Software Testing for Ubiquitous Computing Devices

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

Abstract

We describe an approach for building and testing software for ubiquitous computing. The approach provides application-level emulators of ubiquitous devices. Since each emulator is implemented as a mobile agent, it can dynamically deploy its target software at each of the sub-networks that its device may be connected to and permits the software to interact with other devices and servers in its current sub-network. Therefore, the approach can test software designed to run a ubiquitous device in the same way as if the software was moved with and executed on the device when attached to the sub-network. This paper also presents some experiences with the development of ubiquitous computing software by using the framework.

KEY WORDS
mobile computing, ubiuquitous computing, mobile agent

1 Introduction

Current work on ubiquitous computing often focuses on the creation of small and low-powered devices, user interfaces, and context-aware systems. On the other hand, the tasks of building and testing software for ubiquitous computing have received little attention so far and many existing software for ubiquitous computing have been still developed in an ad-hoc manner. This is a serious impediment to the growth of ubiquitous computing beyond mere laboratory prototypes. To solve this problem, a software development approach suitable to ubiquitous computing is needed. Indeed we introduced a framework, called Flying Emulator, for developing software running on portable (but not ubiquitous) devices in our earlier paper [13]. Although the initial goal of the framework is not to support any ubiquitous devices, this paper proves that the framework is still practical in ubiquitous computing settings and describes lessons learned from exploiting the framework in the development of typical ubiquitous computing devices.

In the remainder of this short paper we describe our design goals (Section 2), an approach for building and testing ubiquitous computing software, our experience with several applications (Section 3), conclusions and future issues (Section 4).

2 Background

A typical ubiquitous computing device has a less powerful processor with less memory and limited user interface such as cramped keyboard and small screen. Therefore, it is difficult to build and debug software for such a device within the device itself. A popular and practical solution to this problem is to offer a software-based emulator of the target ubiquitous device. However, it is not always available in the development of network-enabled devices because of the following characteristics of ubiquitous computing as discussed in [3, 6].

Network-dependency and Interoperability: Cooperation among ubiquitous devices and servers within a domestic or office network is an indispensable feature that complements some missing parts in a device. As a result, the correctness of software running on a device depends not only on the internal execution environment of the device, but also the external environments provided from the network that the device may connect to. Moreover, testing of interoperability among various devices often tends to be tedious, since devices, with which the target device may cooperate, are various and innumerable. However, it is almost impossible for an emulator running on a standalone computer to simulate the whole context that its target device can interact with through networks.

Mobility and Disconnection: Devices may be disconnected from the network of the current location and then reconnected to that of another location. Changing the network and location may imply movement away from the servers currently in use, toward new ones. Even while a device is disconnected from the network, the device may perform its own task independently of other devices for the reason of availability. A precise solution to solve this problem is that the developer actually carries a workstation running an emulator of the target device (or the device itself) for performing an application and attaches it to local networks in the current location. However, this is extremely cumbersome troublesome for the developer and thus should be resorted in only the final phase of software development.

Spontaneous and Plug-and-Play Management: Another solution is enable target software to run on a local workstation and link up with remote devices and servers

through networks. On the other hand, a ubiquitous computing environment requires zero user configuration and administration. Several middleware systems, such as Jini [2] and Universal Plug and Play (UPnP) [10], have been proposed and used as the part of coordination architectures that ensure device interaction with the ultimate aim of simple and spontaneous device management in a plug-and-play manner. These middleware systems are design for managing devices by using multicast communications, where management messages may be transmitted to only the hosts within specified sub-networks, instead of any remote networks. As a result, such a networked emulator cannot remotely access all the multicast-based services supported through by middleware systems. Also, it is inappropriate because of responsiveness and security protection such as firewalls.

In the remainder of this section we briefly survey related work. There have been a few researches to improve the task of building and testing software for ubiquitous devices. Among them, most of them provide middleware and toolkits for building context-aware software suitable to ubiquitous computing, for example, see [14, 9]. To our knowledge, no existing attempts to apply mobile agent technology [4, 7], including mobile code approach, to the development of ubiquitous and mobile computing except for our previous paper [13]. There have been a variety approaches for building and testing software for ubiquitous devices, including standalone emulation approach and field testing approach as stated in this section. The goal of the approach presented in this paper is to complement other existing approaches so that the approach should be properly combined with the existing approaches.

3 Architecture

To satisfy the above requirements, we construct an approach for building and testing software for ubiquitous devices based on the Flying Emulator framework.

3.1 Flying Emulator Framework

We outline the Flying Emulator framework, which is a basis of the approach and presented in a previous paper [13].[1] The goal of the original framework is to test network-dependent software running on a portable computing device in the sense that the software may often access servers on the network that the device is currently connected to either through wired or wireless networks. As the device moves across networks, their environments may change. That is, some new servers become available, whereas others may no longer be relevant. Such software must be tested in the environments of all the networks that the device can be moved into and attached to.

[1] Details of the Flying Emulator framework can be found in our previous paper [11].

The key idea of the framework is to emulate the physical mobility of a portable computing device by using the logical mobility of the targeted software. The framework provides a software-based emulator of portable computing devices. Each emulator is constructed as a Java-based mobile agent. It can carry target software across networks on behalf of a device and allows the software to connect to local servers in its current network in the same way as if the software was moved with and executed on the device when attached to the sub-network.Figure 1 illustrates the correlation between the physical mobility of a running device and the logical mobility of an emulator of the device.
Each mobile agent is just a logical entity and thus must be executed on a computer. Therefore, the framework assumes that each of the networks, into which the device may be moved and attached to, has a special stationary host, called an access point host. The host offers a runtime system for executing and migrating mobile agent-based emulators and allows the software carried by an emulator to connect with various servers running on the network. That is, the physical movement of a portable computing device from one network and attachment to another network is simulated by the logical mobility of a mobile agent-based emulator with the target software from an access point computer in the source network to another access point computer in the destination network. Each mobile agent-based emulator can carry not only the code but also the states of its software to the destination. As a result, the carried software can continue their processes after arriving at another host as if it was moved with its targeted device. Also, the framework allows the software tested successfully in a mobile agent-based emulator to be executed on its target portable device without modifying or recompiling the application. An implementation of the framework is available in the JDK 1.1 or 1.2-compatible Java virtual machine, including Personal Java.

3.2 Software Testing for Ubiquitous Devices

Although the Flying Emulator framework was initially inspired for the development of portable computing devices, the framework is available in ubiquitous computing setting. An emulator of the framework performs application-transparent emulation of its target device for application software written in the Java language. 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 of the devices and servers provided from its visiting emulator. The carried software can keep its previous processes and interact with other devices and servers provided on the current sub-network via the access point host. Therefore, the developer can test his/her target software designed to run on its target device to an access point host in the sub-network that the device may 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 de-

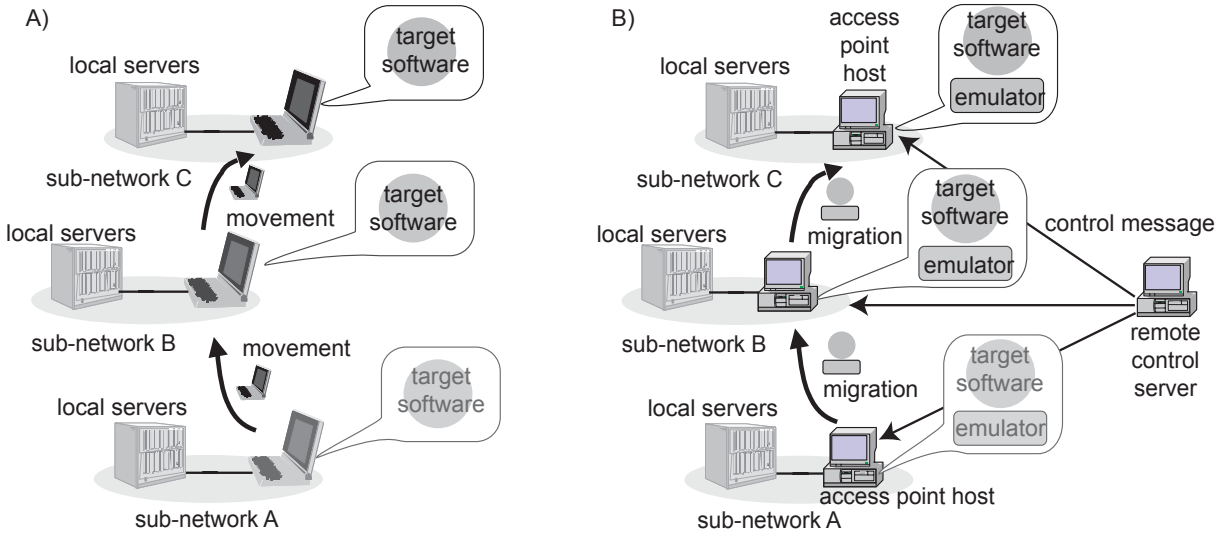

Figure 1. Corelation between (A) the physical mobility of a device and (B) the logical mobility of an emulator of the device with its target software

ployed and performed within the domain of the current sub-network, it can directly receive multicast packets such as Jini's and UPnP's management messages available in the domain. Therefore, the framework satisfies the third requirement and is useful in the testing of interoperability of various protocols for ubiquitous computing.

4 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 [11].[2] As shown in Figure 2, the framework consists of the following three parts:

- The *mobile agent-based emulator* can carry the target software to specified access-point hosts on remote networks on behalf of a target ubiquitous computing device.
- *Access-point hosts* are allocated to each network and allows the software carried by an emulator to connect with various servers running on the network.
- The *remote-control server* is a front-end to the whole system and it allows 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 parts, we provide a runtime system to run on a ubiquitous device and support execution of the tested software. The framework is constructed independently of the underlying system and can thus run on any computer with a JDK 1.1 or 1.2-compatible Java virtual machine, including Personal Java, and as the MobileSpaces system.

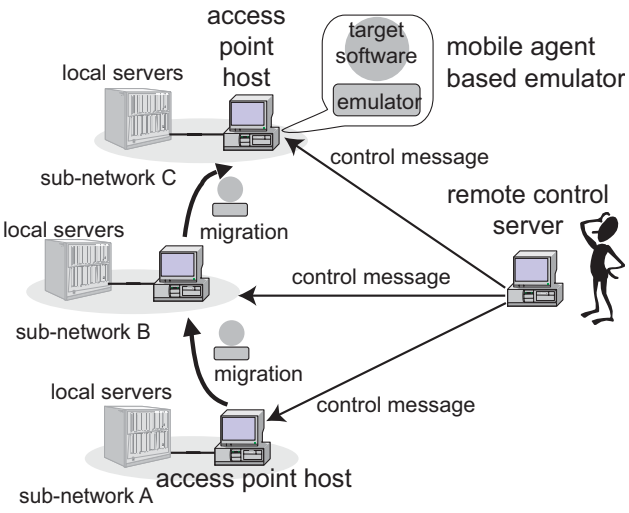

Figure 2. Architecture

4.1 Mobile Agent-based Emulator

The mission of our mobile agent-based emulator is 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

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

system. Since every application is provided as a collection of mobile agent-based components, the emulator can naturally contain more than one mobile agent-based application inside itself and can migrate itself and its inner applications to other places. Since such contained applications are still mobile agents, both the applications running on an 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 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 3 illustrates the structure of a mobile agent-based emulator running an access-point host.

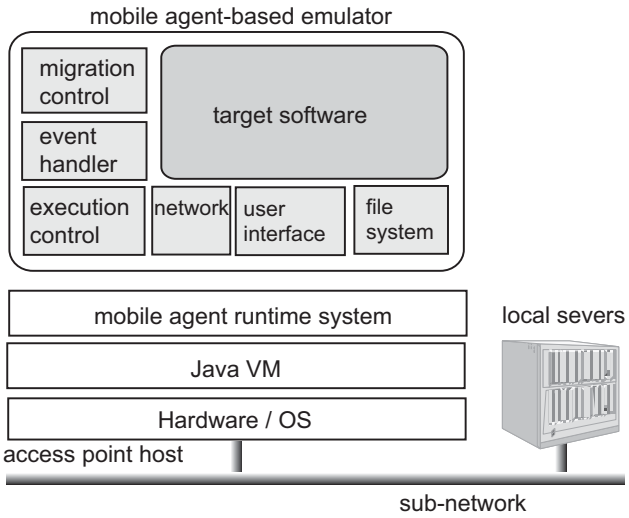

Figure 3. A mobile agent-based emulator running on an access-point host.

The framework assumes that its target software is a 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 it. 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 and 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 visible content for the MobileSpaces system, called MobiDoc, developed by the author in [12]. As will be mentioned later, our framework furthermore enables the developer to view and operate the user interfaces of applications in an emulator on the screen of its local computer, even when the emulator is 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. For example, Figure 4 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.

Serial Port: Each emulator can permit its target software to be Java's communication APIs (Java COMM), provided on the device that the emulator runs. Furthermore, the framework offers a mechanism for allowing its target software to have access to equipment running on remote computers via serial ports. The mechanism consists of proxies whose interface is compatible with Java's communication APIs and which can forward the port's signals between an emulator and the remote-control server through TCP/IP channels. In almost all Intranet situations, a firewall prevents users from opening a direct socket connection to a node across administrative boundaries.

4.2 Access-point Host

As mentioned previously, 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 and migrating the mobile agent-based emulator to another access-point host. When an agent is transferred over a network, the runtime system stores the

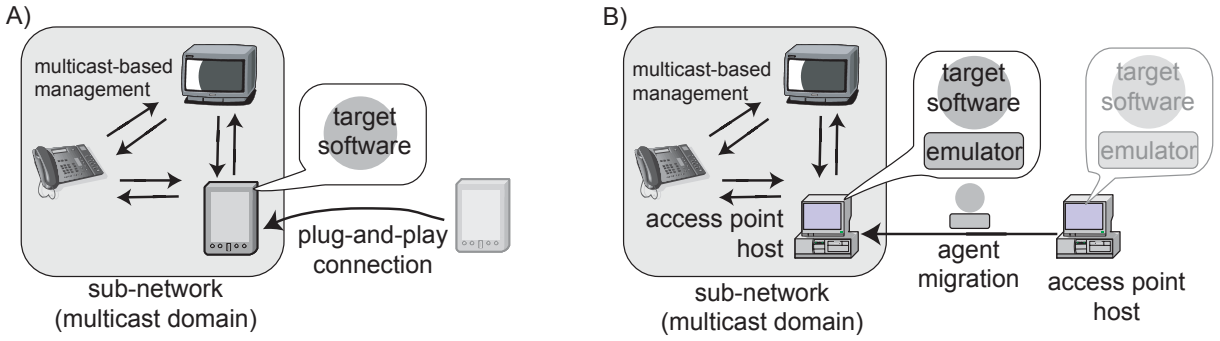

Figure 4. Emulation of (A) the plug-and-play operation of a ubiquitous device by (B) the migration of the emulator for the device between access-point hosts

state and codes of the agent, including software, in a bitstream defined by Java's JAR file format that can support digital signatures for authentication. The MobileSpaces runtime system supports a built-in mechanism for transmitting the bitstream over networks by using an extension of the HTTP protocol. 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 to be 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 built on the Remote Abstract Window Toolkit (RAWT) developed by IBM [5]. 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 host has to offer any graphics services.

4.3 Remote-control Server

This server is a control entity responsible for managing the whole system. It can run on a standard workstation that supports the Java language. It can always track the locations of all the emulators, because each access-point host sends certain messages to the control server whenever 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. Also, it can monitor the status of all the access-point hosts by periodically multicasting query messages to them.

5 Experiences

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

UPnP-based Management System: In our previous project [8], we implemented a subset of the UPnP protocol written in the Java language. Using this framework, we tested the interoperability between 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), for announcing a device's presence to others as well as discovering other devices or services. For example, a joining device sends out an advertisement multicast message to advertise its services to 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 be performed within the domain to receive the messages. Therefore, we constructed a mobile agent-based emulator as just a carrier of the software. When the emulator arrives at an access point host within the domain, the software carried by the emulator can send out an advertisement

multicast message and receive search multicast messages from other devices in the domain as if the emulator's target joined to the domain. In addition, the software tested successfully in the emulator can still be performed in the same way without modifying or recompiling the software. As a result, this example shows that our framework can provide a powerful methodology for testing the interoperability of protocols limited within specified sub-networks for the reasons of security protection and the reduction of network traffics.

Printer Management System: This example illustrates the development of a location-dependent printing service system for moving users in a building. In the current implementation of the system, each floor of the building has one or more printers, which are of various types and managed by the Jini system [2] and is covered by one or more ranges of IEEE802.11b wireless sub-networks without any overlap among the ranges. Also, each moving user has a portable computing device equipped with IEEE802.11b network interface.[3] When moving from floor to floor, the server allocated in the sub-network can automatically advertise its printers to the visiting device. To construct 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 test the system. That is, we constructed a mobile agent-based emulator for the device. The emulator can migrate the client-side software to the sub-network of another floor and then allows the software to interact with Jini's servers to access the printer of the floor. While it was impossible to measure the framework's benefit in a quantitative manner, the framework exempts the developer from the task of going up and down stairs with while carrying a portable device simply to verify whether the device can successfully print out data by using networked printers on its current floor or not.

6 Conclusion

In this paper, we have seen a way to use the framework initially designed for the development of software for portable computing devices as a novel and practical approach for testing software for ubiquitous devices. The key idea of the approach is to construct an emulator of ubiquitous devices as a mobile agent. The emulator can dynamically deploy and test software designed to run on its target portable device with the environment provided from the sub-network in the same way as if the software was moved with and executed on the device when attached to the sub-network. The approach can test most features of ubiquitous devices, such as network-dependency, mobility, multicasting-based management, and so on. Our early experience with this approach suggested that our approach could greatly reduce the time needed to develop software for ubiquitous devices.

Finally, we would like to point out further issues to be resolved. Security is one of the most essential issues in mobile agent technology. Actually, since our approach should be used in the development phase instead of any operation phases, this issue is not serious, compared to other mobile agent-based applications. However, we plan to design scheme to perform security and access control, since the current implementation relies on the JDK 1.1 security manager. Also, our approach should be used to complement other existing software development methodologies for ubiquitous computing. Therefore, we are interested in making a tool for integrating between our approach and other methodologies.

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

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] 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.

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

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

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

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

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] I. Satoh, "Flying Emulator: Rapid Building and Testing of Networked Applications for Mobile Computers", Proceedings of Conference on Mobile Agents (MA'2001), LNCS, pp.103-118, Springer, December, 2001.

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