communication.[2] When a table PC moves from building to building, it receives an advertising message from servers in the subnetwork of the current building and then knows the network address of the servers in the subnetwork. It then tries to access content, such as maps, from the servers and displays the content on its screen.

2. The current implementation uses a simple version of UPnP to locate information servers.

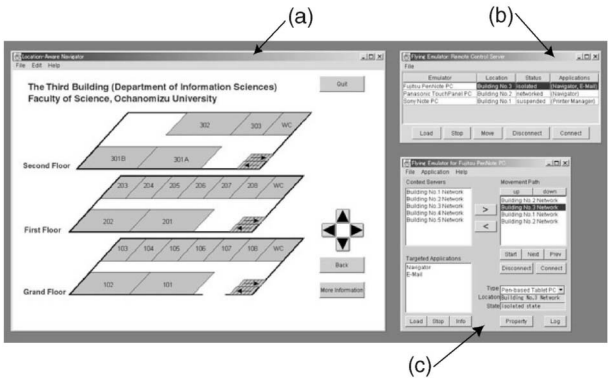

Fig. 8. Screenshot of the remote control server when user navigation system runs on the mobile agent-based emulator.

5.1.1 Software Testing

To test the system with this framework, we constructed a mobile agent-based emulator for the tablet PC. The developer can instruct the emulator to migrate to an access-point host on the subnetwork of another building through the remote control server. Also, since the emulator can define its own itinerary among subnetworks, it can precisely trace the movement of each visitor. The emulator can carry a viewer application designed to run on the tablet PC to the subnetwork of another building. It continues to run the application in the subnetwork and permits the application to directly receive UDP multicast packets, which location information servers only transmit within the domain of a subnetwork. We measured the processing overhead of the emulator, but the performance of an application running in an emulator on an access-point host was not inferior to that of the same application running on the target device, as long as the processor capability of the host was equivalent to that of the device.[3]

By using the RAWT toolkit, this framework allows application-software developers to view and operate the GUIs of the target application on the screen of the remote control server as shown in Fig. 8. Fig. 8a is a window of the viewer application tested in the emulator. Fig. 8b is a user interface of the control server for monitoring several emulators and Fig. 8c shows a window of an emulator for controling itself and its applications. Fig. 9 shows the target tablet PC (Fujitsu PenNote Model T3 with Windows98) running the viewer application. As illustrated in Fig. 8a and Fig. 9, both the application running on the emulator and the application running on the target device can present the same 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 being executed in the emulator. In addition, the software tested successfully in the emulator could still be run in the same way without modifying or recompiling it.

Furthermore, this example shows that the framework can provide a powerful method for easily testing not only application software but also for creating location-dependent content, such as map and annotations about locations.

3. In fact, the former is better than the latter because Java VM for desktop or server computers is often faster than Java VM for mobile computing devices.

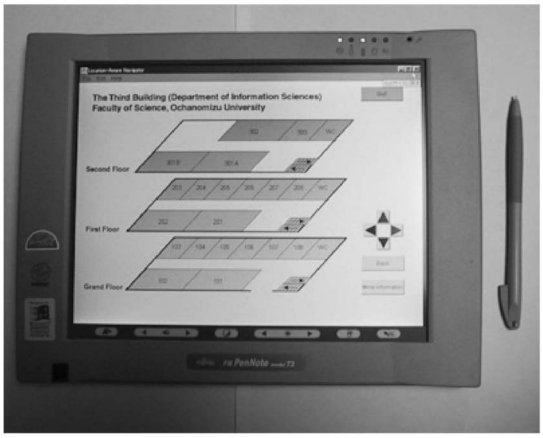

Fig. 9. User navigation system running on pen-based tablet PC.

5.1.2 Discussion

Let us now compare the framework with the other two existing approaches discussed in Section 2. As previously mentioned, most existing emulators of mobile computing devices have been designed for standalone software and are thus not available in testing network-dependent software. A few emulators support testing of network-dependent software by means of the following two approaches.

5. *Comparison with Field Testing Approach*: This approach involves the developer carrying a workstation running an emulator of the target tablet PC or the tablet PC itself and testing the target software in the emulator within the subnetwork in the current location. The developer can directly view and operate the GUI of the map viewer application on the workstation or the device. Like our framework, this approach permits the map viewer application to receive the advertising packets that the location information servers multicast within the subnetwork because the application is running within the domain of the subnetwork. However, the developer walk between the buildings carrying the workstation or the device simply to check whether or not the application is displaying the proper content according to the buildings. This task is extremely cumbersome for the developer. Our framework, however, can replace the physical mobility of the developer with the logical mobility of a mobile agent-based emulator and it thus enables the application to run and link up with servers in the subnetwork.
6. *Comparison with Network-Enabled Emulator Approach*: A few emulators enable software to run on a local workstation and link up with location information servers, on target networks that their target devices may connect to, through networks. Unfortunately, such existing emulators are not available since their purpose is testing particular devices. Instead, we constructed a network-enabled emulator for Java-based software by removing our mobile agent-based emulator's mobility.

The approach has two disadvantages, which our framework does not have. Some subnetworks often restrict the reachable range of multicast packets within their domains due to the need for reducing network traffic and security. Therefore, unlike our framework, this approach needs a mechanism for forwarding UDP packets from location information servers in subnetworks, which the target devices may be moved and connected to, to the map viewer application, so that the application running in a network-enabled emulator on the workstation outside the subnetworks cannot receive advertising packets multicast from servers on the subnetworks. The cost of the mechanism also is inevitable. We implemented proxies for forwarding UDP multicast packets beyond the domains of subnetworks through UDP unicast communication and deployed these proxies at hosts in the subnetworks. Actually, it took about a few milliseconds for our prototype proxies to forward packets to the application.

This approach resulted increased latency and network traffic in communication between the target application and servers, unlike ours, because the application in an emulator had to remotely communicate with the servers via routers and gateways, whereas the target device could be directly connected to the servers. This is a serious problem in testing applications in gathering a large volume of data from servers, vice versa. The reader may wonder about the network traffic and latency in IBM's RAWT toolkit. However, since it only transfers small packets corresponding to Java's GUI APIs instead of bitmaps, it does not seriously affect network costs and, thus, enable the developer to view and operate the GUI of the application in a reasonable response time even when the application runs in remote subnetworks.

5.1.3 Remarks

The framework presented in this paper can be viewed as being between two existing approaches, i.e., field testing emulator and network-enable emulator. It can alleviate the disadvantages of each through the advantages of the other. Therefore, it should complement the two existing approaches but not exclude them.

5.2 Testing Plug-and-Play Management System

Next, we illustrate that our framework is useful in testing application-level protocols for ubiquitous computing devices. In a previous project [13], we implemented a subset of the UPnP protocol written in Java. Using this framework, we tested the interoperability of our UPnP implementation and other UPnP-aware devices. UPnP [11] is an infrastructure for managing various devices such as smart appliances, embedded computers, and PCs. It uses a multicast-based management protocol, called Simple Service Discovery Protocol (SSDP), to announce a device's presence to others as well as to discover other devices or services. For example, a joining device sends out a multicast message to advertise its services to the UPnP's control points. Like the previous application, UPnP aware-software for such a device must be tested within the domain of subnetworks that the device may be connected to. Therefore, we constructed a mobile agent-based emulator just as a carrier for the software. When the emulator arrives at an access-point host within the domain,

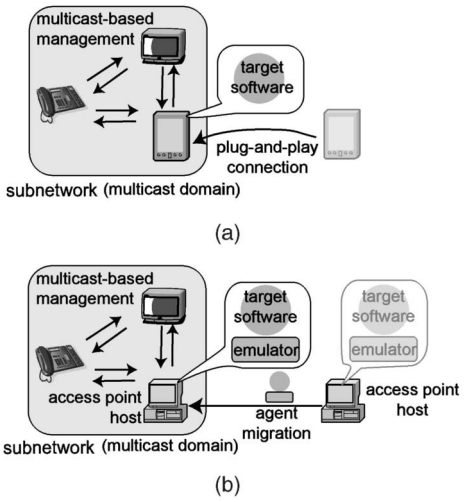

Fig. 10. Emulation of (a) the plug-and-play operation of ubiquitous computing device (UPnP client) by (b) the migration of the emulator for the device between access-point hosts.

the software it carries can send out an advertisement multicast message and receive search multicast messages from other devices in the domain as if the emulator's target were joined to the domain, as shown in Fig. 10.

This example demonstrates that our framework can provide a powerful methodology for testing the interoperability of application-level protocols, limited within specified subnetworks for reasons of security and reduced network traffic. While it is impossible to measure the framework's benefits in a quantitative manner, it eliminates the task of the developer having to carry and connect his/her target device to subnetworks to verify whether software designed for running on the device can successfully coordinate with servers or other devices. Moreover, the framework can test client-side software, which is managed by using other service discover mechanisms, such as Jini [2]. Since a mobile agent-based emulator enables its target software to access Java's standard classes for network processing provided by the current access-point hosts, the software can interact with Jini's servers by using Java's RMI and multicast APIs.

6 Future Work

The current implementation of the framework relies on the JDK 1.1 security manager. Although our framework should only be used just as a development tool, we plan to enhance security and access control. This framework does not support any disconnection operations or addressing schemes for mobile devices. These have been left open for future work. Also, the current implementation supports three kinds of Java-enabled portable computing devices: notebook PCs, pen-based tablet PCs, and PDAs. However, the framework can basically support mobile agent-based emulators of any devices with JDK 1.1 or a later version, including Personal Java. We plan to support other devices, including information appliances. Our approach can complement existing software-development methodologies for ubiquitous computing as well as mobile computing. We are therefore interested in preparing tools to integrate our approach with other methodologies. The location-aware mobile agent infrastructure we developed incorporates RF-based and infrared-based tag sensors [20] and the framework we propose should be able to support these.

7 Conclusion

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

Appendix

Application Program

As previously mentioned, each application, which can be tested in our mobile agent-based emulators, is composed of more than one mobile agent-based component. However, typical Java software units, including Java applets and Java beans, can easily be modified to such components by implementing the following listener interface.

```
interface ApplicationListener
  created(); // invoked after creation
  terminating(); // invoked before termination
  networked();
   // invoked after network enabled
  isolated();
   // invoked after network disconnected
  suspending(); // invoked before suspension
  resumed(); // invoked after resumption
}
```

The above interface specifies callback methods invoked by the emulator and the runtime system on the target device when the life-cycle state of an application, such as the networked running state, isolated running state, and suspended state changes. Each application must define the proper processes in each of these methods to hook and handle such changes. Fig. 11 has the state-transition diagram of an application.

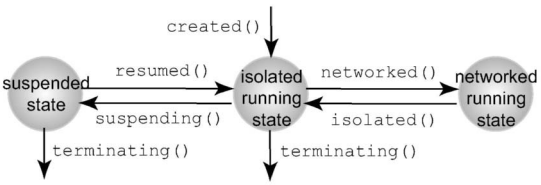

Fig. 11. Callback method invocations in life-cycle state-transition.

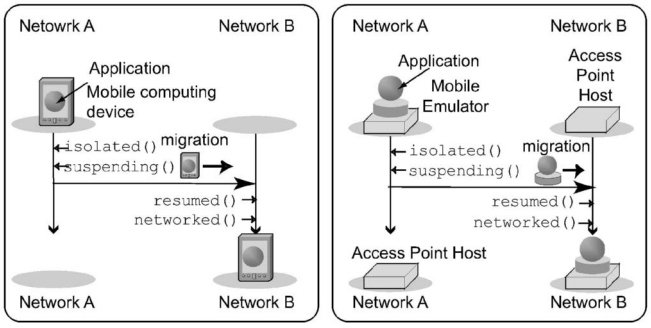

Fig. 12. Movement of computing device and migration of corresponding mobile agent-based emulator.

For example, suppose that a mobile agent-based emulator is just about to migrate from its current host to another host. An application in the emulator is notified through the following process:

1. The `isolated()` method of the application is invoked to handle disconnection from the network, and then the application must release resources, such as sockets and RMI remote references, which are captured by the application and be prohibited from connecting to any servers.
2. Next, the `suspending()` method of the application is invoked to instruct it to do something, e.g., closing its graphical user interface, and then the application is marshaled into a bit-stream.
3. The emulator migrates to the destination as a whole with all its inner applications.
4. After the application is unmarshaled from the bit-stream, its `resumed()` method is invoked to do something, for example, redrawing its graphical user interface.
5. After the `networked()` method is invoked, the application is permitted to connect to servers on the current networks.

Fig. 12 illustrates the correlation between an application running on a mobile computing device and that on its emulator during the above process.

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," *Proc. ACM/IEEE Conf. 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," *Proc. USENIX Symp. Mobile and Location Independent Computing, USENIX*, 1995.

[5] A. Fuggetta, G.P. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Trans. Software Eng.*, vol. 24, no. 5, May 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, 1999.

[8] K. Kangas and J. Roning, "Using Code Mobility to Create Ubiquitous and Active Augmented Reality in Mobile Computing," *Proc. ACM/IEEE Conf. 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," *Proc. Workshop Mobile and Wireless Information Systems*, 1994.

[11] "Universal Plug and Play Device Architecture Version 1.0," Microsoft Corporation, June 2000, http://www.upnp.org/UpnPDevice_Architecutre_1.0.htm.

[12] N. Minar, M. Gray, O. Roup, R. Krikorian, and P. Maes, "Hive: Distributed Agents for Networking Things," *Proc. Symp. Agent Systems and Applications/Symp. Mobile Agents (ASA/MA '99)*, 2000.

[13] T. Nakajima, I. Satoh, and H. Aizu, "A Virtual Overlay Network for Integrating Home Appliances," *Proc. Int'l Symp. Applications and the Internet (SAINT '02)*, pp. 246-253, Jan. 2002.

[14] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker, "Agile Application-Aware Adaptation for Mobility," *Proc. ACM Symp. Operating System Principles*, 1997.

[15] C. Perkins, "IP Mobility Support," *Internet Request For Comments RFC 2002*, 1996.

[16] G. Roman, G. Pietro, and A.L. Murphy, "A Software Engineering Perspective on Mobility," *The Future of Software Eng.*, A. Finkelstein, ed., pp. 241-258, 2000.

[17] I. Satoh, "MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS '00)*, pp. 161-168, Apr. 2000.

[18] I. Satoh, "MobiDoc: A Framework for Building Mobile Compound Documents from Hierarchical Mobile Agents," *Proc. Symp. Agent Systems and Applications/Symp. Mobile Agents (ASA/MA '00)*, 2000.

[19] I. Satoh, "Flying Emulator: Rapid Building and Testing of Networked Applications for Mobile Computers," *Proc. Conf. Mobile Agents (MA '01)*, Dec. 2001.

[20] I. Satoh, "Physical Mobility and Logical Mobility in Ubiquitous Computing Environments," *Proc. Conf. Mobile Agents (MA '02)*, Oct. 2002.

[21] I. Satoh, "Configurable Network Processing for Mobile Agents on the Internet," *Cluster Computing: A J. Computer Software and Comm.*, vol. 7, no. 1, 2001.

Ichiro Satoh received the BE, ME, and PhD 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, he was an associate professor in the same department. Since 2001, he has been an associate professor at the National Institute of Informatics, Japan. His current research interests include distributed and mobile computing. He received Information Processing Society of Japan (IPSJ) paper award, IPSJ Yamashita SIG research award, and Japan Society for Software Science and Technology (JSSST) Takahashi research award. He is a member of six learned societies, including the ACM and the IEEE Computer Society.