ROAM (Resource-Aware Application Migration) System

Hao-hua Chu, Shoji Kurakake
haochu@dcl.docomo-usa.com, kurakake@dcl.docomo-usa.com
DoCoMo Communications Laboratories USA, Inc.
181 Metro Drive, Suite 710, San Jose, CA 95110

Abstract

In this paper, we present the design and implementation of the Roam (Resource-aware Application Migration) system. The Roam system is a framework for building mobile applications that are capable of runtime migration between *heterogeneous computing and communication devices* (e.g., PCs, PDAs, cell phones, and etc). In particular, the Roam system is focused on device heterogeneity in today's mobile device market. The Roam system proposes several innovative solutions that enable migrating applications to become aware of the target device capabilities, and then dynamically reconfigure their code and computation to fit the target device capabilities. We have implemented the Roam system on top of Java VM and PersonalJava VM. We have also conducted some experiments using simple applications running on top of the Roam system, and the experimental results have shown promising feasibility of the Roam system.

Keywords: Mobile Applications, Application Migration, Mobile Agents, Java.

1 Introduction

Roam (*Resource-aware Application Migration*) system is a *framework* for building resource-aware mobile applications that are capable of *runtime migration* between heterogeneous computing and communication devices that have vastly different hardware, software and network capabilities. Some examples of heterogeneous devices are desktop computers, notebooks, personal digital assistants (PDAs), cell phones, or other emerging mobile devices and information appliances.

1.1 Motivations

Given the popularity and widespread use of mobile devices nowadays, there will be no surprise in the near future that a single user may own multiple such mobile devices or information appliances with different sizes, shapes, functionalities, and capabilities. However, a mobile user may use only one particular device at a particular time depending on his/her situation. For example, a user is planning an online trip (via an Applet) on a desktop computer in office. In the middle of the planning, the user receives an urgent call and must leave office for a meeting at a remote site. As a result, the user takes a light-weighted device (PDA or cell phone) to the remote site, and he/she really wants to continue working on trip planning on the light-weighted device when he/she is on a bus or during break time of meeting. Based on this scenario, we think that there exists a need to allow a mobile user, when switching devices, to also move any running applications from one device to any other connected device in an *effortless* manner. The benefit is that the applications can continue to serve the user anywhere anytime on whatever device is accessible and convenient to the user. The Roam system is designed to realize this need for runtime application migration among heterogeneous devices.

There are several alternative approaches to application migration. *Remote display* (e.g., X window) is one approach. Remote display allows windows and graphics to be transmitted and displayed on the remote user terminals, while computation occurs on the server host (e.g., X server). However, remote display is only suitable to situations where terminals and server hosts are connected by relatively fast networks (e.g., LAN), and they must always stay connected. As a result, remote display is not suitable to the mobile environment.

Data synchronization is another approach, and it is a commonly used approach nowadays. Data synchronization is typically used to synchronize data between two *device-specific applications*—different applications that perform the same function on different types of devices. When a user switches from one device to another device, he/she invokes data synchronization software to synchronize data between two different applications running on two devices. In comparison to the application migration used in the Roam system, we believe that it has the following two advantages over the data synchronization approach:

- *Application developers can develop on application that can run on different devices.* This is an improvement over device-specific applications. For examples, we have different email programs on PCs, PDAs, and cell phones, which they all perform the same function – email. However, application developers have to develop different applications that perform the same function on different types of devices. This is a *waste of developers' development time.* Using the Roam system, application developers can develop one application that can run on different types of devices.
- *Mobile users can dispatch running applications effortlessly from one device to any other devices that they plan to carry with them*, given that

devices are connected at time of dispatch. This is an improvement over the existing data synchronization approach across devices. Not all running state of an application can be stored as synchronizable data, and not all data is interchangeable between different applications due to lack of a unified data format and data transformation utilities. When switching devices, a mobile user may need to input non-synchronizable data again on the new device. With the Roam system, application developers can design their applications so that both the running state and data are migrated at runtime from one type of device to another type of device.

There has been an abundance of research in the area of mobile agents that allow applications to move from one host to another host at runtime with little or no loss of running state [1] [2] [3] [4] [5] [6] [7] [8]. These systems are mostly built on top of the Java virtual machines (JVM), which they can take advantage of a common virtual machine environment provided by Java. However, these systems make an important assumption of *device homogeneity*. For an example, the underlying hosts/devices must be PC or PC-like devices with enough hardware/software (HW/SW) capabilities to run the standard JVM. However, device homogeneity is not a realistic assumption in today's mobile computing environment where there exists a wide range of mobile devices and information appliances with different hardware/software (HW/SW) capabilities.

1.2 Requirements

The focus of the Roam system is to address the *device heterogeneity* problem in runtime application migration. In order to solve this problem, it requires *device adaptation*. The Roam system must enable mobile applications to become *aware of HW/SW capabilities* of the devices they are migrating to. It must also work with the mobile applications to *adaptively re-configure* their code and computation so that they fit agreeably with the HW/SW capabilities of the target device. In the Roam system, device adaptation utilizes the following three techniques. (1) *Resource capability specification* allows devices to describe their HW/SW capabilities, and it allows mobile applications to describe the required device capabilities. (2) *Computation apportioning* allows mobile applications running on less capable devices to offload computational or memory intensive tasks to more capable servers. (3) *Dynamic instantiation* allows software components in mobile applications to be instantiated and loaded dynamically based on capability of the underlying device.

2 Design

In this section, we describe the design of the Roam system. It is centered around two design concepts--*Roam System Architecture* and the *Roamlet Component-based Programming Model*. We call an application that runs in the Roam system a Roamlet, and a Roamlet can migrate between any two connected devices that have the Roam system running. Based on these two design concepts, we describe three features that are built into the Roam system to support device adaptation.

2.1 Roam System Architecture

The Roam system architecture is shown in Figure 1. It contains the following components: *Roam agent*, *Roamlet*, and *HTTP server*. Roam agents are the core components, and they must be installed and executed on both the source and target devices *before* a Roamlet can run and *before* any Roamlet migration can occur. During a Roamlet runtime migration, the Roam agent on the source device first negotiates with the Roam agent on the target device. The negotiation involves exchanges of the target device capabilities, the codebase URL where the Roamlet class byte code can be downloaded from, and information on how to adapt the migrating Roamlet to the target device (explained in later sections). In the 2nd step, the Roam agent on the target device downloads the Roamlet class byte code from the HTTP server corresponding to Roamlet's codebase. The HTTP server can reside on the source device or any other devices in the network. In the 3rd step, the Roamlet on the source device serializes its running state and sends its running state to the Roam agent on the target device. In the 4th step, the Roam agent instantiates the Roamlet on the target device.

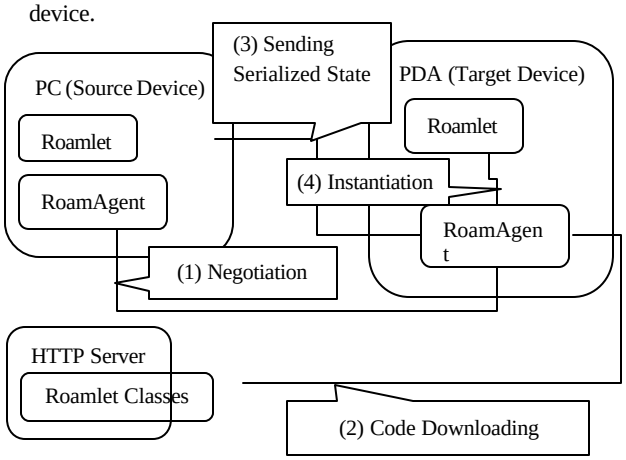

Figure 1: Roam System Architecture

2.2 Roamlet Component-based Programming Model

The Roam system supports a feature called *computation apportioning*, which allows a Roamlet to offload some of its computational intensive components to remote servers. However, this requires the application developers to modulize a Roamlet into distributed software components that can be distributed across multiple devices. Figure 2 shows an example of a Roamlet using this component-based programming model. The Roamlet is broken down

into 3 components—PDA GUI component, PC GUI component, and computation component. In a Roamlet, each component may be a distributed object and may be placed on any suitable devices. This requires that each component be *location transparent*, meaning that it can communicate with any other components regardless of their physical location.

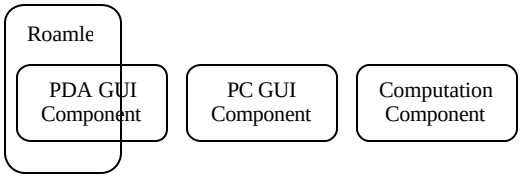

Figure 2: Roamlet Component-based Programming Model

2.3 Roam System Features

The Roam system provides three features to support device adaptation – *resource specification*, *computation apportioning*, and *dynamic instantiation*. We describe them in details in the following sections.

2.3.1 Resource Specification

The resource specification is used by Roamlet to describe what are the required device capabilities to run each of its components, and by the Roam agent to describe the device capabilities of its host device. Figure 3 shows an example on how the resource specification is used in the Roam system. Because components in a Roamlet may have different device requirements, a Roamlet is required to provide a separate device capability requirement for each component. The computation component requires a PC-equivalent device to run it, the PC GUI component also requires a PC-equivalent device, and a PDA GUI component requires a PDA-equivalent device or better.

Since the Roam system is built on top of Java, we choose Java VM configurations and profiles as the resource specification parameter in the current implementation. A summary of Java configurations and profiles is shown in Table 1. Java configurations and profiles are used to define the VM capabilities for different types of devices. Given that a device must have sufficient HW/SW capabilities to run a particular VM configuration and profile, VM configurations and profiles are good indications for the underlying device capabilities.

2.3.2 Dynamic Instantiation

Dynamic instantiation allows a Roamlet to choose and to instantiate appropriate components depending on the target device capability that the Roamlet is migrating to. Using dynamic instantiation, Roamlet developers can program different Roamlet behaviors in separate code segments. At runtime, the Roam system loads only the code segment that exhibits a behavior suited for the target device capability.

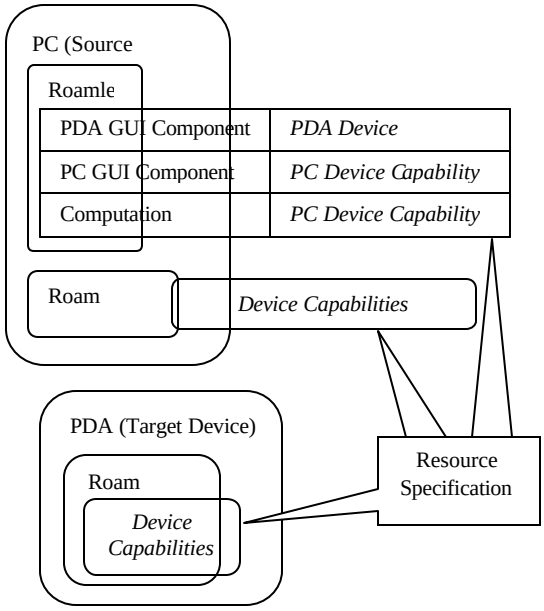

Figure 3: Resource Specification

In order to achieve dynamic instantiation, we require application developers to distinguish between two types of components in a Roamlet—device dependent (DD) components, and device independent (DI) components. Examples of DD-components are, but not limited to, GUI components, which may require different GUI libraries to take advantage of different device input and display capabilities. Application developers would provide one *implementation*, or one DD component, for each device type. On the other hand, DI-components are *functional components*, which they exhibit the same behavior regardless of the types of devices on which a Roamlet is executing. The Roam system performs dynamic instantiation only on the DD-components, which the Roam agent would select the most suitable DD-component for instantiation during a Roamlet migration.

Profiles	Personal Profile *[2.5MB ROM + 1M RAM]*	RMI Profile *[2.5MB ROM + 1M RAM]*	Foundation Profile *[1MB ROM + 512K RAM]*	PDA Profile *[512KB]*	MID Profile *[136KB ROM + 32KB RAM]*
Configurations	CDC *[512KB ROM + 256K RAM]*			CLDC *[160 KB]*	
Virtual Machines	PJVM		CVM	KVM	

Table 1: Java VM Configurations and Profiles

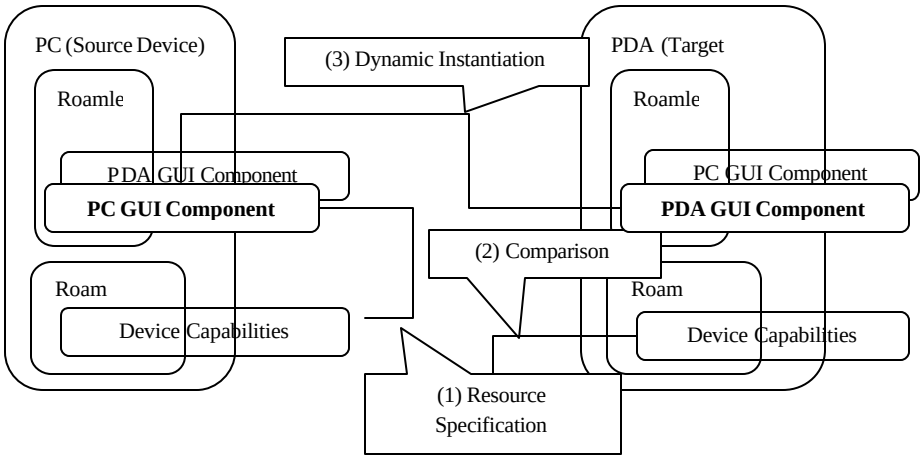

Figure 4: Dynamic Instantiation

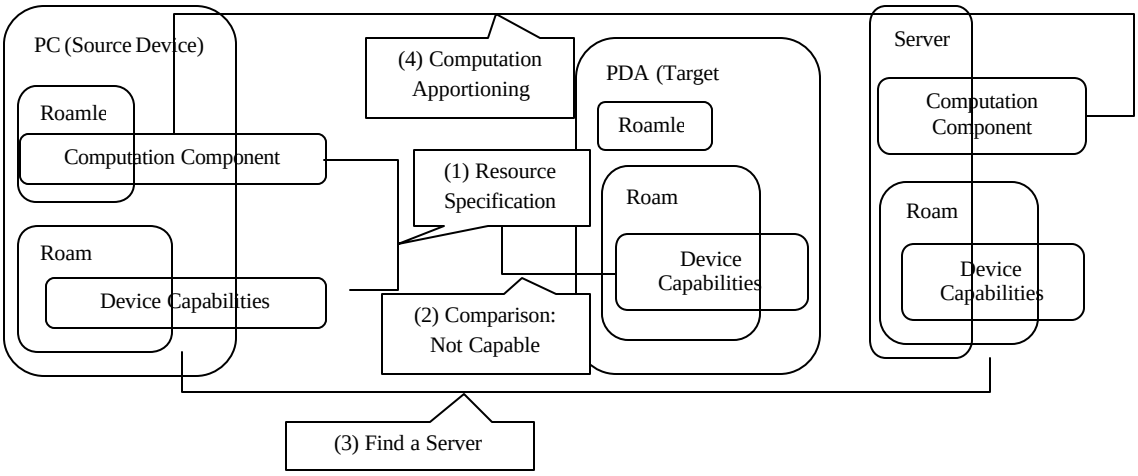

Figure 5: Computation Apportioning

Figure 4 shows an example of dynamic instantiation when a Roamlet migrates from a PC to a PDA. The Roamlet contains two DD components – a PC GUI component, and a PDA GUI component. The Roamlet first informs the Roam agent the required device capabilities to run each DD component – the PC GUI component requires a PC-equivalent device, and the PDA GUI component requires a PDA-equivalent device. In the 2nd step, the Roam agent on the source device compares the target device capability with the component's required device capability. In the 3rd step, the Roam agent determines that PDA GUI is the most suitable GUI component, and the PDA GUI component is instantiated on the target device.

2.3.3 Computation Apportioning

Computation apportioning allows a Roamlet to offload computation and memory intensive components to any remote servers that have the device capability to run them. Figure 5 shows an example of computation apportioning. The Roamlet contains a computation component, which is a DI-component as decribed in section 2.3.2. The Roamlet is migrating from a PC to a PDA. The Roamlet first informs the Roam agent that the computation component requires a PC-equivalent device capability. In the 2nd step, the Roam agent compares the required device capability of the computation component with the target device capability. Since the target device is a PDA, the Roam agent determines that the target device does not have the capability to run the computation component. In the 3rd step, the Roam agent attempts to find a server that satisfies the required device capability of the computation component. Both the Roamlet users and the Roam agent can specify a list of servers that can accept off-loadable components. When a server is found, the Roam agent directs the computation component to be migrated to the server. In the 4th step, the server instantiates the computation component.

Not all components in a Roamlet are off-loadable. The Roam system allows a Roamlet to specify components as either *non-off-loadable* or *off-loadable* components. Non-off-loadable components are

required to run on the target device – for example, the GUI component is typically not off-loadable.

3 Implementation

We have implemented the Roam system using the Java language and the Java language features—RMI, Serialization, and Reflection. The Roam system currently runs on the PCs with JVM, and on Pocket PC devices with Personal Java VM. In the future, we are planning to port the Roam system to CVM or any other Java virtual machines with RMI, Serialization and Reflection support. The current code size of the Roam system is approximately 24 Kbytes, which is small enough to fit on almost all Java-capable devices.

Roamlet
Roamlet APIs
Roam Agent
JVM, PJVM, CVM, …

Figure 6: Roam System and Roamlet APIs

The Roam system provides a set of Roamlet APIs shown in Figure 6. The Roamlet APIs provide an application interface of the Roam agent to the Roamlets. Due to space limitation, we will not describe the Roamlet APIs.

4 Experiments

The purpose of experiments is to evaluate the feasibility of the Roam system in a heterogeneous device environment. It also illustrates the Roam system features—runtime migration, computation apportioning, and dynamic instantiation. The experimental setup is shown in Figure 7. Casio E-125 is a PDA that runs Windows CE OS and PersonalJava VM, and it is connected via a wired LAN. Notebook runs Windows 2000 OS and standard JVM, and it is connected via a wireless LAN. PC runs Windows 2000 OS and standard JVM, and it is connected via a wired LAN. In the experiments, Casio PDA and Notebook act as source and target devices for migrate-able Roamlets. PC acts as the server device to run offloaded components.

We implement an application called HelloWorld on top of the Roam. It contains the following three components shown in Figure 8:

- Click component is an off-loadable device-independent component. It simply holds the number of clicks that a user clicks. We assume that it is computational/memory intensive so that we can show computation apportioning feature of the Roam system.
- Swing GUI component is a non-off-loadable device-dependent GUI component for a PC running JVM. Since it is built on top of Java Swing libraries, it requires the JVM capability.

Awt GUI component is a non-off-loadable device-dependent GUI component for a PDA running PersonalJava VM. Since it is built on top of the Java AWT libraries, it requires the PersonalJava VM or better capability. Note that the Java Swing libraries are not supported in PersonalJava.

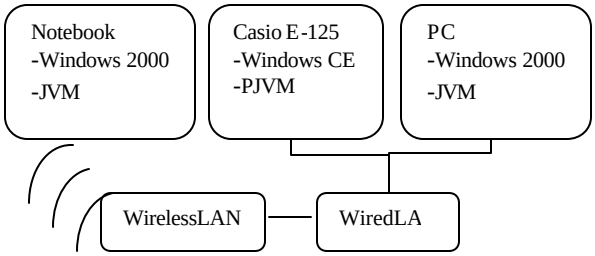

Figure 7: Experimental Setup

When the HelloWorld Roamlet is started on the Notebook (IP address = 172.21.96.17), the Swing GUI component is instantiated, and it has the look shown in the left window of Figure 9. "Number of button clicks" is initially 0, and becomes 3 after repeatedly clicking on "Click!" button. When HelloWorld migrates to the PDA (IP address = 172.21.96.152), the Roam system dynamically instantiates the AWT GUI component, which has the look shown in middle window of Figure 9. During the migration, the _click component is off-loaded to the PC as shown in the right windows of Figure 9. We have measured the amount of time needed for the HelloWorld runtime migration from PC to the PDA (and vice versa) to be less than 5 seconds.

We have implemented a second application, a Connect4 game Roamlet. We have converted the Connect4 Applet [13] into a Roamlet using the Roam APIs. It contains a GUI component (it can run on both the Notebook and the PDA) and an AI component as shown in Figure 10. The AI component computes the computer's next move by building a search tree, which is computation and memory intensive. As a result, the AI component requires the JVM capability.

When the Connect4 Roamlet is started on the Notebook, the AI component is instantiated on the Notebook because the Notebook has the required device capability. When it migrates to the PDA (running PersonalJava VM), the AI component is off-loaded to the PC as shown in Figure 10. We have measured the amount of time needed for the Connect4 runtime migration from PC to the PDA (and vice versa), which is less than 5 seconds.

5 Conclusion

In this paper, we present the design and implementation of the Roam system. We describe the Roam system architecture, the Roamlet component-based programming model, and Roam system features – resource specification, dynamic instantiation, and computation apportioning. We show the feasibility of the Roam system by implementing two simple Roamlet applications.

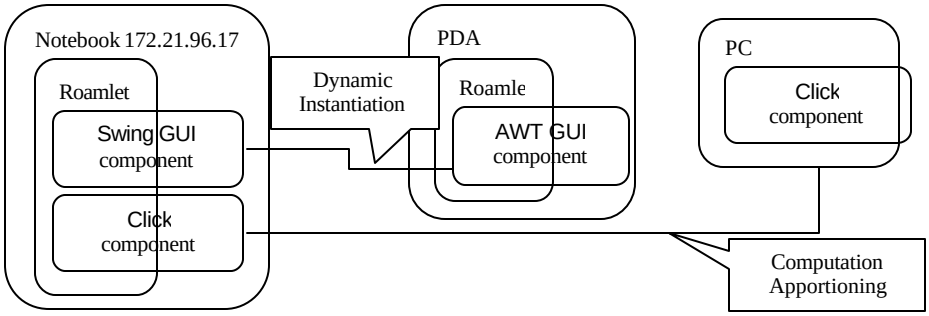

Figure 8: HelloWorld Roamlet: Dynamic Instantiation and Computation Apportioning

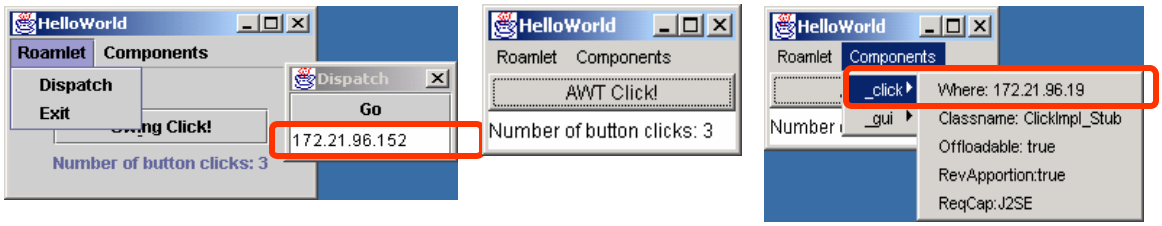

Figure 9: HelloWorld Roamlet

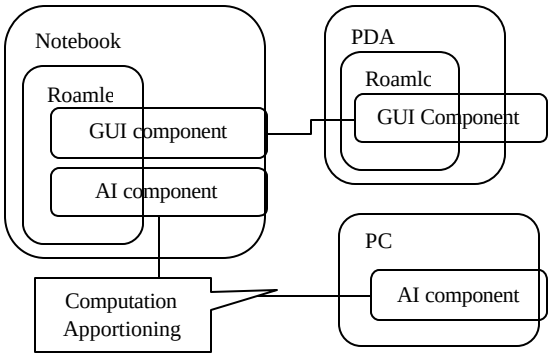

Figure 10: Connect4: Computation Apportioning

We foresee that in near future that people will use wide varieties of mobile devices and information appliances anywhere anytime. These devices come in different sizes, shapes, capabilities, and functionalities. There will be a need to create mobile applications that can move with people to whatever devices they are carrying with little or no human efforts. The Roam system is an ongoing research effort to realize this need.

References

[1] Aramira, Inc., "Jumping Beans™ White Paper", *http://www.jumpingbeans.com/index.html*, October 1999.

[2] A. Acharya, M. Ranganathan, and J. Saltz, "Sumatra: A Language for Resource-aware Mobile Programs". *Mobile Object Systems, J. Vitek and C. Tschudin (eds), Springer Verlag Lecture Notes in Computer Science.*

[3] M. Izatt, T. Brecht, and P. Chan. "Ajents: Towards an Environment for Parallel, Distributed and Mobile Java Applications", *ACM 1999 Java Grande Conference*, June 1999.

[4] D. B. Lange, M. Oshima, "Mobile Agents with Java: The Aglet API", *World Wide Web Journal*, 1998.

[5] D. S. Milojicicc, W. LaForge, D. Chauhan, "Mobile Objects and Agents (MOA)", In *Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, Santa Fe, New Mexico, April 1998.

[6] Mitsubishi Electric ITA Horizon Systems Laboratory, "Mobile Agent Computing A White Paper", January 1998.

[7] ObjectSpace, Inc., "Voyager", *http://www.objectspace.com/products/voyager*.

[8] M. Strasser, J. Baumann, and F. Hohl, "Mole - a Java Based Mobile Agent System. In *2nd ECOOP Workshop on Mobile Object Systems*, pages 28-35, Linz, Austria, July 1996.

[9] Sun Microsystems, "PersonalJava™ Technology -- White Paper", August 1998.

[10] Sun Microsystems, "JSR #000036 J2ME™ Connected Device Configuration", August 2000.

[11] Sun Microsystems, "Java™ 2 Platform Micro Edition (J2ME™) Technology for Creating Mobile Devices, White Paper", May 2000.

[12] Sun Microsystems, Inc. "J2ME CLDC/KVM Palm Release: Release Notes/CLDC 1.0", May 2000.

[13] S. Wiebus, Connect4 Applet.