A Toolkit for Building Migratory Continuous Media Applications

Tatsuo Nakajima and Hiroyuki Aizu
{tatsuo, aizu}@jaist.ac.jp
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi, Ishikawa, 923-1292, JAPAN

Abstract

In this paper, we propose a toolkit for building environment-aware migratory continuous media applications. In future computing environments, various objects in homes and offices will embed computers, and various applications will be moved between these computers according to the location of these users. However, since the computers may have dramatically different hardware and software configurations, the application cannot be moved without taking into account the configuration of the computer that the application is migrated. Therefore, migratory applications should be aware of environments that they are executed.

1 Introduction

In future computing environments, users will expect to use their computing environments in any places even if they go out of their offices. In this case, it is convenient that their applications will be moved with them according to their location. For example, their intelligent agents that monitor their behaviors should give advice to them whenever they need helps. The users expect the agents to be executed in computers that exist near them. Traditionally, the scenario can be realized by carrying computers of respective users, and the agents can be executed on their computers.

The approach requires that the computers may execute heavy computation, and always connect to networks for acquiring information from servers. This means that users should carry powerful but heavy computers with them. The alternative approach is that the agents run on the computers near the agents' users. In future, computers will be embedded in various objects that are placed everywhere, and we want to use these computers for executing applications for helping our work. In the environments, applications will be moved among computers that are embedded in objects existing near their users according to their location. The applications are called *migratory applications*[1, 2].

However, since computers may have dramatically different hardware and software configurations, these applications cannot be moved without taking into account the configurations of the computers that the applications are migrated. Therefore, migratory applications should be aware of environments in which the applications are executed, and should be adapted to respective computing environments to which the applications are moved[4]. Also, these applications may need to process various types of media data such as video and audio for monitoring surrounding environments of the applications' users, or showing video and audio information that is interested by the users in a timely fashion. Therefore, environment-ware migratory continuous media applications are one of the most important classes of migratory applications in future computing environments.

In this paper, we present a system support for building environment-ware migratory continuous media applications. The work described in this paper is one of case studies showing what is required for supporting migratory applications. The experiences with building the system support clarify problems and requirements for realizing migratory applications in large scaled heterogeneous distributed environments.

2 Issues for Supporting Environment-Aware Migratory Continuous Media Applications

For building environment-aware migratory continuous media applications, the following three issues should be taken into account.

- How do applications recognize computing environments in which the applications run, and how do they notify their changes to the applications ?
- How do continuous media applications should be adapted to respective computing environments ?
- How are applications migrated to other computers in the middle of their executions ?

Our system support consists of three components as shown in Figure 1, which answers the above respective issues, and it makes it possible to build environment-aware migratory continuous media applications in a systematic fashion.

The first component that answers the first issue is an environment server that monitors computing environments, and it notifies the changes in the environments to applications. The environment server provides primitives for accessing information about computing environments. Traditionally, respective information about computing environments requires to be accessed by different primitives.

[1] † Currently, Tatsuo Nakajima is a research engineer at Olivetti and Oracle Research Laboratory and a visiting scholar at Department of Engineering of University of Cambridge.

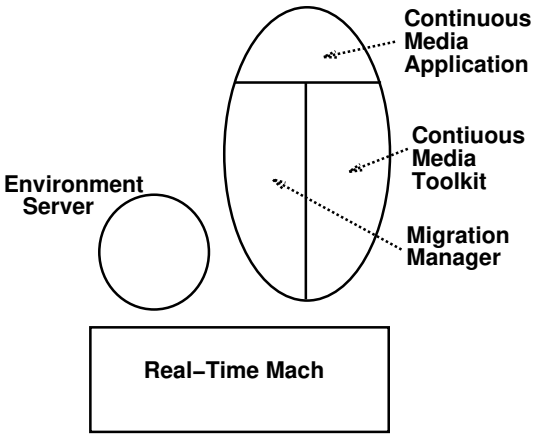

Figure 1: Structure of Environment-Aware Migratory Continuous Media Application

This makes the structures of environment-aware applications complicated. On the other hand, the uniform interface provided by the environment server enables us to build environment-aware applications in a systematic fashion.

The second component that answers the second issue is a continuous media toolkit that enables us to build continuous media applications in a highly configurable way. By using the toolkit, continuous media applications are constructed by composing several modules. The approach has two advantages. The first advantage is that the toolkit enables us to build continuous media applications with a little programming. The second approach is that the toolkit enables applications to change their configurations according to respective computing environments. The advantage makes it possible to build environment-aware continuous media applications systematically.

The last component that answers the third issue is the migration manager. The migration manager that makes continuous media application migratory has two functions. In the first function, the migration manager manages the reconstruction of an application on a computer that the application will be moved. In the second function, it saves the states of applications, and restores the states in applications that are reconstructed on a new computer. In our approach, the amount of states of an application can be changed by taking into account the tradeoff between the time for migrating the application and the bandwidth of a network connected between two computers.

The above three components are currently implemented on Real-Time Mach[8, 5]. The environment server is implemented as a server, and the continuous media toolkit and the migration manager are implemented as libraries that are linked with applications.

3 Environment Server

The *environment server*[7] is important as a basic infrastructure for building environment-aware applications. In traditional operating systems, information about computing environments is managed in a very ad-hoc way, and applications require to access the information using different primitives for respective information. The approach makes the development of applications that are aware of various differences in computing environments very complicated. The *environment server* contains a database that manages environmental information in a uniform way. The interface of the environment server is well defined, and an application can access environmental information by using its uniform interface. This makes it possible to build environment-aware applications with a systematic framework.

4 Toolkit for Building Environment Aware Continuous Media Applications

In our toolkit, continuous media applications are defined as compositions of several modules. This makes it easy to build continuous media applications significantly. Also, the approach enables applications to change their configurations in the middle of the executions. The characteristic makes us to build environment-aware continuous media applications that can adapt to respective computing environments according to the configurations of computers on which the applications run.

Also, our toolkit hides many complicated programming for building continuous media applications such as real-time programming, media synchronization, and dynamic QOS control from programmers for making it easy to build complex continuous media applications[6].

5 Migration Manager

The scripting language that is used for writing environment-aware continuous media applications is an extension of the TCL scripting language. Programmers can write continuous media applications using the script languages since the scripting language provides several primitives for calling the functions of our continuous media toolkit.

5.1 Protocols for Migrating Application

When a continuous media application detects that it should be migrated to other computers, it needs to negotiate to a computer that the application will be migrated. Let us assume that the computer executing the application currently is *computer A* and a new computer that will execute the application after the migration is *computer B*.

First, *computer A* sends a request to *computer B* for creating an application on *computer B*. As described in the previous section, our continuous media application uses the same binary. Thus, *computer B* runs the binary when the request is received. We assume that the binary is created by compiling for respective machine architectures, and it is stored in respective computers that our migratory applications may be migrated.

After creating the application on *computer B*, the newly created application sends a request for acquiring a script from the application executed on *computer A*. Then, the application sends the script to *computer B*. After receiving the script program, the application running on *computer B* configures modules and streams according to the script

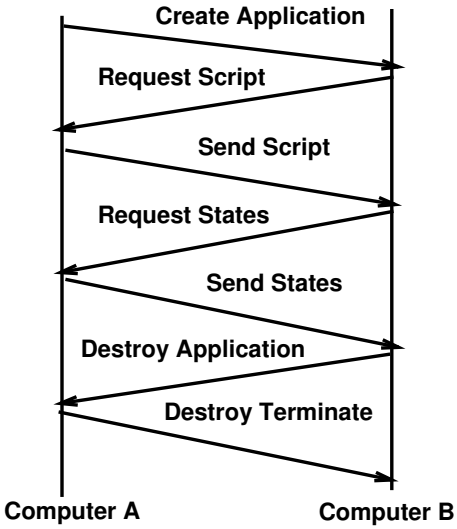

Figure 2: Application Migration Protocol

program as described in Section 5.3. Also, *Initialize* procedure in the script language is executed. The procedure includes necessary initializations for the application on *computer B* such as the registration of conditional statements to the environment server.

Next, the application running on *computer B* sends a request for saving the checkpoint of every module of the application running on *computer A* and transferring the checkpoint as described in Section 5.4. On *computer A*, the checkpoint of an application is taken, and the checkpoint is transformed to the standard format by calling the *Checkpoint* procedure in the script. Then, the checkpoint is transfered to *computer B*. The checkpoint is transformed from the standard format according to the current configuration of the application on *computer B*, and restored in the application's modules by calling the *Restore* procedure in the script. Also, the states for respective streams and GUI are transfered and restored in the application on *computer B*. Lastly, the application running on *computer B* sends a destroy request to the application running on *computer A*. Then, the application on *computer A* executes *Finalize procedure* in the script for unregistering conditional statements in the environment server, and the application is destroyed on *computer A*. Finally, the application on *computer B* starts to be executed after the execution of the destroy request is terminated.

5.2 Reconfiguring Application

In the previous section, we describe how an application is migrated to other computers during the execution. This section describes how the application is adapted according to the configuration of computers that the application is migrated.

Figure 3 shows an example of a continuous media application that captures a video stream from a video camera, converts the video stream, and displays it to a display. Let us assume that the application is migrated between two computer A and B. Computer A has a high resolution video capture, and a window system. When the application is executed on computer A, it connects the capture A module that controls a high resolution video capture board with the convert module that converts a video stream. Also, the convert module is connected with the display A module that draws a video stream to a window. Now, let us assume that the application is migrated to computer B. Since computer B has a low resolution video capture card and a small display, the capture A module and the Display A module cannot be used on the computer B. Therefore, the application changes the configuration by replacing the connections as described in Figure 3 for taking into account the hardware configuration of computer B. After the computer B receives a script program, the application is reconstruct on the computer. Then, the display A module is replaced by the display B module, and the capture A module is replaced by the capture B module. On the other hand, if the application is migrated to computer A again, the configuration of the application is recovered to the original configuration.

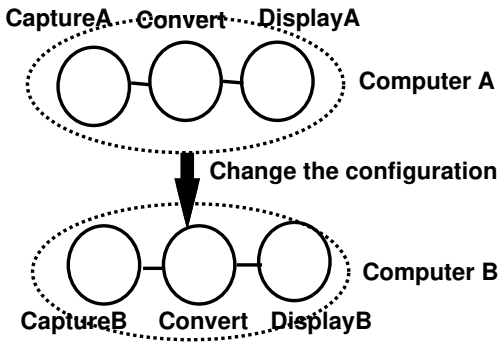

Figure 3: Reconfiguration of Application

In the migration manager, a script program manages the reconfiguration of an application when it is migrated to other computers. When the application is migrated to the computer whose configuration is drastically different from the original computer, the application receives a notification from the environment server, and the script program accesses the environment server for getting the configuration information of the new computer. In the example, the script program acquires information about a capture card and a display device of a new computer from the environment server, and the configuration of the modules is changed by executing the script program.

6 Sample Application

In the example application as described in Figure 4, a computer captures a video stream from a camera in front of a user. Another computer that receives the video stream is displayed to a window. We call the application that captures the video stream a *capture application*, and the application that displays the video stream a *display application*.

Now, let us assume that the *capture application* runs on computer 2, and the *display application* runs on computer 1. The capture application is migrated from computer 2 to computer 3 when the user is moved to near computer

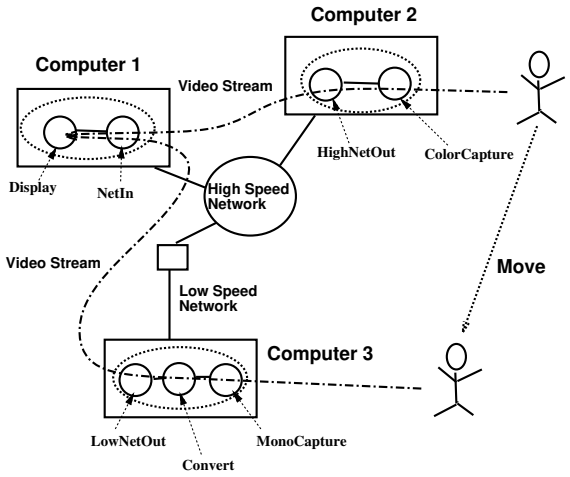

Figure 4: Application Mobility Support

3. Computer 2 is connected to a high speed network, and it has a color capture card. On the other hand, computer 3 is connected to a low speed network, and it has a monochrome capture card.

Before the application is migrated, the capture application is executed on computer 2, and it connects the HighNetOut module and the ColorCapture module. When the application is migrated to computer 3, its configuration is changed as described in Figure 4 since computer 2 and computer 3 have different types of capture cards. Also, since the connected network of computer 3 is low bandwidth, the captured video stream should be scaled down by degrading its quality. In this application, media elements buffered in computer 2 need not to be transfered to computer 3 since these media elements are not important for reconstructing the capture application after its migration.

7 Related Work

Teleporting[10] provides user interface mobility. In this system, when a user moves to other rooms, his windows that he is now working on will be moved to the nearest computer from him according to his location. The system takes into account the different characteristics of display devices that windows are migrated, but the ability is restricted by the limitation of the X window system.

Mobile agents[3, 9] is one of promising approaches for future computing environments. The agent moves among world wide computers and gathers information for helping his user's work. From a view of system research, the most important point of mobile agents is the mobility awareness. This means that the agent receives a notification when it is moved to other computers, and does a job according to the location of the agent. The approach does not take into account the adaptation of the agents according to computing environments. Therefore, traditional mobile agents degrade their performance if the agents are migrated to the computers with the machine configurations that the agents do not assume. On the other hand, our approach solves the problem by changing the configurations of applications according to computing environments by generalizing a notion of migration awareness to a notion of environment-awareness.

8 Conclusion

In this paper, we described a toolkit for supporting environment-aware migratory continuous media applications, and presented three components that we have been developed. The first component is the environment server. The second component is the continuous media toolkit. The last component is the migration manager. The components enable us to build environment-aware migratory applications in a systematic way. We implemented a prototype version of the components on Real-Time Mach, and build an example program of environment-aware continuous media applications.

References

[1] K.A. Bharat, L. Cardelli, "Migratory Applications", SRC Research 138 Report, Digital, 1996.

[2] L. Cardelli, "A Language with Distributed Scope", Computer Systems, 8(1), MIT Press, 1995.

[3] C. Harrison, D. Chess, and A. Kershenbaum, "Mobile Agent: Are they a good idea ?", Research Report RC19887, IBM Research Division, 1995.

[4] A.Hokimoto, K.Kurihara, T.Nakajima, "An Approach for Constructing Mobile Applications using Service Proxies", The 16th International Conference on Distributed Computing Systems, May, 1996.

[5] T.Nakajima,T.Kitayama,H.Arakawa,and H.Tokuda, "Integrated Management of Priority Inversion in Real-Time Mach", *In Proceedings of the Real-Time System Symposium*, 1993

[6] T. Nakajima, "A Toolkit for building Continuous Media Applications", In Proceedings of the 4th International Workshop on Real-Time Computing, Systems, and Applications, 1997.

[7] T. Nakajima, H. Aizu, M. Kobayashi, K. Shimamoto "Environment Server: A System Support for Adaptive Distributed Applications", In Proceedings of the International Conference on World Wide Computing and its Applications'98, 1998.

[8] H. Tokuda, T. Nakajima, and P. Rao, "Real-Time Mach: Towards a Predictable Real-Time System", *In Proceeding of the USENIX 1st Mach Symposium*, October, 1990.

[9] J.E. White, "Telescript Technology: The Foundation for the Electric Marketplace", General Magic Inc., 1994.

[10] K.R. Wood, T. Richardson, F. Bennett, A. Harter, A. Hopper. "Global Teleporting with Java: Toward Ubiquitous Personalized Computing", *Computer* February 1997, Volume 30, Number 2, IEEE.