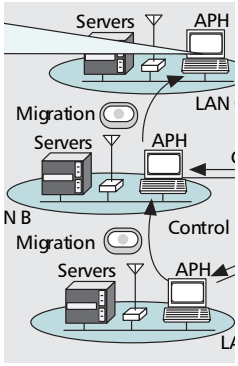
SOFTWARE TESTING FOR WIRELESS MOBILE COMPUTING

ICHIRO SATOH, NATIONAL INSTITUTE OF INFORMATICS



To construct correct software to run in mobile terminals for 4G wireless networks and wireless LANs, it must be tested in all the networks to which the terminal could be moved and connected to. This article presents a new approach, called Flying Emulator, to test software designed to run on mobile terminals.

ABSTRACT

4G wireless networks make it increasingly difficult to develop and test application software for mobile terminals in comparison with 3G or earlier generations. These 4G networks will incorporate wireless LAN technologies, and mobile terminals can access the services provided by LANs as well as global network services. Therefore, software running on mobile terminals may depend on not only its application logic but also on services within the LANs to which the terminals are connected. To construct correct software to run in mobile terminals for 4G wireless networks and wireless LANs, it must be tested in all the networks to which the terminal could be moved and be connected. This article presents a new approach, called Flying Emulator, to testing software designed to run on mobile terminals. Like existing approaches, the approach provides software-based emulators of its mobile terminals for software designed to run the terminals. It is unique because it constructs emulators as mobile agents that can travel between computers. These emulators can carry the target software to the networks to which the terminals are connected and allow it to access services provided by the networks in the same way as if it was moved with and executed on the terminals connected to the networks. This article describes the idea of the approach, its implementation, and our experience with a typical application.

INTRODUCTION

The development and testing of software for fourth generation (4G)-enabled mobile terminals may become more difficult than for third generation (3G) or earlier generations. 4G wireless networks will incorporate wireless LAN (WLAN) technologies in addition to cellular networks and satellite-based network technologies. Then WLANs will not only provide high-speed wireless connectivity for mobile terminals, including smart mobile phones, notebook PCs, tablet PCs, and PDAs, but will also see a shift in the nature of mobile terminal applications. At present, the goal of most WLANs is still to connect mobile terminals to wide area networks, such as the Internet. However, WLANs can have servers within WLANs. In fact, several recent commercial and noncommercial WLAN hotspots have offered printing services and database services about resident information (e.g., restaurants and maps) that are available only within their coverage areas. As WLANs and 4G wireless networks expand and develop, such location- or network-dependent services will become more commonplace.

However, it is very difficult to develop and test software designed to run on mobile terminals for 4G mobile networks and WLANs. To motivate the development of mobile terminal software for 4G wireless networks, it is helpful to have a single example scenario in WLANs. For example, a user will visit an office building and have a terminal equipped with a short-range wireless link (e.g., IEEE802.11x or Bluetooth). Each floor in the building has a subnetwork with WLAN access points and can provide different resources, such as database servers, for location-dependent navigation information on each floor. As a user moves from floor to floor with his or her terminal, some new servers will become available from software running on the terminal, or it may no longer be able to access previous servers. As a result, such software depends not only its aonpplication logic but also on the services available on the current subnetworks. To construct correct software, the developer must test the software using the services available on each of the subnetworks to which the terminal might be connected. This is troublesome for the developer because nobody wants to run up and down stairs, physically carrying a terminal to test whether it can successfully access contents from appropriate databases in the current subnetworks and properly display the contents on the terminal. This is a serious obstacle to the growth of 4G mobile networks that incorporate WLANs. Nevertheless, the task of testing network- or location-dependent software has attracted little attention.

To overcome this problem, we need a software testing approach suitable for 4G-enabled mobile terminals. Indeed, have we introduced an approach, called Flying Emulator, to developing software that runs on smart mobile terminals such as notebook PCs, tablet PCs, and PDAs.[1]

[1] *The reader can find detailed information about this approach in our previous paper [1] .*

1536-1284/04/$20.00 © 2004 IEEE

The key idea of the approach is to provide a mobile-agent-based emulator of a mobile terminal, where a mobile agent is autonomous software that can travel from computer to computer. Like other software-based emulators, the emulator performs application-transparent emulations of its target terminal for application software. Since it is implemented as a mobile agent, it can also carry its target software to LANs to which its target device may connect and test the software inside the environments of LANs. That is, it emulates the physical mobility of terminals by using its own logical mobility over LANs.

Background

The approach aims at testing network-dependent application software designed to run on mobile terminals that access services available on WLANs. Emulating the performance of wireless networks (i.e., bandwidth and connectivity latency) cannot be addressed.

Requirements

To test software for mobile terminals using WLANs and 4G wireless networks, our approach should satisfy the following requirements.

Mobility and Network Dependency — Cooperation among mobile terminals and servers within a domestic or office network is indispensable because such cooperation will offset various features missing in the terminals. As terminals move, they may be disconnected from the current network and then reconnected to another network. A change in network and location may indicate movement away from the servers currently in use toward new ones. A lot of work has been proposed to transparently mask variations in mobility. For example, mobile IP technology may be useful for maintaining a terminal's connections to services in the source networks even if the terminal moves across LANs. On the other hand, terminals are often required to link up local services provided by the current LANs to access information available at the current locations as well as remote servers. As a result, application-level software running on moving terminals must treat the differences between the services available in the source networks and those available in the destination networks.

Spontaneous and Plug-and-Play Management — Mobile terminals are often managed by using service discovery mechanisms that use multicast communications to find their servers and terminals to be managed, such as Universal Plug and Play (UPnP) [2] and Sun's Jini [3]. These multicast messages can only be transmitted to the hosts within specified subnetworks. Therefore, to test the target software designed to run on the terminal, we must execute and test it inside a subnetwork to which the terminal can be connected.

Ease of Use — The approach needs to avoid the movement of the developer in the testing of software. It also should be simple enough for end users to use in testing. It must be able to run on servers without requiring custom hardware. Application software often has its own graphical user interface (GUI), so the approach should enable the developer to test his or her target software, including its GUI. All applications successfully tested in the emulator should be performed in the same way without being modified or recompiled.

Supports to Content Creators/Designers — The creation of location- or network-dependent contents, (e.g., museum guides on portable terminals) is as difficult as the development of location- or network-dependent software, because the creators must test their content, which is designed for being displayed on terminals, whether or not the content is valid at the locations to which the terminals move. The approach should display and operate visual content for terminals.

Existing Approaches

There have been several approaches to testing software that will run on mobile terminals. We can classify these approaches into four types, as follows.

The first approach is to carry the target terminal to the LANs to which the terminal may move and connect, and test the target software running on the terminal with the servers. However, compared to desktop/notebook computers, typical mobile terminals have only limited computational resources (e.g., restricted levels of processing power and memory capacities) and poor user interface devices (e.g., clamped keyboards and small screens). Therefore, it is difficult to debug and monitor software within the terminal itself. This should only be resorted to in the final phases of software development. In other approaches software-based emulators are useful on behalf of the target terminals to solve this problem. In fact, many commercial mobile terminals, such as palm-sized PDAs and smart mobile phones, provide software-based emulators that can run on workstations and simulate terminal execution environments. However, most existing emulators simulate only the internal execution environments of their target terminals. On the other hand, the correctness of the software running on the terminal depends on its internal execution environment and also the external environments provided by the network to which it connects.

The second approach is for the developer to physically carry the workstation that runs an emulator corresponding to the target terminal and connect it to LANs to which the terminal might move. It can also inherit the advantages from software-based emulators, including fast turnaround time, source-level debugging, and fast execution. Like the first approach. its target software is executed within the LANs, so it can directly access the servers provided by the LANs. However, this is difficult to actually carry the workstation to another location and connect it to the LANs in that location, even when the workstation is a portable notebook PC.

The third approach is to simulate the terminal's external environments inside local workstations. For example, Nokia provides a network server interface simulator, called Nokia Mobile

Compared to desktop/notebook computers, typical mobile terminals have only limited computational resources. Therefore, it is difficult to debug and monitor software within the terminal itself.

When an emulator migrates between APHs, its target software is transformed into a bit-stream along with the states and codes of the target software with the emulator. Then the software and emulator are transferred to the destination APH.

Server Services, in addition to terminal emulators for their terminals. The simulator enables messages to be sent from server-side applications to a terminal emulator, and forwards messages from the terminal emulator to server-side applications. Also, several integrated development environments (IDEs), such as Microsoft Visual Studio and Eclipse, can be opened to provide tiny database servers and directory servers inside local workstations. However, it is almost impossible to exactly simulate all the services and content available in each of the actual LANs to which the target terminal may connect to inside local workstations.

The fourth approach enables software designed for mobile terminals to run in software-based emulators on local workstations and remotely access actual servers, instead of pseudo ones, through networks (e.g., the InfoPad project at Berkeley [4] and Lancaster University's network emulator [5]). However, accomplishing this in a responsive and reliable manner is difficult, and the emulators may not be capable of remotely accessing all the services within the LANs because of security protection. Moreover, the approach is not suitable for testing software using multicast communications, including service discovery mechanisms such as Jini and UPnP, and some ad hoc networking protocols, because the reachable domains are limited within specified subnetworks to reduce network traffic. It is almost impossible to forward a large quantity of multicast packets, which will spill over into subnetworks, to the target software running on an emulator in other networks via the Internet.

The Flying Emulator Approach

The existing approaches described above have their own strengths and weaknesses. This article proposes an innovative approach to testing network-dependent software on mobile terminals. The goal of this approach is not to compete with the existing approaches but to complement them and solve some of their problems. Like the three other approaches, our approach provides software-based emulators for terminals. The key idea of this approach is to implement emulators as mobile agents that can travel from computer to computer under their own control (e.g., [6, 7]). When an agent moves to another location, the agent transfers its state, as well as its code, to the destination. After arriving at the destination, it can still continue its execution. Therefore, our mobile agent emulators can carry the code and the state of their target software to the destinations, so the carried software can basically continue its processing after arriving at the new host in the same way as if it had been physically moved with the target terminal.

This approach consists of the following three components:

- *A mobile-agent-based emulator* that provides the target software with the internal execution environment of its target terminal, and also carries the software to specified access point hosts on remote networks on behalf of the terminal
- *Access point hosts* (APHs) that are allocated to each network and enable the software carried by the emulator to connect with various services running on the network
- *A remote control server* (RCS) that is the front-end to the whole system and enables the moving emulator and its target software to be monitored and operated, by remotely displaying their GUIs on the screen

As we can see from Fig. 1, the physical movement of a mobile computing computer from one LAN to another is simulated by the logical mobility of a mobile-agent-based emulator with the target software moving from an APH in the source LAN to another APH in the destination LAN. Each emulator permits the target software to access servers and multicast-based services provided in current networks and have its own itinerary that corresponds to the physical movement pattern of its target terminal. Each APH is a server offering a runtime system for the execution and migration of mobile agents, including mobile-agent-based emulators. Additionally, it is lightweight and does not need any custom hardware.

Mobile Terminal Emulation

Each mobile-agent-based emulator can carry and test software designed to run on its target terminal. Figure 2 shows the structure of a mobile-agent-based emulator running on an APH. The current implementation of the approach is built on Java and provides mobile-agent-based emulators for typical notebook PCs, PDAs, tablet PCs, and smart phones. Although the approach itself can support native software, hereafter we describe mobile emulators for testing software written in Java (J2SE, JDK 1.1, Personal Java, or J2ME CDC) to illustrate how to emulate the mobility and connectivity of terminals rather than the terminals' execution environments.

Emulation of Terminal Mobility

When an emulator migrates between APHs, its target software is transformed into a bitstream along with the states and codes of the target software with the emulator. Then the software and emulator are transferred to the destination APH. The destination APH retrieves the software and emulator from the bitstream. The current implementation uses Java's standard JAR file format, which can support digital signatures for authentication, to pass them. The developer can control the movement of the emulator interactively through the RCS. Also, each emulator can have its own itinerary, a list of APHs that corresponds to the physical movement pattern of the target terminal.

Typical mobile terminals assign execution modes, which stationary computers may not have, on their application-level software. Accordingly, we divide the life cycle states of the target software into three phases: networked-running, isolated-running, and suspended. Networked-running mode refers to the target software running and connecting to a LAN in the current location. In this mode the software running in the emulator is allowed to link to servers on the LAN through the current APH. Isolated-running mode means that the terminal is still running but

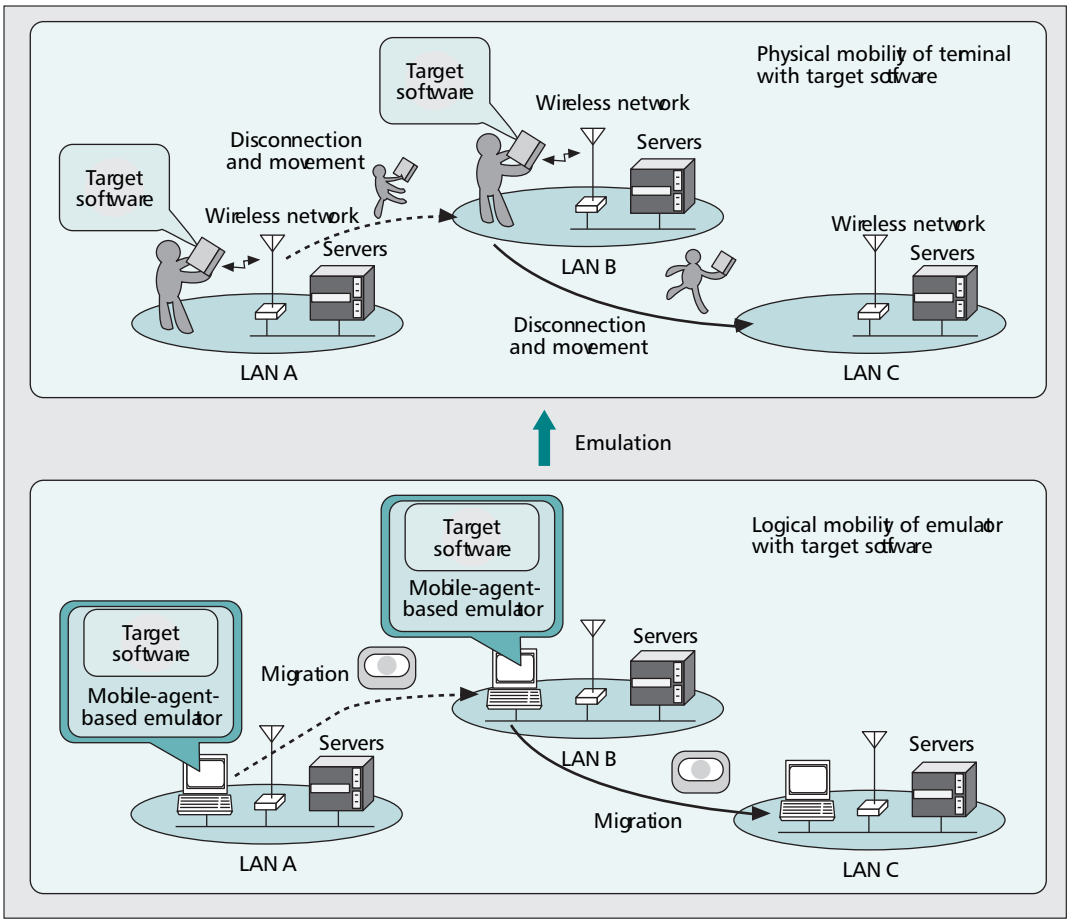


Figure 1. *Emulation of physical mobility through logical mobility.*

Since each mobile agent is a general program, the approach can embed a processor-instruction interpreter, which can execute software for the processor in the same way as if it were being executed by the processor, into a mobile agent.

disconnected from the LAN. In this mode the software still runs in the emulator, but is prohibited from communicating with any servers on the current LAN. Suspended mode corresponds to that of a terminal that is sleeping to save battery life, and avoids the risk of accidental damage while moving. In this mode the emulator stops the target software.

This approach assumes that each terminal can connect to at most one LAN through a wired or wireless network. When a terminal is moved and reconnected to a different LAN, the software running on the moving terminal goes into suspended mode and then enters the networked- or isolated-running mode. The emulator can dispatch certain events to the target software to explicitly restart (or stop) its activities and acquire (or release) the computational resources of the current APH when the life cycle state of the software is changed.

Emulation of Computational Resources

The emulation of the internal execution environments of terminals is basically the same as that of other existing software-based emulators. Since each mobile agent is a general program, the approach can embed a processor-instruction interpreter, which can execute software for the processor in the same way as if it were being executed by the processor, in a mobile agent. If its target software is written in Java, the Java virtual machine can shield the target software from many features of the hardware and operating systems of mobile terminals. As a result, mobile emulators for Java software can be simple. They permit their target software to have access to the standard Java classes commonly supported by the Java virtual machine as long as their target terminals offer the classes.

Each emulator offers its target terminal's typical computational resources (e.g., file storage and I/O ports). To store and access files, the emulator maintains a database as a pair consisting of its file/directory path name pattern and the content inside it. The emulator provides basic primitives for file operation, such as creation, reading, writing, and deletion; it allows developers to insert files into itself through the RCS. Furthermore, the approach 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 application programming interfaces (APIs) and can forward the port's signals between the emulator and the RCS.

Emulation of Networking

The target software running in a mobile-agent-based emulator can interact with servers and other software running on the same or different emulators on the current LAN, and the Internet when the LAN is connected to the Internet. The

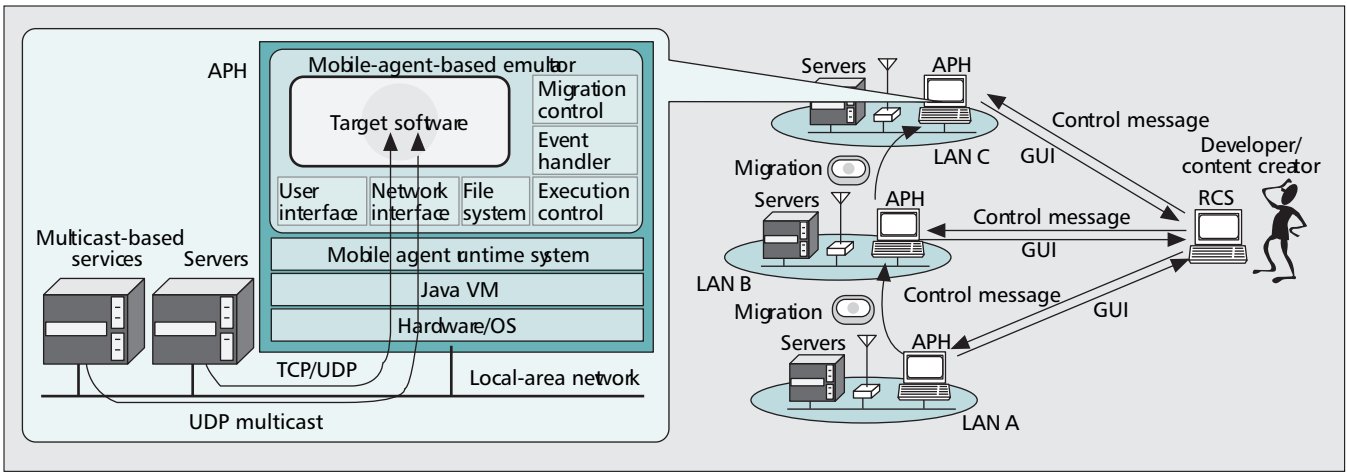


Figure 2. *The mobile-agent-based emulator.*

current implementation simply maps the terminal's TCP/IP stack onto the TCP/IP stack provided in the current APHs to simulate IP connectivity. Mobile-agent-based emulators for Java software inherit most network resources, including the current APH's IP address and TCP/UDP ports, through Java classes for networking, such as `java.net` and `java.rmi` packages. An emulator cannot have its own network identifier, but this is not a serious problem as long as the target software is client-side.

This approach also has a mechanism for simulating many basic characteristics of wireless networks, such as disconnection and reconnection. This mechanism overrides Java's classes for network operations (e.g., `java.net.Socket` and `java.net.ServerSocket`) with customized classes that emulate those characteristics of wireless networks by using bytecode rewriting techniques. Our bytecode rewriting tool is based on Apache Software Foundation's byte code engineering library (BCEL) [8], which enables bytecode manipulations of Java classes, entirely written in Java, and does not have to extend the Java virtual machine. This mechanism detects certain classes in target software and transforms them into the corresponding customized classes when the original classes are loaded. The current implementation of this approach provides customized TCP socket classes and can be explicitly disconnected and reconnected by the RCS.

Emulation of User Interfaces

The displays and keyboards with which most mobile terminals are equipped are limited. Each emulator can explicitly constrain such user interfaces available through the target software by using a set of its customized Java AWT classes. It can also provide pictures of the target terminal's physical user interface as it would appear to the end user. A typical mobile terminal will include a screen that may allow the content to be displayed. Therefore, the screen is seamlessly embedded into the terminal pictures, and the basic controls of the terminal can be simulated through mouse-clickable buttons. We can display the user interface of the target software with the pictures of the terminal on the RCS's screen and operate it from the RCS's standard input devices, such as a keyboard and mouse.

Software Testing

Figure 3a shows the RCS's screen. To save the developer the trouble of testing the target software running in an emulator, our approach allows the developer to sit in front of the RCS. The developer can control the migration of each emulator and the execution mode of its target software through the GUI displayed on the RCS (the right window in Fig. 3a). The RCS can run on standard workstations without any custom hardware. It is responsible for managing the whole system. It can always track the locations of all the emulators, because each APH sends certain messages to the RCS whenever a moving emulator arrives or leaves. It can also monitor the status of all APHs by periodically multicasting query messages to them. Software successfully tested in the emulator could still be run in the same way on the device, without modifying or recompiling it.

Each APH provides a runtime system for executing the mobile-agent-based emulator and migrating it to another APH. The current implementation uses our Java-based mobile agent system, called MobileSpaces [9], but the approach itself is independent of the system and can be easily built on other mobile agent platforms. 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.

The left window of Fig. 3a is the picture of a terminal in which the user interface of the target software tested in an emulator on a remote APH is embedded.[2] To test the user interface of the

[2] *The target software is a map-viewer application mentioned in the following section.*

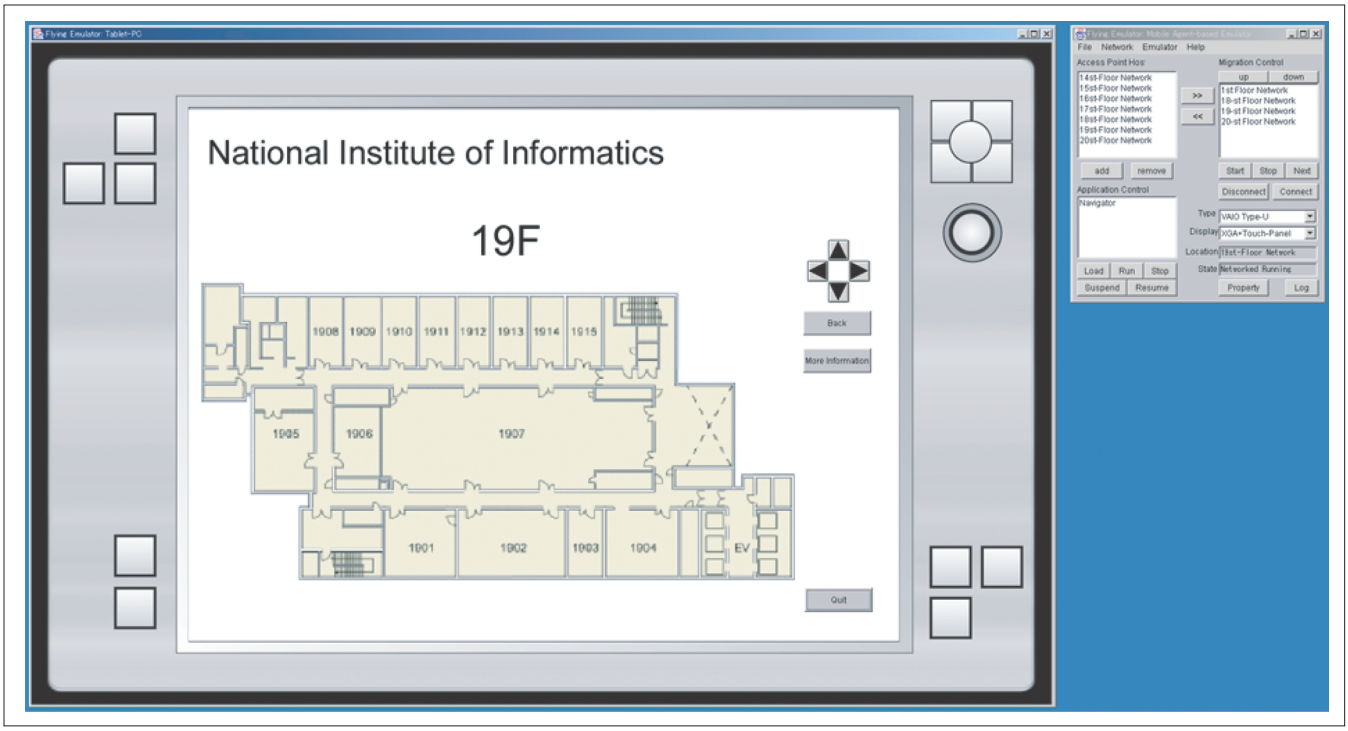


Figure 3. *A screenshot of the RCS when a map viewer application runs in the mobile-agent-based emulator on an APH.*

target software running on remote APHs, we used existing remote desktop systems. The current implementation uses the Remote Abstract Window Toolkit (RAWT) developed by IBM [10].[3] 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 APH, thus enabling the GUIs of application software running in a visiting emulator to be displayed on the RCS's screen, and operated using the RCS's keyboard and mouse. Therefore, the developer can always test his or her target software, including their GUIs, within the RCS, and the APHs do not have to offer user-input and user-output devices, or any graphics services.

Remarks

The current implementation uses Java's object serialization mechanism as a mechanism for transforming the emulator for Java software and the target software into bitstreams that can marshal the heap blocks of a program into bitstreams but not stack frames. Therefore, it is impossible for any active threads to migrate from one virtual machine to another while preserving its execution state. This problem can be solved when APHS supports persistent Java virtual machines (e.g., PJava developed by Sun Microsystems and the University of Glasgow) that preserve all the Java objects' execution states, including threads and stack frames.[4]

Security is essential in mobile agent or code techniques. This problem in our emulators becomes not serious because they are used only in the process of software development. APHs can inherit the security mechanism of the underlying Java virtual machine and mobile agent system to restrict agents so that software running in an emulator can only access specified resources to protect the APHs from incorrect or malicious software.

Experience

In this section we describe some experience with the approach through testing a prototype navigation system, which is a typical application of mobile terminals equipped with conventional wireless LAN interfaces. The system, described in the first section of this article, guides a visitor to the National Institute of Informatics' building. Each floor has its own LANs and one or more WLAN access points. We assume that each visitor has a small tablet PC to access location-dependent information from the database servers provided within the subnetwork of the current floor via IEEE 802.11b. As a visitor moves from floor to floor, the tablet PC leaves the previous subnetwork and joins a new one. A map viewer application running on the terminal automatically accesses location-dependent information (e.g., maps on the current location) from the database and displays the information on the tablet PC's screen. To test the system, we constructed a

[3] *The approach has supported two similar mechanisms, Virtual Network Computing (VNC) and X-Windows. Since RAWT is optimized to Java's GUI, it enables our target applications to be more efficiently displayed on the RCS than with other mechanisms.*

[4] *However, such persistence extensions of Java are still premature for attaining our goal, because they cannot transfer most computational resources and do not often coexist with essential optimization techniques for the Java language.*

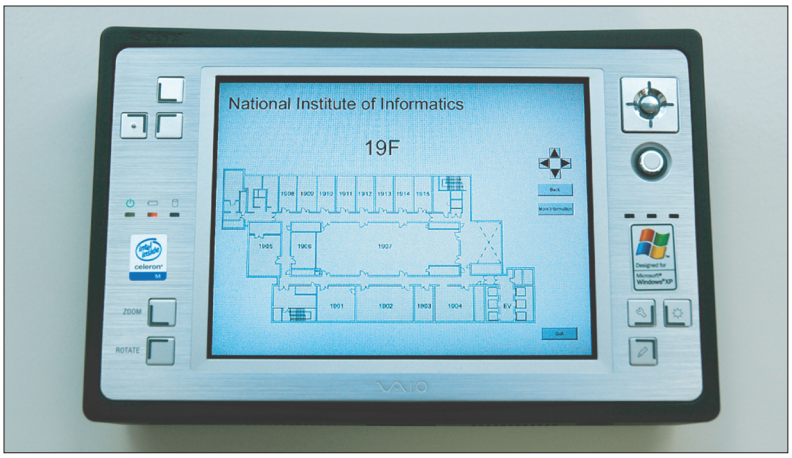


Figure 4. *The map viewer application running on a tablet PC.*

mobile-agent-based emulator that corresponds to the tablet PC. The emulator could migrate a map viewer application to an APH in the subnetwork of another floor, and enable the application to access the local database of the floor via the APH and display maps on the RCS's screen.

Figure 4 shows the target tablet PC (Sony VAIO Type-U with Windows XP) running the target software tested in our emulator as shown in Fig. 3. As illustrated in Figs. 3 and 4, both the application running on the emulator and the one running on the target terminal presented the same navigation information. That is, the tested application ran in the target terminal in the same way as it was executed in the emulator. The system uses a multicast-based service discovery technique for visiting tablet PCs to automatically detect database servers. That is, each server periodically multicasts advertising messages within the domain of its subnetwork to provide its network address to visiting tablet PCs. Since software running in the emulator directly uses the current host's APIs for UDP-based networking, the emulator receives UDP multicasting messages from servers in the domain as if the software were joined to the domain.

Furthermore, this example shows that this approach can provide a powerful method of testing not only application software for mobile terminals but also creating location-dependent contents, such as maps and annotations about locations. It can test server-side systems and contents when client-side terminals access them. Moreover, by using a remote desktop system (IBM's RAWT toolkit), this approach enables the content creator to view the location-dependent information that should be displayed on the tablet PC on the screen of his or her stationary RCS, and operate the software from the RCS even while the viewer application runs in an emulator on remote APHs and accesses servers at its current location. Also, since the emulator can define its own itinerary for multiple access points, it can automatically follow the complex mobility patterns of visitors and test the contents displayed on the screen of the tablet PC. This is useful in the development of track-dependent contents in addition to location-dependent ones.

Conclusion

We have described an approach to testing software designed to run on mobile terminals for 4G wireless networks and wireless LANs. The approach enables application-level software to be executed and tested with the services and resources provided through its current network as if the software were being moved and executed on that target device when attached to the network. Software tested successfully in the emulator can be run in the same way on the target device without being modified or recompiled. Our early experience indicated that by using the approach we can greatly reduce the time and cost required to develop network-dependent software for mobile terminals with WLANs.

Further issues need to be resolved. Our approach will be useful in emulating and testing application-level ad hoc networking, since it can model the deployment of mobility of mobile terminals on different reachable subdomains. The approach does not support terminals that connect to multiple networks and reconnect to new networks while they are running. This limitation does not lessen the utility of the approach. Nevertheless, we plan to support emulation of seamless roaming of terminals, including multi-input multi-output connectivity.

References

[1] I. Satoh, "A Testing Framework for Mobile Computing Software," *IEEE Trans. Software Eng.*, vol. 29, no. 12, 2003, pp. 1112–21.
[2] Microsoft Corporation, "Universal Plug and Play Device Architecture Version 1.0" June, 2000. http://www.upnp.org/UpnPDevice_Architecutre_1.0.htm
[3] K. Arnold *et al.*, *The Jini Specification*, Addison-Wesley, 1999.
[4] M. Le, F. Burghardt, and J. Rabaey, "Software Architecture of the Infopad System," *Wksp. Mobile and Wireless Info. Sys.*, 1994.
[5] N. Davies *et al.*, "A Network Emulator to Support the Development of Adaptive Applications," *Proc. USENIX Symp. Mobile and Location-Independent Comp.*, 1995.
[6] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Trans. Software Eng.*, vol. 24, no. 5, 1998.
[7] B. D. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, 1998.
[8] M. Dahm, "Byte Code Engineering Library," http://jakarta.apache.org/bcel/index.html
[9] I. Satoh, "MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System," *Proc. Int'l. Conf. Distrib. Comp. Sys.*, Apr. 2000, pp. 161–68.
[10] International Business Machines Corporation, "Remote Abstract Window Toolkit for Java," http://www.alphaworks.ibm.com/, 1998.

Biography

ICHIRO SATOH [M] (ichiro@nii.ac.jp) received his B.E., M.E, and Ph.D. degrees in computer science from Keio University, Japan, in 1996. From 1996 to 1997 he was a research associate in the Department of Information Sciences, Ochanomizu University, Japan, and from 1998 to 2000 he was an associate professor in the same department. Since 2001 he has been an associate professor at the National Institute of Informatics, Japan. He received the IPSJ paper award, IPSJ Yamashita SIG research award, and JSSST Takahashi research award. He is a member of six learned societies, including ACM. He is serving as Publicity Chair for IEEE PerCom 2005 and on technical program committees of several IEEE conferences. His current research interests include distributed, mobile, and pervasive computing.