Testing Mobile Wireless Applications

Ichiro Satoh

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

Abstract

A framework is described that can be used to build and test application-level software for wireless mobile computing. It emulates the physical mobility of wireless devices by using the logical mobility of software-based emulators of the devices and target software. Since each emulator is implemented as a mobile agent, it can dynamically carry the target software to each of the sub-networks to which its device is connected on behalf of the device, permitting the software to interact with other servers in the current sub-network. That is, it can test software designed to run on a wireless device in the same way as if the software were disconnected from the network, moved with the device, and reconnected to and operated on another network. Also described are the lessons learned from exploiting the framework in developing typical software for wireless devices.

1 Introduction

The development of software for portable computing devices is very difficult due to the limited computational resources these devices have, even if they are standalone. Recent advances in networking technology have enabled portable computing devices to link up with servers through wireless networks, such as IEEE802.11b and Bluetooth, to access information from them, and to delegate heavy tasks to them. A typical IEEE802.11b wireless LAN consists of more than one base station (i.e. access point), whose typical radio area is within at most a few hundred meters, connected through a local area network. When a user moves from location to location, his/her mobile computing device may be disconnected from the current network and reconnected to another network. Several researchers has explored mechanisms to transparently mask variations in mobility at the network or system level, such as Mobile-IP [14] and host mobility [19]. These approaches have been designed to enable packet delivery while a mobile device is away from its home base.

However, the notion of mobility transparency is not always suitable for wireless applications, such as user navigation systems in a city or museum and printing services. Such applications need to access local servers on a local area network in the current

location. That is, a change in the network and location implies movement away from the servers currently in use, toward new ones. Therefore, software development for mobile computing devices that use short range wireless networks is often tedious and extremely susceptible to change. To construct a correct application, it must be tested in all the networks to which the device could be moved and hooked up to. Unfortunately, the task of testing software for mobile computing devices has attracted little attention so far. This is a serious impediment to its growth beyond mere laboratory prototypes.

To overcome this problem, a software testing approach suitable to wireless devices, including PDAs and wireless appliances, is needed. We introduced a framework, called Flying Emulator, for developing software running on portable computing devices in our earlier paper [17]. The key idea of the framework is to offer a mobile agent-based emulator of a mobile computing 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 inside the environments of those networks. However, the framework presented in the previous paper had no mechanism for simulating wireless networks or the small form factor displays and controls of mobile wireless devices such as PDAs. Therefore, it is not always suitable for testing applications for small wireless devices, in particular PDAs and wireless appliances. The goal of this paper is to enhance the existing framework to solve these problems. The enhanced framework can have a mechanism to simulate characteristics of wireless networks by using Java bytecode rewriting technique. That is, it can automatically replace Java classes for network processing in applications by customized classes, which enable the developer to control features of wireless networks such as network disconnection. It also provides a graphical front end for target devices so that it is easy enough for end-users of wireless devices to use in testing applications and evaluating contents.

The remainder of this paper is organized as follows. Section 2 discusses requirements in testing wireless applications and then outlines a framework for testing wireless applications. Section 3 presents the design and implementation of the framework and Section 4 demonstrates the usability of the framework through two real-world examples. Section 5 surveys related work and Section 6 provides a summary and discusses some future issues.

2 Approach

The framework presented in this paper aims testing network-dependent application-level Java-based software, including application-level protocols designed to run on mobile devices, such as PDAs and notebook PCs, and which may often access servers on local networks in the device's current location either through short-range wireless networks such as the IEEE802.11b or Bluetooth networks. This framework does not address mobile phones because wireless networks for mobile phones provide a global view in the sense that they offer continuous access to services and resources through a land-based network, even when the device's location changes. Emulation of the performance of wireless networks, such as bandwidth and connectivity latency, is also

beyond the scope of the framework.

2.1 Requirements

To test applications designed to run on wireless devices, the framework should satisfy the following requirements.

Network-dependency and interoperability: Cooperation among mobile computing devices and servers within a domestic or office network is indispensable because it complements various features missing in the device. As a result, the appropriateness of the software running on the device not only depends on its internal execution environment but also on the external environments, including servers, provided by the network that it connects to. Moreover, testing the interoperability of various devices tends to be tedious, since there are countless varieties of devices with which the target device can interface.

Mobility and disconnection: Wireless access points, so-called *hotspots*, are being installed in various places, such as airports, hotels, and cafes. While a wireless device roams among the radio cells of the base stations within a hotspot, it can continue to access servers provided within the hotspot as well as global networks such as the Internet. On the other hand, when a device moves outside of the area of the current hotspot, it is disconnected and cannot always enter the area of another hotspot, because the areas of hotspots are still sparse. Such devices must often sleep to save battery life and avoid the risk of accidental damage while moving.

Ease of use: The framework should be simple enough for end-users of wireless devices to use in testing. It must be able to run on servers without any custom hardware and enable easy operation of applications using a graphical user interface displayed on a stationary computer in front of the developer. Many applications have their own GUIs. The framework should enable the developer to test his/her target applications including their GUIs.

Spontaneous and plug-and-play management: When a wireless mobile device is reconnected, it may have to detect servers on the network to perform its task. To achieve this, several middleware systems, such as Jini [2] and Universal Plug and Play (UPnP) [12], have been used to manage devices. These mechanisms use multicast communications whose packets can be transmitted to only the hosts within specified sub-networks. Therefore, the software target to run on a wireless device must be tested inside the sub-networks to which the device can be connected.

2.2 Overview of the Framework

It is difficult to build and debug software designed wireless mobile devices within the devices themselves, because it has a less powerful processor with less memory and a limited user interface with a clamped keyboard and small screen. A popular solution is

to use a software-based emulator for the target device. However, existing emulators are not always usable for the development of software dependent on resources provided in networks, because an emulator running on a standalone computer cannot simulate all the resources provided in the networks to which its target device may connect. The simplest 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) and to attach it to the local networks in the current location. This is of course troublesome for the developer and consequently should only be resorted to in the final phase of software development.

Our framework aims to solve these problems through the use of a software-based emulator that can simulate the internal execution environment of its target device, like the approaches taken in previous works. The key idea of the framework is to emulate the physical mobility of a wireless device between networks by using the software's logical mobility, which has been designed to run on the device between the networks, as shown in Figure 1. The framework constructs a software-based emulator as a mobile agent, which can travel from host to host under its own control. When a mobile agent-based emulator and its target software moves among networks, it transfers the code and state of the software to the destination network. The carried applications can access servers provided in the destination and continue their processes as if they had been physically moved with the target device.

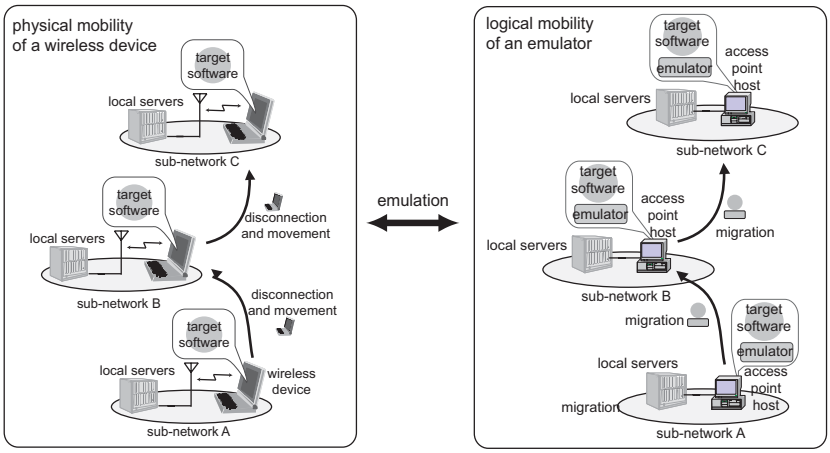

Figure 1: Correlation between physical and logical mobilities

Each mobile agent is simply a logical entity that must thus be executed on a computer. Therefore, we assume that each network to which the device may be moved and attached has more than one special stationary host, called an access-point host that offers a runtime system for mobile agents. Each access-point host has a runtime environment that enables applications running in a visiting emulator to connect to local servers in its network.

3 Design and Implementation

The current implementation of this framework is based on a Java-based mobile agent system called MobileSpaces[15].[1] As Figure 2 shows, the framework has the following three components:

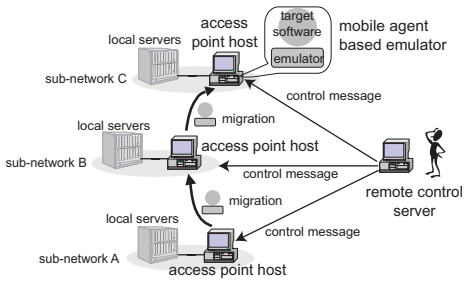

Figure 2: Architecture of the framework.

- A *mobile agent-based emulator* that can carry the target software to specified access-point hosts on remote networks on behalf of a target wireless device.
- *Access-point hosts* that are allocated to each network and enable the software carried by an emulator to connect with various servers running on the network.
- A *remote-control server* that is a front-end to the whole system, enabling the moving emulator and its target software to be monitored and operated by remotely displaying their GUIs on its screen.

In addition, we provided a runtime system that runs on a wearable computer and supports the execution of the tested software. 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.

3.1 Mobile Agent-based Emulator

Our mobile agent-based emulator can carry and test software designed to run on its target wireless device. Figure 3 shows the structure of a mobile agent-based emulator running on an access-point host.

Emulation of Mobility:

The developer can interactively control the movement of the emulator through the graphical user interface displayed the remote-control server as Figure 4 shows. Also,

[1]The framework itself is independent of the MobileSpaces mobile agent system and can thus work with other Java-based mobile agent systems.

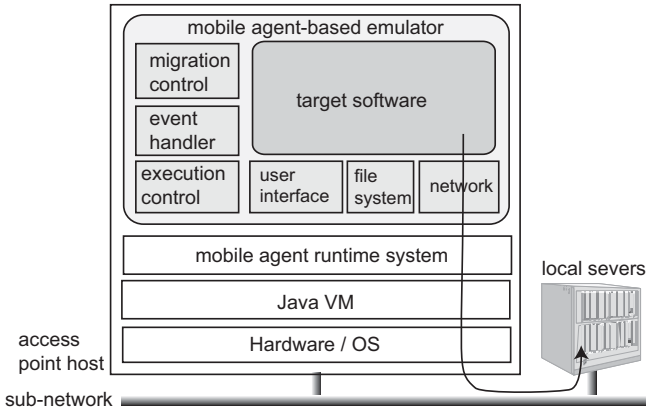

Figure 3: Mobile agent-based emulator running on an access-point host.

each emulator can have its own itinerary, a list of hosts corresponding to the physical movement pattern of its target wireless device.

When a wireless device moves in physical space, it may still be running. However, our emulator cannot migrate over networks when 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, we designed our framework to divide the life-cycle state of each application into the following three phases: networked running, isolated running, and suspended. In the networked running state, the software is running in its emulator on an access-point host and is allowed to link to with servers on the network. In the isolated running state, the software is still running but is prohibited from communicating with any servers or devices on the network. This means that the device is disconnected from the network. In the suspended state, the emulator stops its target software and maintains the execution states, such as program variables, for the software by marshaling itself into a bit stream along with the states and codes of its target software.

When an emulator is suspended or migrated over networks, it can marshal itself into a bit stream along with the heap blocks and codes of its target software since it is implemented as a mobile agent. The emulator also dispatches certain events to its target software to explicitly restart (or stop) its activities and acquire (or release) the computational resources of the current host when the life-cycle state of the software is changed. In addition, our framework can provide each mobile wireless device with lightweight middleware to monitor the environment of the device, such as network connectivity and location, and dispatch certain events to its target as a mobile agent-based emulator corresponding to the device.

Emulation of Wireless Networking

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

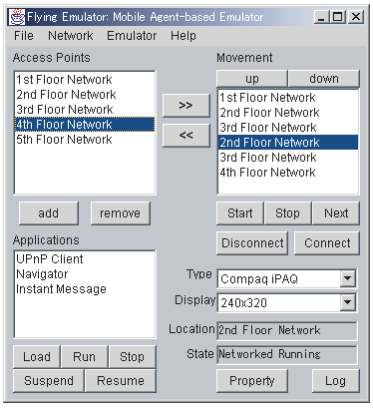

Figure 4: User interface for controlling mobile agent-based emulators.

current implementation, a moving emulator cannot have its own network identifier, such as an IP address and port number, but this is not a serious problem because our target software is a client-side program as mentioned previously. Applications running on an emulator can interact with other applications running on different emulators and servers on the current sub-network and the Internet if the sub-network is connected to the Internet.

The current implementation simply maps the wireless device TCP/IP stack onto the desktop TCP/IP stack to simulate IP connectivity. Some issues in wireless device use can be simulated as well as to the extent that the medium of a desktop computing environment allows; but there may be other issues, like network disconnection, latency, and bandwidth. This framework has a mechanism to simulate characteristics of wireless networks. The mechanism overrides Java's classes for network operation, such as `java.net.Socket` and `java.net.ServerSocket`, with customized classes that emulate that characteristics of wireless networks by using a bytecode rewriting technique. Our bytecode rewriting tool is based on Byte Code Engineering Library (BCEL) [5], which enables bytecode manipulations of Java classes and is also entirely written in Java and does not have to extend the Java virtual machine. Our mechanism detects certain classes in target applications and then transforms them into the corresponding customized classes when the original classes are loaded at runtime. The current implementation of this framework provides customized TCP socket classes that are subclasses of `java.net.Socket` and `java.net.ServerSocket` and can be explicitly disconnected and reconnected by the remote control server as shown in Figure 5. Moreover, the developer can easily define customized classes to specify other characteristics of wireless networks.

Emulation of Computing Environment

The framework assumes that its target software will be Java application programs. Accordingly, the Java virtual machine can actually shield the target software from many

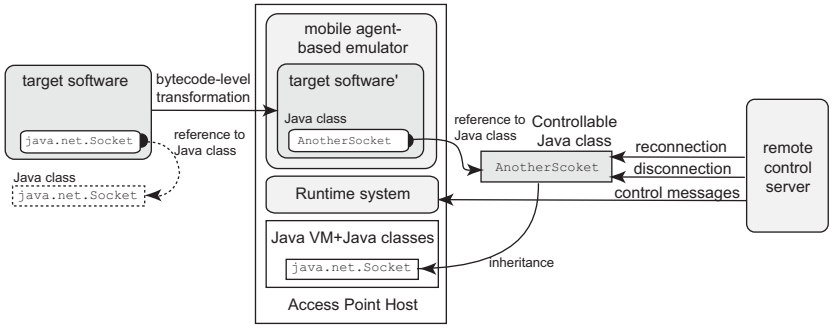

Figure 5: Bytecode transformation for customizing a class for network operation.

features of the hardware and the operating system of mobile wireless devices. Each emulator permits its target software to have access to the standard classes commonly supported by the Java virtual machine as long as the target device offers them. In addition, the current implementation of our emulator supports several typical resources of mobile wireless devices. Each emulator maintains 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; it allows a user to insert files into it through its GUI. Each emulator can permit its target software to be Java's communication APIs (Java COMM) if they are provided on the device on which the emulator runs. Furthermore, the framework offers a mechanism that enables its target software to access equipment running on remote computers via serial ports. The mechanism consists of proxies whose interfaces are compatible 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.

Emulation of User Interface

The user interfaces of most handheld computers are limited by their screen size, color, and resolution, and they may not be equipped with traditional input devices such as a keyboard or mouse. Each emulator can explicitly constrain the size of the user interface being used from its inner applications by using a set of classes for the visible content of the MobileSpaces system, called MobiDoc [16]. Also, it can have images of the physical user interface of the target device as it would appear to the end-user. Typical handheld devices will include a screen that may allow content to be displayed. Therefore, the screen is seamlessly embedded into the pictures of the device, and the basic controls of the device can be simulated through mouse-clickable buttons.

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 a keyboard and mouse. This mechanism is constructed on the Remote Abstract Window

Toolkit (RAWT) developed by IBM [8]. 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 windows of applications in a visiting emulator to be displayed on the screen of the remote control server and operated using the keyboard and mouse of the server. 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 and 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. Figure 6 (A) shows a screenshot of the remote control server and (B) is a picture of a tablet PC running a user navigator application. The left window in Figure 6 (A) is the window of a mobile agent-based emulator of the tablet-PC, where the emulator tests the application.

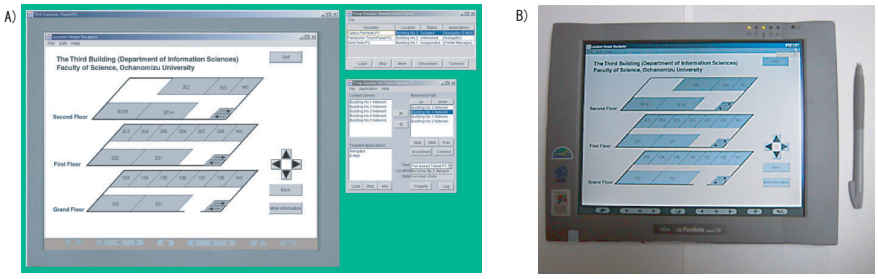

Figure 6: (A) Screenshot of remote control server and (B) Picture of a tablet PC

3.2 Access-point Host

We assume that more than one access-point host is 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 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 a firewall as HTTP POST requests, and responses can be retrieved as HTTP responses.

3.3 Remote-control Server

The remote-control server is a control entity responsible for managing the whole system. It can run on a standard workstation that supports Java and does need any custom hardware. 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, enabling the developer to freely instruct moving emulators to migrate to other locations and terminate, through its own graphical user interface. It also can monitor and record the status of all access-point hosts by periodically multicasting query messages to them.

4 Application

To demonstrate the utility of our framework, we used it to test two typical mobile wireless applications.

4.1 Testing of Network-dependent Software

Portable computing devices have been used in several other projects to provide user navigation in a city [1, 4]. Similarly, we developed a prototype navigation system for a building of the National Institute of Informatics using PDAs with IEEE 802.11b wireless LAN connectivity. Each floor in the building has its own local area networks and one or more wireless LAN base stations. The system provides each visitor with a PDA that obtains location-dependent information from servers allocated on the sub-network of the current location via IEEE802.11b wireless networks. As a visitor moves from floor to floor, the PDA automatically displays a map of the current floor. To test the system, we constructed a mobile agent-based emulator for the PDA. The emulator can migrate a map viewer application designed to run on the PDA to the sub-network of another floor and enable the application to access the local database of the floor and display suitable maps.

Figure 7 (A) shows the window of the map viewer application tested in the emulator forwarded from an access point host to the remote control server by using the RAWT toolkit and (B) is a picture of the target PDA (Compaq iPAQ) running the map viewer application. As illustrated in Figure 7 (A) and (B), both the application running on the emulator and the application running on the target device can present the same navigation information. That is, the tested application can run in the target device in the same way as it was executed in the emulator. Furthermore, this example shows that the framework can provide a powerful method for testing not only application software for portable computers but also for creating location-dependent contents, such as map and annotations about locations. Moreover, by using the RAWT toolkit, this framework enables a content creator to view the location-dependent information that should be displayed on the PDA on the screen of his/her stationary computer. Also, since the emulator can define its own itinerary among multiple access points, it can easily and exactly trace the mobility of each visitor and test the contents displayed on the screen

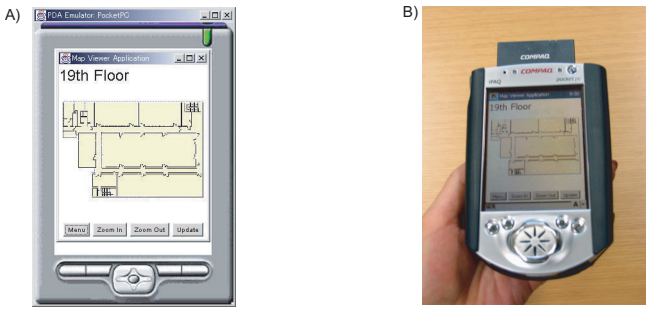

Figure 7: (A) the window of a mobile agent-based emulator with a map viewer application running on an access point host displayed on the screen of remote control server and (B) the same application running on a PDA.

of his/her PDA.

4.2 Testing of Multicast-based Protocol

Since wireless devices move from network and network, they need to be managed in an ad-hoc manner. Universal Plug and Play (UPnP) [12] is a powerful infrastructure for enabling a device to dynamically join a network, obtain an IP address, convey its capabilities upon request, and learn about the presence and capabilities of other devices. Using it, we easily tested the interoperability of UPnP-aware Java application software designed to run wireless devices and a subset implementation of the UPnP protocol in our previous project [13]. UPnP uses a multicast-based management protocol, called simple service discovery protocol (SSDP), with which a device can announce its presence to others as well as to discover other devices or services. For example, a joining device multicasts messages to advertise its services to the UPnP's control points. Since such multicast messages are available within the domain of specified sub-networks, UPnP-aware software designed to run on a device must operate within the domain to receive the messages. Therefore, we constructed a mobile agent-based emulator as a carrier for the software. When the emulator arrives at an access-point host within the domain, the software it carries can multicast advertisement-messages to hosts in the domain and can receive search-messages multicasted from other devices in the domain as if the emulator's target were joined to the domain, as shown Figure 8. We demonstrated that the software in an emulator running on a host can interact with software in another emulator running on a different host as well as on the same host through UPnP protocols. In addition, software tested successfully in the emulator could still be run in the same way on the device without modifying or recompiling it. Our framework can thus provide a powerful methodology for testing the interoperability of protocols, within limited specified sub-networks, for reasons of security and reduced network traffic.

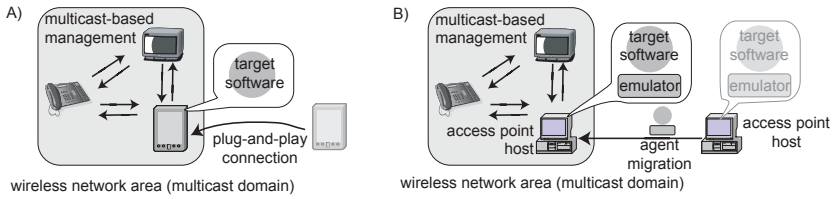

Figure 8: Emulation of (A) the plug-and-play operation of a mobile device by (B) the migration of the emulator for the device between access-point hosts.

5 Related Work

As mentioned above, software-based emulators of portable computing devices facilitate development and testing of standalone software running on the devices, but most existing emulators do not support wireless devices in the sense that they cannot simulate the whole network in which the target device will interact. An extreme solution is to actually carry around portable devices and attach them to local networks at the destinations, but this would be extremely troublesome for developers and content creators. Another solution is to let the target software run on a local workstation and link to remote devices and servers through networks, e.g., the InfoPad project at Berkeley [11] and Lancaster University's network emulator [6]. 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 mechanisms. 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 [12], are often used to manage devices. These systems use multicast communications to find their management servers and devices, but the multicast messages can only be transmitted to hosts within specified sub-networks. Therefore, the software that is to run on ubiquitous computing devices must be tested within the sub-networks to which the devices may be connected.

There have been various attempts to apply mobile agent technology [3, 7, 10], including the mobile code approach, to wireless mobile computing, but their goals are to handle network disconnection using the mobility of agents, instead of any testing of software for mobile computing. Our previous framework presented in [17] lacks a mechanism for simulating the user interface of wireless devices and the characteristics of wireless networks. It also does not support multicast networking, so that it cannot be used to test wireless appliances managed in a plug-and-play manner.

6 Conclusion

We have described a framework for building and testing application-level Java-based software designed to run on wireless devices. The framework provides software-based emulators for target software by incorporating a Java virtual machine. Since these emulators are constructed as mobile agents, they can carry software on behalf of the target device to networks that the device may be moved and connected to. That is, testing software is provided with the services and resources provided through its current network as if the software were being executed on the target device when attached to the network. Software successfully tested in the emulator can be run in the same way on the target device without being modified or recompiled. Moreover, the framework allows emulators to easily simulate other characteristics of wireless networks by using a runtime bytecode rewriting technique, Our early experience indicated that we can greatly reduce the time required to develop software for wireless devices using the framework.

There are, however, further issues that need to be resolved. Security is one of the most serious concerns in mobile agent technology. However, since our framework is used in the development phase instead of the operation phases, this issue is not as serious as it is in other mobile agent-based applications. Nevertheless, we plan to devise schemes to guarantee security and to control access, since the current implementation relies on the JDK 1.1 security manager. Since our framework can complement existing software-development methods for wireless computing discussed in Section 5. Therefore, we are interested in developing a tool to integrate our approach with other methods. The location-aware mobile agent infrastructure we developed incorporates RF-based and infrared-based tag sensors [18] and the framework we propose should be able to support these sensors.

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] G. Cabri, L. Leonardi, F Zambonelli, “Engineering Mobile Agent Applications via Context-Dependent Coordination”, pp.1039-1055, IEEE Transaction of Software Engineering, Vol. 28, No. 11, November 2002.

[4] 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.

[5] M. Dahm, “Byte Code Engineering Library”, http://jakarta.apache.org/bcel/index.html

[6] 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.

[7] A. Fuggetta, G. P. Picco, and G. Vigna, “Understanding Code Mobility”, IEEE Transactions on Software Engineering, 24(5), 1998.

[8] International Business Machines Corporation, “Remote Abstract Window Toolkit for Java”, http://www.alphaworks.ibm.com/, 1998.

[9] J. Jing, “Client-Server Computing in Mobile Environments”, ACM Computing Survey.

[10] B. D. Lange and M. Oshima, “Programming and Deploying Java Mobile Agents with Aglets”, Addison-Wesley, 1998.

[11] M. Le, F. Burghardt, and J. Rabaey, “Software Architecture of the Infopad System”, Workshop on Mobile and Wireless Information Systems. 1994.

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

[13] T. Nakajima, I. Satoh, and H. Aizu, “A Virtual Overlay Network for Integrating Home Appliances”, Proceedings of International Symposium on Applications and the Internet (SAINT’2002), pp.246-253, IEEE Computer Society, January, 2002.

[14] C. Perkins, “IP Mobility Support”, Internet Request For Comments RFC 2002, 1996.

[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, “Flying Emulator: Rapid Building and Testing of Networked Applications for Mobile Computers”, Proceedings of Conference on Mobile Agents (MA’2001), LNCS, Vol.2240, pp.103-118, Springer, December, 2001.

[18] I. Satoh, “SpatialAgents: Integrating of User Mobility and Program Mobility in Ubiquitous Computing Environments”, to appear in Wireless Communications and Mobile Computing (Accepted), Vol. 3, John Wiley, 2003.

[19] A. C. Snoeren and H. Balakrishnan, “An End-to-End Approach to Host Mobility”, Proceeding of Conference on Mobile Computing and Networking (MobiCom’02), pp.155-166, ACM Press, 2002.

Appendix: Application Program

As mentioned previously, 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 be easily modified to such components by implementing the following listener interface.

```
1:  interface ApplicationListener
2:    created() // invoked after creation
3:    terminating() // invoked before termination
4:    networked() // invoked after network enabled
5:    isolated() // invoked after network disconnected
6:    suspending() // invoked before suspension
7:    resumed() // invoked after resumption
8:  }
```

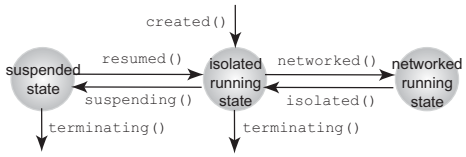

Figure 9: Callback method invocations in the life-cycle state-transition.

The above interface specifies callback methods invoked by the emulator and the runtime system on the target device when the life-cycle states of an application, such as

networked running state, isolated running state, and suspended state as shown in Figure 9. Each application must define proper processes in each of these methods to hook and handle such changes. For example, suppose that a mobile agent-based emulator is just about to migrate from its current host to another host. As shown in Figure 10, an application contained in the emulator is notified through the following process:

1. The `isolated()` method of the application is invoked to handle the disconnection from the network, and then the application must release resources, such as sockets and rmi's 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, for example, 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 servers on the current networks.

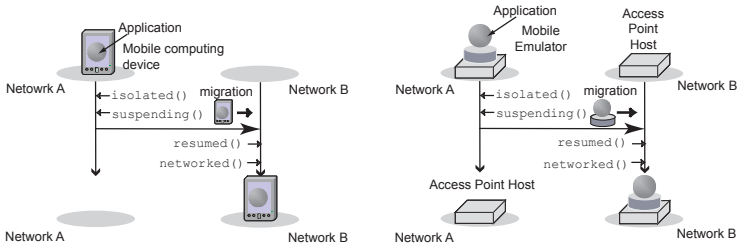

Figure 10: The movement of a computing device and the migration of the corresponding mobile agent-based emulator.