Old Wine in New Bottles

Applying OS Process Migration Technology to Mobile Agents

Dejan S. Milojicic, Shai Guday and Richard Wheeler
{d.milojicic, s.guday, r.wheeler}@opengroup.org

The Open Group Research Institute
11 Cambridge Centre
Cambridge, MA 02142

Abstract

We describe how our experience with classic operating system level process migration mechanisms influenced the design of MOA, a mobile agents project. A case is presented for implementing mobile agents using language environments and overcoming the limitations of those environments by employing mechanisms adapted from classic process migration implementations.

1 Introduction

In this paper, we describe how we have applied our experience with classic operating system level process migration mechanisms to mobile agents. The need to have agents that migrate between nodes with different architectures and operating systems favors moving support for mobility from the operating system level up to the language environment levels, but the wealth of experience with the operating system level mechanisms should be used to guide these higher level implementations.

The particular area of mobile agents is but one example of operating system functionality moving to other system levels. The operating system program here at The Open Group Research Institute no longer addresses only the development of new operating systems, but rather it also demonstrates how operating system techniques apply to different system levels. A second example of the use of OS mechanisms in higher system levels is the SHAWS project [22]. SHAWS, the **S**calable, **H**ighly **A**vailable **W**eb **S**erver, uses operating system techniques, such as fault-tolerance and load balancing, to enhance off-the-shelf Web browsers and servers.

Although the language environments and Web interfaces used by mobile agents make it irrelevant which base operating system is used, most of the functionality traditionally provided by operating systems is still required. It remains an open and active area of research how and on what system level this support should be provided.

The rest of this document is organized in the following manner. Section 2 describes operating system support for mobility. Section 3 provides a general description of mobile agents. Section 4 discusses benefits and challenges of language environments for mobile agents. Section 5 describes the Mobile Objects and Agents project at the Open Group Research Institute and how we applied our experience with OS migration to mobile agents. Section 6 overviews related work and Section 7 concludes the paper.

2 Operating System Support for Mobility

Process migration is a well researched area. Process migration mechanisms usually rely on help from the underlying operating system, both when implemented inside the operating system, as in MOSIX [7], Sprite [10], Accent [31], and the V kernel [27], or when implemented on top of it, as done in Mach [13] and UNIX [4]. A survey of process migration can be found in [15]. The following examples represent some of the support useful for mobility of processes:

Naming, addressing and locating processes. In operating systems that support process migration, giving processes unique names was difficult. Process names, typically process identifiers (PIDs), have to maintain existing single node semantics and yet uniquely identify each process in the global namespace. The complexity involved is proportional to the desired degree of transparency. For example, many systems encode the process' creating host identifier in the PID which has ramifications for scalability.

Process migration used several schemes to track and maintain up to date information about the location of migrated processes. One popular scheme, employed in MOSIX [7] and Sprite [10], used the creation site, or *home machine*, of each process to track its location. On migration, the home machine was always informed about the process' new location. Charlotte [6] used a forwarding scheme which basically left a forwarding address at each node visited. This scheme is useful when agents make a small number of migrations. Searching schemes, similar to those used in the V Kernel [27], can be applied in cases where the number of nodes to be visited is small.

Controlling migrating processes. Given a process' current location, various mechanisms were used to provide control. For example, to terminate or suspend a remote process, a local proxy can be used. In Locus and OSF/1 AD, the *vproc* acts as a local proxy, keeping track of the current process location, and forwarding process requests to the process' current node. In Mach, control is achieved by transparently sending messages to any task regardless of location via a distributed IPC service that supports transparent forwarding of messages to remote nodes.

Exporting process state. Migrating a process at any arbitrary point in time requires complex support for encapsulating the process' state. Beside exporting various registers, operating system activity must be cleanly interrupted at a restartable point of execution. This state can either be transferred to another node in a distributed operating system, such as in MOSIX [7], or exported into the user space and then transferred, as in case of Mach [12]. The latter is required as processes can not transfer any internal operating system state between nodes.

Process data transfer. A process has several types of data, such as *bss*, stack or memory mapped files. After migration, the operating system can transparently *page-in* this data to its current location on demand. This mechanism was provided to various degrees in Accent [31], MOSIX [7] and Mach [13]). This is called *lazy evaluation*.

Lazy evaluation of an address space, as well as any other state that might be too large or costly to transfer eagerly, has proven to be a useful optimization in the case of process migration. Lazy evaluation however, is very complex to support. This is especially true for complex address space implementations, as in the case of the Mach microkernel.

Transparent communication. Many systems, such as Charlotte [6], Mach [1] and Amoeba [26], provide support for transparent communication of processes independent of their location. Mobility generally discards the connection between two processes. Messages sent during migration can be discarded or received out of order. In order to prevent such cases, complex algorithms are required to maintain communication in the presence of migration. Typically these algorithms employ some type of proxy. Maintaining communication in the presence of migration has turned out to be one of the most complex components of migration systems.

File Systems. In many cases, process migration is supported by the underlying file system (e.g. Sprite [28]). This support consists of transparently moving the open connections for files and other abstractions represented by files, such as pipes. A process can open files on one node, migrate to another node and still access its opened files. In addition, a distributed file system allows a process to open any file in the system from any location using a universal name.

Security. Operating systems provide a secure trusted environment that can be shared by untrusted abstractions. Security is provided by supporting capabilities, as done in Amoeba and Mach, or by using access control lists (ACLs), as done in Locus. These single node abstractions have been extended in various ways to support distributed applications executing on multiple nodes.

3 Mobile Agents

Mobile agents are active abstractions that can independently and often autonomously roam a distributed system (either the Internet at large or corporate intranets) and perform useful work. Mobile agents maintain state information and use this information to make intelligent decisions. They differ from downloadable applets in that they can independently roam from machine to machine on the Web. Some areas where mobile agent applications have been used include data mining, electronic commerce, mobile computing and system administration.

Mobile agents are a relatively new paradigm on the Web. However, mobile agents researchers have a wealth of related research to draw on, including work done by operating system researchers into process migration and artificial intelligence researchers into software agents. Although both of the related areas have been criticized as "mechanisms searching for a problem", the prospects for mobile agents, currently seem to be quite encouraging.

Mobile objects and agents provide benefits similar to those provided by classic process migration without incurring many of the problems of operating system mechanisms. These benefits include: improved load distribution, fault tolerance and performance as well as reduced network traffic. Traffic reduction is achieved by allowing applications to migrate to the host node of a large data set instead of sending a large number of messages or transferring large amounts of remote data. Agents can also be used to represent a user that is temporarily not present on the network.

There are already a number of companies, such as General Magic and FTP Software, which provide agents as products. There are also many agent research projects in academia (see Section 6). Mobile agents fit naturally with the trend of other "active" functionality, such as active networks and active network attached storage devices. Finally, there are already a number of agent standardization processes, such as OMG, FIPA and the Agent Society. Additional details about mobile agents can be found in [14].

4 Language Environment Support for Agents

Recent programming languages and their environments, including Java [11], Tcl/Tk [18], Python, Perl, etc., have proved to be a suitable environment for mobile agents (see the examples in Section 6). However, they lack some features needed to support agent mobility. This section overviews the advantages of and functionality missing from language environments for agents.

The advantages provided by these languages and environments consist of:

Mobile code. These languages have been designed with mobile code in mind. Mobile code is supported by using interpreted languages, by transferring ascii code, or by using bytecode, as in Java. Note that process migration is typically supported only between homogeneous nodes. Recently, there has been some work on heterogeneous migration, such as in Emerald [24] and Tui [23].

Remote method invocation. Some language environments, including the Java Development Kit (JDK) and CORBA from OMG [17], support remote method invocation as a part of the virtual machine [29]. Remote method invocation is an alternative to mobility and in many cases it represents the best solution for distributed computing. Remote method invocation can be used for implementing mobile agents.

Object serialization. Object serialization provides a means for encoding a local object or tree of objects into a serial form suitable for transmission over a network. This is useful for mobile agents but is required independently of mobility, since objects can be passed to remote sites as parameters. Object serialization is supported by a couple of projects including the Java development kit [21].

Operating system independence. Java and interpreters for scripting languages can execute on top of any processor or operating system.

Semantic simplicity. Mobile agents do not support operating system process semantics, for example signals and file locking. Such support has proven to be complex to design and implement in a distributed manner.

The challenges posed by mobile agents to these language environments include the following:

Naming, addressing and locating agents. *Naming* is a way of distinguishing who and what an agent is. An agent may have multiple names, or a name may refer to multiple agents. *Addressing* is a means of specifying the location of an agent. For example, using an address, an agent can be communicated with. Agents are typically named, but only the locations they visit are addressable. *Locating* is a way of finding the current address (location) of a mobile agent. Operating systems typically provide support for naming, addressing and locating mobile processes; language environments lack equivalent support.

Controlling mobile agents. While operating systems provide basic support for controlling processes and other operating system abstractions, this infrastructure is missing for mobile agents. At best, only remote method invocation is available. The missing controls include facilities for killing, suspending and resuming agents.

Exporting agent execution state. Due to the complexity, most language environments do not export an agent's execution state. Therefore, the agent application has to provide an additional state machine for maintaining the actual execution point at the time of migration. This lack of transparency is not a significant limitation for a simple agent application, where the agent makes decisions about migration independently, but it hinders complex agent applications.

Security. It is hard to achieve security for mobile code even when the limitations introduced for applets, such as connection restrictions, are considered. Mobile agents levy stringent security requirements. Sensitive data exists both within the mobile agent and at the host site. Not only the sensitive data requires protection, but the agents and sites must in turn be protected from each other. Mobile agents may require various types of security, such as encryption, authentication and authorization. Agent data can be encrypted in order to prevent the host nodes from inspecting data it has collected on other nodes. Authentication is required to verify the owner of the agent, as well as to authenticate the host node to prevent a node from impersonating another node. Finally, an agent needs to be authorized to visit and to perform certain actions on a node. Nodes needs similar authorizations to interact with agents. While some of this support is provided for mobile code, a great deal of infrastructure has yet to be added.

The challenges above are frequently handled by operating systems. Many of the techniques used by operating systems to provide this support can be applied to different settings, such as language virtual machines, or as a part of middleware level solutions. In Section 5, we show how the MOA project utilized these techniques and applied them to mobile agents.

5 Mobile Objects and Agents (MOA) Project

The Mobile Objects and Agents (MOA) project at the Open Group Research Institute addresses mobile agents on top of Java. In doing so, we have attempted to draw upon lessons that have been learned during our previous work on process migration. The MOA system is composed of three subsystems:

1. The **Core** subsystem supports agents, places, the agent environment and other abstractions that comprise the agent system.
2. The **Name Server** performs tracking of agent locations.
3. The **Monitor** agent subsystem assists in controlling the agents and related abstraction. Agents can be created, killed, suspended, resumed, searched for, etc. Places can be inspected for residing agents.

Applications are run on top of the core agent system. MOA subsystems are described in Figure 1 which depicts a set of three nodes that support the basic functionalities required

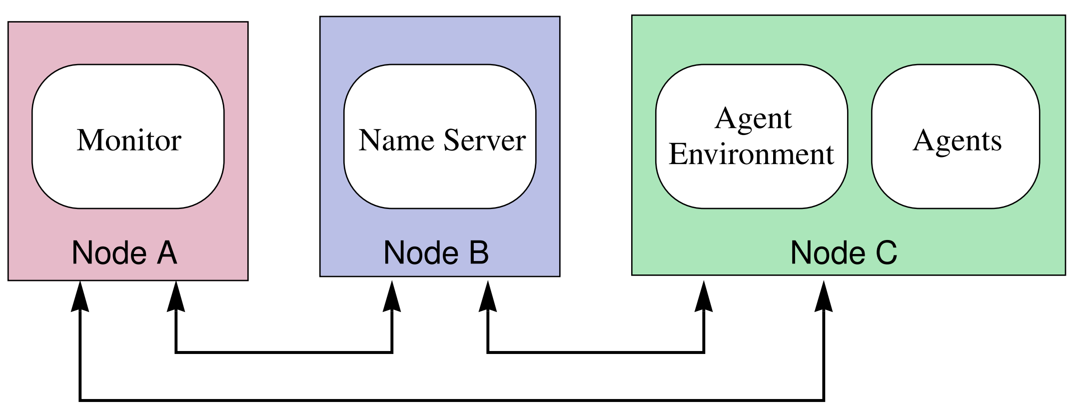

FIGURE 1. MOA Project Subsystems

by mobile agents. In some scenarios it is conceivable that a single node or subset of nodes might contain all the subsystems shown. Node A, the origin site that created a set of agents, maintains a monitoring subsystem that supports both agent creation, agent logging, and agent control. In order to assist in the locating agents, Node B supports a nameserver. The actual agent environment, consisting of places, connections, security support, and resource constraints, is supported on Node C. Thus Node C is the only node in this case which contains agents.

Agents are the basic MOA abstractions. Agents are written by application programmers, typically in Java. The underlying support for agents is provided by the Agent Environment. An agent consists of constant code and variable data that accumulates as it migrates between nodes. An agent typically invokes a "*move*" method (also called "*go*", in the Telescript model [30], or "*migrate*" in process migration) autonomously, i.e. by interpreting application code.

Places are abstractions representing locations that agents visit. Places are introduced for reasons of security and resource management, and provide a convenient means for inter agent communication. They can also serve as repositories of agent data. Agents that are collocated at the same place may communicate more efficiently. Before allowing an agent to arrive at a place, a negotiation occurs consisting of a verification dialogue conducted between an agent and a place. During the dialogue, security requirements, security constraints, and resource availability issues are resolved.

Like Telescript places, MOA places are non-mobile objects and can be nested. Like an Aglet proxy object, MOA places can serve to protect an agent from an agent environment and vice versa. However, MOA places are unique in that within the confines of a single agent environment, a one to one mapping exists between a place and an agent. An agent can have a number of places in different agent environments. Places are not first class objects and can not be created without an associated agent. Place names are defined as a combination of the agent name and the agent environment name. MOA places can be active objects with their own threads. These threads can be used, for example, to communicate with the agent after it migrated away.

Communication channels can be established with either places or agents. Negotiation policies can be set on a per place basis, which allows each place to tailor a security policy that addresses security and resource requirements of (possibly nested) places. Resource management is also performed at the place level, and different places can have different resource limits.

MOA Logging. A logging and auditing module supports storing information about the agent activity. The following information about agent is logged: places visited, communication, meeting, cloning, etc. Similarly, information about places can be logged: the agents that visited the place, that met at a place, the communication that took place, etc. Because of the large amounts of data, not all of the information is always logged, nor is it logged for all agents. It is possible to specify which data and for which agents and/or places will be logged, both in a static and a dynamic manner.

Naming addressing and locating agents. The naming of agents and places is modeled on Universal Resource Locators (URLs). Agent names are based on two components that reflect the place of their creation, similar to the notion of the home node, and a local identifier.

Agent names reflect their lineage, as shown in Figure 2. In the figure, each agent created by the user serves as the ancestor of a *family*, e.g. **Agent**. Families are composed of one or more agents derived from ancestors via cloning. Each family ancestor is differentiated by a suffix denoting the ancestor's number, e.g. **Agent-1**. Family names are inherited by descendant clones, with each generation appending a new suffix to denote its identity

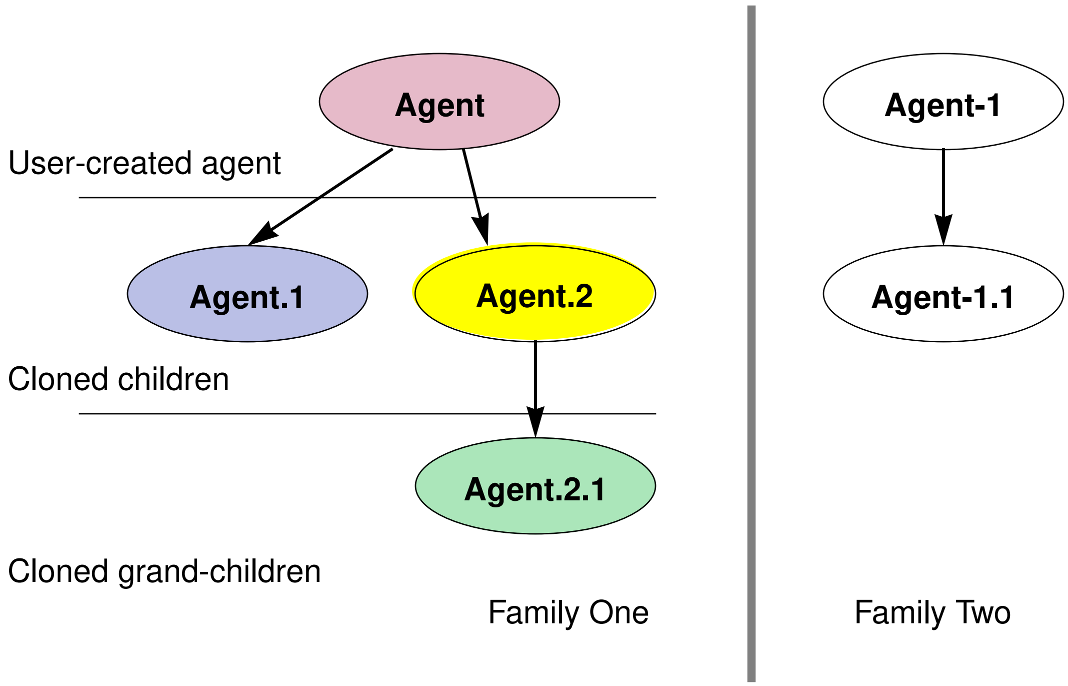

FIGURE 2. Naming The Agent Hierarchy

within that generation. Thus the first child of **Agent** is named **Agent.1** and the second child **Agent.2**. The first grandchild of **Agent** via **Agent.2** is named **Agent.2.1**.

No two agents, or two places in the WWW, have the same name and address at the same time. A naming authority represented either by an agent's home node or a surrogate node, resolves an agent's name if all other naming attempts have failed.The naming scheme is comprehensible to users and convenient for use by programmers because the names of places and agents correspond to their creation history. No assumptions are made upon the agent ancestry information embedded into the names. This information is just a convenient informative indication for the agent's user, while the actual information is maintained at the home node. This scheme does not exhibit problems with scalability common to PID-based distributed operating system schemes.

Location tracking schemes, described in Section 2, depend upon the connectivity of all the nodes in the system. In the case of mobile agents, frequently the home node may not be present or reachable at the time of an agent's migration. This requires an extended version of the home node location scheme which utilizes surrogate home nodes. MOA offers several variations of the operating system methods presented earlier.

Controlling agents. The control of agents is similar to the control of migrating processes in distributed operating systems. Although the scaling, performance and failure semantics differ between Internet environment and clusters of workstations, mobile agents still need to be suspended, resumed, killed, located, etc. Some of the design and implementation of control mechanisms may differ slightly because of the different setting, but much of the functionality and experience gained can be directly deployed in an agent setting.

MOA employs a monitor process, similar to the local proxy used in operating systems, to forward control messages to an agent. This monitor process is not required to be executed on the home node of an agent in order to control that same agent.

Transparent communication. One of MOA's unique features is its support for inter-agent communication. Agents and places communicate by means of channels. A MOA communication channel is a logical channel. Upon migration, the MOA system is responsible for reconnecting an agent's channels. No agent intervention is required to maintain an open channel with any other agent or place, regardless of the other object's location. Like interprocess communication in the case of process migration, no migration-related constraints are placed upon these channels.

Opening or reopening a channel begins with locating the agent using the MOA name server. After agent migration, channels can be reopened either eagerly or lazily. Each agent, place or MOA environment can accept or reject the creation of a non-solicited channel subject to its own resource limits and/or application preferences. For example, a specific agent may not accept channels originating from certain nodes, families or generations of agents. Channels can be used by agent applications for synchronization by exchanging messages.

Agent communication and cooperation are one of the focuses of this project. Although many projects deal with mobile agents, none of them stress agent cooperation. We believe that the mobility and dynamics needed to support a wider variety of agent communication models require additional support, currently nonexistent in Java.

Exporting agent state. MOA agent state consists of agent application classes, agent system state, and an agent application execution point. Agent application classes are serialized prior to migration. This is a straightforward process which is done by writing the agent object into the object stream. In a similar fashion, the agent system state is serialized prior to migration. Once system and application state have been serialized, MOA uses a low level data mover to perform the actual data transfer.

Execution point state is not easy to maintain across migration as it requires changes to the Java VM, as done in Sumatra [20]. As a result, agents usually restart after a migration or they maintain a small, internal state machine as an application object (e.g. as in Aglets).

A typical MOA agent migration begins by the agent application invoking the MOA system *move* method. At this point, the MOA system serializes the agent application and system state, negotiates with the destination place and then invokes the low level data mover to complete the actual migration.

MOA data transfer. Mobile agent data transfer is performed at the class level, giving application programmers some control over application state transfer. Specifically, agent application classes and system state can be declared as transient data which will be discarded before migration. In the typical process migration scheme, the entire process address space had to be copied to the target site. The object serialization process maps an agent's many small objects into one contiguous buffer and the agent's system state into a second buffer. This process typically reduces the total amount of data that is transferred and reduces the need to optimize data transfer by using techniques such as copy-on-reference or lazy evaluation.

File systems. The world wide web provides a universal namespace, using URLs, that allow agents to access files regardless of their location. Unlike process' that employ universal file handles, agents must reopen needed files after migration. Files without URLs can be accessed only on the file's local node.

Security. The security issue, in the case of process migration, is handled differently depending on what kind of operating system support is provided. As such, it was not applicable in the case of mobile agents.

The MOA project will address mainly the authentication aspects of security. Authorization will be re-used from another project [32]. Authentication of the code is done only once by the well know authority. Authentication of the data must be repeated at each site that an agent visits. This is performed by the cumulative signing of all of the previous data extended by the data provided by the current node. Note the important distinction between a migrated process and agent. Processes typically require huge amounts of data. The relevant data that an agent carries along its migration is typically small. In some cases, an agent might not have to transfer any data, for instance when it has not found the searched for entity, no data will be transferred from a node.

The impact on the trust model is still open to research. For example, should data be signed cumulatively or separately. The visiting node can strip the previous data in either case. Syncing the intermediate data occasionally prevents the intermediate nodes from pursuing such attacks.

Resource management. Although the Java environment maintains its own notion of resource management, the agent system imposes additional restrictions on the resources agents can consume. In other words, proper agent systems should be self-limiting, good citizens of the Internet World [8]. Therefore, limitations are imposed on the various resources an agent can consume on the visited node. E.g., the number of clones, the time to live, the number of places it can create, etc. Some of these assumptions help in garbage collecting, for example, when the time expires, the agent and resources owned by the agent can be freed.

The above functionality can easily parallel the operating system development, but it is simpler in many cases.

6 Related Work

In this section we mention a number of agent systems, implemented in different languages, such as Java [11], Tcl/Tk [18], Telescript [30] and Python [19].

Telescript [30], developed by General Magic, is the first well-known system that supports agents as a commercial product. Originally, Telescript was aimed at closed networks, such as a collection of personal digital assistants, but has since been refocused on the Internet.

The **Aglets** project [3], conducted by IBM Japan, aims at the standardization of an agent facility. Aglets represent a Java implementation of a standard agent transfer protocol and an agent facility allowing the interoperation of agents written in the same language.

The **Mole** project [16] at the University of Stuttgart is one of the first academic efforts to provide mobile agents in Java. Mole relies on Java and RMI. It is being used by Siemens for experimenting with mobile agents and AI, as well as by some other industrial sponsors, such as Daimler-Benz and Tandem.

Sumatra [20] is a Java-based agent project at the University of Maryland. Sumatra provides support equivalent to RMI, but enhanced with the checkpointing of the execution point. Sumatra requires changes to the Java virtual machine, similar to object serialization. This is a major compatibility issue. Agents are used for optimization of communication between different nodes.

CyberAgents [9] is a commercial product developed by FTP Software, Inc. It is a software package that supports mobile agents for various tasks, such as system administration, virus checking, or platform inventory information collection. CyberAgents supports sophisticated user interfaces and agent control.

The **Agent Tcl** project [2] is one of the first to provide agents in Tcl/Tk. It provides rich semantics for agents and agent interaction. It is intended for mobile computers, supporting temporarily disconnected operation. Agent Tcl also deals with navigation and adaptation.

Ara [5] started as a Tcl/Tk based mobile agent project, at the University of Kaiserslautern. Recently it has also moved towards Java, and claims multiple language support.

The **Tacoma** project [25] is conducted as a collaboration between the University of Tromso at Norway and Cornell University. The Tromso part develops basic mechanisms and applications; the Cornell part addresses fault tolerance and security.

Python [19] is being used as a base for mobile agents, under the name Knowbots. Knowbots are targeted for applications consisting of multiple cooperating agents that travel and communicate among themselves.

None of the mentioned systems depends on the operating systems directly, although each of them uses some of the functionality typically associated with operating systems.

7 Conclusion

Process migration algorithms and techniques developed originally for operating systems have direct relevance to language environment mobile agent implementations. While recent programming languages (and their environments) have provided some benefits for mobile agents, many features are still lacking. Features lacking in language environments that we have successfully leveraged for the Open Group's MOA project include techniques and schemes for naming and locating agents, transparent agent communication, data transfer, resource management and controlling agents.

The MOA project has not addressed all of the functionality missing in language environments. There are some operating system characteristics, such as real-time behavior, fault-tolerance, and security, that remain open to investigation. These areas will be addressed in future work.

8 References

[1] Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A., and Young, M., “Mach: A New Kernel Foundation for UNIX Development”. *Proceedings of the Summer USENIX Conference*, pp. 93–11, 1986.

[2] Agent Tcl home page, *http://www.cs.dartmouth.edu/~agent/*.

[3] Aglets home page, *http://www.trl.ibm.co.jp/aglets/*.

[4] Alonso, R. and Kyrimis, K., “A Process Migration Implementation for a Unix System”. *Proceedings of the USENIX Winter Conference*, pp. 365–372, Feb. 1988.

[5] Ara home page, *http://www.uni-kl.de/AG-Nehmer/Ara/*.

[6] Artsy, Y. and Finkel, R., “Designing a Process Migration Facility: The Charlotte Experience”. *IEEE Computer*, pp. 47–56, Sep. 1989.

[7] Barak, A., Guday, S. and Wheeler, R., *The MOSIX Distributed Operating System: Load Balancing for UNIX*, Springer-Verlag, 1993.

[8] Condict, M, et al., “Towards a World-Wide Civilization of Objects”, *Proceedings of the 7th European SIGOPS Workshop Systems Support for World-wide Applications*, Connemara, Ireland, Sep. 1996.

[9] CyberAgents home page, *http://www.ftp.com/product/whitepapers/4agent.htm*.

[10] Douglis, F. and Ousterhout, J., “Transparent process migration: Design alternatives and the Sprite Implementation”, *Software-Practice and Experience*, 21(8):757-785, Aug. 1991.

[11] Gosling, J., et al. *The Java Language Specification*, Addison-Wesley, May 1996.

[12] Milojicic, D., Zint, W., Dangel, A., and Giese, P. “Task Migration on the top of the Mach Microkernel”, *Proceedings of the third USENIX Mach Symposium,* pp. 273–290, Apr. 1993.

[13] Milojicic, D., *Load Distribution, Implementation for the Mach Microkernel*, Vieweg, Germany, 1994.

[14] Milojicic, D. Bolinger, D. Zurko, M.E., and Mazer, M., Mobile Object and Agents, *The Open Group Research Institute Technical Report*, Nov. 1996 (appears in *Collected Papers Vol. 5*, Mar. 1997).

[15] Milojicic, D., Douglis, F, Paindaveine, Y., Wheeler, R. and Zhou, S., "Process Migration", *The Open Group Research Institute Technical Report*, Oct. 1996 (appears in *Collected Papers Vol. 5*, Mar. 1997).

[16] Mole project home page, *http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html.*

[17] Object Management Group, CORBA web Page, *http://www.omg.org/corba/.*

[18] Ousterhout, J., *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[19] Python Home Page, *http://www.python.org/.*

[20] Ranganathan, M., et al., "Network-aware Mobile Programs", *Proc. USENIX 1997 Technical Conf.,* Jan 1997.

[21] Riggs, R., et al.,"Pickling State in the Java System," *Proc. USENIX 1996 Conf. on Object-Oriented Technologies (COOTS)*, pp 241-250.

[22] SHAWS home page, *http://www.osf.org/RI/PubProjPgs/SFTWWW.htm.*

[23] Smith, P., Hutchinson, N., "Heterogeneous Process Migration: The Tui System", TR 96-04, University of British Columbia, Feb. 1996.

[24] Steensgaard, B., Jul, E., "Object and Native Code Thread Mobility" *Proceedings of the 15th SOSP*, pp. 68-78, Dec. 1995.

[25] Tacoma home page, *http://www.cs.uit.no/DOS/Tacoma/.*

[26] Tanenbaum, A., Experiences with the Amoeba Distributed Operating System for the 1990s. *Communication of the ACM*, 33(12):46–63, Dec. 1990.

[27] Theimer, M., Lantz, K., and Cheriton, D., Preemptable Remote Execution Facilities for the V System. *Proceedings of the 10th ACM Symposium on OS Principles*, pp. 2–12, Dec. 1985.

[28] Welch, B.,Naming, State Management and User-Level Extensions in the Sprite Distributed File System. *Technical Report UCB/CSD 90/567, Ph.D. Thesis, CSD (EECS), University of California, Berkeley*, Apr. 1990.

[29] Wollrath, A., et al., "A Distributed Object Model for the Java System," *Proc. USENIX 1996 Conf. on Object-Oriented Technologies (COOTS)*, pp. 219-231.

[30] White, J., "Telescript Technology: Mobile Agents", General Magic White Paper," *http://www.genmagic.com/agents/Whitepaper/whitepaper.html*.

[31] Zayas, E., "Attacking the Process Migration Bottleneck", *Proc. of the 11th SOSP*, pp. 13-24, Nov. 1987.

[32] Zurko, M. E., Simon, R., "User-Centered Security", *Proc. of New Security Paradigms Workshop*, 1996 (to be published).