Seamless Applications over Roam System

Hao-hua Chu, Henry Song, Candy Wong, Shoji Kurakake
Multimedia Application Laboratory, DoCoMo Communications Laboratories USA, Inc.
181 Metro Drive, Suite 300, San Jose, CA 95110 USA,
{haochu, csyus, wong, kurakake}@dcl.docomo-usa.com

Abstract

One of the biggest challenges in future application development is device heterogeneity. In the future, we expect to see a rich variety of computing devices that can run applications. These devices have different capabilities in processors, memory, networking, screen sizes, input methods, software libraries, and etc. This creates a challenge to developers on how to make their applications available on as many types of devices as possible. We also expect that a future user is likely to own many types of devices. Depending on the situation he/she is in, a mobile user may choose to switch from one type of device to another type of device that brings the best combination between the need for functions and the ease of mobility (size and weight). Based on this scenario, we believe that there is a need for an application framework that can both assist developers to build applications that can run on heterogeneous devices and allow a user to move/migrate a running application among heterogeneous devices in an effortless manner. We define a *seamless application* to be an application that can run on heterogeneous devices and migrate among heterogeneous devices. In this paper, we present the challenges and the design of our seamless application framework called the Roam system.

1 Introduction

The era of PC-dominated applications is about to end. Nowadays, we see a widespread use of mobile devices that have sufficient computing and networking capabilities to run different applications. They come in a variety of form factors – from smart pagers, cell phones, PDAs (Pocket PCs), Handheld PCs, car navigation systems, to Notebook PCs. As a matter of fact, an average person may already own multiple such devices. It is expected that in the near future, the number of such devices will far exceed the number of PCs. As a consequence, we expect that the main software platform will be shifted from PCs to mobile devices. This creates a challenge for developers to build applications that can run on different mobile device platforms. To address this challenge, we believe that there is a need for an application framework that can assist developers to build applications that can run on heterogeneous devices.

We also expect that a mobile user may choose to switch from one type of device to another type of device that brings the best combination between the need for functions and the ease of mobility (size and weight) given his/her current situation. For example, a user starts planning an online trip using 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. The user would like to continue planning the trip on the bus or during break time between meetings. Given the need for mobility, the user switches to a lightweight device (PDA or cell phone) away from office. Based on this scenario, we think that there is 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 him/her.

Based on these two needs, we define a *seamless application* to be an application that can run on heterogeneous devices and migrate among heterogeneous devices. 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][5][6][7]. These systems are mostly built on top of the Java virtual machines (VM), which they can take advantage of a common Java virtual machine environment. However, these systems make an 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 Java VM. However, device homogeneity is not a realistic assumption in today's heterogeneous mobile computing environment.

Roam (ResOurce-aware Application Migration) system is our *seamless application framework*. It allows developers to build *resource-aware* seamless applications that are capable of *runtime migration* in *heterogeneous device environment*. By resource-aware, we mean that applications are aware of the underlying device capabilities (hardware, software and network capabilities), which may change when they migrate from one type of device to another type of device. In order to migrate applications to devices with different capabilities, *adaptation* is needed. The Roam system makes use of the following three adaptation strategies:

1. *Transformation*: it allows software components to be transformed to fit the target device capabilities. Developers can provide platform-independent software components (e.g., using scalable graphical user interface library described in section 3), and they are transformed into platform-specific components at migration time.
2. *Dynamic instantiation*: it allows a different set of software components to be loaded and instantiated based on capabilities of the target device. An application can contain multiple software components that perform the same function, but each software component is implemented/re-implemented to run on a specific type of device.
3. *Offloading Computation*: it allows applications to make use of distributed computing resources to run software components that are beyond the target device capabilities. These software components may be CPU, memory, or network intensive, so that it is necessary to dispatch them on remote servers that have such capabilities.

We are currently in the process of building a prototype implementation of the Roam system. In our prototype, we restrict the device platforms to three popular mobile devices – cell phones, Pocket PCs (PDAs), and Notebook PCs. In addition, we will focus exclusively on Java applications and those devices that are capable of running Java VMs, e.g., JVM on Notebook PCs, Personal Java [8] VM on Pocket PCs, and J2ME [9] DoJa profile [4] on cell phones (DoCoMo 503i).

2 Roam System Design

The Roam system architecture is shown in Figure 1. It contains the following components: *Roam agent*, *Roamlet*, *Computing server*, and *HTTP server*. We call an application running in a Roam system as a Roamlet. The job of a Roam agent is to assist a Roamlet to run on and migrate among heterogeneous devices. Roam agents must be installed and executed on both source and target devices. Figure 1 shows an example of a Roamlet migrating from PC to PDA. The Roamlet has four components: computation component, PDA/PC GUI components, and scalable GUI component. The byte code for each software component is stored separately on a HTTP server. To illustrate all three adaptation strategies, we put both scalable GUI presentation (transformed from scalable GUI component), and non-scalable GUI component in a Roamlet on a device; however, only one of them is used in a real application. The Roam agent selects an appropriate adaptation strategy for each software component:

- The *scalable graphical user interface (GUI) component* is implemented using our scalable GUI library (see section 3 for details). The transformation strategy is used for adaptation. Since transformation can be computationally intensive, the target Roam agent may dispatch this job to a *GUI transformation manager* on a remote computing server. The transformation manager generates a platform-specific (i.e., PDA) GUI presentation that meets the target device screen size, input method, and GUI library. The PDA GUI presentation is then sent back to the target device.
- The *PDA and PC GUI components* are non-scalable GUI components that are implemented/re-implemented for each device platform using platform-specific GUI libraries. The Roam agent applies the dynamic instantiation strategy by instantiating the component (e.g., PDA GUI component for PDA device) that is most suitable for the target device capabilities.
- The *computational component* has only one implementation, and it prefers PC-equivalent device capabilities. If the target device does not have enough computing capability, the Roam agent applies the offloading computation strategy by finding a computing server, and then offloading the computation component to it.

The execution flow for a Roamlet migration from PC to PDA is described as follows:

1. 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 and the codebase URL where the Roamlet component byte code can be downloaded from. The appropriate adaptation strategy is also decided.
2. The Roam agent on the target device downloads the PDA GUI component byte code from the HTTP server.
3. The Roamlet on the source device serializes its running state and sends its running state to the Roam agent on the target device.
4. The Roam agent executes the appropriate adaptation strategy for each software component.
    a. Transformation is used to generate PDA GUI presentation from the scalable GUI component in the case that scalable GUI is used.
    b. Dynamic instantiation is applied for the PDA GUI component in the case that non-scalable GUI component is used.
    c. Computational component is offloaded if necessary.

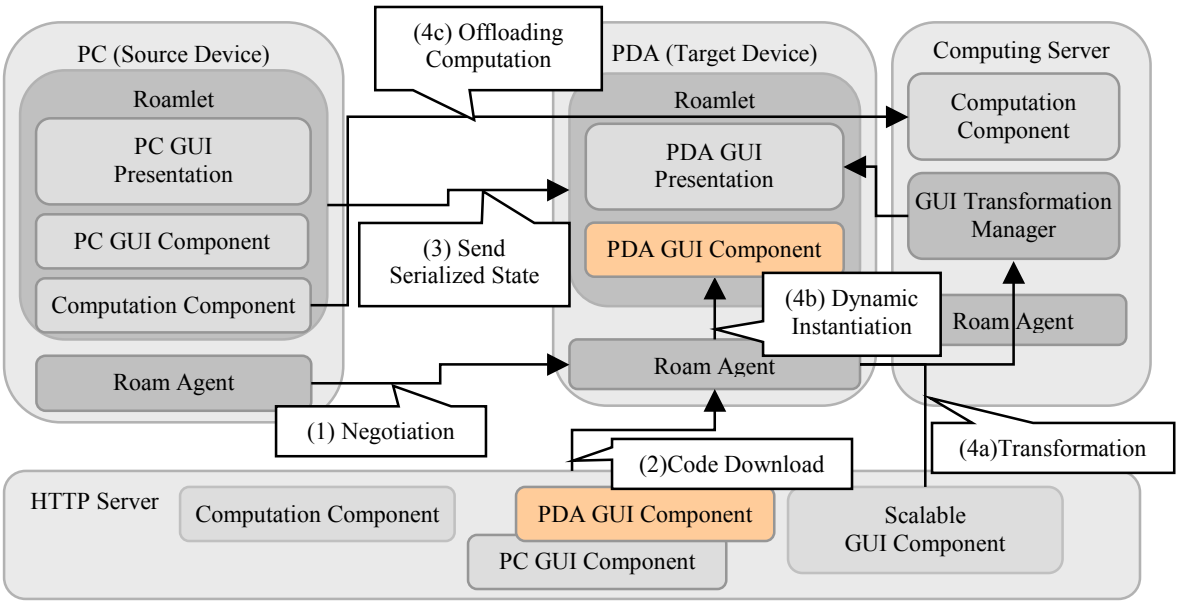

Figure 1: Roam System Architecture

3 Scalable Graphical User Interface (SGUI)

In cross-platform application development, UI is often the most challenging part for developers because devices can differ significantly on their human interface capabilities. The Roam system provides scalable GUI to assist developers to create GUI that can be transformed into different platform-specific presentations. SGUI is targeted to address the following three commonly found challenges:

- *Display size*: different device platforms have different display sizes. A DoCoMo 503i cell phone has a 120x130 pixels display, a Compaq IPAQ Pocket PC has a 320x240 pixels display, and an average Notebook PC has 1024x768 pixels display. Display size dictates the layout of the GUI components. A large display (e.g., PC) can accommodate big GUI components, and many GUI components can be presented at once on the same page. On the other hand, a small display can only accommodate highly compact GUI components, and only a few GUI components can be displayed at once on the same page. SGUI provides dynamic layout and GUI transformation for a seamless application so that the final presentation fits the target device display size.
- *Input method*: different device platforms have different input methods. A cell phone uses a keypad, a Pocket PC uses a stylus, and a Notebook PC uses a keyboard and a mouse-like pointing device. SGUI provides an abstract and unified GUI event model for a seamless application to process user input from different input methods.
- *User interface (UI) library*: different device platforms support different GUI libraries. For examples, a DoCoMo 503i cell phone supports J2ME DoJa profile, a Pocket PC supports the Personal Java AWT, and a Notebook PC can support J2SE SWING. SGUI provides a scalable GUI API and library that is supported on different device platforms.

The SGUI design is shown in Figure 2. It contains 3 main components – *scalable GUI library*, *task manager*, and *transformation manager*. Scalable GUI library provides APIs for seamless applications to construct platform independent GUIs. The scalable GUI library is divided into two parts – a *scalable GUI component library* and a *scalable GUI event model*. The scalable GUI component library contains a set of device independent GUI components, which are mapped to the equivalent platform-specific GUI components on each device platform. The scalable GUI event model provides *unified* APIs to represent events for scalable GUI components that may receive events generated from different device specific input methods, e.g., clicked by a PC mouse, tapped by a PDA stylus, or pressed by a cell phone keypad. Scalable GUI library also contains a tool to build an intermediate representation, where a seamless application can arrange its GUI components in a logical structure with layout constraints, the task model, and other information. Developers can build this intermediate representation for a seamless application using the scalable GUI library, and the intermediate representation is transformed into the appropriate platform specific presentation through the *task manager* and the *transformation manager* at migration time.

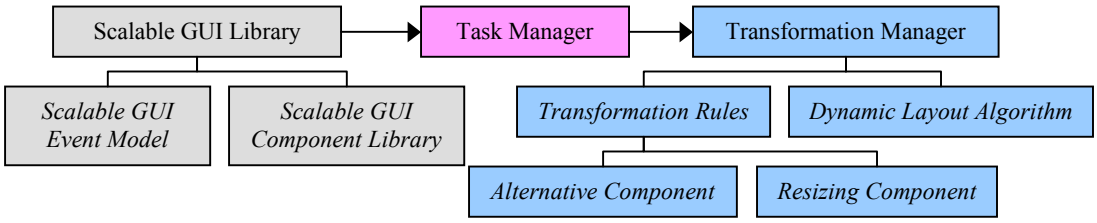

Figure 2: Scalable Graphical User Interface Architecture

The job of the *task manager* is to choose the appropriate tasks for the target device platform and to remove GUI components that are used by tasks not chosen. For an example, it is troublesome for users to input lots of texts on a cell phone using a keypad; as a result, the application may choose not to display fancy GUIs for the editing function on a cell phone. The core of the *transformation manager* is the *dynamic layout algorithm*. Its job is to place SGUI components onto *pages* (or presentation units) according to the layout structure and layout constraints specified by the application, and according to the platform constraint that a page should not exceed the target device display size. To meet the display size constraint, the dynamic layout algorithm can apply *transformation rules* on the SGUI components to generate a set of possible platform-specific presentations of varying sizes. Then it can select the most desirable platform-specific presentation. Transformation manager uses two types of transformation rules – *resizing the GUI components,* or *finding alternative GUI components* that consume less screen space.

4 Future Work

Due to page limitation, we mention only a few of our ongoing/future work:

- We are looking at resource specification that can fully describe the capabilities of the target device.
- Proper security policy must be enforced to ensure that only authorized applications are allowed to migrate from one device to another. We are working on proper security policies and mechanisms for seamless application migration between two trusted personal devices as well as between an un-trusted public device and a trusted personal device.
- The SGUI transformation manager can generate many possible presentations with different layouts and different choices of GUI components. We do not expect our SGUI transformation manager to produce perfect presentation for an application with complex graphical user interface. We are exploring usability analysis that can provide useful feedbacks to refine transformation, and design tools that can allow developers to customize presentation based on the generated GUI.
- Preserving the running state is necessary when a seamless application migrates from one device to another device. However, the use of SGUI transformation can create a situation where the source device has a different GUI presentation than the target device. We are looking at running state translation that can map the running state of one GUI presentation to another GUI presentation.

References

[1] 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,* April, 1997.
[2] S. Fünfrocken, "Transparent Migration of Java-based Mobile Agents (Capturing and Reestablishing the State of Java Programs)", In *Proceedings of the 2nd International Workshop on Mobile Agents*, Stuttgart, Germany, September 1998.
[3] R. S. Gray, D. Kotz, S. Nog, D. Rus, and G. Cybenko, "Mobile Agents for Mobile Computing", In *Proceedings of the Second Aizu International Symposium on Parallel Algorithms/Architectures Synthesis*, Fukushima, Japan, March 1997.
[4] NTT DoCoMo, Inc., "i-mode Java Content Developer's Guide", http://www.nttdocomo.com/i/java/eng_jguide_rel11.pdf, May 2001.
[5] D. B. Lange, M. Oshima, "Mobile Agents with Java: The Aglet API", *World Wide Web Journal*, 1998.
[6] 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.
[7] 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.
[8] Sun Microsystems, "PersonalJava™ Technology -- White Paper", August 1998.
[9] Sun Microsystems, "Java™ 2 Platform Micro Edition (J2ME™) Technology for Creating Mobile Devices, White Paper", May 2000.