Software Testing for Mobile and Ubiquitous Computing

Ichiro Satoh
National Institute of Informatics / Japan Science and Technology Corporation
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
E-mail: ichiro@nii.ac.jp

Abstract

We describe a framework that is used to build and test software for ubiquitous and mobile computing. The approach involves software-level emulators for computing devices. Since each emulator is implemented as a mobile agent, it can dynamically carry its target software to each of the sub-networks that its device is connected to, on behalf of the device, and it permits the software to interact with other servers in its current sub-network. The framework can simulate the mobility and reconnection of mobile or ubiquitous computing devices by using the logical mobility of mobile agent-based emulators. That is, it can test software designed to run on a mobile or ubiquitous device in the same way as if the software were disconnected from the network, moved with the device, and reconnected to and executed on another network. This paper describes the lessons we learned from exploiting the framework in developing typical application software for mobile and ubiquitous computing devices.

1 Introduction

The development of software for ubiquitous and mobile computing devices is very difficult due to the limited computational resources that these devices have. Furthermore, recent advances in networking technology have made software development tedious and extremely susceptible to change, because modifications to network connectivity and location may lead to very sudden and unpredictable changes in contextual information. That is, a change in the network and location implies movement away from the servers currently in use, toward new ones. For example, a handheld device with a short-range radio link, such as IEEE802.11b and Bluetooth, which is carried across the range of 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 suitable software, the developer must test it in all the possible network environments that the device might be connected to. However, current work on ubiquitous and mobile computing has often focused on creating network and system infrastructures, small and low-power devices, user interfaces, and context-aware systems. Unfortunately, the task of building and testing the software for it has attracted little attention so far. This is a serious impediment to its growth beyond mere laboratory prototypes. To solve this problem, a software development approach that is suited to ubiquitous and mobile computing is needed. In fact, we previously introduced a framework for the development of software that ran on portable (but not ubiquitous) devices in an earlier paper [17]. However, since the goal of the original framework was not to support ubiquitous devices, it could not be used to test the software that used the discovery and management services for these devices, such as Jini and UPnP.

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

The remainder of this paper is organized as follows. Section 2 is a survey of related work and Section 3 explains the framework for building and testing mobile computing applications. Section 4 is a brief review of our mobile agent system and the design and implementation of the framework are presented in it. Section 5 demonstrates the usability of the framework through two real-world examples. Section 6 has the conclusion and provides suggestions for future work.

2 Requirements

The goal of this paper is to offer a framework for testing network-dependent application software, which is designed to run on ubiquitous or mobile computing devices, such as information appliances, smart sensors, mobile phones, PDAs, and notebook-PCs, and which may often access servers on local networks in the device's current location either through wired networks such as the Ethernet or short-range wireless networks such as the IEEE802.11b or Bluetooth.

Software that is to be tested for ubiquitous or mobile computing devices needs to satisfy the following requirements.

Network-dependency and Interoperability

Cooperation among ubiquitous devices and servers within a domestic or office network is an indispensable because it complements various missing features 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 the external environments provided by the network that it connects to. Moreover, testing the interoperability of various devices often tends to be tedious, since there are countless varieties of devices, with which the target device can cooperate.

Spontaneous and Plug-and-Play Management

Since a ubiquitous 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) [14], have often been used to manage devices. These middleware systems use multicast communications to find their management servers and devices, where multicast-based messages may only be transmitted to the hosts within specified sub-networks. Therefore, the target software to run on ubiquitous computing devices must be tested within the sub-networks that the devices can be connected to.

Mobility and Disconnection

Mobile devices may be disconnected from the network of the current location and then reconnected to that of another location. Changing the network and location implies movement away from the servers currently in use, toward new ones. That is, as a device moves across sub-networks, or joins or leaves a sub-network, 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 to or attached to. Even when a device is disconnected from the network, it may perform its own tasks independently of other devices for reasons of availability.

3 Related Work

A typical ubiquitous or mobile computing device has a less powerful processor with less memory and a limited user interface with a clamped keyboard and 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 portable 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.

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 sub-networks. 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 an application and to attach it to local networks 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 solution is to let the target software run on a local workstation and link up with remote devices and servers through networks, e.g., the InfoPad project at Berkeley [11] and Lancaster University's network emulator [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 local networks because of security concerns. Moreover, this approach is inappropriate since network traffic increases when a large amount of data is exchanged between the emulator and the remote servers. Also, since multicast communications are often transmitted within specified sub-networks due to reduced network traffic, they cannot be received outside the sub-networks. As a result, target software running on a workstation in a sub-network cannot access all multicast-based services, including service discovery systems, such as Jini, UPnP and SDP, that can be accessed in other sub-networks.

To our knowledge, there have been attempts to apply mobile agent technology [6, 10], including mobile code approach, to developing ubiquitous and mobile computing apart from that described in our previous paper [17].

4 Approach

Our framework aims to solve the problems in the previous section through a software-level emulator, which can simulate the internal execution environment of its ubiquitous or mobile computing device like the approaches taken existing works. 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 networks. Physical mobility entails the movement and reconnection of mobile computing devices between sub-networks (see Figure 1), while logical mobility involves software, such as mobile codes and mobile agents, that migrates among hosts on the sub-networks (see Figure 2).

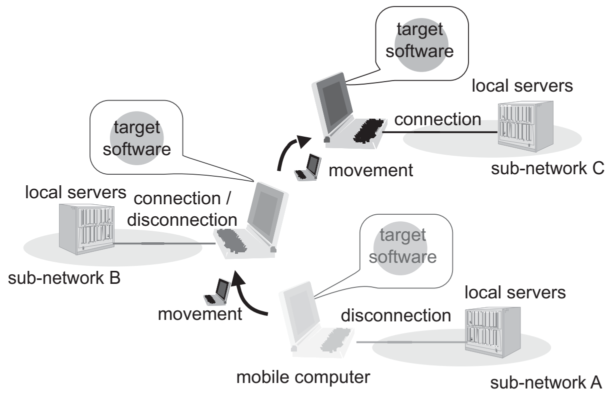


Figure 1. Physical mobility of a portable device.

Each mobile agent is just a logical entity and must thus be executed on a computer. Therefore, this framework assumes that each of the networks into which the device may be moved and attached to, has more than one special stationary host, called an access-point host, which offers a runtime system for mobile agents. Each access-point host is a runtime environment allowing applications running in a visiting emulator to connect to local servers in its network. That is, the physical movement of a portable computing device from one network and its attachment to another network is simulated by the logical mobility of a mobile agent-based emulator with the target applications moving from an access-point computer in the source network to another access-point computer in the destination network. Since each emulator is a mobile agent, it can basically carry not only the code but also the states of its applications to the destination, so the carried applications can basically continue their processes after arriving at another host as if they have been physically moved with the targeted device. Since the framework itself and applications are written in the Java language, the target portable devices must support this language. Moreover, while the framework does not require any custom hardware, its current implementation requires its target devices to offer TCP/IP communication over wired or wireless networks.

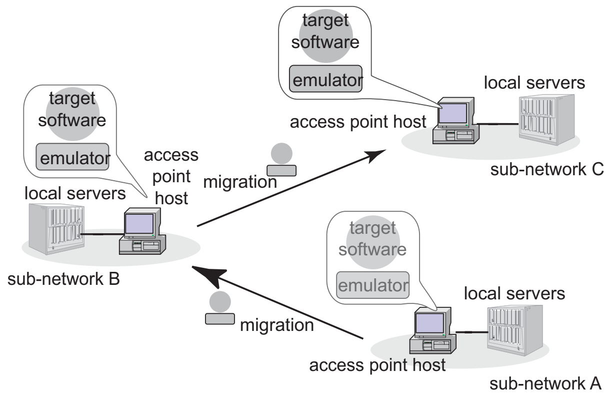


Figure 2. Emulation of physical mobility through logical mobility.

Since each emulator is implemented as a mobile agent, it can carry its target software to an access-point host, which can be treated as a peep provided from its visiting emulator for the devices to communicate with servers. The carried software can maintain its previous processes and interact with other devices and servers on the current sub-network via the access-point host. Therefore, the developer can test his/her target software that has been designed to run on its target device on the access-point host in the sub-network that the device can be moved and connected to. This means that the framework can satisfy the first and second requirements discussed in the previous section. Moreover, since the carried software is deployed and executed within the domain of the current sub-network, it can directly receive multicast packets such as Jini's and UPnP's management messages that are available in the domain. Therefore, the framework satisfies the third requirement and is useful in testing the interoperability of various protocols for ubiquitous computing.

5 Design and Implementation

This section describes our mobile agent-based framework. The current implementation of this framework is based on a

Java-based mobile agent system called MobileSpaces [15].[1] As Figure 3 shows, the framework has the following three components:

- The *mobile agent-based emulator*, which can carry the target software to specified access-point hosts on remote networks on behalf of a target-mobile or ubiquitous computing device.
- The *access-point hosts*, which are allocated to each network and allow the software carried by an emulator to connect with various servers running on the network.
- The *remote-control server*, which is a front-end to the whole system allowing us to monitor and operate the moving emulator and its target software by remotely displaying their graphical user interfaces on its screen.

In addition to the above, we provided a runtime system to run on a ubiquitous or mobile device and support 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.

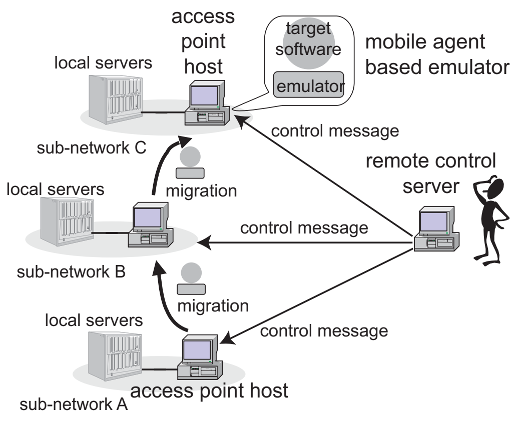


Figure 3. Architecture

5.1 Mobile Agent-based Emulator

We designed our mobile agent-based emulator to carry and test applications that have been designed to run on its target computing device. Each mobile agent-based emulator is just a hierarchical mobile agent of the MobileSpaces system. Since every application is provided as a collection of mobile agent-based components, the emulator can naturally contain more than one mobile agent-based application and can migrate itself and its inner applications to other places. Since such contained applications are still mobile agents, both the applications running on emulator and the applications running on the 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 the differences between them as much as possible. In addition, the Java virtual machine can actually shield applications from most features of the hardware and operating system of target computing devices. Figure 4 has the structure of a mobile agent-based emulator running an access-point host.

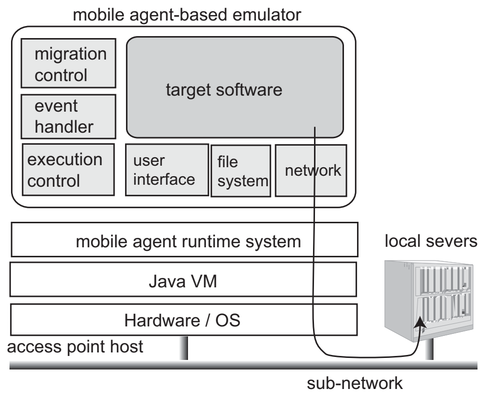


Figure 4. A mobile agent-based emulator running on an access-point host.

Emulation of Physical Mobility

Each emulator can have its own itinerary, listing hosts that correspond 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 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 and is statically defined by the user through its graphical user interface as Figure 6 shows. 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 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

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

is running in its emulator on an access-point host and is allowed to link up with servers on the network. In the isolated running state, the software is still running but is prohibited from communicating with any servers and 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 as a whole with the states and code of its target software. For example, the reconnection of a disconnected device is emulated by a combination of the isolated running state and the networked-running state of the software designed to run on the device.

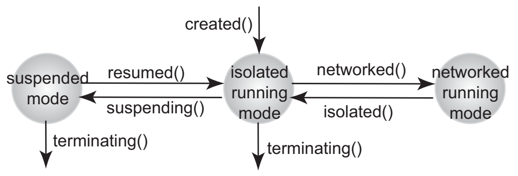
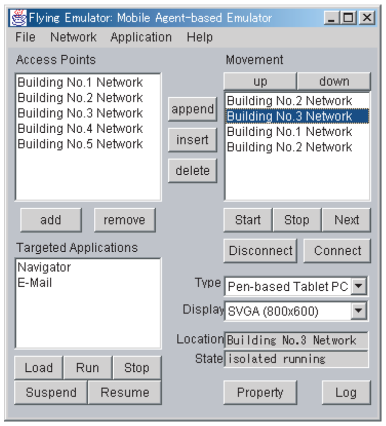


Figure 5. Callback method invocations in execution mode transition

When an emulator is suspended or migrated over networks, it can marshal itself into a bit stream as a whole 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 lifecycle state of the software is changed, as shown in Figure 5. In addition, our framework can provide each ubiquitous device with a lightweight middleware to monitor the environment of the device and dispatch certain events to its target as a mobile agent-based emulator corresponding to the device.



Access Points
Building No.1 Network
Building No.2 Network
Building No.3 Network
Building No.4 Network
Building No.5 Network

Movement
Building No.2 Network
Building No.3 Network
Building No.1 Network
Building No.2 Network

Targeted Applications
Navigator
E-Mail

Type Pen-based Tablet PC
Display SVGA (800x600)
Location Building No.3 Network
State isolated running

Figure 6. The user interface of a mobile agent-based emulator.

Emulation of Computing Devices

The framework assumes that its target software will be Java application program. Accordingly, the Java virtual machine can actually shield such target software from many features of the hardware and operating system of ubiquitous 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 these. In addition, the current implementation of our emulator supports several typical resources of ubiquitous devices as follows:

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 itself through its graphical user interface.

User Interface: The user interfaces of most handheld computers are limited by their screen size, color, and resolution, and they may be not equipped with traditional input devices such as a keyboard or mouse. 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 the visible content of the MobileSpaces system, called MobiDoc, developed by the author [16]. As will be discussed later, our framework also 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 being deployed at remote hosts.

Network: When anchored at an access-point host, each emulator can directly inherit most network resources from the host, such as `java.net` and `java.rmi` packages. In the current implementation, a moving emulator cannot have its own network identifier, such as an IP address and port number. However, this is not a serious problem because most applications on a computing device are provided as client-side programs, rather than server-side ones, as discussed in [8]. For example, Figure 7 shows the emulation of a ubiquitous device when the device is connected to a sub-network in a plug-and-play manner and the software running on it is interacting with other devices through multicast-communications. For example, when arriving at an access-point host, each emulator can directly exploit most network resources from the host, such as `java.net` and `java.rmi` packages. Although a moving emulator cannot have its own unique network identifier, such as an IP address and port number, it can inherit the identifier of the access-point host that it is running on.

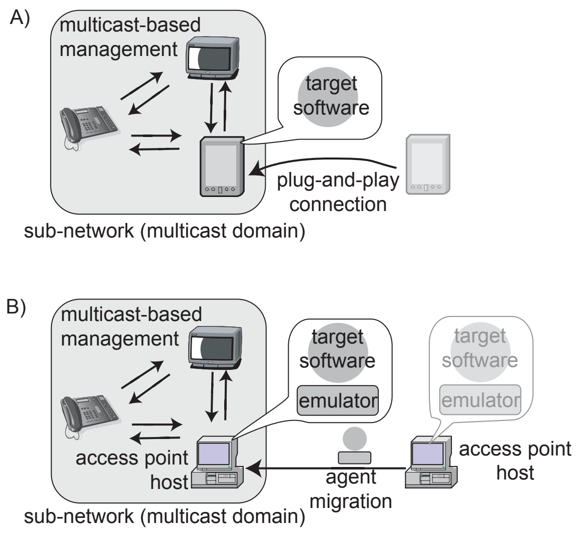


Figure 7. Emulation of (A) the plug-and-play operation of a ubiquitous device by (B) the migration of the emulator for the device between access-point hosts

Serial Port: Each emulator can permit its target 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 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.

5.2 Access-point Host

As previously mentioned, the framework presented in this paper is built on the MobileSpaces mobile agent system. Each access-point host offers a MobileSpaces runtime system for executing the mobile agent-based emulator and migrating it to another access-point host. When an agent is transferred over a network, the runtime system stores the state and codes of the agent, including 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 through using an extension of the HTTP protocol. In almost all Intranet situations there is a firewall that prevents users from opening a direct socket connection to a node across administrative boundaries. Since the mechanism is based on a technique called HTTP tunneling, it enables agents to be sent outside a firewall as HTTP POST requests, and responses to be retrieved as HTTP responses.

Also, each access-point host is treated as a peep of the resources and services provided in its network from the applications in a visiting emulator. This framework assumes more than one access-point host is allocated in each network, to which the target computing device may be attached. Each access-point host is constructed based on a common runtime system that can be used for targeted devices and run on a standard workstation without any custom hardware. Many applications have their own graphical user interfaces. To test such applications, our framework should offer a mechanism for remotely viewing and operating these user interfaces on the screen of the remote control server, instead of on the screen of their current hosts. The mechanism is constructed on the Remote Abstract Window Toolkit (RAWT) developed by IBM [7]. This toolkit allows Java programs that run on a remote host to display GUI data on a local host and receive GUI data from it. Each access-point host can be incorporated with the toolkit, thus allowing all the windows of applications in a visiting emulator to be displayed on the screen of the control server and operated using the keyboard and mouse of the server. Therefore, no access-point hosts do not have to offer any graphics services.

5.3 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. Moreover, by incorporating with a server of the RAWT toolkit, it 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.

5.4 Runtime System on Target Devices

This framework offers a lightweight runtime system to each target computing device. Each runtime system supports the execution of the software tested by our mobile agent-based emulators and monitors the environment of the device, including its characteristics such as network connectivity and location, to make the software aware of environmental changes. When detecting changes, it invokes certain of the methods of the software. Moreover, the runtime system provides a collection of service methods to allow software to have access to the device, without any particular knowledge of the operating system or hardware of its target device. There is no need to worry about the overheads of the runtime system, because the performance of software running on the minimum runtime system is almost equal to that of corresponding applications executed directly on the Java virtual machine.

5.5 Current Status

The current implementation of the framework supports emulators for four kinds of computing devices: standard notebook PCs, pen-based tablet PCs, palm-sized PDAs, and embedded computers that monitor and control equipment via RS232C-based serial ports. The features of the framework are summarized as:

- Like other computer emulators, this framework can provide software-level emulation of its target portable device for software designed by incorporating a Java virtual machine.
- Depending on the movement and (dis)connection patterns of its target device, the mobile agent-based emulator can carry software on its behalf to networks that the device may be moved and connected to.
- The emulator allows us to test and debug software with the services and resources, including multicast-based services, provided through its current network as if the software were being executed on the target device when attached to the network.
- The framework does not need any special equipment. Most existing Java-enabled application-specific servers can be used as access-point hosts by supplying runtime systems for mobile agent-based emulators. The remote-control server is operated on an ordinary PC.
- Software tested successfully in the emulator can still run in the same way without being modified or recompiled.

As previously mentioned, it is constructed on a Java-based mobile agent system called MobileSpaces [15], but we believe that the framework itself can work with other Java-based mobile agent systems.[2]

[2]Although the MobileSpaces system is characterized by the notion of hierarchical mobile agents, the framework presented in this paper is reconstructed independently of the system, unlike the original framework presented in our previous paper [17].

6 Applications

To demonstrate the utility of our framework, we tested two typical systems in ubiquitous computing settings.

6.1 UPnP-based Management System

In a previous project [12], 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 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. Since UPnP's multicast messages are available within the domain of specified sub-networks, our 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 just as a carrier for the software. When the emulator arrives at an access-point host within the domain, 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. In addition, the software tested successfully in the emulator can still be run in the same way without modifying or recompiling it. This example demonstrates that our framework can provide a powerful methodology for testing the interoperability of protocols, limited within specified sub-networks for reasons of security and reduced network traffic.

6.2 Printer Management System

Through this example, we explain the development of a location-dependent printing service system for users that are moving through a building. In the current implementation of the system, each floor of the building has one or more printers of various types that are managed by the Jini system [2]. Each floor is covered by one or more ranges of IEEE802.11b wireless sub-networks without any overlap in these ranges. Also, all users moving in a building have a portable computing device equipped with an IEEE802.11b network interface.[3] As the user is moving from floor to floor, the server allocated in the sub-network automatically advertises its printers to the visiting device. To construct

[3]The implementation assumes our target device is a PC-based portable computer because other devices such as PDAs and mobile phones cannot currently support Jini.

Jini client-side software designed to run on a portable computing device, the developer needs to carry the device, attach it to the sub-network of each floor, and then check whether it can successfully access every printer on the current floor. This framework could successfully be used to test the system. We constructed a mobile agent-based emulator for the device that can migrate the client-side software to the sub-network of another floor. It then allows the software to interact with Jini's servers to access the printer for the floor. While it is impossible to measure the framework's benefits in a quantitative manner, it eliminates the task of the developer having to go up and down the stairs while carrying a portable device simply to verify whether it can successfully print out data from the networked printers on the current floor.

7 Conclusion

In this paper, we provided a framework for building and testing software that had been designed to run on ubiquitous or mobile devices. The key idea behind the framework was to construct an emulator for target computing devices that was implemented as a mobile agent. The emulator could carry, deploy, and test software designed to run on its target portable device with the environment provided by the sub-network in the same way as if the software had been moved with and executed on the device when attached to the sub-network. This approach could test most features of ubiquitous or mobile devices, such as network-dependency, mobility, and multicasting-based management. Our early experience indicated that we could greatly reduce the time required to develop software for ubiquitous and mobile devices by using the framework.

There are also further issues that need to be resolved. Security is one of the most serious concerns in mobile agent technology. However, since our approach can be used in the development phase instead of operation phases, this issue is not serious, as it is in other mobile agent-based applications. However, we plan to devise schemes to guarantee security and control access, since the current implementation relies on the JDK 1.1 security manager. Also, our approach can be used to complement existing software-development methodologies for ubiquitous computing. Therefore, we are interested in making tools to integrate our approach with other methodologies. 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 theses.

Acknowledgments: We would like to thank the anonymous reviewers for their significant comments.

References

[1] G.D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, "Cyberguide: A Mobile Context-Aware Tour Guide". ACM Wireless Networks 3, pp.421–433. 1997.

[2] K. Arnold, A. Wollrath, R. Scheifler, and J.Waldo, "The Jini Specification". Addison-Wesley, 1999.

[3] K. Cheverst, N. Davis, K. Mitchell, and A. Friday, "Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project", Proceedings of ACM/IEEE Conference on Mobile Computing and Networking (MOBICOM'2000), pp.20–31, 2000.

[4] N. Davies, G. S. Blair, K. Cheverst, and A. Friday, "A Network Emulator to Support the Development of Adaptive Applications", Proceedings of USENIX Symposium on Mobile and Location Independent Computing, USENIX, 1995.

[5] W. K. Edwards and R. E. Grinter "At Home with Ubiquitous Computing: Seven Challenges", Proceedings of Ubiquitous Computing (Ubicomp'2001), pp.256-272, LNCS, Vol. 2201, Springer, 2001.

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

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

[8] J. Jing, "Client-Server Computing in Mobile Environments", ACM Computing Survey.

[9] T. Kindberg and A. Fox, "System Software for Ubiquitous Computing", Pervasive Computing, Vol.1, No.1, pp.70-81, IEEE Computer Society, 2002.

[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] 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.

[13] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: Aiding the development of context-enabled applications" Proceedings of Conference on Human Factors in Computing Systems (CHI'99), pp.434-441, ACM Press, 1999.

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

[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, "Physical Mobility and Logical Mobility in Ubiquitous Computing Environments", Proceedings of 6th International Conference on Mobile Agents (MA'2002), LNCS, Vol.2535, pp.186-202, Springer, October, 2002.

[19] B. Schilit, N. Adams, R. Want, "Context-Aware Computing Applications" Proceeding of Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, 1994.