Grid-based Asynchronous Migration of Execution Context in Java Virtual Machines

Gregor von Laszewski[1], Kazuyuki Shudo[2], and Yoichi Muraoka[2]

[1] Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL, U.S.A.
gregor@mcs.anl.gov

[2] School of Science and Engineering, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo 169–8555, Japan {shudoh,muraoka}@muraoka.info.waseda.ac.jp

Abstract. Previous research efforts for building thread migration systems have concentrated on the development of frameworks dealing with a small local environment controlled by a single user. Computational Grids provide the opportunity to utilize a large-scale environment controlled over different organizational boundaries. Using this class of large-scale computational resources as part of a thread migration system provides a significant challenge previously not addressed by this community. In this paper we present a framework that integrates Grid services to enhance the functionality of a thread migration system. To accommodate future Grid services, the design of the framework is both flexible and extensible. Currently, our thread migration system contains Grid services for authentication, registration, lookup, and automatic software installation. In the context of distributed applications executed on a Grid-based infrastructure, the asynchronous migration of an execution context can help solve problems such as remote execution, load balancing, and the development of mobile agents. Our prototype is based on the migration of Java threads, allowing asynchronous and heterogeneous migration of the execution context of the running code.

1 Introduction

Emerging national-scale *Computational Grid* infrastructures are deploying advanced services beyond those taken for granted in today's Internet, for example, authentication, remote access to computers, resource management, and directory services. The availability of these services represents both an opportunity and a challenge an opportunity because they enable access to remote resources in new ways, a challenge: because the developer of thread migration systems may need to address implementation issues or even modify existing systems designs. The scientific problem-solving infrastructure of the twenty-first century will support the coordinated use of numerous distributed heterogeneous components, including advanced networks, computers, storage devices, display devices, and scientific instruments. The term *The Grid* is often used to refer to this emerging infrastructure [5]. NASA's Information Power Grid and the NCSA Alliance's National Technology Grid are two contemporary projects prototyping Grid systems; both build on a range of technologies, including many provided by the Globus project. Globus is a metacomputing toolkit that provides basic services for security, job submission, information, and communication.

The availability of a national Grid provides the ability to exploit this infrastructure with the next generation of parallel programs. Such programs will include mobile code as an essential tool for allowing such access enabled through *mobile agents*. Mobile agents are programs that can migrate between hosts in a network (or Grid), in order to find places of their own choosing. An essential part for developing mobile agent systems is to save the state of the running program before it is transported to the new host, and restored, allowing the program to continue where it left off. Mobile-agent systems differ from process-migration systems in that the agents move when they choose, typically through a *go* statement, whereas in a process-migration system the system decides when and where to move the running process (typically to balance CPU load) [9]. In an Internet-based environment mobile agents provide an effective choice for many applications as outlined in [11]. Furthermore, this applies also to Grid-based applications. Advantages include improvements in latency and bandwidth of client-server applications and reduction in vulnerability to network disconnection. Although not all Grid applications will need mobile agents, many other applications will find mobile agents an effective implementation technique for all or part of their tasks. The migration system we introduce in this paper is able to support mobile agents as well as process-migration systems, making it an ideal candidate for applications using migration based on the application as well as system requirements.

The rest of the paper is structured as follows. In the first part we introduce the thread migration system MOBA. In the second part we describe the extensions that allow the thread migration system to be used in a Grid-based environment. In the third part we present initial performance results with the MOBA system. We conclude the paper with a summary of lessons learned and a look at future activities.

2 The Thread Migration System MOBA

This paper describes the development of a Grid-based thread migration system. We based our prototype system on the thread migration system MOBA, although many of the services needed to implement such a framework can be used by other implementations.

The name *MOBA* is derived from *MOB*ile *A*gents, since this system was initially applied to the context of mobile agents [17][22][14][15]. Nevertheless, MOBA can also be applied to other computer science–related problems such as the remote execution of jobs [4][8][3]. The advantages of MOBA are threefold:

1. **Support for asynchronous migration**. Thread migration can be carried out without the awareness of the running code. Thus, migration allows entities outside the migrating thread to initiate the migration. Examples for the use of asynchronous migration are global job schedulers that attempt to balance loads among machines. The program developer has the clear advantage that minimal changes to the original threaded code are necessary to include sophisticated migration strategies.
2. **Support for heterogeneous migration**. Thread migration in our system is allowed between MOBA processes executed on platforms with different operating systems. This feature makes it very attractive for use in a Grid-based environment, which is by nature built out of a large number of heterogeneous computing components.

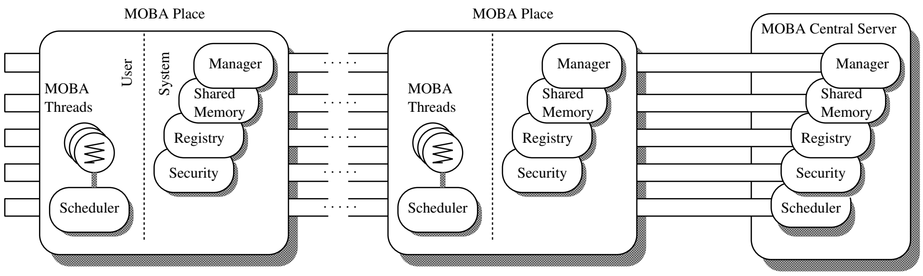

Fig. 1. The MOBA system components include MOBA places and a MOBA central server. Each component has a set of subcomponents that allow thread migration between MOBA places.

3. **Support for the execution of native code as part of the migrating thread.** While considering a thread migration system for Grid-based environments, it is advantageous to enable the execution of native code as part of the overall strategy to support a large and expensive code base, such as in scientific programming environments. MOBA will, in the near future, provide this capability. For more information on this subject we refer the interested reader to [17].

2.1 MOBA System Components

MOBA is based on a set of components that are illustrated in Figure 1. Next, we explain the functionality of the various components:

Place. Threads are created and executed in the *MOBA place* component. Here they receive external messages to move or decide on their own to move to a different place component. A MOBA place accesses a set of MOBA system components, such as manager, shared-memory, registry, and security. Each component has a unique functionality within the MOBA framework.

Manager. A single point of control is used to provide the control of startup and shutdown of the various component processes. The manager allows the user to get and set the environment for the respective processes.

Shared Memory: This component shares the data between threads.

Registry: The registry maintains necessary information — both static and dynamic — about all the MOBA components and the system resources. This information includes the OS name and version, installed software, machine attributes, and the load on the machines.

Security: The security component provides network-transparent programming interfaces for access control to all the MOBA components.

Scheduler: A MOBA place has access to user-defined components that handle the execution and scheduling of threads. The scheduling strategy can be provided through a custom policy developed by the user.

2.2 Programming Interface

We have designed the programming interface to MOBA on the principle of simplicity. One advantage in using MOBA is the availability of a user-friendly programming interface. For example, with only one statement, the programmer can instruct a thread to migrate; thus, only a few changes to the original code are necessary in order to augment an existent thread-based code to include thread migration. To enable movability of a thread, we instantiate a thread by using the `MobaThread` class instead of the normal Java `Thread` class. Specifically, the `MobaThread` class includes a method, called `goTo`, that allows the migration of a thread to another machine. In contrast to other mobile agent systems for Java [10][12][6], programmers using MOBA can enable thread migration with minor code modifications.

An important feature of MOBA is that migration can be ordered not only by the migrant but also by entities outside the migrant. Such entities include even threads that are running in the context of another user. In this case, the statement to migrate is included not in the migrant's code but in the thread that requests the move into its own execution context. To distinguish this action from the `goTo`, we have provided the method `moveTo`.

2.3 Implementation

MOBA is based on a specialized version of the Java Just-In-Time (JIT) interpreter. It is implemented as a plug-in to the Java Virtual Machine (JVM) provided by Sun Microsystems. Although MOBA is mostly written in Java, a small set of C functions enables efficient access to perform reflection and to obtain thread information such as the stack frames within the virtual machine. Currently, the system is supported on operating systems on which the Sun's JDK 1.1.x is ported. A port of MOBA based on JDK 1.2.x is currently under investigation. Our system allows heterogeneous migration [19] by handling the execution context in JVM rather than on a particular processor or in an operating system. Thus, threads in our system can migrate between JVMs on different platforms.

2.4 Organization of the Migration Facilities

To facilitate migration within our system, we designed MOBA as a layered architecture. The migration facilities of MOBA include introspection, object marshaling, thread externalization, and thread migration. Each of these facilities is supported and accessed through a library. The relationship and dependency of the migration facilities are depicted in Figure 2. The introspection library provides the same function as the reflection library that is part of the standard library of Java. Similarly, object marshaling provides the function of serialization, and thread externalization translates a state of the running thread to a byte stream.

The steps to translate a thread to a byte stream are summarized in Figure 3. In the first step, the attributes of the thread are translated. Such attributes include the name of the thread and thread priority. In the second step, all objects that are reachable from the thread object are marshaled. Objects that are bound to file descriptors or other local resources are excluded from a migration. In the final step, the execution context is serialized. Since a context consists of contents of stack frames generated by a chain of method invocations, the externalizer follows the chain from older frames to newer ones and serializes the contents of the frames. A frame is located on the stack in a JVM and contains the state of a calling method. The state consists of a program counter, operands to the method, local variables, and elements on the stack, each of which is serialized in machine-independent form.

Together the facilities for externalizing threads and performing thread migration enabled us to design the components necessary for the MOBA system and to enhance the JIT compiler in order to allow asynchronous migration.

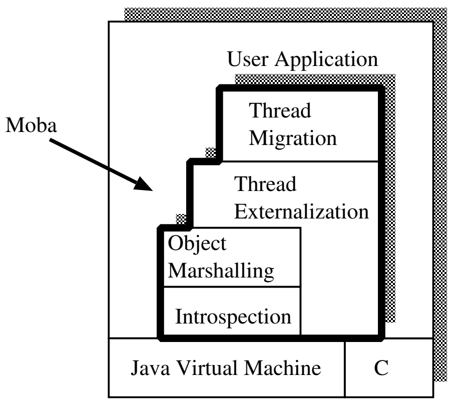

Fig. 2. Organization of MOBA thread migration facilities and their dependencies.

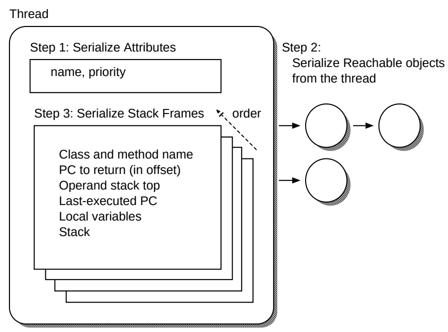

Fig. 3. Procedure to externalize a thread in MOBA.

2.5 Design Issues of Thread Migration in JVMs

In designing our thread migration system, we faced several challenges. Here we focus on five.

Nonpreemptive Scheduling. In order to enable the migration of the execution context, the migratory thread must be suspended at a *migration safe point*. Such migration safe points are defined within the execution of the JVM whenever it is in a consistent state. Furthermore, asynchronous migration within the MOBA system requires nonpreemptive scheduling of Java threads to prevent threads from being suspended at a not-safe point. Depending on the underlying (preemptive or nonpreemptive) thread scheduling system used in the JVM, MOBA supports either asynchronous or cooperative migration (that is, the migratory thread determines itself the destination). The availability of green threads will allow us to provide asynchronous migration.

Native Code Support. Most JVMs have a JIT runtime compiler that translates bytecode to the processors native code at runtime. To enable heterogeneous migration, a machine-independent representation of execution context is required. Unfortunately, most existing JIT compilers do not preserve a program counter on bytecode which is needed to reach a migration safe point. Only the program counter of the native code execution can be obtained by an existing JIT compiler. Fortunately, Sun's HotSpot VM [18] allows the execution context on bytecode to be captured during the execution of the generated native code since capturing the program counter on bytecode is also used for its dynamic deoptimization.

We are developing an enhanced version of the JIT compiler that checks, during the execution of native code, a flag indicating whether the request for capturing the context can be performed. This polling may have some cost in terms of performance, but we expect any decrease in performance to be small.

Selective Migration. In the most primitive migration system all objects reachable from the thread object are marshaled and replicated on the destination of the migration. This approach may cause problems related to limitations occuring during the access of system resources as documented in [17]. Selective migration may be able to overcome these problems, but the implementation is challenging because we must develop an algorithm determining the objects to be transferred. Additionally, the migration system must cooperate with a distributed object system, enabling remote reference and remote operation. Specifically, since the migrated thread must allow access to the remaining objects within the distributed object system, it must be tightly integrated within the JVM. It must allow the interchange of a local references and a remote references to support remote array access, field access, transparent replacement of a local object with a remote object, and so forth. Since no distributed object system implemented in Java (for example, Java RMI, Voyager, HORB, and many implementations of CORBA) satisfies these requirements, we have developed a distributed object system supported by the JIT compiler shuJIT [16] to provide these capabilities.

Marshaling Objects Tied to the Local Resource. A common problem in object migration systems is how to maintain objects that have some relation to resources specific to, say, a machine. Since MOBA does not allow to access objects that reside in a remote machine directly, it must copy or migrate the objects to the MOBA place issuing the request. Objects that depend on local resources (such a file and socket descriptors) are not moved within MOBA, but remain at the original fixed location [8][13].

Types of Values on the JVM Stack. In order to migrate an object from one machine to another, it is important to determine the type of the local object variables. Unfortunately, Sun's JVM does not provide a type stack operating in parallel to the value stack, such as the Sumatra interpreter [1]. Local variables and operands of the called method stay on the stack. The values may be 32-bit or 64-bit immediate values or references to objects. It is difficult to distinguish the types only by their values.

With a JVM like Sun's, we have either to infer the type from the value or to determine the type by a data flow analysis that traces the bytecode of the method (like a bytecode verifier). Since tracing bytecode to determine types is computationally expensive, we developed a version of MOBA that infers the type from the value. Nevertheless, we recently determined that this capability is not sufficient to obtain a perfect inference and validation method. Thus, we are developing a modified JIT compiler that will provide stack frame maps [2] as part of Sun's ResearchVM.

3 Moba/G Service Requirements

The thread migration system MOBA introduced in the preceding sections is used as a basis for a Grid-enhanced version which we will call MOBA/G. Before we describe the MOBA/G system in more detail, we describe a simple Grid-enhanced scenario to outline our intentions for a Grid-based MOBA framework. First, we have to determine a subset of compute resources on which our MOBA system can be executed. To do so, we query the Globus Metacomputing Directory Service (MDS) while looking for compute resources on which Globus and the appropriate Java VM versions are installed and on which we have an account. Once we have identified a subset of all the machines returned by this query for the execution of the MOBA system, we transfer the necessary code base to the machine (if it is not already installed there). Then we start the MOBA places and register each MOBA place within the MDS. The communication between the MOBA places is performed in a secure fashion so that only the application user can decrypt the messages exchanged between them. A load-balancing algorithm is plugged into the running MOBA system that allows us to execute our thread-based program rapidly in the dynamically maintained MOBA places. During the execution of our program we detect that a MOBA place is not responding. Since we have designed our program with check-pointing, we are able to start new MOBA places on underutilized resources and to restart the failed threads on them. Our MOBA application finishes and deregisters from the Grid environment.

To derive such a version, we have tried to ask ourselves several questions:

1. What existent Grid services can be used by MOBA to enhance is functionality?
2. What new Grid services are needed to provide a Grid-based MOBA system?
3. Are any technological or implementation issues preventing the integration?

To answer the first two questions, we identified that the following services will be needed to enhance the functionality of MOBA in a Grid-based environment:

Resource Location and Monitoring Services. A resource location service is used to determine possible compute nodes on which a MOBA place can be executed. A monitoring service is used to observe the state and status of the Grid environment to help in scheduling the threads in the Grid environment. A combination of Globus services can be used to implement them.

Authentication and Authorization Service. The existent security component in MOBA is based on a simple centralized maintenance based on user accounts and user groups known in a typical UNIX system. This security component is not strong enough to support the increased security requirements in a Grid-based environment. The Globus project, however, provides a sophisticated security infrastructure that can be used by MOBA. Authentication can be achieved with the concept of public keys. This security infrastructure can be used to augment many of the MOBA components, such as shared memory and the scheduler.

Installation and Execution Service. Once a computational resource has been discovered, an installation service is used to install a MOBA place on it and to start the MOBA services. This is a significant enhancement to the original MOBA architecture as it allows the shift from a static to a dynamic pool of resources. Our intention is to extend a component in the Globus toolkit to meet the special needs of MOBA.

Secure Communication Service. Objects in MOBA are exchanged over the IIOP protocol. One possibility is to use commercial enhancements for the secure exchange of messages between different places. Another solution is to integrate the Globus security infrastructure. The Globus project has initiated an independent project investigating the development of a CORBA framework using a security enhanced version of IIOP.

The services above can be based on a set of existing Grid services provided by the Globus project (compare Table 1). For the integration of MOBA and Globus we need consider only those services and components that increase the functionality of MOBA within a Grid-based environment.

Table 1. The Globus services that are used to build the MOBA/G thread migration system within a Grid-based environment. Services that are not available in the initial MOBA system are indcated with •.

MOBA/G Service	Service	Globus Component
MOBA Place startup	Resource Management	GRAM
MOBA Object migration	Communication	GlobusIO
• Secure Communication, Authentication, Secure component startup	Security	GSI
MOBA registry	Information	MDS
• Monitoring	Health and Status	HBM, NWS
• Remote Installation, Data Replication	Remote Data Access	GASS

Before we explain in more detail the integration of each of the services into the MOBA system, we point out that many of the services are accessible in Java through the Java CoG Kit. The Java CoG Kit [20][21] not only allows access to the Globus services, but also provides the benefit of using the Java framework as the programming model. Thus, it is possible to cast the services as JavaBenas and to use the sophisticated event and thread models as used in the programs to support the MOBA/G implementation. The relationship between Globus, the Java CoG Kit, and MOBA/G is based on a layered architecture as depicted in Figure 4.

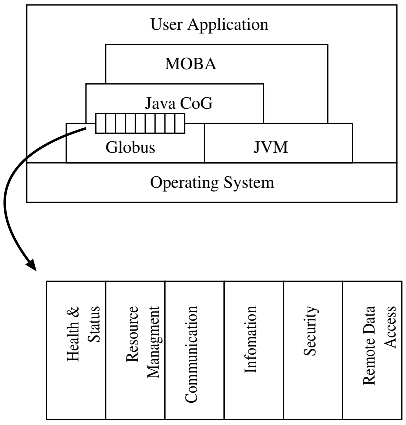

Fig. 4. The layered architecture of MOBA/G. The Java CoG Kit is used to access the various Globus Services.

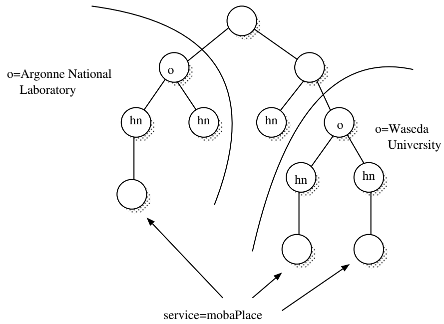

Fig. 5. The organizational directory tree of a distributed MOBA/G system between two organizations using three compute resources (hn) for running MOBA places.

3.1 Grid-based Registration Service

One of the problems a Grid-based application faces is to identify the resources on which the application is executed. The Metacomputing Directory Service enables Grid application developers and users to register their services with the MDS. The Grid-based information service could be used in several ways:

1. The existing MOBA central registry could register its existence within the MDS. Thus all MOBA services would still interact with the original MOBA service. The advantage of including the MOBA registry within the MDS is that multiple MOBA places could be started with multiple MOBA registries, and each of the places could easily locate the necessary information from the MDS in order to set up the communication with the appropriate MOBA registry.

2. The information that is usually contained within the MOBA registry could be stored as LDAP objects within the distributed MDS. Thus, the functionality of the original MOBA registry could be replaced with a distributed registry based on the MDS functionality.

3. The strategies introduced in (1) and (2) could be mixed while registering multiple enhanced MOBA registries. These enhanced registries would allow the exchange of information between each other and thus function in a distributed fashion.

Which of the methods introduced above is used depends on the application. Applications with high throughput demand but few MOBA places are sufficiently supported by the original MOBA registry. Applications that have a large number of MOBA places but do not have high demands on the throughput benefit from a total distributed registry in the MDS. Applications that fall between these classes benefit from a modified MOBA distributed registry.

We emphasize that a distributed Grid-based information service must be able to deal in most cases with organizational boundaries (Figure 5). All of the MDS-based solutions discussed above provide this nontrivial ability.

3.2 Grid-based Installation Service

In a Grid environment we foresee the following two possibilities for the installation of MOBA: (1) MOBA and Globus are already installed on the system, and hence we do not have to do anything; and (2) we have to identify a suitable machine on which MOBA can be installed. The following steps describe such an automatic installation process:

1. Retrieve a list of all machines that fulfill the installation requirements (e.g., Globus, JDK1.1, a particular OS-version, enough memory, accounts on which the user has access, platform-supported green-threads).
2. Select a subset of these machines on which to install MOBA.
3. Use a secure Grid-enabled ftp program to download MOBA in an appropriate installation space, and uncompress the distribution in this space.
4. Configure MOBA while using the provided auto-configure script, and complete the installation process.
5. Test the configuration, and, if successful, report and register the availability of MOBA on the machine.

3.3 Grid-based Startup Service

Once MOBA is installed on a compute resource and a user decides to run a MOBA place on it, it has to be started together with all the other MOBA services to enable a MOBA system. The following steps are performed in order to do so:

1. Obtain the authentication through the Globus Security service to access the appropriate compute resource.
2. List all the machines on which a user can start a MOBA place.
3. For each compute resource in the list, start MOBA through the Java CoG interface to the Globus remote job startup service.

Depending on the way the registry service is run, additional steps may be needed to start it or to register an already running registry within the MDS.

3.4 Authentication and Authorization Service

In contrast to the existing MOBA security system, the Grid-based security service is far more sophisticated and flexible. It is based on GSI and allows integration with public keys as well as with Kerberos. First, the user must authenticate to the system. Using this Grid-based single-sign on security service allows the user to gain access to all the resources in the Grid without logging onto the various machines on the Grid environment on which the user has accounts, with potential different user names and passwords. Once authenticated, the user can submit remote job request that are executed with the appropriate security authorization for the remote machine. In this way a user can access remote files, create threads in a MOBA place, and initiate the migration of threads between MOBA places.

3.5 Secure Communication Service

The secure communication can be enabled while using the GlobusIO library and sending messages from one Globus machine to another. This service allows one to send any serializable object or simple message (e.g., thread migration, class file transfer, and commands to the MOBA command interpreter) to other MOBA places executed under Globus-enabled machines.

4 Conclusion

We have designed and implemented migration system for Java threads as a plug-in to an existing JVM that supports asynchronous migration of execution context. As part of this paper we discussed various issues, such as whether objects reachable from the migrant should be moved, how the types of values in the stack can be identified, how compatibility with JIT compilers can be achieved, and how system resources tied to moving objects should be handled. As a result of this analysis, we are designing a JIT compiler that improves our current prototype. It will support asynchronous and heterogeneous migration with execution of native code. The initial step to such a system is already achieved because we have already implemented a distributed object system based on the JIT compiler to support selective migration. Although this is an achievement by itself, we have enhanced our vision to include the emerging Grid infrastructure. Based on the availability of mature services provided as part of the Grid infrastructure, we have modified our design to include significant changes in the system architecture. Additionally, we have identified services that can be used by other Grid application developers. We feel that the integration of a thread migration system in a Grid-based environment has helped us to shape future activities in the Grid community, as well as to make improvements in the thread migration system.

Acknowledgments

This work was supported by the Research for the Future (RFTF) program launched by Japan Society for the Promotion of Science (JSPS) and funded by the Japanese government. The work performed by Gregor von Laszewski work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. Globus research and development is supported by DARPA, DOE, and NSF.

References

1. Anurag Acharya, M. Ranganathan, and Joel Saltz. Sumatra: A language for resource-aware mobile programs. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems*. Springer Verlag Lecture Notes in Computer Science, 1997.
2. Ole Agesen. GC points in a threaded environment. Technical Report SMLI TR-98-70, Sun Microsystems, Inc., December 1998. http://www.sun.com/research/jtech/pubs/.

3. Bozhidar Dimitrov and Vernon Rego. Arachne: A portable threads system supporting migrant threads on heterogeneous network farms. *IEEE Transaction on Parallel and Distributed Systems*, 9(5):459–469, May 1998.
4. M. Raşit Eskicioğlu. Design Issues of Process Migration Facilities in Distributed System. *IEEE Technical Comittee on Operating Systems Newsletter*, 4(2):3–13, Winter 1989. Reprinted in *Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press.
5. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann, 1998.
6. General Magic, Inc. Odyssey information. http://www.genmagic.com/technology/odyssey.html.
7. Satoshi Hirano. HORB: Distributed execution of Java programs. In *Proceedings of World Wide Computing and Its Applications*, March 1997.
8. Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-Grained Mobility in the Emerald System. *ACM Transaction on Computer Systems*, 6(1):109–133, February 1988.
9. David Kotz and Robert S. Gray. Mobile agents and the future of the internet. *ACM Operating Systems Review*, 33(3):7–13, August 1999.
10. Danny Lange and Mitsuru Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley Longman, Inc., 1998.
11. Danny B. Lange and Mitsuru Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, March 1999.
12. ObjectSpace, Inc. Voyager. http://www.objectspace.com/products/Voyager/.
13. M. Ranganathan, Anurag Acharya, Shamik Sharma, and Joel Saltz. Network-aware mobile programs. In *Proceedings of USENIX'97*, January 1997.
14. Tatsuro Sekiguchi, Hidehiko Masuhara, and Akinori Yonezawa. A simple extension of Java language for controllable transparent migration and its portable implementation. In *Springer Lecture Notes in Computer Science for International Conference on Coordination Models and Languages(Coordination99)*, 1999.
15. Tatsurou Sekiguchi. JavaGo manual, 1998. http://web.yl.is.s.u-tokyo.ac.jp/amo/JavaGo/doc/.
16. Kazuyuki SHUDO. shuJIT—JIT compiler for Sun JVM/x86, 1998. http://www.shudo.net/jit/.
17. Kazuyuki Shudo and Yoichi Muraoka. Noncooperative Migration of Execution Context in Java Virtual Machines. In *Proc. of the First Annual Workshop on Java for High-Performance Computing (in conjunction with ACM ICS'99)*, Rhodes, Greece, June 1999.
18. Inc. Sun Microsystems. The Java HotSpot performance engine architecture. http://www.javasoft.com/products/hotspot/ whitepaper.html.
19. Marvin M. Theimer and Barry Hayes. Heterogeneous Process Migration by Recompilation. In *Proc. IEEE 11th International Conference on Distributed Computing Systems*, pages 18–25, 1991. Reprinted in *Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press.
20. Gregor von Laszewski and Ian Foster. Grid Infrastructure to Support Science Portals for Large Scale Instruments. In *Proc. of the Workshop Distributed Computing on the Web (DCW)*, pages 1–16, Rostock, June 1999. University of Rostock, Germany.
21. Gregor von Laszewski, Ian Foster, Jarek Gawor, Warren Smith, and Steve Tuecke. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In *ACM 2000 Java Grande Conference*, San Francisco, California, June 3-4 2000. http://www.extreme.indiana.edu/java00.
22. James E. White. *Telescript Technology: The Foundation of the Electronic Marketplace*. General Magic, Inc., 1994.
23. Ann Wollrath, Roger Riggs, and Jim Waldo. A Distributed Object Model for the Java System. In *The Second Conference on Object–Oriented Technology and Systems (COOTS) Proceedings*, pages 219–231, 1996.