

System Support for Migratory Continuous Media Applications in Distributed Real-Time Environments

Tatsuo Nakajima, Mamadou Tadiou Kone and Hiroyuki Aizu

Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi, Ishikawa, 923-12, JAPAN

Abstract. In this paper, we propose system support for building adaptive migratory continuous media applications in distributed real-time environments. In future distributed 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. These environments are considered as one of typical future distributed real-time environments. 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 distributed 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' owners. 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*[BC96, Car95].

These computing environments are considered as one of typical distributed real-time environments, and many techniques developed for distributed real-time environments such as real-time scheduling and synchronization can be used for building migratory applications. 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[HKN96]. 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 real-time 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. 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

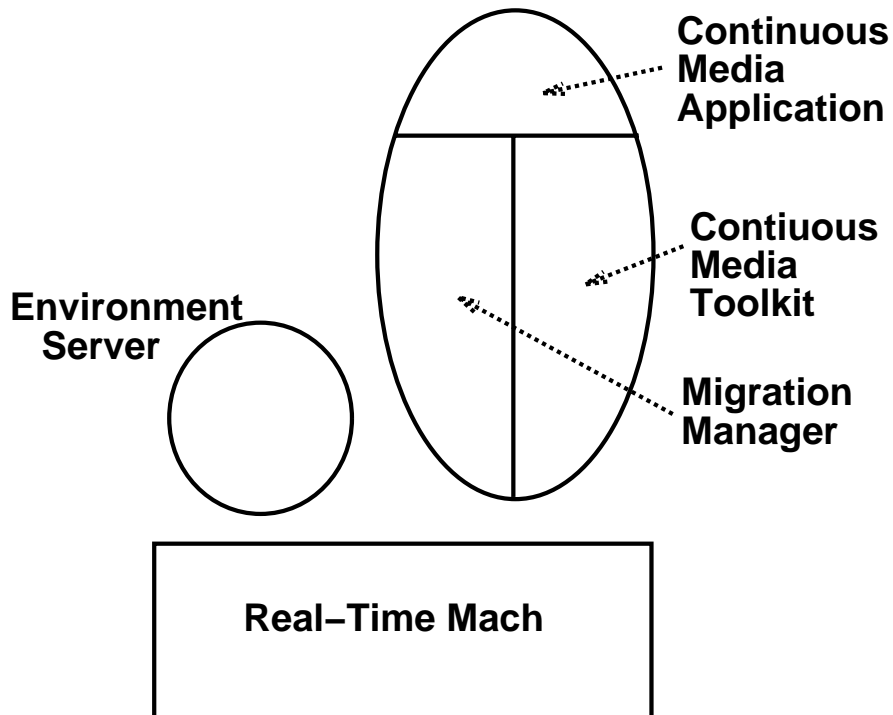


Fig. 1. Structure of Environment-Aware Migratory Continuous Media Application

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 [TNR90, NKAT93]. 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* [NAKS98] 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 Support 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[Nak97].

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*. In the current implementation, 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 program. 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. On *computer A*, the checkpoint of an application is taken,

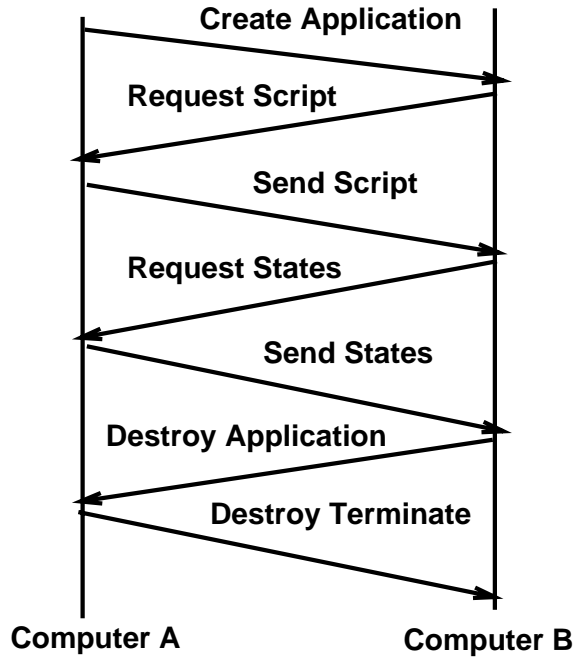


Fig. 2. Application Migration Protocol

and the checkpoint is transformed to the standard format by calling the *Checkpoint* procedure in the script. Then, the checkpoint is transferred 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 transferred 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

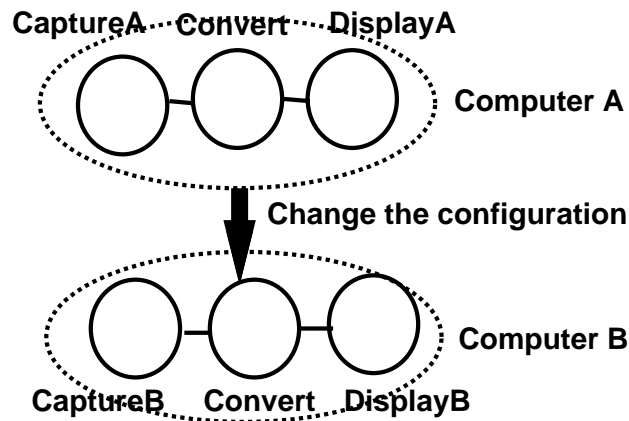


Fig. 3. Reconfiguration of Application

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.

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 Agent Support

Migratory continuous media applications presented in this paper can be considered as mobile agents. In fact, our system provides several supports for im-

plementing mobile agents. This section describes a brief overview of our mobile agent support.

6.1 Basic Model

In the design of the proposed architecture, four entities come into play: a virtual host, a virtual host interface, a place and a resource manager. A place, located inside a virtual host, is responsible for providing the adequate computation resources for the execution of an agent and grants it the desired QOS. A place confines the actions of the agent to a restricted environment for portability and security reasons. Each place is configured specially by an interface according to the requests submitted by the mobile agent. This interface is the *virtual host interface* which is also in charge of translating the subjective user defined QOS parameters into system compliant parameters. The virtual host interface receives mobile agents and translates their data (subjective QOS) into system defined QOS. With this data, it configures an adequate place inside a virtual host. Then, the mobile agent enters that place and uses the services provided there.

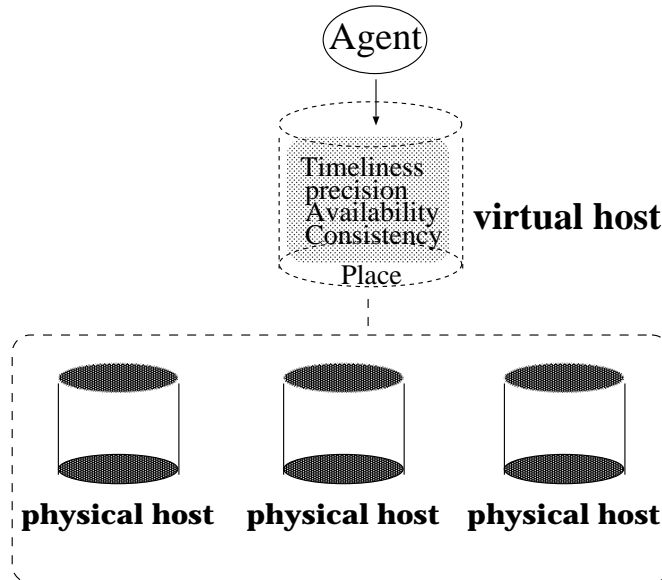


Fig. 4. A Virtual Host Spanning Multiple Machines

The virtual host interface separates the responsibilities of QOS negotiation and resources management. That is to say: the mobile agent system takes care of QOS negotiation at the user level while the virtual host deals with resource management at the machine level. The whole system acts more like a split-QOS server. In fact, we propose in this paper a protocol for QOS control that relies on the movements of mobile agents inside and between places. In order to satisfy

its owner's QOS request, the mobile agent in our system implements an effective policy of QOS control inside places created on demand.

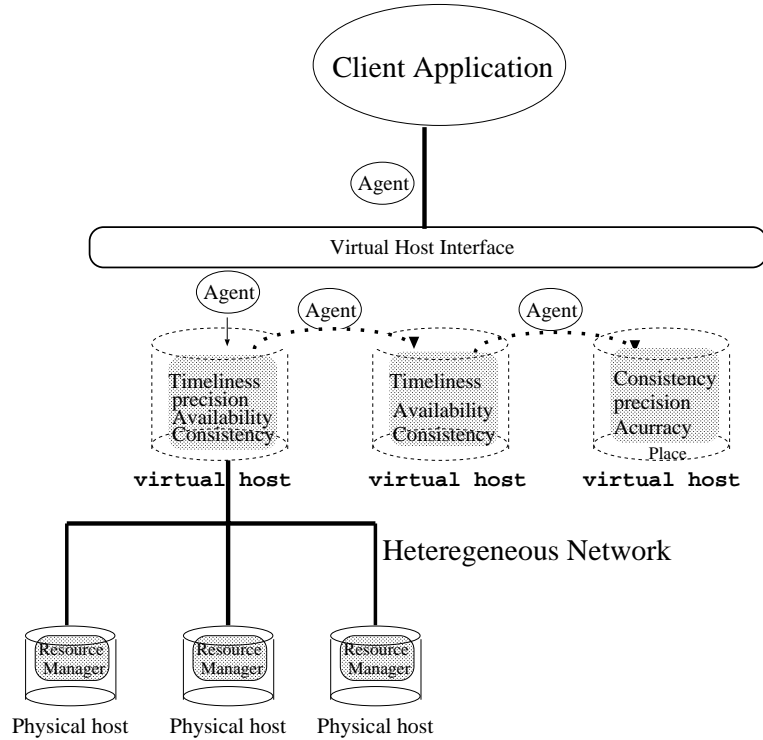


Fig. 5. A Client View of the Mobile Agent System

For example, when an application with strict time constraints like a video application makes a timeliness request together with accuracy, precision and consistency; the system gives precedence to the timeliness QOS and assigns it a high priority. In addition, if each of these QOS type is expressed in terms of “hard” or “soft” guaranty, then some tradeoff becomes necessary and the agent might visit several virtual hosts to achieve the best possible result.

6.2 QOS Assumption

In our system, we assume that a user targets a fixed number of QOS: timeliness for systems with time constraints, accuracy, precision, availability and consistency. We assume also that a number of host machines scattered across the network have the necessary resources to satisfy the user needs. Each place inside a virtual host in conjunction with the adequate resource managers located inside the host machine provides several types of QOS but not necessarily the entire set of QOS. In addition, in our context we assume that a resource reservation

scheme exists at the level of each physical host machine. For matter of simplicity, here we do not deal with the case of multiple applications competing for the same resources.

6.3 Agents Movements

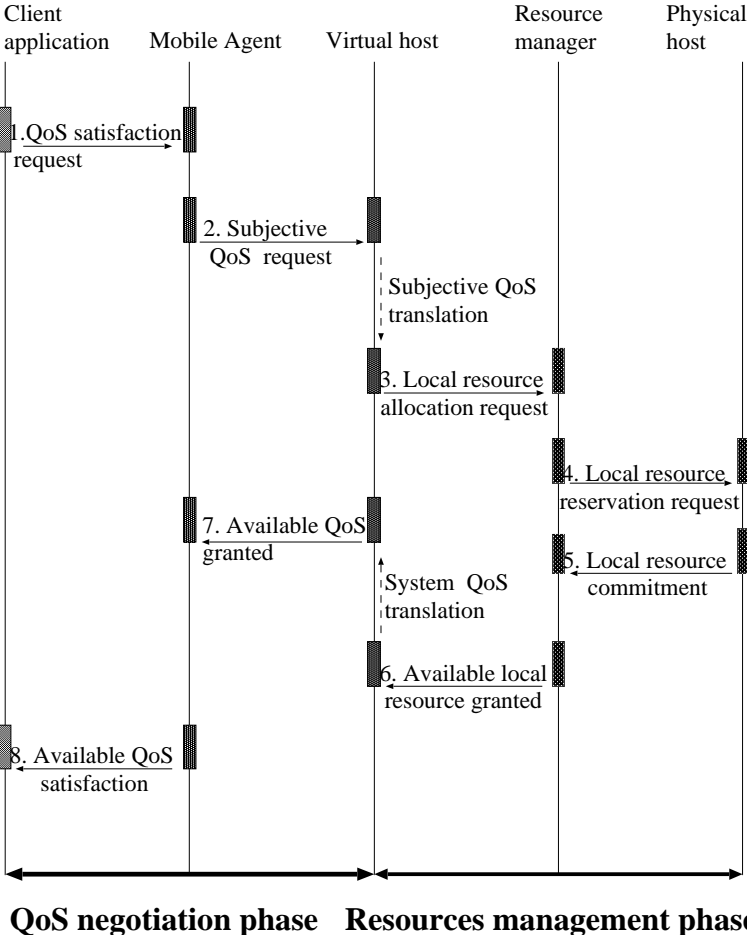


Fig. 6. The QOS Negotiation Process

Multiple QOS requests from users expressed as timeliness, accuracy, precision, availability and consistency may need several resources in order to be satisfied. In this case, the mobile agent in charge migrate with these requests to an appropriate virtual host. Before any operation, the subjective QOS is translated by the virtual host interface into system defined QOS. With this data,

the interface configures a suitable place within the virtual host. Each virtual host spans, over the network, a number of resource managers and host machines that contain the needed resources. When a place is built, the mobile agent freely enters it. Then, it tries to secure the available QOS and migrate - if necessary - to other places to satisfy the complete set of requested QOS. In particular, when a strict time constraint is set by the client application, the mobile agent must assign a high priority to the timeliness parameter and lower priority to other QOS. Assuming that the execution environment is reliable, the agent migrates throughout the entire network, from virtual host to virtual host in order to satisfy all QOS requests.

6.4 Inter-Agents Communication

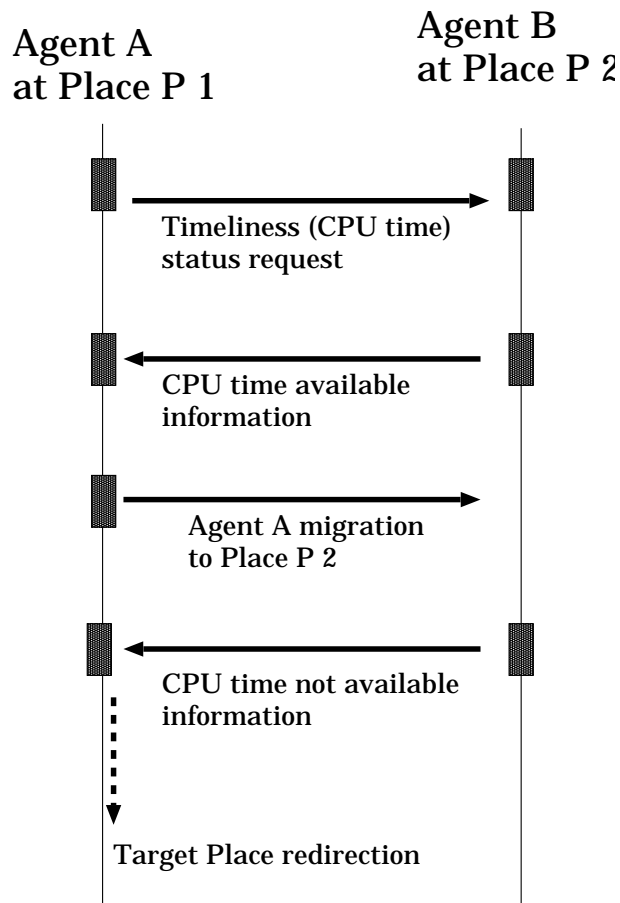


Fig. 7. Inter-Agents Communication

As a lot of agents migrate across the network on behalf of their owner to satisfy different QOS requests, they need to cooperate for optimization purposes. Agents communicate with one another by passing messages about new environmental conditions at their respective place. This way, an agent can have an impact on the computation of other agents. For example, when an agent, on behalf of the real time system user, plans to migrate to a place where timeliness (CPU time) used to be available, another agent staying at the targeted place knows that a failure of the local resource makes this QOS no longer satisfiable. So the latter agent, upon request, sends a message to the first agent which then changes its path. Also, the level of QOS available locally may not be sufficient and the agent in need looks elsewhere for better service. The inter-agent communication is achieved through asynchronous messages rather than Remote Procedure calls (RPC) for flexibility purposes. In this scheme, an agent calls a communication primitive and supplies the identity of the receiving agent and the message as arguments.

7 Conclusion

In this paper, we described system support 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

- [BC96] K.A. Bharat, L. Cardelli, "Migratory Applications", SRC Research 138 Report, Digital, 1996.
- [Car95] L. Cardelli, "A Language with Distributed Scope", Computer Systems, 8(1), MIT Press, 1995.
- [HCK95] C. Harrison, D. Chess, and A. Kershenbaum, "Mobile Agent: Are they a good idea?", Research Report RC19887, IBM Research Division, 1995.
- [HKN96] 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.
- [NKAT93] 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
- [Nak97] T. Nakajima, "A Toolkit for building Continuous Media Applications", In Proceedings of the 4th International Workshop on Real-Time Computing, Systems, and Applications, 1997.
- [NAKS98] 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.
- [TNR90] 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.
- [Whi94] J.E. White, "Telescript Technology: The Foundation for the Electric Marketplace", General Magic Inc., 1994.

[WRB97] 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.

This article was processed using the L^AT_EX macro package with LLNCS style