Case Studies in
Security and Resource Management
for Mobile Object Systems

Dejan Milojicic[†], Gul Agha[‡], Philippe Bernadat*, Deepika Chauhan**
Shai Guday[††], Nadeem Jamali[‡], Dan Lambright[‡‡] ,and Franco Travostino[†††]

[†]HP Labs, [‡]University of Illinois at Urbana-Champaign, *Silicomp Research Institute, **Nokia, [††]Microsoft, [‡‡]EMC, and [†††]Nortel Networks

Abstract

Mobile objects have gained a lot of attention in research and industry in the recent past, but they also have a long history. Security is one of the key requirements of mobile objects, and one of the most researched characteristics related to mobility. Resource management has been somewhat neglected in the past, but it is being increasingly addressed, in both the context of security and QoS. In this paper we place a few systems supporting mobile objects in perspective based upon how they address security and resource management. We start with the theoretical model of Actors that supports concurrent mobile objects in a programming environment. Then we describe task migration for the Mach microkernel, a case of mobile objects supported by an operating system. Using the OMG MASIF standard as an example, we then analyze middleware support for mobile objects. Mobile Objects and Agents (MOA) system, is an example of middleware level support based on Java.The active networks project, Conversant, supports object mobility at the communication protocol level. We summarize these projects, comparing their security and resource management, and conclude by deriving a few general observations on how security and resource management have been applied and how they might evolve in the future.

Keywords: security, resource management, mobility

1. Introduction

In a distributed system, objects may migrate from one node to another for a variety of reasons. The new location may provide a more suitable computational environment, it may offer cheaper resources, or it may have data needed by the agent to satisfy its goals. The ability of an object to exist in a resource space that is not entirely dedicated to its own computation raises security concerns for the object itself, and raises performance and security concerns for the host environment. Individual objects or groups of objects may exhibit undesirable *resource consumptive behavior.* These reasons make it important to study ways of controlling resources used by mobile objects.

Most of the work on security and resource management has been performed for stationary systems, such as the security of client server models and of programming environments, and resource management in operating systems. A mobile object system must protect its own resources, including the underlying environment (such as the operating system or a virtual machine, e.g. JVM, or Tcl/Tk interpreter), file system, devices, memory, other (non)mobile objects, etc. In order to achieve this, the mobile object system has to identify the incoming object as well the sender of the object. The system must recognize the access rights of the incoming object and optionally support its confidentiality. There are various possible threats and attacks on a mobile object system, including denial of service attacks, unauthorized access, corruption of data, spamming, spoofing, trojan horses, replay and eavesdropping [50]. These attacks can be performed through a mobile object application or through mobile object systems, utilizing weaknesses in the communication infrastructure, language, or mobile object system implementation.

The security and safety services of the underlying communication infrastructure and the programming language enforce the security policies. The security policy is applied using the factors such as the credentials of the communicating parties, the object classes, and the authority of the mobile object. Policies may restrict or grant mobile object capabilities by setting resource consumption limits and access permissions to resources. Multiple security policies may be set by the authority that is represented by the mobile object and its system.

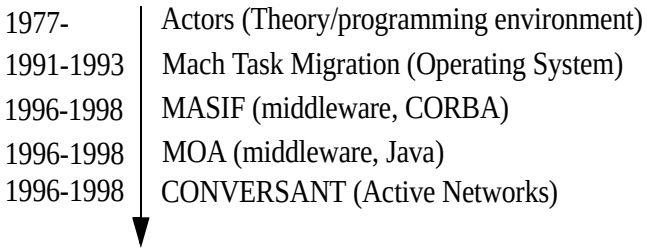

Figure 1. Systems described.

Mobile object systems use communication transport calls to transfer objects between systems. Mutual authentication of object systems is typically required for these calls to succeed. Objects are authenticated based on the authentication of the source object system, information about the object itself, and possibly information about trusted authorities that can authenticate the object.

Security and resource management significantly interplay in any system and this is especially expressed in case of mobile object systems. In particular, denial of service security attacks are typically achieved by exhausting resources of an attacked entity, e.g. by consuming all memory, processing cycles or the communication bandwidth. A careful resource management can prevent such attacks. In a similar manner, system security provides guarantees that resources are appropriately used by users who have permission to do so. In case of mobile object systems, both security and resource management are much harder to achieve because of the distributed and possibly disconnected state. Aspects such as scalability, fault tolerance, and real-time complicate security and resource management for mobile object systems even further.

This paper is an overview of the research topics in security and resource management in mobile object systems. We provide some practical experience drawn from five examples of mobile object systems that we were involved in specifying, designing, and implementing. These systems have achieved various stages of implementation, usage, and technology transfer, offering an interesting mix of platforms and goals.

The rest of the paper is organized in the following manner. The five sections that follow describe five different mobile objects systems and the security and resource management issues are described. In particular, Section 2 describes the Actors model. Section 3 presents task migration

on top of the Mach microkernel. Section 4 analyzes MASIF, an OMG standard for mobile agent systems. Section 5 describes Mobile Objects and Agents (MOA) project. Section 6 describes CONVERSANT, an active networks project. In Section 7 we summarize the paper by making some general observations garnered from the different systems we described. Given the survey type of the paper, we briefly mention related research in each of the five sections that describe a type of mobile object system.

2. Actors

Actors were initially developed by Carl Hewitt [33] for conceptual modeling of open systems. The idea was further refined by Gul Agha [2] to serve as a computational model for concurrent computation in distributed systems.

Actors extend the concept of objects to concurrent computation. Specifically, objects encapsulate a state and a set of procedures that manipulate the state; actors extend this by also encapsulating a thread of control. Actors are mobile. Each actor potentially executes in parallel with other actors and may send messages to actors it knows the address of. Actor addresses may be communicated in messages, allowing dynamic interconnection. Finally, new actors may be created; such actors have their own unique address. A more concrete way to think of actors is as an abstraction over concurrent architectures. An actor runtime system provides the interface to services such as global addressing, memory management, fair scheduling, and communication. Actors are self-contained, interactive, autonomous components of a computing system that communicate by asynchronous message passing [2, 4]. This model abstracts over issues of low-level synchronization by encapsulating the state of an object and its execution thread, and limiting communication to asynchronous message passing.

The model has been used as a basis for a number of programming languages and systems. Actor implementations include Acore for AI applications at MIT [48], Cantor for scientific computations at Caltech [7], Rosette for systems architectures at MCC as a part of the InfoSleuth project [73, 80], ActTalk for object-oriented programming at Université Paris [15], Act++ for Real-time Systems at Virginia Tech [40], and ActorFoundry for Actor based mobile computation in Java at

Illinois [60]. Recent agent implementations such as Aglets [46] and Concordia [81] also follow actor semantics to varying degrees.

2.1 Resource Management

In an open system, agents may migrate from one node to another searching for computation environments suitable for completing their tasks at affordable costs. These agents may also spawn child agents to pursue subtasks. This makes it important to study ways of controlling the *resources* that such agents or their ensembles could use in serving some particular interest. On the one hand, we need a *bounded resources* model to control the amount of computational resources consumed by agents serving an interest; on the other hand, we need a *bounded autonomy* model for allowing coordination among agents.

The Actor model has been extended to explicitly model the location of agents on particular *hosts* and the *bounded computational resources* that they may use. Hosts are actors that manage physical and logical resources of a node and offer them to agents interested in paying for them. A *universal currency* is used to pay for the cost of these resources. The behavior of an agent may be interpreted in a suitable framework, e.g., the belief, desire, intent model [57].

To support a system where agents can use resources available ‘‘elsewhere’’ in a satisfactory way, it is important to have some notion of an economy. Such an economy would provide the basis on which nodes would allow agents to use their resources, and would serve as an environment that would enable nodes and agents to get into binding contracts about the services needed.

Resource allocation in multi-agent systems is a problem that raises issues of reciprocity as well as performance and security concerns. Nodes on the worldwide web, for instance, may be willing to be part of a multi-agent system if they receive something in return for allowing foreign agents to use their resources. From the performance and security perspective, agents migrating to a node may exhibit undesirable resource consumptive behaviors, either individually, or as ensembles. Similarly, network channels are a scarce resource requiring controls on how they may be used.

An economic model can be used to protect against resource consumptive behavior of agents in a multi-agent system [3]. Note that control in agent systems is not based solely on programming

structures, as agents may create or invoke other autonomous agents. Such autonomy makes it important to devise explicit mechanisms for controlling the extent to which an expanding group of agents, working on a single task, can utilize a system's resources. In an open distributed system, the problem is compounded by the ability of agents to exist in a resource space not entirely dedicated to their computations alone. We need mechanisms to support bounding the resource utilization of individual agents, or ensembles of agents working together, according to the terms under which they are allowed access to those resources.

To implement an economic model, we will use the notion of a universal currency. Specifically, resource allocation will be measured in a common currency called **GCU** (for *global currency unit*). Every computational activity must be allocated **GCU**s which may be used in completing the task. Each agent is allotted some subsistence **GCU**s at the time of its creation by its creator, and because activity in message-based systems is triggered by a message, **GCU**s must also be allocated at the time of sending a message. The **GCU**s so transferred are deducted from the accounts of the creator or the sender, respectively.

An agent interested in migrating to a particular host must negotiate a contract with the host ahead of the actual migration. Independent of the actual negotiation protocol, the purpose is to agree on a function that would determine costs of resources that will be made available, possibly dependent on the state of the host. Once the contract has been agreed, an agent may arrive at the node with a certain number of **GCU**s. The **GCU**s held by an agent may be spent for purchasing computational resources from the host as it computes, according to the negotiated contract.

To separate environmental concerns (such as the suitability of current host) from the application code of agents, a reflective model is used. Reflection allows an application to monitor execution of the underlying system and to modify it dynamically. Specifically, detection of an agent's need to migrate and the process of migration are handled by a facilitator, that belongs in the meta-architecture which enables agents to continuously interaction with their environment.

Application specific schedulers enhanced with a scheme to dynamically assign priorities to agents may also be useful in bounding the autonomy of agents, allowing them to cooperate by resource sharing. An agent may fine-tune priorities of its child agents or peer agents may themselves

choose to lower their own priorities to allow others, working on more time-sensitive parts of the application, to compute faster.

2.2 Security

The Actor model assumes that actor names are unforgeable and that message delivery is guaranteed; the model abstracts over implementation protocols which provide these guarantees. Other security features have to be explicitly implemented in the model by enforcing interaction policies. The resource management model described here serves to secure the system from chaotic resource consumptive behavior which may otherwise be exhibited by actors or ensembles of actors.

2.3 Actors Contributions and Lessons Learned

Actors provide a natural extension of the object-oriented paradigm to concurrent and distributed computation. They support encapsulation, description as behavior templates, and re-usability via libraries accessed using message-passing protocols. The locality properties of actors guarantee that changes of representation and elaborations can be made independent of the interaction with, and behavior of, other actors. Thus actors can support local instrumentation and monitoring which provide important tools for analysis and debugging. Because the internal behavior of an actor is encapsulated and cannot be observed directly, the Actor model supports heterogeneous, variable grained objects. Specifically, the behavior of individual actors may be defined using any programming language.

A rigorous theory of actor systems is developed in [5]. Specifically, various notions of testing equivalence on actor expressions and instantaneous snapshots (configurations) are designed and studied. The model provides fairness, namely that any enabled transition eventually fires. Fairness is an important requirement for reasoning about eventuality properties. It is particularly relevant in supporting modular reasoning: if we compose one configuration with another which has a non-terminating computation, computation in the first configuration may nevertheless proceed as before, for example, if actors in the two configurations do not interact.

The work on Actor-based agents extends this body of work to mobile resource-bounded agents by explicitly representing locations of agents and their resource allocations. This extension allows a study of ways in which resource consumption may be controlled in an open distributed system. As a result, support for resource management and security is provided in terms of dynamic control of resource access and utilization by agents and agent ensembles.

2.4 Related Work

Actors is one of the earliest mobile object systems, however there were a few others, such as Emerald [39], COOL [6], and SOS [68]. Similar to Actors, these systems rely on programming language support for mobility. Ether [43] was the first language to address explicit allocation of resource in concurrent systems. Sponsors were assigned to processes to support their computations. This idea was later incorporated in an actor language, Acore. Sponsor actors accompanied computation requests, and they carried *ticks* that could be used in processing a request. Using a similar scheme, in Telescript [79], processes were awarded funds in a currency (“teleticks”) which they were supposed to use to accomplish their results. Quantum [56] uses a similar idea where computations require *energy* to execute; energy is used as a transferable currency to access computational resources.

3. Task Migration on top of Mach

The whole area of process and task migration is quite relevant for the today's mobile object systems, yet the community largely is oblivious of this fact. A great deal of experience and many lessons learned are relevant to today's systems and could be utilized to avoid making similar mistakes or reinventing the same wheels. Examples include communication among the mobile entities, delegation of the rights to mobile objects, resource management, etc.

The task migration project was conducted in the period 1991-1993 at the University of Kaiserslautern. A few versions of task migration and load distribution have been designed and implemented on top of the Mach microkernel. All versions relied on distributed Interprocess Communication (IPC) and in varying degrees, on distributed memory management support. An

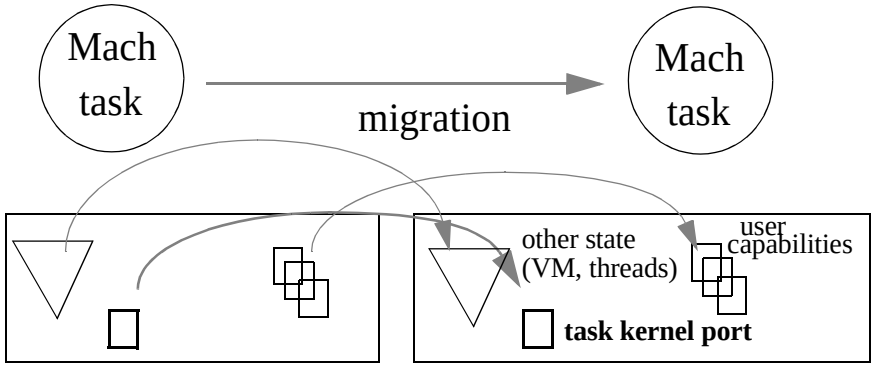

Figure 2. Interposing Task Kernel Port. *The task kernel port is extracted (interposed) on the source- and inserted on the destination-node. Messages sent to control the task are queued, and reactivated as a part of the second interposition on the destination node. Tasks that sent messages are blocked until migration finishes.*

in-kernel version and a user level version relied entirely on distributed memory management, while another user level version relied on an external implementation of the migration pager. This work was used as the basis for two implementations of process migration for the OSF/1 server [61, 70].

3.1 Security

In the task migration project, security was mostly addressed in exporting the task and kernel thread ports representing the corresponding task and the associated threads. These ports are part of the kernel internal state that has to be extracted and transferred to a destination node (see Figure 2). Whoever holds a task port is capable of controlling the task (e.g. to suspend, resume task and threads, read/write into address space, manage IPC space, etc.) by sending messages to the task kernel port.

In all versions of task migration, the task kernel port needed to be temporarily suspended, so that all messages sent to it are not interpreted as commands, but rather as true messages that can be queued and then restarted after migration. In normal operation, such messages are guaranteed to execute in finite time. It is possible to compromise security when migrating the task kernel port. While the port is in transfer, all messages sent to it are not interpreted as control messages that

invoke certain kernel procedures, but are queued as regular messages. This changes the expected behavior and in the case of problems with the transfer of the task kernel port, could indefinitely delay the sender. The only possible way to protect arbitrary users from waiting on their task control message is to associate a timer with their request. Timeouts were not part of the task control interfaces, so they have to be a configurable constant per kernel.

In user space migration implementations, the task kernel port is extracted from the user space (see Figure 3). This is particularly dangerous since an arbitrary task can have its kernel port swapped. There is no particular solution to this problem. Whoever possesses the task kernel port can harm this underlying task with or without interposition. From the perspective of security, interposing on a task kernel port can compromise its privacy and allow for arbitrary monitoring all tasks activities. This also compromises the integration of the kernel, since part of its internal state is exported, and the kernel port semantics are changed. In kernel implementations this problem does not exist as ports are always passed between the kernels. The port transfer is secured by standard (non-mobile) security techniques between two stationary points.

Task kernel port interposition is related to subsequent work in the mobile agents area where agent credentials are extracted prior to migration and the delegated to the remote instance. This is further discussed in Section 4.

3.2 Resource Management

As a part of the task migration project, accounting and profiling of Mach tasks was performed for the purpose of load balancing. The local and remote IPC, local and remote paging, lifetime, processing time, and the number of migrations were all accounted for. Figure 4 describes how accounting was extended for remote IPC (Mach already supported local IPC and VM accounting). More details about accounting can be found in [55]. The microkernel was instrumented to collect information at the node, task, and internal objects level. At the node level, information was accounted for the number and size of messages across the network.

The information collected by kernel accounting was used for the purpose of distributed scheduling decisions: where to migrate the tasks. Figure 5 demonstrates the benefits of taking IPC into

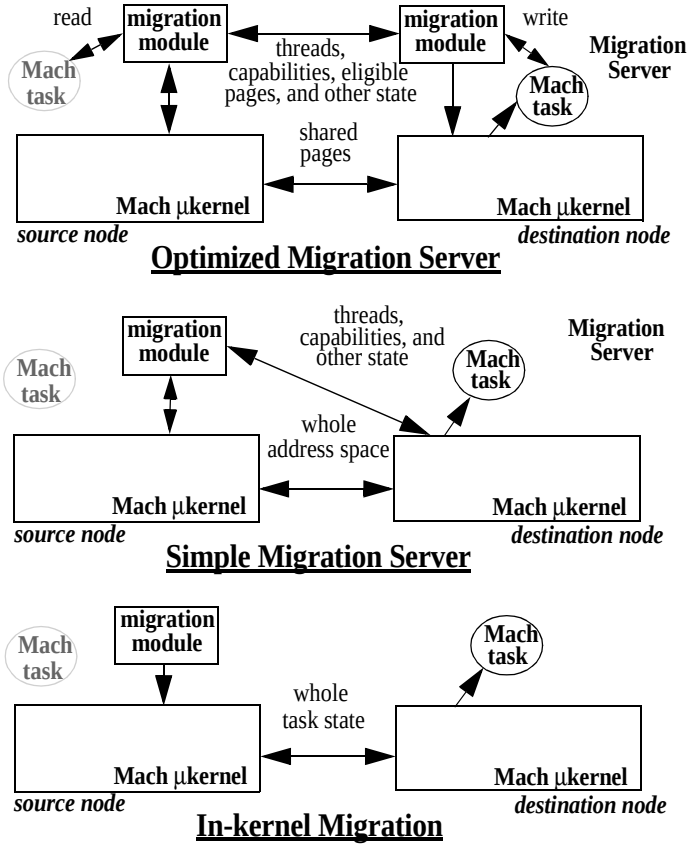

Figure 3. Task State Transfer for Various Task Migration Versions: *in all versions shared pages are migrated by the kernel, Simple Migration Server and Optimized Migrating introduce various levels of user space migration. Depending on the level at which state is transferred, the secure state might be exported outside of the same protection domain, exposing the migration to security risks.*

account when migrating tasks. A number of tasks were started on a task migration testbed. A sender initiated distributed scheduling algorithm was activated in two versions, one that considered IPC in making decisions and another that did not. More details on this experiment can be found in [53]. This experiment demonstrates that in some cases load distribution decisions could be improved by using information about the network IPC. The performance improvement is driven by two factors. The obvious advantage comes from the fact that moving a client towards the server improves the average task execution time. The less obvious advantage is the consequence of the limited IPC bandwidth which could, similar to processor overload, become a bottleneck. Distributing IPC load from the client to a node that is not the server node still improves the performance because although a single node is still the server, the client/server communication takes place in parallel among the client nodes.

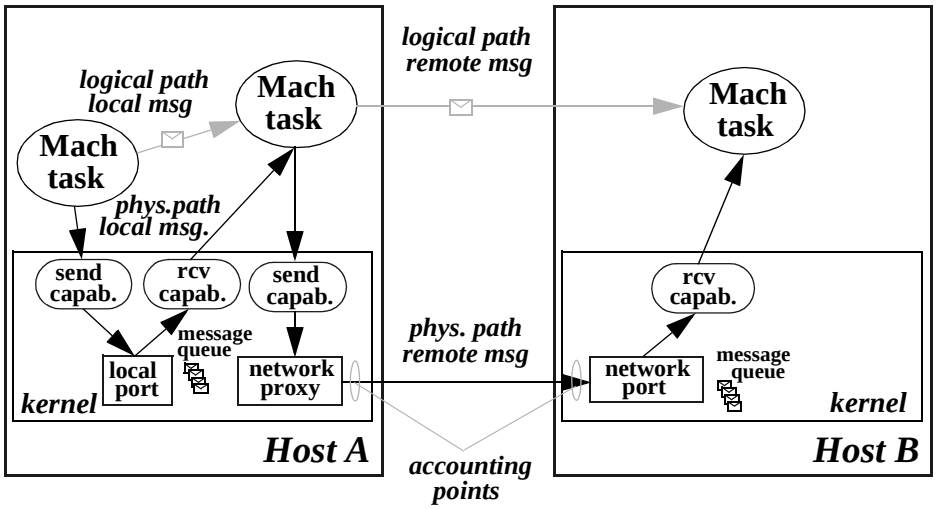

Figure 4. Network and Local IPC Paths and Accounting.

We also used the user space IPC profiler for the transparent instrumentation of the IPC traffic of a task. The profiler was intended for off-line testing (its overhead renders it unsuitable for runtime experiments). The profiler tracks the number of messages, their size and destination, the number of capabilities, etc. To intercept message transfer, all capabilities are extracted from the profiled task and substituted with proxy capabilities. Every message sent or received using an original

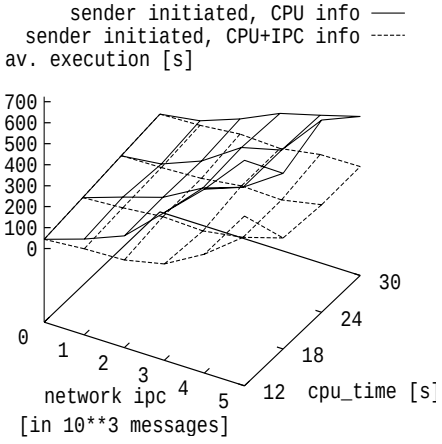

Figure 5. Using IPC Accounting to Improve Distributed Scheduling Decisions. Figure demonstrates that better overall performance can be achieved when load distribution algorithm takes into account information on the communication of tasks, by migrating tasks to the servers they communicate with (above some threshold amount of communication).

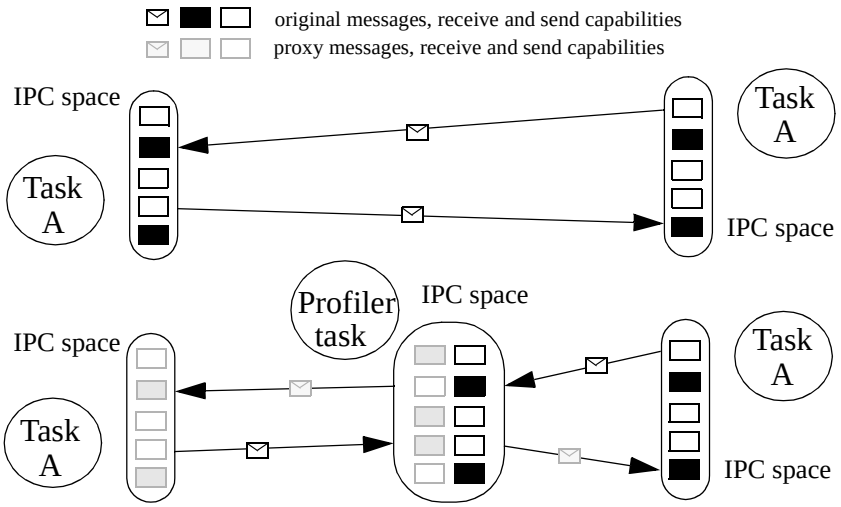

Figure 6. Profiling Application's IPC Activity: *Profiling Application's IPC Activity: after activating the IPC profiler, the original capabilities in the profiled task are substituted with the interpose capabilities, while the original capabilities are migrated to the profiler IPC space. All messages that are sent to or from the profiled task are accounted. Capabilities transferred in messages are also interposed.*

capability is received by the profiler. After being analyzed, messages are forwarded to their original destination.

During profiling, the ports carried in messages need to be exchanged to preserve consistency for the receiver of the message and to allow the profiler to intercept the future messages communicated with the profiled task through transferred ports. Figure 6 describes how the profiler interposes between the profiled task and the other tasks in the system. Profiling is similar to the kernel port interposition that is applied for user space migration. The kernel port interposition is also performed for the profiled task in order to instrument the messages sent to the task kernel port. The threat that such systems are exposed to is possible only if the task kernel port is possessed, which is required in order to do the interposition.

3.3 Mach Task Migration Contributions and Lessons Learned

A more complete list of the contributions of the Mach task migration project can be found in [49]. Security and resource management related contributions are the following.

This project demonstrated that it was easy to implement and insulate from the other modules task migration on top of the microkernels. This is on contrary to other migration implementations on top of the monolithic operating systems.

A few task migration implementations demonstrated that capability based operating systems and in particular microkernels are a convenient platform for mobility. Capabilities are suitable mechanism for carrying and delegating access rights. Microkernels minimize the amount of resource that need to be managed: processing, memory, and IPC (including their distributed extensions) have been the only resources that were used for distributed scheduling.

Distributed IPC was found to be a powerful mechanism that significantly simplified implementation of task migration, however, it was also a source of significant complexity that required a lot of attention and continuous improvements. Most of the complexity for transparent migration of communication channels has been pushed from the mobility layer into the communication layer.

Fault tolerance was hard to achieve. A lot of residual dependency remained at each node, either explicitly (as a part of the mobility support) or implicitly at other layers, such as distributed IPC or distributed memory management.

3.4 Related Work

Security for task migration is impacted by the underlying environment, which consists of clusters of systems connected in a local area network, typically within the same security domain. In these systems there was not as much need for a secure communications infrastructure as is required for wide area networks or the Web. Security work was done at CMU [65] and also by Trusted Information Systems [66].

Interposition was researched by Jones [37] as a means for introducing new code into the system. The concept of mediation [41] is also related to interposition and capability based systems. In order to make scalable systems, the concept of the Key Distribution Center (KDC) is introduced, which mediates the keys to all nodes. If some node wants to talk to other nodes, it needs to talk first to KDC [41, 58]. In the case of Mach and capability interposition, all kernels on all nodes are

part of a trust domain and all kernel ports are trusted. User level task migration breaks this concepts.

The work by Ford et al. [24, 25] bridges the gap between OS and agent work. Ford et al. introduce the recursive virtual machines that allow the code to execute in a separate trusted domain. SPIN system addresses security by enabling the secure execution of the code within the kernel. In particular it addresses the kernel extensibility by safely executing extensions within the kernel. [11].

Operating systems were instrumented to provide information on various physical and logical resources, such as in COCANET [63], and in the work by Huang et al. [35]. This was done for the purpose of load distribution and performance evaluation. In the Stealth scheduler [45], VM was prioritized in order to reduce the impact of the visiting task. The same was planned for IPC but was never actually implemented. Load information was used to decide whether to migrate processes and to where [21].

For a general overview of process migration and load information management we recommend a thorough survey paper [51] and an edited book on mobility [54].

4. OMG MASIF

MASIF is an OMG standard for interoperability among mobile agent systems. It has been developed by Crystaliz, General Magic, GMD Fokus, IBM, and The Open Group. It extensively addresses security but it leaves resource management for future standardization. The goal was interoperability between the agent systems, in particular: agent transfer, class transfer, agent management, agent and agent system names, agent system types, and location syntax. Non-goals include interfaces between applications and the system, agent communication, and language

interoperability. Choice of selected functionality for standardization and the level of complexity are described in Table 1.More details about MASIF can be found in [17, 50, 59].

Table 1. Functionality Standardized by MASIF and the Level of Complexity.

Function	Addressed by MAF	Level of Complexity to Support
agent management	YES	straightforward
agent tracking	YES	straightforward
agent communication	NO	N/A
agent transport	YES	complex

4.1 Security

The security capabilities of current CORBA implementations can be categorized as: no security, proprietary security services, and conforming to CORBA security services [19]. The CORBA security services can conform to CSI level 0, 1 or 2 as defined in [20]. Secure ORBs exchange security information (a *Credential* object) about principals for remote operations. The information in the *Credential* may be used for authentication.

CORBA security services offer client authentication services via the PrincipalAuthenticator object. The client invokes the authenticate operation to establish its credentials. When the client makes a request to create an agent, it makes the credentials available to the destination agent system. The principal for the new agent is then obtained via these credentials. The agent system uses this information to find and apply the appropriate security policies. A non-secure ORB does not provide client authentication. If a client creates an agent in such an environment, the agent will be marked as “not authenticated”.

CORBA security services can require mutual authentication of agent systems during migration. Authentication can be one-hop and multi-hop. Most systems currently support one-hop authenticator in which a one-hop authentication can be applied to objects traveling only one hop from their source system. For any communication, the requestor may be able to specify its integrity, confidentiality, replay detection, and authentication requirements. MASIF currently addresses only one-hop authentication.

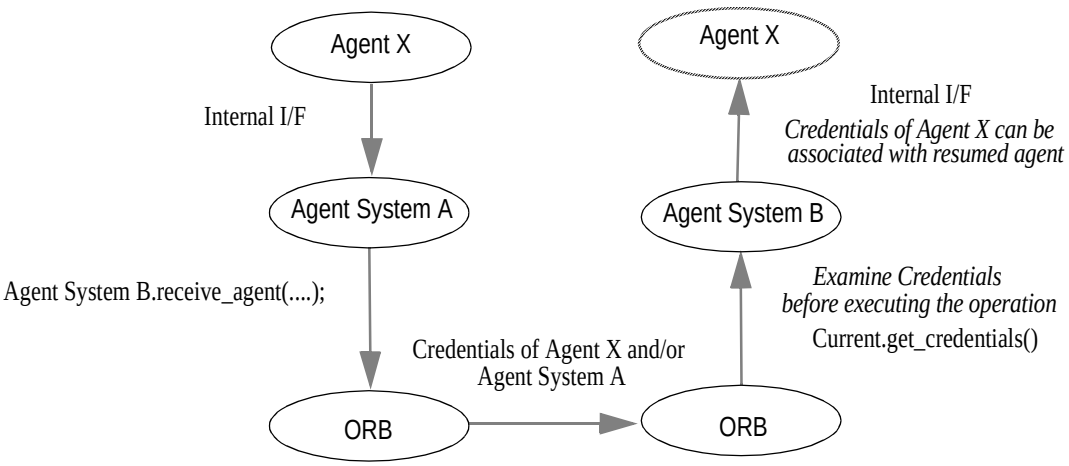

Figure 7. Delegation of credentials: *one or more credentials can be propagated; the destination system chooses credentials it will use.*

Both the source and destination agent systems transfer credentials before an agent transfer occurs, making it possible to apply security policies before transferring the agent, protecting agents from illegitimate agent systems and agent systems from illegitimate agents. A non-secure ORB does not provide mutual authentication of agent systems. An agent initially marked as authenticated is marked as "not authenticated" if it visits a non-authenticated agent system.

The destination agent system must identify the principal on whose behalf an agent is acting. That principal need not be authenticated by the agent system, certain applications may use application-defined authentication. An agent system maintains the following information about an agent it is hosting:

- The agent's name (principal and identity)
- Whether the principal has been authenticated
- The algorithm used to authenticate agent

It is desirable that secure ORB implementations propagate the agent's credentials across migration. If the destination agent system requires the sending agent system credentials, then this is only possible using composite delegation, which involves both parties in the transfer request, and propagates the credentials of the agent and the sending agent system.

Upon receiving an agent's credentials, the receiving agent system establishes the agent's credentials as the invocation credentials of the agent. As a result, any operations invoked by the agent will be subject to the policies associated with the agent's principal. This approach also ensures the continued propagation of the agent's credentials when the agent makes other transfers.

If an agent system receives an agent from an untrusted agent system, it may choose to weaken the agent's credentials. For example, it may wish to treat the agent as unauthenticated.

The propagation of both agent credentials and agent system credentials is only possible with composite delegation, available with ORB implementations that conform to CSI level 2. Delegation of credentials is needed to identify an agent's principals when an agent invokes a method on CORBA objects. In CSI level 0 and 1 implementations, only one of the credentials of the agent or the agent systems can be transmitted. If mutual authentication between agent systems is not required (e.g. in a trusted environment), the agent's credentials may be propagated to the destination agent system in lieu of the agent system's credentials. In non-secure ORB implementations, an agent's credentials are not propagated between agent systems.

Secure ORBs allow specification of the quality of security service and security level by setting the security features of the invoker's credentials, or the quality of protection in an object reference. These include: integrity, confidentiality, replay detection, misordering detection, and target authentication.

4.2 Resource Management

MASIF has limited the amount of effort dedicated to resource management for a few reasons. MASIF was intended to be a minimal specification and addressing resources would have violated this goal. It is also hard to provide a single standard covering resource management for different platforms and systems. Finally, it was intended for practical usage for a first reference implementation, and other agent features would be addressed later. For the time being, only the existing CORBA support can be used for resource management.

4.3 MASIF Contributions and Lessons Learned

MASIF was a useful experience that demonstrated collaboration of implementors of a few mobile agent systems who solved similar problems, sometimes finding different solutions. MASIF correctly limited the standard only to a limited set of mobile agent system interoperability features.

MASIF demonstrated how to support mobility on top of an industrial middleware-based system, such as CORBA. It demonstrated re-use of CORBA services, such as lifecycle, naming, externalization, and security.

MASIF offered an extensive analysis of industrial security support: what was possible to standardized at the time and what had to wait until security for mobility support matures, such as composite delegation.

Nevertheless, MASIF happened too early in the development stage of the field of mobile agents. There were only a few systems that adopted it and that were MASIF compliant, such as Grasshopper [14] and IBM Aglets [46].

4.4 Related Work

Within the FIPA organization there is another effort pursuing standardization of agent mobility [22]. FIPA standardizes agents in general, and addresses mobility as just another capability of agents. The MASIF team is participating in FIPA efforts and MASIF is a contender for agent mobility support. FIPA explicitly addresses both security and resource management as requirements for agents and mobile agents. A number of projects at the recent ACTs event in Europe have indicated their support for either the MASIF or FIPA standard [1].

5. Mobile Objects and Agents (MOA)

The MOA project has been developed in The Open Group Research Institute between 1996 and 1998 for SECOM, utilizing funding provided by MITI, Japan. A few sites have been using MOA: SECOM, University of Denver, INRIA, France, and University of Coimbra, Portugal. The goals

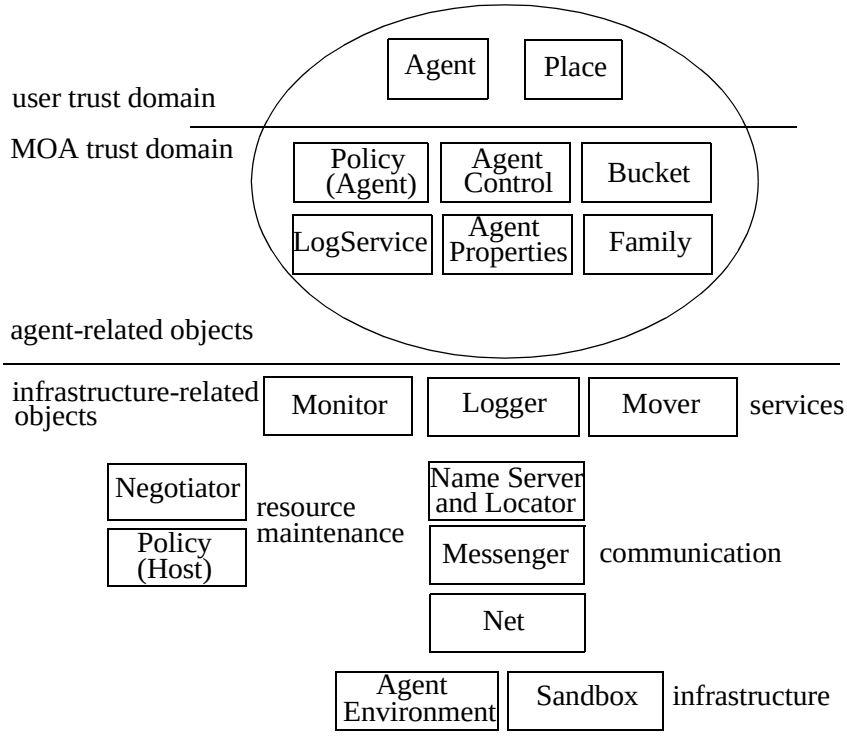

Figure 8. MOA Objects: *these are objects in the user trust domain and objects in the MOA infrastructure trust domain; the agent environment object is a container for all MOA objects.*

of the project were maintaining communication channels across migration, resource management and utilizing a component based model. More details about MOA can be found in [52].

5.1 Security

MOA is compatible with the JDK 1.1 security manager, however no security manager has actually been implemented. Many security features were left open for the next release, such as the work on authentication, and authorization of agents. We have actually implemented only the following features.

The MOA object model was divided into the agent system and agent related parts (see Figure 8 for more details on each particular object see [52]). Furthermore, the agent-related part was divided into the user and MOA system trust domain. The application can not make any changes to the system part and only the owner of the agent application can modify parts of the agent proper-

ties, while the rest of them are system dependent (see the following section for the more details on agent properties).

Thread switching was employed to extend the Java security model. Services are provided by threads containing only trusted classes. When a MOA system thread has to switch trust boundaries (for example in the case of an incoming message, or opening a channel), the request is passed to a system queue serviced by a pool of application threads allocated for that specific trust domain. A thread from the pool processes requests by calling application specific methods. The request resumes either upon receipt of the response, or upon the timeout, whichever happens first. This prevents the application from stalling the system by thread exhaustion, or by impacting performance through overuse of system threads. Resource usage is tracked on a per sandbox basis. The performance implications of thread switching were not investigated in the initial MOA implementation, cause it was assumed that the penalties were not significant compared to the agent transfer time from the perspective of agent performance. From the perspective of the server performance, scalability was lower priority than the deployment of agents. Therefore, we assumed that the initial design and implementation based on context switching was an acceptable solution.

Each agent has its own name space as defined by the bucket in which it is transported. A name space consists primarily of bytecodes and serialized objects. One complication arises when an agent returns to a place that it has left. In this case, the name space is a combination of the original bytecodes and the returning objects. This is achieved by nesting the returning bucket (with the meaning of classloader in this context) within the bucket of the remaining place.

MOA uses the standard JAR file format for passing agents. This format has provisions for digital signatures, allowing for authentication. At the moment, although the design and implementation of authentication mechanisms have been considered, MOA has not addressed authentication. For example, the agent's authenticity will be maintained as a part of the agent's name object. The agent name is internal to the MOA agent system and maintained only as a String, although it was considered to upgrade it to an object in order to maintain security related information.

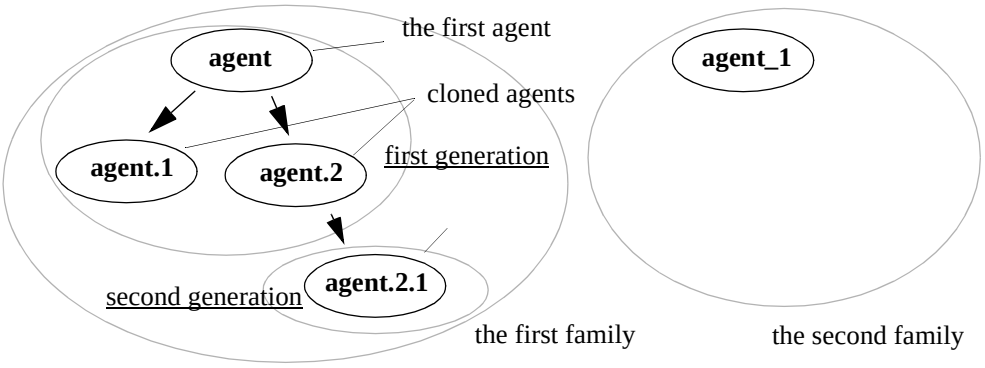

Figure 9. Naming of agents: *agent names are organized around agent families and generations. Cloned agents always carry the name of its ancestor as a part of their name.*

Agents are organized in families and generations, as presented in Figure 9 First generation agents are named after the agent system where they were created. If cloned, agents are named after the agent system (also called agent environment in MOA) of their first ancestor, extended with their generation number, irrespective of the agent system where they were actually cloned. In other words, each agent bears the sign of the original site responsible for initiating the agent family. This is used as an ultimate source of information on the current agent location: as a last resort, the home agent system can be queried for current location information. The place name consists of the name of the agent it belongs to, extended with the agent system name where place currently resides. The agent name syntax is presented in Figure 10. Agent environments are named using host name and port number. Servers (e.g. name, application) have the same naming syntax.

5.2 Resource Management

One of the initial goals of the MOA project was to support extensive resource control of various MOA resources. Resource management is deeply ingrained in the design decisions of many MOA

AEname *(ae)*: ***h:p***
AgentName *(a)*:$ae_{home}\#f\_l.g$
PlaceName: $a_{owner}\%ae_{residing}$
ServerName: ***h:p***

h - host name
p - port number
f - family name
l -launch number
g - generation number

Figure 10. MOA Named Objects: *Agent Environment, Agent, Place and Server.*

layers and components. This support would have been hard to add as an afterthought. The following limits are enforced on MOA resources:

- agent: lifetime, places, hops, channels, clones
- place: lifetime, nested places, channels, agents
- agent environment: agents, places, channels

These limits are verified by each MOA function that can impact the values, such as moving, or opening a channel. Should the limits be exceeded, the function is interrupted and the appropriate exception is thrown to the component that invoked the function.

Prior to being accepted at a node, the agent negotiates for MOA resources: which and how many it can utilize at the target MOA system. This is achieved by calculating local policy from the agent policy and host policy. The agent local policy is enforced during its lifetime at the visiting MOA system. Figure 11 presents how the policies are configured in MOA system. More details how component based computing is applied in MOA is described in [52].

MOA did not address resource management unsupported by the JVM (e.g. memory, processing, and communication). Whereas it would be possible to enforce some of these by modifying the JVM, we refrained from any deviation from *de facto* standard solutions. Imposing resource limits has impacted the design and implementation of the MOA system. Resource accounting and checking of limits is performed at various levels of the system, such as in the communication module, negotiation module, sandbox module, etc.

5.3 MOA Contributions and Lessons Learned

MOA demonstrated that many operating systems techniques can be applied in the development of middleware systems. A substantial experience was drawn from the area of operating systems,

AgentProperties (owner, familyName, home/alternateHomeAE, lifeTime)
AgentPolicy (maxLifeTime, timeRemaining, maxChannel, remainingPlaces, maxThreads)
HostPolicy (maxLifeTime, timeRemain, maxChnl, remainPlaces, maxThrd)

Figure 11. MOA Configuration of agent properties and agent and host policy.

such as communication channels and messaging protocols; locating and naming of mobile agents; resource management; negotiation policies, synchronization among agents, etc. Applying these techniques at the middleware level is easier, and more robust, than at the operating system level.

Nevertheless, achieving transparency in communication (maintaining channels across migration) was more complex to support than originally thought, despite relaxing compared to operating system transparency.

Applying some of the OS techniques to separating agents and agent system protection domains proved to be useful for the initial prototype. It was an appropriate assumption to separate agent application and system state.

Resource management was straightforward to design and implement, especially since it was planned for from the very start of development. Resource management contributed in terms of protecting hosts from overly demanding agents, by putting limitations on resource consumption.

Component-based computing has somewhat slowed us down during the development. The costs existed both in terms of development effort, as well as run-time. The learning curve was high to get accustomed to Java Beans; e.g. to provide additional methods to inspect/set properties; to take care that all classes are serializable; to create jar files for both the agent application and agent system; and to link (or wire) components once they are loaded. Nevertheless, the benefits have at least returned the investment.

Immediate benefits of complying with the component model were stronger enforcement of component boundaries than is the case with object boundaries. The components are loaded instead of constructed and component boundaries enforced careful design of what is serialized, particularly useful for application development.

5.4 Related Work

There is much related work on mobile agents and on security, Telescript provided seminal treatment of security and resource management [79]. Other examples include: Tacoma addresses security and fault tolerance [38], Ara addresses security and resource management by modifying the

JVM [62], Agent Tcl provides some basic security support [28, 44], the Mole project [9], and Concordia addresses security together with persistence. M0 [75] and Voyager [26] investigate mobile agents as mobile objects. A capability based system is suggested in work by Hagimont and Ismail [31] and protection of an agent application is researched in [64]. A new capability based system was developed by Shapiro et al. [67]. Bryce and Vitek developed a mobile agent kernel that addresses Java security limitations [16]. Security of mobile agents has been presented in a book by Vigna [76]. A book by Bradshaw represents a good source of information on agents in general [13]. Chess discusses security among other features of mobile agents [18].

6. Conversant

Conversant is an active network prototype being built under a DARPA active networks program. Active networks move extensibility properties into the network layer by allowing code to be dynamically loaded into intermediate nodes on the network [72]. This has been demonstrated to be useful for applications ranging from customized congestion control to web caching [12, 82]. Generally, the active network concept differs from the agent concept in that mobile code acts at a lower level of abstraction. That is, the node's network layer (e.g. protocol drivers and device drivers) is visible and modifiable.

6.1 Security

Conversant uses Java as its execution environment and vehicle for mobile code. Java's security features, such as type-safety and namespace partitioning, are useful in enforcing controllable non interference between mutually suspicious code. Although these features impose some runtime overhead to ensure that safety, other research projects (e.g. SPIN [23, 69]) have shown that type safe languages can efficiently be used for low level programming.

Authentication and authorization are done using cryptographic key techniques. The end user is the principal, and is authenticated using public key encryption. A special user, the system manager, controls policies which map resource rights to users. A policy is a code module which determines the validity of a resource request based on arbitrary criteria. We will build this infrastructure using

features from Java's latest version (JDK 2), which include classes that simplify the bookkeeping necessary to maintain this information, as well as libraries for performing cryptography. Note that the Conversant project has not yet addressed whether a secure transport mechanism (SSL, etc.) will be used, or what the mechanisms for key distribution will be.

A problem with Java is that its security model was designed to protect the host platform from malicious code, but not the Java code from performance interference or outright denial-of-service attacks by other malicious Java code. Thus, the resulting environment is poorly suited as a multi-processing environment [8]. Although the language's type-safety prevents pointers from being forged to point into another users private data space, numerous denial of service attacks on resources are still possible. Interestingly, the hierarchical composition of units of mobile code in Conversant fully exposes Java's vulnerability to denial-of-service attacks. This becomes clear in the following example. Although both units A and B do pass individual authentication and integrity checks, the composition of the two units (either A over B, or B over A) may inadvertently result in a forever loop, or a resource hog (e.g., due to incompatibilities between A and B).

Because Conversant's goal is reliable communications, it is crucial to thwart this type of attacks.

6.2 Resource Management

The Conversant project has found it useful to characterize denial of service attacks in two ways. *First order* and *second order* attacks. First order attacks may be carried out with relative ease; the code capitalizes on overt loopholes, is simple to write, and it may cause damage quickly. Second order attacks employ more subtle attacks, are difficult to prevent, and may be harder to implement.

In Java, first order denial of service attacks on physical resources come in two varieties. Firstly, attacks may be made on the system heap. There is only one heap that is shared by all classes, and there is no way to limit a particular thread from taking all of it. Secondly, a JVM which implements a thread scheduling algorithm that reschedules only when threads block (such as FIFO) would allow a thread to enter an endless loop, thus denying other threads CPU cycles.

Second order denial of service attacks come in numerous flavors but frequently result from side effects of VM implementation choices. With sufficient knowledge of the VM internals it is possible voluntarily to conduct such attacks. We hand-crafted evil programs to exacerbate such side effects. In a first test case, the program relies on a feature of most VM implementations, where the hashCode() method returns the virtual address of the object as an integer. Storing these integer values in a memory area that is known to be scanned by a conservative garbage collector (such as the stack) prevents the virtual machine from removing unreachable objects. In a second test case an object finalizer is written such that it never returns. This rapidly hangs a VM wherein if it is implemented such that a single thread is dedicated to handle object finalizations. One may also play tricks against the heap fragmentation, to the degree that allocations can no longer be made before the garbage collector can perform compaction. Worse yet, malicious code can set up non-malicious code to trip over these deficiencies and carry the blame.

Clearly, in solving the aforementioned problems, changes must be made somewhere in the Java system. The Conversant designers have evaluated four mechanisms:

- **Byte code editing**. The byte code is modified prior to execution to conform to particular coding conventions, such as placing a yield within loops. There is no dependency on the VM implementation but the accuracy is severely limited and second order attacks may be hard to detect.

- **Explicit coding conventions**. This technique imposes coding restrictions at the language and standard packages levels. The classes are either statically verified and signed or dynamically verified by a dedicated class loader. As compared to byte code editing, this mechanism does not alter the original byte code, but may severely limit the expressiveness of the JAVA language.

- **Multiple JVM instantiations**. In this “wrapper” approach, a process executes the JVM for each and every principal. Thus, the underlying operating system, rather then the JVM, enforces how often a given principal is scheduled and how much memory it receives. This solution is attractive because it necessitates few changes to the JVM However, it may not scale well as the number of principals - and thus the burden on the kernel in managing

many processes - grows. Another problem is that we would like principals need to share data and code amongst each other, and the IPC mechanisms necessary to enable such communication across processes would hinder performance.

The scalability problems of the wrapper approach may be mitigated if for each principal a new JVM needs to be instantiated within the same process. Presumably distinct kernel threads would be associated with each instantiation. This solution poses some complicated technical problems: the JVM would have to be modified such that it could distinguish the different JVM incarnations running within a single process, and a means would have to be found to enforce memory limits on a per thread set basis.

- **Direct JVM Modifications:** Rewriting the JVM such that it partitions physical resources fairly or on the basis of a “pay for what you use” policy in a multiuser environment. One can modify it such that kernel threads are used, thereby ensuring the host OS’s scheduler fairly shares the CPU. Fairly sharing the heap, on the other hand, is more complicated. On the other hand, it is far more complicated to share the heap in a fair way (or unfairly, if the policy says so).

The problem is choosing whether to modify the runtime JVM, restrict the language and standard packages or modify the semantics of the Java language itself. Modifying the language would conflict with Conversant’s goals of interoperability. The developer should continue to write in pure Java (or some controlled subset), rather then a Conversant-specific derivation. On the other hand, we expect the JVM to be specialized software that is eventually integrated into special-purpose machines such as routers. The Conversant JVM already must be modified to meet the engineering constraints of an embedded system. Thus, Conversant’s resolution is to thwart denial of service attacks by changing the JVM without changing the language [10]. We chose to experiment our resource control mechanisms within the Kaffe Java VM [42].

CPU control

Java threads and synchronization primitives were mapped to the equivalent verbs in the POSIX standard. For configurations required to apply a *fair access* policy, we set the thread’s scheduling

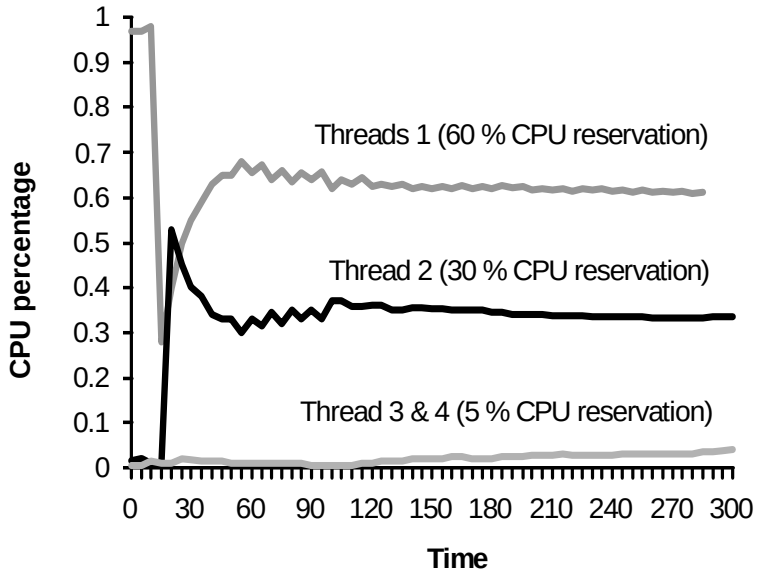

Figure 12. Constraining 4 threads to their respective CPU quota.

policy to round-robin, with a common priority value and the default quantum. For the pay for what you use policy a feed-back based scheduling mechanism is more appropriate. A JVM Internal thread running at the highest priority periodically samples actual thread speed, defined as the number of cycles used in the unit of time.

If actual speed is greater than maximum one, the thread priority is temporarily depressed. A proactive kernel mechanism would be more efficient and is the subject of further studies.

Memory control

As illustrated in Figure 13., we have modified the VM's heap management allows so that a new private, independently garbage-collected heap is bound to newly created name-spaces (via class loader instantiations). There is no need for objects in different name space to be in the same heap, in that the Java model ensures that they are not mutually visible. Java defines a "null classloader" which loads system classes (e.g. java.lang.* and java.util.*). These classes are shared by all entities within the JVM. Access still has to be made somehow to a "system heap" with its own garbage collector, in which information pertaining to the null classloader would be stored. This shared heap also needs to be impervious to denial of service attacks. So it is crucial that cross ref-

erences from one heap to another cannot be made. Cross references are caught at runtime and exceptions are thrown. The rules that determine which objects/classes get their memory estate from which heap are shown in Table 2..

Table 2. Heap allocation rules. Class instances allocation rules. In contrast, class objects are always loaded and initialized in the heap the classloader belongs to.

	User Class	System Class
User Thread	User heap (foo.bar, ...)	User heap (java.*, threads, ...)
System Thread	User heap	**System heap**

With an implementation of such mechanisms, we have demonstrated that pathological resource hog test cases do not intervene with concurrent well-behaved flows. The final prototype required minimal changes to our active network test bed (ANTS) such that it be cross reference free.

Most of Conversant's efforts were devote to on controlling the CPU and memory resources. Resources such as network bandwidth or cache entries are more likely to be effectively controlled via dedicated APIs.

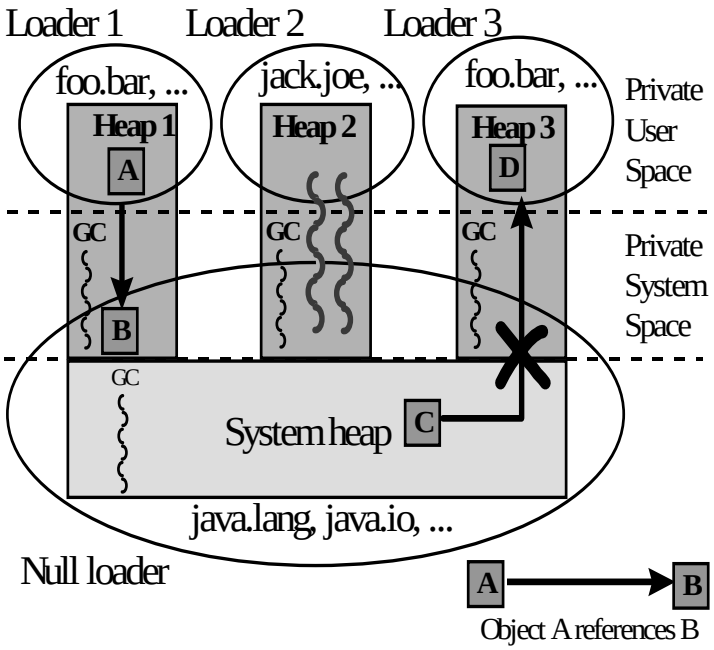

Figure 13. Binding private independent heaps to newly created JAVA name spaces.

6.3 Conversant Contributions

Conversant is a runtime for active network nodes that

- is explicitly designed to implement dynamic pay-for-what-you-use policies with the comfort of a commodity type-safe language (Java) and with added systemic resilience to resource theft and denial-of-service attacks among mutually suspicious code (above and beyond Java traditional security);

- provides a framework for the hierarchical composition of primitive building blocks (micro-protocols); and

- provides APIs and toolkits to successfully negotiate resources (be it computation or communication resources) in the Java-derived resource-safe system.

Nortel Networks' recent Active Networks prototypes [36, 74] build upon many Conversant concepts.

6.4 Related Work

ANTS is a an environment for processing active code that is written in pure Java [77, 78]. As in MOA, this limits resource control to logical resources and bases most of its security on Java abstractions (namespace partitioning, etc.) Joust, in contrast, is written using a custom JVM built on top of the Scout operating system [32]. Although the custom JVM enables physical resource control in principal, the emphasis in the Joust literature is fast-path optimizations rather then security. PLAN is a programming language designed explicitly for active networks [30]. Resource consumption is implicitly controlled because the language's rules force programs to be written in a way that consume a bounded amount of physical resources, the amount of which is statically verifiable. To enable this, the rules are very strict (e.g. no loops are allowed in programs). APIs to the host environment are provided to perform actions not expressible in the PLAN language; the APIs are guarded using access control lists.

7. Summary

In this section we summarize our observations in the areas security, resource management, and in a general category.

7.1 Security Observations

In the field of security we have made the following observations:

- **Security from historical perspective.** Early theoretical systems and systems employing task migration were generally not focused on security, though a few special dedicated exceptions exist [65, 66]. Instead they concentrated on resource management issues. They were designed for closed system where security does not have high priority, and resource control is focused on performance, but not on protecting system against distributed attacks. Early middleware systems addressed security thoroughly, however resource management was somewhat neglected. This appears due to the dependency on the underlying operating system for resource management, consequently there is less possibility for control from the middleware system. Recent middleware systems, such as active networks and mobile agents are focusing on both security and resource management.

- **Security has many facets**. This is particularly true in those systems that admit mobile code and distributed behavior. In most cases, designers quite aptly care about the integrity of the system, and about the integrity of the communication among systems. This is necessary, but not sufficient. Increasingly, the sophistication of Quality-of-Service classes and pay-for-what-you-use policy models require the designers to appreciate non-interference guarantees and resilience to denial-of-service attacks. Both the former and the latter have hitherto fell off of the roadmap and feature lists of most products. We are five years into the Java era, for instance, and yet it is still trivial for a Java application to mount a denial-of-service attack, or to doctor system resources in the runtime in a way that some other Java code will fail to execute next. Conversant's atypical approach was to make non-interference and resource safety the first, major stepping stone upon which the rest of the

Conversant environment unfolds. Security is the most-researched aspect of mobile agents, including MOA, MASIF as well as many other mobile agent systems [14, 27, 28, 38, 46, 62].

- **Security is more easily compromised for mobile than for stationary systems.** In the case of Mach, this is obvious from the examples of interposition both at the kernel and user level. The former even allows for compromising the kernel consistency. In the case of today's mobile agent systems, this is indicative based on the amount of existing work on security with respect to mobility.

- **The client server security model is not an acceptable solution for mobile objects.** This is a model where an intermediate server is responsible for authentication and authorization chores. Capability-based authentication is much more suitable for mobile agent systems, such as Actor's unforgeable names, Mach ports or MASIF's use of certificates as a part of the name. Capabilities appear to be highly appropriate for mobile objects because they are easily transferred and do not introduce residual dependencies.

- **Naming is closely related to authentication.** This is true in all systems described. Some perform authentication exclusively based on the name String (without introducing any cryptography). Actors abstracts the naming and security at the higher level. In others, names are protected by operating system guarantees (e.g. in Mach), whereas in others the mobile object identity consists of a combination of the authenticator and the name.

- **Authorization policies for mobile object systems are still in an emergent state.** In older systems, they were not needed within the confines of a single trust domain. With the requirement for larger systems, e.g. middleware systems such as CORBA and later Java, authorization is being introduced into mobile object systems. As evidenced by the evolution of MASIF or new security models of the JVM, these authorization policies are still evolving.

- **Protecting the arriving object from the visited system is rarely addressed.** None of the systems described in this paper have addressed this problem. This has not been an issue in the past, however it is drawing interest in the past few years [64]. In the past more

attention was paid that the incoming object does not impact the surrounding as a side effect, e.g. to get clobbered memory in the case of Actors, or loose some state in the case of migration. In the case of networking (Conversant) protecting the existing objects from arriving ones has always been an issue. You need at least to look at the packet type (header) to reject it. So this a potential source of attack, denial of service.

- **Mediation of shared resources was always a hard problem to solve.** In the case of Mach, the distributed shared memory was a single hardest problem to solve even for correctness only. Conversant has not addressed the problem of how to mediate shared resources such as caches, networking tables, user credentials, such that it does not leave the door open to denial of service attacks (e.g. creating too many entries over short period of times). There is a clear distinction between resource partitioning (such as CPU, Memory) and resource sharing, where principals access read/write some shared data. As soon as you start to share resources (mostly for efficiency, to avoid duplicating resources) then one must be ready to give up a bit on security against denial of service.

7.2 Resource Management Observations

The following observations have been drawn on the topic of resource management.

- **Past systems have addressed resource management issues in an ad hoc manner.** Actors early on supported the notion of sponsorship of activities. In task migration, resources were accounted for both system and the incoming objects (tasks). Recently, with the advent of QoS has this subject gained substantial attention. Currently, issues are handled subject to the limitations of the underlying environment. For example, in Mach, modifications to the microkernel were required to profile it for remote communication and paging, while in Conversant, it was necessary to make modifications to the JVM. One of the top conclusions for Conversant is that Java was a major step forward regarding mobility and security but really lacks physical resource management. However, because of the standardization and world-wide deployment, the changes for resource management are much harder to deploy in nowadays systems than it was the case with early systems.

- **Mobile object systems typically offer a combination of logical and physical resource management.** All of the system types surveyed adhere to this model. Earlier systems were more weighted towards logical resource management, which is much easier to achieve but often has only a limited effect upon the system performance. For example, Actors supported the notion of the meta-architectures. Physical resource management is harder to manage and often requires modifications to the underlying environment. Load distribution in Mach addressed processor, VM, and communication bandwidth. Recent QoS requirements have caused newer systems to address these issues, as can be seen by the work done in Conversant. The java language naturally fits in mobile object systems, since it is an interpreted language, object oriented, especially designed for security and has a well defined set of API packages, available at any node. Nevertheless, as the MOA and Conversant projects point out the basic physical resources (CPU and Memory) are very poorly handled in Java. Furthermore, there is a duality between resource management and security, since insufficient resource management may lead to denial of service.

- **Recent systems provide some form of negotiation between the arriving object and the receiving object system.** This negotiation is utilized in order to verify whether the necessary resources are available at the system and to guarantee that the agent will not abuse the system's available resources. In Actors, the negotiation is abstracted at the meta-architecture level. In MOA system, the resources are explicitly negotiated prior to accepting the object at the visited host. Negotiation is typically asymmetric in that host is privileged over the arriving object. Conversant also negotiates resources before the packets (capsules) are evaluated. The negotiation is in fact performed end to end first.

- **Clean OS abstractions and frameworks can be a useful starting point for mobile object systems.** Conversant's experience in bringing resource safety into a system (Java) that is inherently resource unsafe has once more demonstrated that clean OS abstractions (e.g., Java's name spaces) and frameworks (e.g., Java's classloader) go a long way. Conversant literally fork lifted the standard notion of Java name space and overloaded with the new notion of non-interference guarantees. Without having name spaces as a

starting point, it would have been woefully hard (or impossible) to bring resource safety into Java and yet preserve compatibility with the Java legacy world. Similar observation applies for security model and naming and locating in MOA.

7.3 General Observations

Some of the lessons learned from the presented cases of mobile object systems that could be readily applied to future mobile object systems consist of the following:

- **Mobility is easy to implement in prototype stage.** There are numerous implementations of the Actors model. A few versions of Mach task migration have been implemented (and many other variations of process migration). There are numerous implementations of mobile agent systems. Active networks also achieved multiple incarnations and derivations.

- **Transparent communication is a powerful but also very complex and fragile mechanism to support.** In the case of Mach, this is a single feature thad caused most trouble and required most support. Nowadays, this is obvious from the difficulties in supporting transparent communication and the lack of standardization for mobile systems. End-to-end guarantees are hard to satisfy in the case of active network implementations.

- **Fault tolerance on top of mobility is hard to support.** Early Actors systems addressed fault tolerance by modular, dynamically loadable protocols. In the case of Mach, fault tolerance is tied to dependencies that were sprinkled throughout the cluster. Today, failure tolerance is still an issue for mobile agents. However, some of the mechanisms for migration can be readily used for checkpointing.

- **Mobility is an interesting topic that attracts interest of researchers.** It is evident from the case studies we described as well as from numerous other implementations of the mobile object systems. It may be interesting because it is a hard problem in general, and as such it attracts researchers. Security and resource management make hard.

- **There are few if any commercially successful products in mobility area.** This is true in all classes of mobile systems we described. There were only few products supporting task migration, only a couple of commercially available mobile agent systems, and active networks still have not been deployed. However, some aspects of mobility are encroaching into the networking area and networking seems to be the most promising area for mobility deployment.

8. Conclusion

In this paper we presented security and resource management as implemented in five different types of mobile object systems. For each system type, for the purpose of clarity, we selected a single project as a representative and noted related project work. It is important to note that the wide range of system types provides a broader base from which we draw our conclusions. We began with a theoretical model, then described task migration, a standard specification for mobile agent systems, a mobile agent system, and an active networks project. For each of the systems presented we have analyzed how they supported security and resource management and briefly mentioned related work for each of the system types. Our findings are tersely summarized in Tables 3-5. A more detailed comparison is beyond the scope of this paper.

Security and resource management will continue to be important aspects of mobile object systems. Both are crucial for the safety and protection of the mobile objects as well as the hosting systems. Security and resource management are addressed at various levels of the system and this is likely to continue to be the case in future systems. QoS requirements supporting the secure and guaranteed behavior of applications will require the utilization of a combination of techniques at the networking, operating system, middleware and application levels. This paper attempted to present a few examples illustrating how security and resource management have been addressed at these different system levels. We believe that exciting developments await us, and having a historical perspective going forward can be useful.

Acknowledgments

The following people deserve credit for their involvement: Carl Hewitt did the original work on Actors. Task migration on top of Mach was significantly supported by the work of Joe Barrera. OMG MASIF is the result of a large team of people from Crystaliz, General Magic, GMD, IBM and The Open Group. A significant part of MOA has been implemented by Bill LaForge. We would also like to thank anonymous reviewers of this and past versions of this paper whose comments significantly improved contents and presentation of the paper. Thanks to Jan Vitek for encouraging us to make a journal version of the workshop paper and providing an excellent feedback. Finally thanks to Danny Lange, the invited editor of this special issue, for encouraging us to submit the paper.

References

[1] ACTS Domain 5, *Agent Cluster Baseline Document*, editor T. Magedanz, January 1998.

[2] G. Agha, “Actors: A Model of Concurrent Computation in Distributed Systems,” *Artificial Intelligence Series*, MIT Press, Cambridge, Mass., 1986.

[3] G. Agha, and Jamali, N., “Concurrent Programming for Distributed Artificial Intelligence,” in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, MIT Press, editor Gerhard Weiss, chap 12, 1999.

[4] G. Agha, S. Frölund, W. Kim, R. Panwar, A. Patterson, and D. Sturman, “Abstraction and Modularity Mechanisms for Concurrent Computing,” *Parallel and Distributed Technology*, IEEE Computer Society, vol 1, no 2, pp 3-15, May 1993

[5] G. Agha, I. Mason, S. Smith, and C. Talcott, “A Foundation for Actor Computation”, Journal of Functional Programming, vol 7, pp 1-72, 1996.

[6] P. Amaral, C. Jacqemot, Jensen, P., Lea, R., and Mirowski, A., “Transparent Object Migration in COOL-2,” *Proceedings of the ECOOP*, June 1992.

[7] W. Athas, and N. Boden, “Cantor: An Actor Programming System for Scientific Computing,” *Proceedings of the NSF Workshop on Object-Based Concurrent Programming*, editors G. Agha, P. Wegner, and A. Yonezawa, pp 66-68, ACM, April 1989.

[8] D. Balfanz, L. Gong, “Experience with Secure Multi-Processing in Java,” *Technical Report 560-97*, Department of Computer Science, Princeton University.

[9] J. Baumann, F. Hohl, K. Rothermel, M. Straßer, “Mole - Concepts of a Mobile Agent System,” WWW Journal, Special Issue on Applications and Techniques of Web Agents, 1(3):123-137, pp 2-13, *Baltzer Science Publishers.*

[10] P. Bernadat, D. Lambright, and F. Travostino, “Towards a Resource safe Java,” IEEE workshop on Programming Languages for Real-Time Industrial Applications (PLRTIA), December 1998.

[11] B. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczinski, D. Becker, C. Chambers, and S. Eggers, “Extensibility, Safety and Performance in the SPIN Operating System,” *Proceedings of the 15th Symposium on Operating Systems Principles*, pp 267–284.

[12] S. Bhattacharjee, K. Calvert, E. Zegura, “On Active Networking and Congestion”. Technical Report GIT-CC-96-02, College of Computing, Georgia Tech.

[13] J. Bradshaw, *Software Agents*, AAAI Press / The MIT Press, Cambridge, MA, 1997.

[14] M. Breugst, T. Magedanz, “Mobile Agents — Enabling Technology for Active Intelligent Network Implementation,” IEEE Network, May/June 1998, vol 12, no 3, pp 53-60.

[15] J.-P. Briot, “Actalk: a Testbed for Classifying and Designing Actor Languages in the Smalltalk-80 Environment,” *Proceedings of the European Conference on Object-Oriented Programming (ECOOP’89)*, pp 109-129, July 1989.

[16] C. Bryce, and J. Vitek, “The JavaSeal Mobile Agent Kernel,” *Proceedings of the First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents (ASA/MA’99)*, October 1999, pp 103-117.

[17] D. Chang, and S. Covaci, “The OMG Mobile Agent Facility: A Submission,” *Proceedings of the First International Workshop on Mobile Agents*, Lecture Notes in Computer Science no 1219, Springer Verlag, pp. 98–110, April 1997.

[18] D.M. Chess, B. Grossof, C. Harrison, D. Levine, C. Parris, and G. Tsudik, “Mobile Agents: Are They a Good Idea,” IBM Research Report, RC 19887, October 1994.

[19] CORBA Security, The OMG Specification, OMG Document Number 95-12-1.

[20] Common Secure Interoperability, The OMG Specification, OMG Document Number; orbos/96-06-20.

[21] D. Eager, E. Lazowska, and J. Zahorjan, May “Dynamic Load Sharing in Homogeneous Distributed Systems,” *IEEE Transactions on Software Engineering*, 12(5):662–675, 1986.

[22] FIPA http://drogo.cselt.it/fipa/.

[23] M. Fiuczynski, B. Bershad, “An Extensible Protocol Architecture for Application-Specific Networking,” in *Proceedings of the 1996 Winter USENIX Conference,* San Diego, CA., pp. 55-64, January 1996.

[24] B. Ford, M. Hibler, J. Lepreau, P. Tullman, G. Back, S. Clawson, “Microkernels Meet Recursive Virtual Machines,” *Proceedings of the USENIX Second Symposium on Operating Systems Design and Implementation (OSDI’96)*, Seattle WA, October 1996, pp 137-152.

[25] B. Ford, M. Hibler, J. Lepreau, R. McGrath, P. Tullman, “Interface and Execution Models in the Fluke Kernel,” *Proceedings of the USENIX Third Symposium on Operating Systems Design and Implementation (OSDI’99),* New Orleans La, February 1999, pp 101-116.

[26] G. Glass, “ObjectSpace Voyager Core Package Technical Overview”. ObjectSpace, White Paper, Reprinted in [54].

[27] Grasshopper, http://www.ikv.de/products/grasshopper.html.

[28] R. Gray, D. Kotz, G. Cybenko, and D. Rus, “D’Agents: Security in a Multiple-Language, Mobile-Agent System,” In Giovanni Vigna, editor, *Mobile Agent Security,* Lecture Notes in Computer Science, Springer-Verlag, 1998.

[29] A. Grimshaw, t al. “The Legion Vision of a Worldwide Virtual Computer,” CACM, vol 40, no 1, Jan. 1997, pp 39-45.

[30] C. Gunter, M. Hicks, P. Kakkar, J. Moore, S. Nettles, J. Carl, “PLAN: Language-Based Safety and Security for Active Networks”. SOSP 97. October 5-8, 1997, Saint Malo, France.

[31] D. Hagimont, and L. Ismail, “A Protection Scheme for Mobile Agents on Java,” *Proceedings of the 3rd ACM/IEEE International Conference on Mobile Computing and Networking*, September 1997.

[32] J. Hartman, L. Peterson, A. Bavier, P. Bigot, P. Bridges, B. Montz, R. Piltz, T. Proebsting, “Joust: A Platform for Communication-Oriented Liquid Software”. TR97-16. Dept. of Computer Science, The University of Arizona.

[33] C. Hewitt, “Viewing Control Structures as Patterns of Passing Messages,” Journal of Artificial Intelligence, vol 8-3, June 1977, pp 323-364.

[34] P. Homburg, M. van Steen, and A. Tanenbaum, “An Architecture for A Wide Area Distributed System,” *Proceedings of the Seventh SIGOPS European Workshop*, Connemara Ireland, September 1996, pp 75-82.

[35] K. Hwang, W. Croft, B. Wah, F. Briggs, W. Simons, and C. Coates, “A Unix-Based Local Computer Network with Load Balancing,” *IEEE Computer*, 15:55–66, April 1982.

[36] R. Jaeger, R. Duncan, T. Lavian, J. Hollingsworth, and F. Travostino, "Dynamic Classification in Silicon Based Forwarding Engine Environments", Proceeding of LANMAN'99 Workshop, 1999.

[37] M. Jones, “Interposition Agents: Transparently Interposing User Code at the System Interface,” *Proceedings of the 14th ACM Symposium on Operating Systems Principles (SOSP’93),* Asheville, NC, December 1993, pp 80-93.

[38] D. Johansen, R. van Renesse, and F. Schneider, “Operating system support for mobile agents,” *Proceedings of the 5th. IEEE HOTOS Workshop*, 4th-5th May, 1995.

[39] E. Jul, H. Levy, N. Hutchinson, and A. Black, “Fine-Grained Mobility in the Emerald System,” *ACM Transactions on Computer Systems*, 6(1):109–133, February 1988.

[40] D. Kafura, M. Mukherji, and G. Lavender, “ACT++ 2.0: A Class Library for Concurrent Programming in C++ Using Actors,” Journal of Object-Oriented Programming, vol 6, no 6, October 1993, pp 47-62.

[41] C. Kaufman, R. Perlman, M. Speciner, “Network Security,” Prentice Hall, 1995.

[42] Kaffe Java virtual machine. http://www.transvirtual.com/products/downloads.html

[43] W.A. Kornfeld, and C. Hewitt, “The Scientific Community Metaphor,” IEEE Transactions on System, Man, and Cybernetics, vol 11, no 1, pp 24-33, January 1981.

[44] D. Kotz, et al., “Mobile Agents for Mobile Internet Computing,” July 1997, IEEE Internet Computing, vol 1, no 4, pp 58-67.

[45] P. Krueger, R. Chawla, “The Stealth Distributed Scheduler,” *Proceedings of the 11th International Conference on Distributed Computing Systems,* June 1991, pp 336-343.

[46] D. Lange, and M. Oshima, “Mobile Agents with Java: The Aglet API,” *World Wide Web*, 1(3), September 1998.

[47] P. Maes, “Computational Reflection,” Ph.D. Thesis, Vrije University, Artificial Intelligence Laboratory, Technical Report, no 87-2, 1987.

[48] C. Manning, “Introduction to Programming Actors in Acore,” in Towards Open Information Systems Science, MIT Press, 1990, editors Hewitt, C. and Agha, G., chap 2, pp 33-80, Cambridge, Mass.

[49] D. Milojicic, *Load Distribution*, Vieweg Verlag, Advanced Studies in Computer Science, Wiesbaden, Germany, 1994

[50] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White, “MASIF - The OMG Mobile Agent System Interoperability Facility,” *Personal Technologies*, Springer Verlag, (1998), 2:117-129. Also appeared in the *Proceedings of the International Workshop on Mobile Agents*, Stuttgart, September 1998, pp 50-67.

[51] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, S. Zhou, “Process Migration Survey,” to appear in ACM Computing Surveys during 2000.

[52] D. Milojicic, W. LaForge, D. Chauhan, “Mobile Objects and Agents, Design, Implementation and Lessons Learned,” Distributed Systems Engineering, IEE, 5 (1988), 1-14. Also appeared in the *Proceedings of the Fourth USENIX Conference on Object-Oriented Technologies and Systems (COOTS ‘98)*, April 27-30, 1998, Santa Fe, New Mexico, pp 179-194.

[53] D. Milojicic, W. Zint, A. Dangel, P. Giese, “Task Migration on the top of the Mach Microkernel,” *Proceedings of the USENIX Mach Symposium*, pp 273-290, Santa Fe, USA, April 1993.

[54] D. Milojicic, F. Douglis, and R. Wheeler, “Mobility – Processes, Computers, and Agents,” Addison Wesley and ACM Press, February 1999.

[55] D. Milojicic, P. Giese, W. Zint, “*Experiences with Load Distribution on top of the Mach Microkernel*,” *Proceedings of the 4th USENIX Symposium on Experiences with Distributed and Multiprocessor Systems*, pp 19-36, San Diego, September 1993.

[56] L. Moreau, C. Queinnec, “Distributed Computations Driven by Resource Consumption,” *Proceedings of the IEEE International Conference on Computer Languages (ICCL’98),* May 1998, Chicago, Illinois, pp 68-77

[57] M.P. Singh, “Multiagent Systems,” Springer-Verlag, no 799, Lecture Notes in Artificial Intelligence, 1994.

[58] R. Needham, M. Schroeder, “Using Encryption for Authentication in Large Networks of Computers,” Communication of the ACM, vol 21, December 1978, pp. 993-999.

[59] OMG MASIF, OMG TC Document ORBOS/97-10-05.

[60] Open Systems Laboratory, “The Actor Foundry: A Java-based actor programming environment,” Available for download at http://www-osl.cs.uiuc.edu/~m-astle/foundry.html.

[61] Y. Paindaveine, D. Milojicic, “Process vs. Task Migration,” *Proceedings of the 29th HICSS,* Jan. 3-6, 1996, pp 636-645.

[62] H. Peine, and T. Stolpmann, “The Architecture of the Ara Platform for Mobile Agents,” *Proceedings of the First Intl Workshop on Mobile Agents MA’97,* April 7-8. LNCS 1219, Springer Verlag.

[63] L. Rowe, and K. Birman, “A Local Network Based on the UNIX Operating System,” *IEEE Transactions on Software Engineering*, March 1982, SE-8(2):137–146.

[64] T. Sander, C. Tschudin, “Towards Mobile Cryptography,” *Proceedings of the IEEE Symposium on Security and Privacy,* Spring 1998.

[65] R. Sansom, “Building a Secure Distributed Computer System, “CMU-CS-88-141, Ph.D. Thesis, Carnage Melon University, May 1988.

[66] J. Sebes, “Overview of the Architecture of Distributed Trusted Mach,” *Proceedings of the Second USENIX Mach Symposium,* November 1991, pp 251-262.

[67] J.S. Shapiro, J.M. Smith, D.J. Farber, “EROS: a Fast Capability System,” *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP’99)*, Kiawah Island Resort, December 1999, pp 170-187.

[68] M. Shapiro, P. Gautron, and L. Mosseri, “Persistence and Migration for C++ Objects,” *Proceedings of the ECOOP 1989–European Conference on Object-Oriented Programming*, July 1989.

[69] E. Sirer, S. Savage, P. Pardyak, G. DeFouw, M. Alapat, B. Bershad, “Writing an Operating System Using Modula-3” Workshop on Compiler Support for System Software, February 1996.

[70] M. Swanson, L. Stoller, Critchlow, T., Kessler, R., “The Design of the Schizophrenic Workstation System,” *Proceedings of the third USENIX Mach Symposium*, Santa Fe, New Mexico, April 1993, pp 291-306.

[71] J. Tardo, and L. Valente, “Mobile Agent Security and Telescript,” *Proceedings of COMPCON’96*, pp. 52–63, February 1996.

[72] D.L. Tennenhouse, J.M. Smith, W. Sincoskie, D.J. Wetherall, G.J. Minde, “A Survey of Active Network Research,” IEEE Communications Magazine. vol 35, no 1, pp.80-86. January 1997.

[73] C. Tomlinson, W. Kim, M. Schevel, V. Singh, B. Will, G. Agha, “Rosette: An Object Oriented Concurrent System Architecture,” Sigplan Notices, vol 24, no 4, 1989, pp 91-93.

[74] F. Travostino, “Towards an Active IP Accounting Infrastructure,” Proceedings of the OpenArch 2000 Conference, Tel Aviv, Il, Mar 2000.

[75] C. Tschudin, “The Messenger Environment M0 - A Condensed Description,” In Mobile Object Systems: Towards the Programmable Internet, pp 149-156, April 1997, LNCS no 1222, Springer Verlag.

[76] G. Vigna, editor. Mobile Agents Security. Lecture Notes in Computer Science. Springer-Verlag, 1998.

[77] D. Wetherall, J. Guttag, Tennenhouse D., “ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols” IEEE OPENARCH’98, San Francisco, CA, April 1998.

[78] D. Wetherall, “Active Network vision and Reality: Lessons from a Capsule-Based System,” *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP’99),* Kiawah Island Resort, December 1999, pp 64-79.

[79] J. White, “Telescript Technology: Mobile Agents,” in Bradshaw, J., Software Agents, AAAI/MIT Press, 1996.

[80] D. Woelk, M. Huhns, and C. Tomlinson, “InfoSleuth Agents: The next generation of Active Objects,” Object Magazine July/August 1995.

[81] D. Wong, et al., “Concordia: An Infrastructure for Collaborating Mobile Agents” *Proceedings of Workshop on Mobile Agents MA’97*, Berlin, April 7-8th. LNCS 1219, Springer Verlag.

[82] L. Zhang, V. Jacobson, S. Floyd, “Adaptive Web Caching,” http://irl.cs.ucla.edu/AWC/.

Appendix Summary of Presented Mobile Object Systems

General Description	Actors	Mach Task Migration	MASIF Standard	MOA	Conversant
Type	middleware	operating system (microkernel)	specification (standard)	mobile agent system	active networks
Goals	modeling and building distrib. applications	load distribution, transparent migration at the µkernel level	interoperability among mobile agent systems	persist. comm. chn, resource manag. component model	dynamic comm. protocols, improved assurance & QoS
Status	active	frozen (Mach still lives)	frozen	ended	ended
Time frame	1985-	1991-1993	1995-	1996-	1996-1998
Who used it	MCC, MIT, University of Paris, etc.	Universities of Kaiserslautern, Utah and Louvain-la-Neuve	Grasshopper [27] (now) Aglets and MOA (in progress)	Cambridge, Grenoble RI, SECOM, INRIA, U. Denver & Portugal	N/A
References	[2, 4]	[53, 61]	[50, 59]	[52]	[10]

Table 3. General Information about Systems Being Compared.

Resource Management	Actors	Mach Task Migration	MASIF Standard	MOA	Conversant
physical resources	CPU, memory, disk, network bandwidth, and access rate guarantees	lifetime, processing, remote IPC & paging	not addressed (relies on the OS)	lifetime	CPU, memory, bandwidth
logical resources	objects, messages	communication channels	future work	MOA messages, places, channels, agents, etc.)	object cache, resource and policy descriptors
object granularity	Single objects of encapsulated groups	single task	single agent	single agent and families of agents	user-user connection
resource policies	fair scheduling, interleaved execution	local processing	N/A	suspend agent on overflow	fair share & pay for what you use
resource negotiation	resource usage costs, access rate guarantees	kernel port transfer	N/A	asymmetric negotiation, logical resources, arbitrary requirements	asymmetric
QoS	Customization of language abstractions	N/A	part of the transport	N/A	high-confidence via end-to-end flows (“paths”)

Table 4. Comparison of Resource Management of Mobile Object Systems.

Security	Actors	Mach Task Migration	MASIF Standard	MOA	Conversant
Object owner	autonomous (meta model)	task kernel port owner(s)	creator	creator of the first generation	creator
Authentication	name based policies	kernel trusted capabilities	one-hop, authenticator part of the name object	name based (temporary)	private/public key encryption
Authorization	name based policies	same trust domain capab. owner has all rights	credentials transferred during migration	name based (temporary)	Java references (capabilities)
Delegation	spawn new actor	passing capabilities	non-composite (for the time being)	can spawn, & “fork”	none
Transport	asynchronous message passing	in-kernel protocols NORMA IPC/DIPC	CORBA IIOP	sockets	ANEP
Naming	strings/patterns	capability based	string, authenticator, agent sys type	string	SNMP MIB
Management	meta-level services	transparent extent. of kernel i/f: suspend, resume, kill	kill, suspend, resume	(group-based) kill, suspend, resume, debug	System administrator
Auditing	multilevel services	interposition of communication ports	N/A	logging	logging

Table 5. Comparison of Security Aspects of Mobile Object Systems.