CONTEXT-AWARE STATE MANAGEMENT for UBIQUITOUS APPLICATIONS

Pauline P. L. Siu, N. Belaramani, C. L. Wang and F. C. M. Lau

Department of Computer Science,
The University of Hong Kong, Pokfulam, Hong Kong.
{plpsiu, moti, clwang, fcmlau}@cs.hku.hk

Abstract. In a ubiquitous computing environment, users continuously access computing services and interact with smart spaces, while moving from one place to another. Application migration, therefore, is an important and necessary feature. State capturing, migration and restoration play a significant role to enable application migration. However, current software systems usually capture the state at the source and restore it in the destination as it is, without any modification. We believe that application migration in the ubiquitous computing environment has to be context-aware. In this paper, we introduce a context-aware state capturing and restoring mechanism that can achieve context-aware application migration. Our context-aware migration scheme has been implemented in our Sparkle Pervasive Environment, and we demonstrate it with a Universal Browser Application. We believe with this mechanism, applications can be more adaptive to the changing environment as the internal program states will be processed to suit the new context environment at the destination device and mobile code could be brought in from the network to adapt to the new execution context.

Keywords: ubiquitous computing, context-aware, state management, mobility support, component-based system.

1 Introduction

The ultimate vision of ubiquitous computing is to enable users to carry out computing anywhere, anytime and on any device. This introduces the concept of mobile users: users move from one location to another while carrying out their tasks on their devices, such as PDA. On the other hand, users can also move their tasks from one device to another, for example from their PCs to their PDAs. Application migration, thus, becomes an important and necessary feature for ubiquitous environments.

State management, including state capturing, migration, and restoration, plays an important role to enable application migration. State capturing can occur at several levels:

Machine State Level. States of the machine are captured. The states are machine-dependent and hence are compatible with machines only of the same architecture and configuration.

Process State Level. The current process state of the running application, including the program counter, stack and instance variables, are captured.

Object State Level. Object states, including program code, data state, and execution state are captured.

Application State Level. Higher-level states which are user perceivable are captured.

Different systems capture state at different levels, and use it for migration. However, most of them will restore the state without any modification and proceed to resume the execution on the target device. We argue that using the captured state as it is, without any processing is not suitable for ubiquitous environments.

Ubiquitous environments are characterized by heterogeneity. When an application migrates to another device, there may be a big change in context (for example, available computing resources, location, etc.). In addition, applications in ubiquitous environments tend to be context-aware, i.e. the application execution is context-dependent. Thus, restoring a captured state without taking into account the new context will have adverse impact on the execution of the application. Thus we believe that for ubiquitous environments, state management must take into account changes in context during state capturing, and state restoration, i.e. state management has to be context-aware.

We have implemented a state migration mechanism in the Sparkle Pervasive Computing Environment [1]. Our technique encompasses state capturing at the application level. With context information, these application states could be adjusted to suit the environment.

The rest of this paper is organized as follows: Section 2 describes our Mobility Subsystem in the Sparkle System. Section 3 describes the proposed context-aware migration method. Section 4 describes our implementation of a Universal Browser with context-aware state migration. Section 5 discusses various related projects. Conclusions and future work are given in Section 6.

2 Mobility Subsystem in the Sparkle System

Sparkle is an ideal platform for context-aware application-level state management. For all applications running on Sparkle, the application state is centralized in the container. Thus, during migration, only the state in the container needs to be captured, processed, migrated and restored.

In this section, we first discuss the overall design and architecture of our Sparkle system, followed by the details of the mobility subsystem that supports context-aware migration of ubiquitous applications.

2.1 Sparkle System Overview

This section provides a brief overview of the Sparkle Pervasive Computing Environment. Readers may refer to our previous work [1] [2] for more details.

In the Sparkle system, applications are composed of facets. Facets are pure functional units which are independent of data or user interface. A facet has a single publicly callable method and has no residual state. Every facet is associated with a manifest which contains a description of its resource requirements, run-time behavior, context dependencies, etc.

Facets do not interact with the user and do not maintain any application state. Hence, every application is associated with a container. The container contains the user-interface (UI) and stores the data and execution state. It also stores the set of functionalities that the application can offer.

Facets are housed on facet servers on the network. An application, during execution, will request for a certain functionality. There may be more than one facet which fulfills the same functionality. At run-time, the facet which is most suitable for the run-time environment is brought in and executed. Once used, it can be thrown away. Consequently, when you run an application in different contexts, to carry out the same task or to continue a task that has been suspended, the actual components used may be different.

2.2 Mobility Subsystem

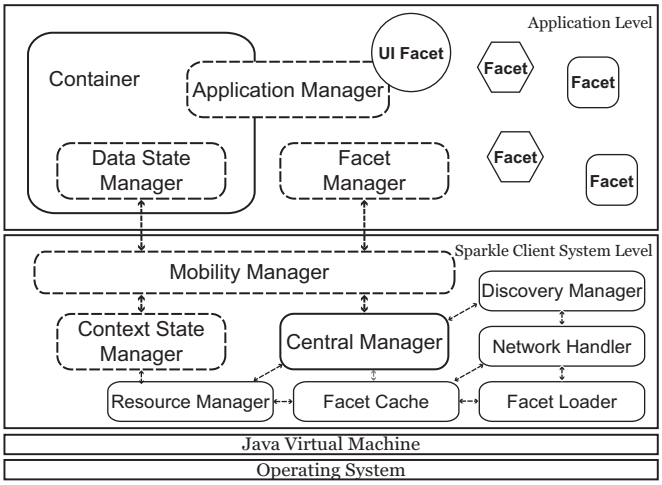

Fig. 1. Design of the Sparkle System with the Mobility subsystem. The Mobility subsystem consists of new components represented in dotted line.

The structure of the Sparkle Client System is shown in Fig. 1. In this section, we discuss several modules which interact with each other to provide context-aware application migration support. They are the Facet Manager, the Data State Manger, the Application Manager, the Context State Manger, the Central Manager and the Mobility Manager.

Facet Manager. The Facet Manager gathers the usage statistics of facets. This helps the Mobility Manager to pre-load facets (on the destination side) when the application migrates.

Data State Manager. When there is a migration request, the Data State Manager is responsible to capture the state of the application, i.e. the state of the container.

Application Manager. The Application Manager acts as the bridge between the Container and the user interface. It eases the job of programmer when saving states, and it helps the communication among graphical components.

Context State Manager. The context of the environment or machine status will change from time to time. The Context State Manager gathers the context information periodically. It gets the current time, location, user schedule, machine status, etc.

Central Manager. It is the central entity of the Sparkle Client System which co-ordinates the activities of the various modules of the client system.

Mobility Manager. This is the core part of the mobility system. The Mobility Manager checks if user needs to migrate. If yes, it will predict the destination. Mobility Manager performs processing on the captured state before (on the source side) and after (on the destination side) application migration.

3 Context-aware Migration

Since context-aware state migration is essential in the ubiquitous environment, in this section, we discuss our migration schemes that have been incorporated in our Sparkle System, followed by the implementation details.

3.1 Context-aware Migration Phases

The migration mechanism takes places in five logical phases as shown in Fig. 2, namely State Capturing (at source), State Process (at source), State Transmission, State Process (at destination) and State Restore (at destination). The details are described below.

State Capturing Phase (at source). The Data State Manager captures the application state and generates an XML document, DataState_XML, representing the state of the application. DataState_XML contains three categories of data states:

(1) *fixed*. States that are essential for the restoration of the application and are context-independent. They will not change under migration.

(2) *mutable*. States that are essential for the restoration of the application but are context-dependent. They may change depending on the context of the application.

(3) *droppable*. States that are not required for the restoration of the application. These may give extra information for the restoration of the application.

DataState_XML will be sent to the Mobility Manager for further processing. The Context State Manager generates ContextState_XML capturing the current

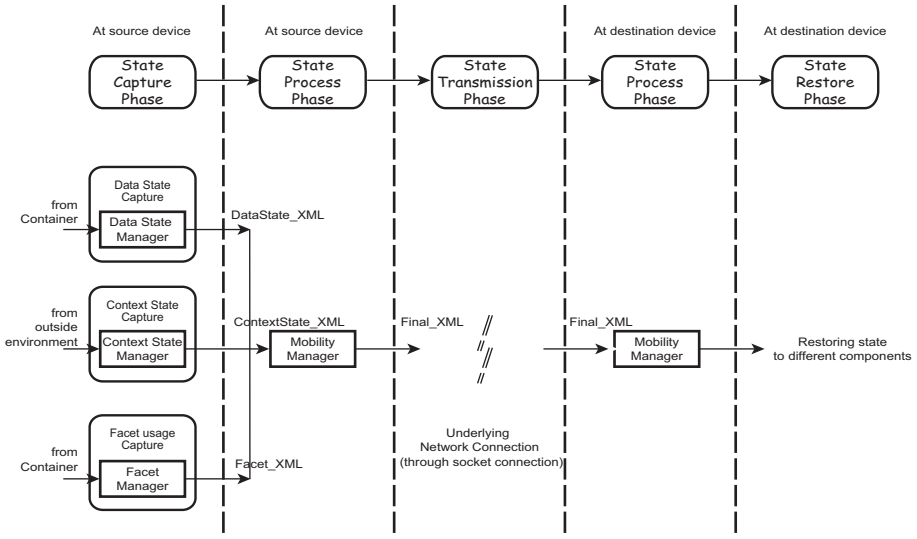

Fig. 2. Context-aware Migration Process

context of the application and passes it on to the Mobility Manager. The Facet Manager gathers the statistics of facet usage and generates Facet_XML to send to the Mobility Manager.

State Process Phase (at source). The Mobility Manager receives Facet_XML, ContextState_XML, and DataState_XML. If it knows where the application is going to migrate to, it will send Facet_XML to the target device for pre-loading of facets. Then, it will adjust the mutable part in DataState_XML to suit the future context. It will also throw away the droppable states if space is at a premium. Finally it will consolidate the state and generate Final_XML.

State Transmission Phase. In this phase, a socket connection will be set up between Sparkle clients. Final_XML will be transferred to the destination device through the underlying network connection.

State Process Phase (at destination). At destination, if the context is different from the predicted one before migration, mutable data states in Final_XML may need to be re-adjusted to suit the current context. The Mobility Manager at destination will obtain the latest ContextState_XML from Context State Manager; then it will analyze Final_XML. On the other hand, if no adjustment on Final_XML is needed, it will go to the next phase, State Restore Phase, immediately.

State Restore Phase (at destination). Lastly, in order to restore the application, the container will be reconstructed. The related data states are restored based on Final_XML.

3.2 Implementation Details

The whole system is implemented in Java. State capturing is done through Java reflection [9]. In Sparkle, programmers can migrate their program by triggering the command *Migrate(destination)*. Once the *Migrate(destination)* method is called, the underlying mechanisms will take care of all the migration related chores. Programmers need not explicitly program the migration details of the application. Byte streams are used for the transmission of states from the source to the destination. At destination, the reconstruction of the container is carried out through dynamic class loading.

4 Universal Browser

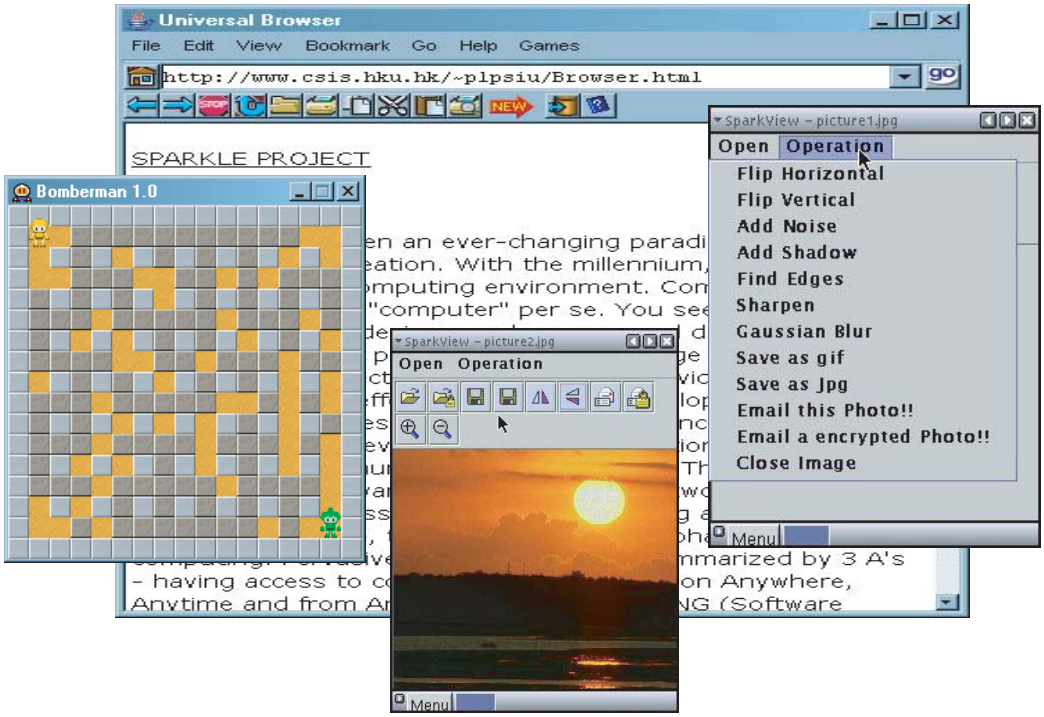

Fig. 3. Screenshot of Universal Browser

In this section, we will discuss the implementation of a Sparkle application, named Universal Browser. Unlike traditional browsing tools, the Universal Browser is not a tool only for browsing HTML files. It is a special user interface for users to run any functions they want. For example, you can play games with

peers nearby, get current location's floor plan, or send messages to your friends. The Universal Browser is built using the facet programming model. Fig. 3 shows the Universal Browser running on a PC.

4.1 Context-aware Migration Support in Universal Browser

When a user migrates the Universal Browser from the PC to the PDA, data state is captured. Fig. 4 shows the states in XML format. The data state will be divided into three categories. For fixed types, they will not be changed no matter how the context changes. For mutable types, they will adjust depending on context. For example, the screen resolution as a mutable type may reduce from 1400×1050 to 320×480. For droppable ones, they will be removed to keep the XML size small.

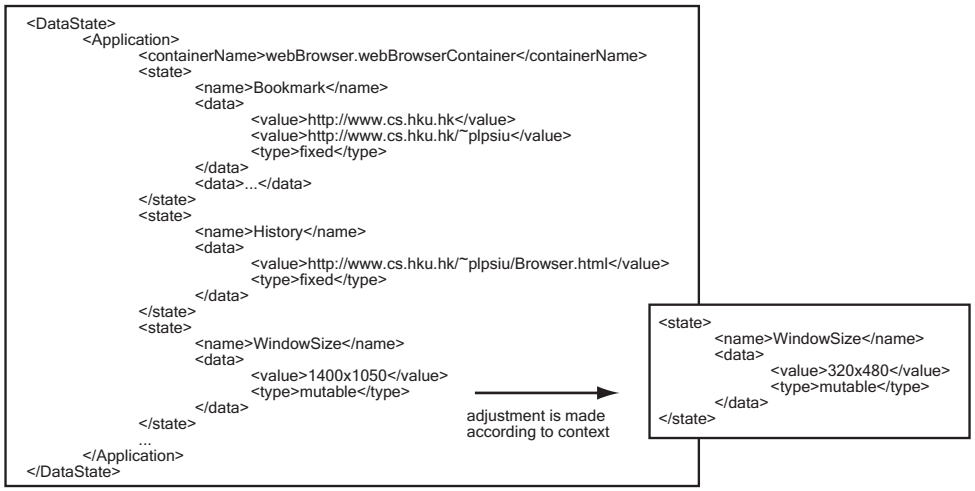

Fig. 4. The DataState_XML: Data states in different categories—fixed, mutable and droppable. For mutable ones, adjustment will be made to suit the future context.

4.2 Development and Performance

The original Universal Browser program has approximately 2,500 lines of code. When the Universal Browser is written using facet programming, the main Universal Browser program is reduced to 1,500 lines. When the Universal Browser needs different functionalities, it needs to bring in facets. These facets are approximately of 50 to 200 lines depending on the facet complexities.

We have measured the migration latency of application migration. Migration latency is the time elapsed from when the migration is initiated, states are captured, and then processed to the moment when the application is completely

restored on the target device. We have measured the latency for the Universal Browser, and the Bomberman and Blackjack games migrating from one desktop PC (Intel Pentium 4 CPU 2.26GHz running Windows 2000) to a notebook PC (Intel Pentium III Mobile CPU 1133 MHz running Windows XP). In addition, we have measured the size of transferred state using our context-aware scheme.

Applications	Migration latency(ms)	Size transferred(bytes)
Universal Browser	3837	1984
Bomberman	4038	2542
Blackjack	3933	2375

Table 1. Migration latency and size transferred

Measurements show that the migration latency of the Universal Browser is less than that of the other two games. This is mainly because less states need to be captured and processed in the Universal Browser. More, the size transferred in the Universal Browser is much less than that of the other two games. This is due to the fact that most of the states in the Universal Browser can be dropped.

5 Related Work

Currently, there are many systems supporting mobility through capturing different kinds of states. Boyd and Dasgupta [3] suggest three migration methods based on the type of state information required: minimal state migration, full state migration, and full distributed state migration. In minimal state migration model, they suggest to collect and restore minimum amount of state information. They have the same idea as ours to move less data when migration is needed. However, their model does not suit the pervasive computing environment since some of the states may change according to external context.

Perimorph [6] is a system that supports compositional adaptation. They suggest the transfer of non-transient state between old components and their replacement, which has a similar goal as our design. However, they only focus on the concept of collateral change. In our system, we focus on state management and adaptation. Our system supports the transformation of state in order to adapt them to the new facet.

ROAM [5], is a seamless application framework that assists developers to build multi-platform applications that can run on heterogeneous devices. They also allow a user to move/migrate a running application among heterogeneous device. In their project, the user interface (presentation) is device-specific. During migration, they need to save all states and map running states between the source presentation and the target presentation. Hence, extra time is needed to

transform states between different GUI components. In our system, we categorize and process data states before migration, and less data is kept and transferred. More, our states are in XML format, which is a machine-independent format, so no transformation of states is needed.

6 Conclusion

Ubiquitous computing environment is the coming trend of the computing world. In such environment, a new software architecture design is needed. With our software infrastructure, Sparkle, it is possible to perform computing and information access anytime, anywhere, from any device. Furthermore, in ubiquitous environment, users are usually of high mobility and contexts are changing continuously. Thus, we have proposed a new mobility system to achieve better application mobility and adaptability. Since most current systems usually transfer captured states to destination devices without further processing, thus, transferred states may not be suitable to the destination context. Hence, our proposed mobility system is designed to capture states, and to do some processing on states before they are transmitted. With the processing of states, the states will suit the next context, making the application more adaptable to the destination environment, and users' satisfaction would be increased.

Acknowledgments. This research has been partly supported by HKU Large Equipment Grant 01021001 and Hong Kong RGC Grant HKU-7519/03E. We would like to thank our colleagues for their assistance.

References

1. N. Belaramani, Y. Chow, V.W.M. Kwan, C.L. Wang, and F.C.M. Lau: A Component-based Software Architecture for Pervasive Computing Intelligent Virtual World: Technologies and Applications in Distributed Virtual Environments, World Scientific Publishing Co.
2. N. Belaramani, C.L. Wang and F.C.M. Lau, "Dynamic Component Composition for Functionality Adaptation in Pervasive Environments," in the 9th International Workshop on Future Trends of Distributed Computing Systems (FTDCS2003), pp. 226-232, San Juan, Puerto Rico, USA, May 28 to 30, 2003.
3. T. Boyd and P. Dasgupta, "Process Migration: A Generalized Approach using a Virtualizing Operating System," in the 22nd International Conference on Distributed Computing Systems, July 2002.
4. G. Chen and D. Kotz., "A Survey of Context-Aware Mobile Computing Research," in Technical Report, Dartmouth Computer Science Technical Report TR2000-381, 2000.
5. H. Chu, H. Song, C. Wong, S. Kurakake, and M. Katagiri "ROAM, A Seamless Application Framework," in Journal of Systems and Software, Vol. 69, Issue 3, pp. 209-226, 2004

6. E. P. Kasten and P. K. McKinley, "Perimorph: Run-Time Composition and State Management for Adaptive Systems," in Proceedings of the Fourth International Workshop on Distributed Auto-adaptive and Reconfigurable Systems (with ICDCS 2004), March 2004.
7. V.W.M. Kwan, F.C.M. Lau, and C.L. Wang, "Functionality Adaptation: A Context-Aware Service Code Adaptation for Pervasive Computing Environments," in IEEE/WIC International Conference on Web Intelligence, Halifax, Canada, October 13-17, 2003.
8. L. Silva, P. Simões, G. Soares, P. Martins, V. Batista, C. Renato, L. Almeida, and N. Stohr., "JAMES: A Platform of Mobile Agents for the Management of Telecommunication Network," in 3rd International Workshop on Intelligent Agents for Telecommunication Applications, Stockholm, Sweden, August 1999.
9. Sun MicroSystems Inc, Java Core Reflection
10. Sun MicroSystems Inc, Java Object Serialization Specification