

Melding Abstractions with Mobile Agents

Antonio Corradi, Marco Cremonini, Cesare Stefanelli

Dipartimento di Elettronica, Informatica e Sistemistica

Università di Bologna

Viale Risorgimento 2- 40136 Bologna - ITALY

Ph.: +39-51-6443001 - Fax: +39-51-6443073

E-mail: {acorradi, mcremonini, cstefanelli}@deis.unibo.it

Abstract

Mobile Agents (MA) seems to be the most suitable technology for distributed systems to integrate the Internet in a synergic way. One of the problems that should be faced when considering MA models for distributed applications is the lack of a thorough model capable of describing the Internet world composed of interconnected networks, each of them with its peculiar policies (for administrative, management and security purposes). We propose a Mobile Agent system based on a model designed to consider and favour aggregations of abstract and protected (network) domains: the use of this model makes easy the development of Internet applications. The paper describes the MAMA system (Melding Abstractions and Mobile Agents) and its implementation in the Java language. An application for distributed monitoring provides the results achieved within the MAMA system.

Topic areas:

Engineering, deploying, and evaluating multi-agents

Multi-agent Programming Framework

Agent model and Architecture

Practical Applications

Information Agent on the Internet

1 Introduction

Distributed programming is obtaining increasing importance due to the widespread diffusion of internetworking. The large dimension of Internet and the huge amount of information already available has focused the attention on code mobility as an alternate solution to a simple message passing solution [FPV97].

Instead of traditional client/server programming, new emerging models are the Remote Evaluation (REV) [StG90], the Code On Demand (COD) [FPV97][ArG96] and the Mobile Agents (MA) [ViT97]. The first two models can be considered as complementary. In the REV model, any client can send the code to a remote server that can use this code both in the execution of the current operation and to add up new behaviour to its features: this approach enlarges the normal client/server with the possibility of extending the server behaviour at run-time. The COD model can make possible for the client to enlarge its capacity of execution by dynamically downloading code from the server. The typical COD application configures the server as a code repository and makes the client load the code by need.

The distinguished point of the MA model is to allow the mobility of entities, even in execution. In general, the agent is the execution entity - composed of code, data, execution state - that has started on an initial node. Agents are capable of moving to a node different from the current one and to resume execution there.

These new programming models could apply to many areas, such as electronic commerce, network management, information retrieval, CSCW, etc. These models can describe distributed applications, but lack the capacity of modelling some common Internet situations, composed of a very large number of LANs interconnected by bridges, routers, and firewalls. In addition, LANs and their interconnection have peculiar policies for network management, resource administration, and security handling. We argue that this scenario should be considered in the definition of a general programming model. For instance, security should be taken into account in the first phases of the application design and, consequently, cannot be lacking in the basic layers of the model. The same quality level cannot be achieved by adding a posteriori a security tool to a non secure-born application.

This paper presents a system called **MAMA** (**M**elding **A**bstractions with **M**obile **A**gents) for the development of Mobile Agents applications. MAMA

follows a model that considers the Internet heterogeneity and provides several abstractions to suit the common localities in the Internet. In particular, we introduce the place as the abstraction of an execution node, while the domain is the locality considered an abstraction for the LAN belonging to a single organisation. Different domains can be connected by gateways that represent the interconnection points for different LANs.

MAMA led the design of an MA system architecture, where any abstraction of locality finds its concrete counterpart. The system architecture has chosen Java as the implementation language, to achieve, on the one hand, portability, interoperability, and rapid prototyping, and, on the other hand, easy integration with the Web scenario. Our programming language choice seems not to address the efficiency issue; however, the growing interests Java receives ensure that efficiency is also dealt with by all implementors and it is going to be more and more improved.

MAMA assisted in the rapid development of several applications. The paper reports a distributed on-line monitoring tool that is employed in the MAMA system to ascertain the global application and system state: applications can exploit monitoring information to enlarge their knowledge of the dynamic state of the system, for instance, for load balancing.

2 The MAMA Model

The MAMA model addresses the requirements of a typical Internet and Web integrated distributed application: portability, interoperability, rapid prototyping, and capacity of conforming to the Internet scenario. MAMA follows a few guidelines in answering the requirements:

- the execution model is based on agent strong mobility;
- several locality abstractions in a hierarchy model Internet LANs and their interconnections;
- security is considered as an integral part of the design and is integrated in any system level.

MAMA represents everything as either an agent or a resource. The agents are the execution units capable of moving between different locality of execution in order to perform their specific tasks. The agents in our model require full mobility: they can move to a different node, wholly reestablishing there, by migrating their code and by copying their execution state [FPV97]. The resources represent the logical and the physical entities available in any node

where the agent executes: examples of node physical resources are printers and local devices, of logical ones are blackboards and tuple spaces.

Agents may be granted/denied access to the local resources depending on the security policy adopted. A typical resource available in a node is an interface toward a local database: it is available through the permitted operations to all currently residing agents. Let us note that a resource can become private to a local agent. In this case, it can be accessed only by the agent itself that becomes its manager: anyone in need of the resource operations should direct its request to the manager agent.

A very important feature of the MA model is its structure composed of different levels of abstraction representing locality domains that describes the real structure of an internetworking environment (see Figure 1). The first abstraction is the place, where agents execute and are enabled to tightly interact by directly cooperating with each other. At a higher level of abstraction, there is the domain, a logical entity that groups a set of places that shares common policies and privileges: inside one domain, places have the visibility of one another and can exploit locality to provide common management policies and to adopt uniform security policies.

The idea of structuring the system in locality of different levels of abstraction, with domains containing set of places, is of paramount importance in granting security. In fact, different locality may have peculiar security policy: each place may have a proper strategy, to satisfy, for instance, the security needs of the place owner, while the domain models a common strategy of a group of places, to enforce the security policy of one department of an organisation. A set of domains matches the internal structure of an organisation. The two levels of abstractions create a double enforced protection: any action is checked first against the domain security strategy and, if it is authorised, it is passed for control to the final place of execution. We believe this layering of security policies be fundamental for the modelling of real agent applications.

The MA model takes into account security both for the agents and for the current locality. Agents should be granted integrity of their internal information, while both domains and places must be protected against malicious code operations.

With regard to agent coordination, we have already stated that agents inside a place can interact by sharing common resources. Whenever an agent needs to share a resource with another agent residing in a remote place, it is forced to

migrate to the remote place. Mobile agents can directly migrate from place to place inside one domain. Whenever an agent needs to move outside the domain, it needs to rely on a specific entity, the gateway, in charge of the inter-domain routing functionalities. The gateway is part of the default place of a domain. One agent that migrates to a new domain is constrained to move to its default place.

Outside the scope of the place, agents can interact only by means of message exchange. The MAMA model assumes that messages can be eventually delivered to agents even when they migrate.

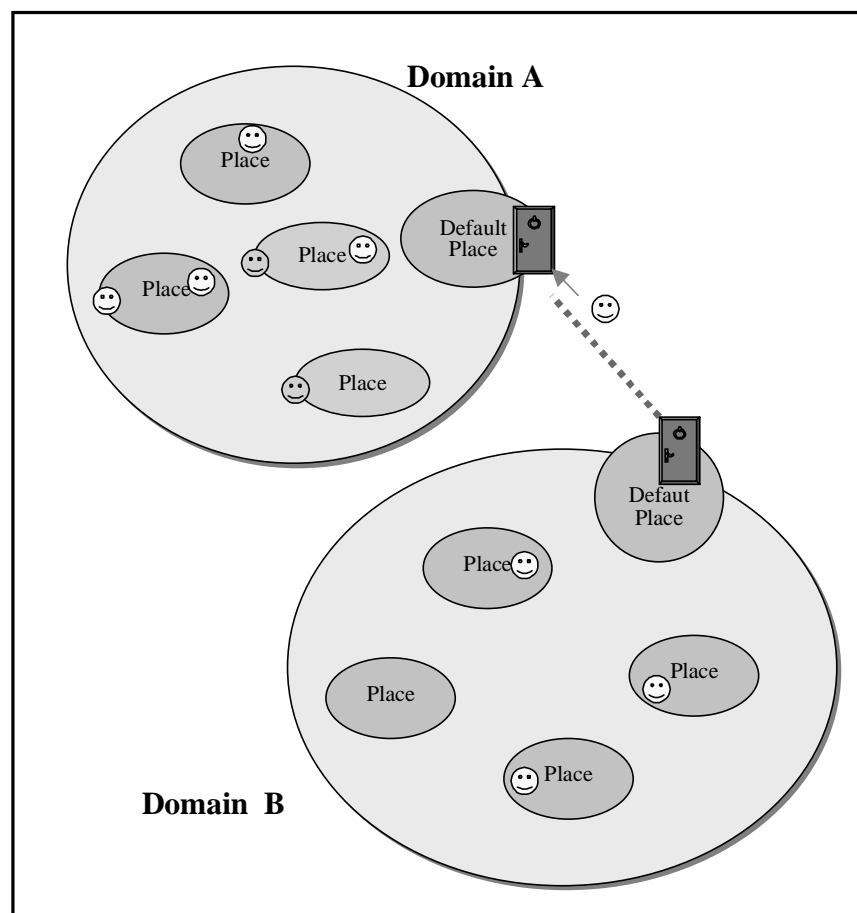


Figure 1. The MA model

3 The MAMA architecture

The Mobile Agent model described in the previous section is naturally mapped in the architecture of the MAMA system. The agent execution environment is the realisation of the place concept and the allocation of each agent execution

environment is constrained into a single node. There cannot be agent execution environments spanning over different nodes (whereas there can be several agent execution environments in the same node) because its goal is to provide an interface to the physical machine. At a higher level of locality abstraction, the domain concept is mapped in the network domain that can contain several execution environments sharing common management and security policies. Each network domain embeds also a default place (agent execution environment) containing the gateway for inter-domain communication. The default place is in charge of handling the agents entering or exiting the domain and of enforcing the domain policies.

3.1 The agent execution environment

Agent execution environments contain the resources that can be accessed by agents. In our architecture, local resources are represented at a higher level of abstraction as objects directly offering their interface. A possible extension of this level of abstraction could be the realisation of a new type of agents, called static agents, providing the functionality of Resource Manager in order to control and mediate the interaction between mobile agents and local resources. The interaction style between mobile agents and local resources evolve in the sense of introducing a higher degree of abstraction in the system.

The agent execution environment offers agents services by means of the modules shown in Figure 2:

- the **Agent Manager** offers the basic functionalities for agent mobility and for communications outside the place. Mechanisms for agent mobility are offered by the MAMA run-time support. This module provides message-passing style of communication for agents residing into different places and even different domains. It also manages the local naming of agents and the possibility of defining several aliases for the same agent.
- the **Local Resource Manager** is the interface for place services and for the node resources that agents may access. This module makes possible the agent access to local resources via their object interface. This makes possible a tight form of interaction between agents in the same place by means of shared objects. For instance, agents can share a blackboard object and a tuple space. This module controls the authorisation of agent in the access to local objects and it ensures the respect of the place security policy.

- the **Distributed Information Service** is responsible for looking up information about agents and places in remote nodes. The visibility is limited to one domain, mapped on different physical nodes, but logically viewed as a unique context. In particular, it provides a Domain Name Service functionality to support a service of Remote Search and Remote Monitoring within a domain. This module can be viewed as an application service, composed of dedicated agents, supported by the previous modules (the Agent Manager and the Local Resource Manager).

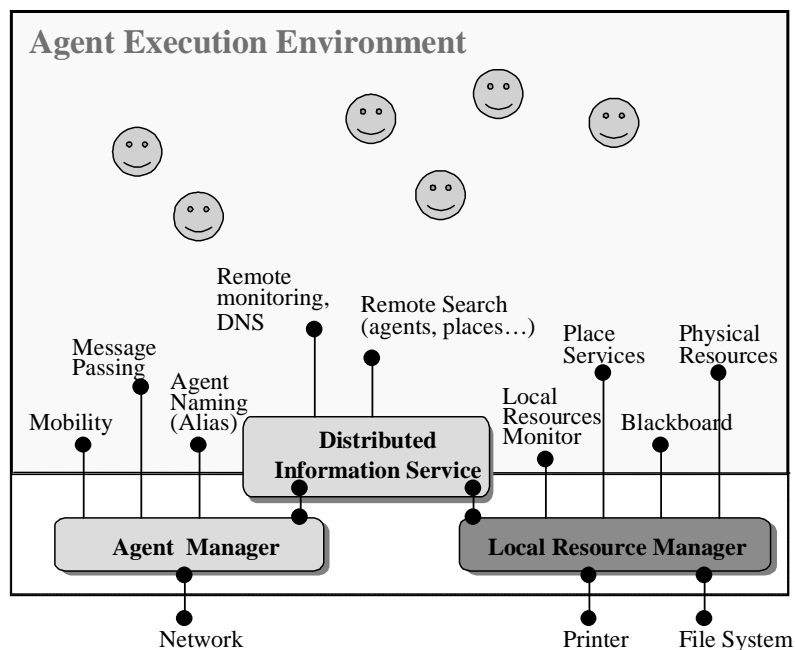


Figure 2. The agent execution environment

The agent execution environment is the realisation of the concept of place: this locality is where agents may interact with each other and with local resources. In particular, it is important to examine the security issues raised by the interaction of an agent with its current agent execution environment. Both parties of the interaction need to be protected against reciprocal malicious behaviour. On the one hand, it is necessary to protect the execution environment from unauthorised agents that could cause damage or steal information. On the other hand, it is necessary to protect the agent against possible unauthorised modifications by hostile execution environments.

The security of the place in our architecture is achieved by enforcing a level of trust between agents and places. Each agent is authenticated and places grant different levels of authorisation depending on the confidence associated

with the agent creator. Any place decides its minimal set of operations permitted in the default case, for unknown agents.

Our mobile agents are protected in terms of data privacy and code integrity. Let us note that it is not possible to prevent the complete destruction of an agent, but this event is similar to the event of a fault in distributed system and the application level can handle it by adopting fault tolerant replication-based protocols. ARA, for instance, adopts a checkpointing technique to recover a previous agent state, in case of such a loss [Pei97].

3.2 The network domain

The domain concept presented in Section 2 identifies a logical locality mapped at the implementation level in the network domain. The network domain groups a set of places and enforces the domain security policy. For instance, the network domain can represent a LAN network or subnetwork of a department, and will be regulated by the organisation policy stating and defining the user authorisation and capabilities, the resources available to users, etc. Inside the network domain, each place can have (and usually has) a specific security policy that rules the actions inside a place. Domain and place policies usually differ, representing the different needs of users and organisations. The agents can access to the resources of a place depending on the restriction imposed by both the domain and place security policies.

Agents execute typically inside one domain, where they can communicate asynchronously via message passing: each agent owns its mailbox where it can receive messages when moving from place to place. The agent support should guarantee message delivery even in the presence of agent mobility. It is not possible for two agents to open stream connection, instead they can decide to move to the same place of execution, where they can share some common objects.

Message passing and agent migration are performed directly between places belonging to the same domain, because places inside one domain have the visibility of all the others. Messages and mobile agents crossing the domain boundaries involve the default place (containing the gateway), that is responsible for:

- routing all messages to/from the domain, acting in a way similar to traditional IP gateways; it is worth to stress that the gateway handles only messages exchanged between agents belonging to different domains, that produces a usually limited traffic of messages;

- handling the incoming/outcoming agents; the gateway receives agents arriving from other domains and it is in charge of performing all the security checks required by the domain policy on the newly arrived agents. Agents exiting the domain are sent to the gateway for further routing to the new destination domain;
- dispatching the agents arriving in the domain to the correct destination place; in practice, it contains a name server for the other places of the domain. This functionality is required only locally to the gateway and only for agents entering the domain and thus it does not represent a bottleneck;
- physically separating the domain from all others; in this case, it acts as a proxy, capable of transferring agents and messages between the domain and the rest of the world.

Let us consider that, from the security point of view, the default domain manager offers the same functionalities and services typically offered by firewall systems. In particular, we propose a solution with the proxy agent separated from the gateway: we implement a screened host firewall architecture, where the default place is the application bastion host and the proxy agent is the router connecting the domain to Internet [ChB95].

Main advantage with respect to traditional firewall implementation is in its flexibility because of the agent-based implementation. In fact, modifications of the gateway architecture in order to reflect the specific security needs of different domains is easier to implement than in traditional solutions. The implementation of the proxy in terms of agents makes also possible the dynamic extension of the supported protocols to support different kind of Internet services.

3.3 The MAMA implementation

The MAMA architecture has been implemented by using the Java language [ArG96]. Java provides an intrinsic portability and interoperability in heterogeneous environments. The object-oriented nature of the Java language is suitable for the design of MAMA: the encapsulation principle suits the abstraction needs of both resources and agents; the classification principle makes possible to inherit behaviour from already specified components instead of starting the design from scratch.

The main problem encountered with Java is that it directly supports only a weak concept of mobility; in other words, it is not possible to save the whole state of a thread before its migration to a different node. It is possible to

overcome this restriction either by modifying the Java Virtual Machine or by providing a new operation at the application level. We have chosen the latter solution for preserving the MAMA portability. The new operation for mobility is a `go` operation that allows an agent to move itself during its execution. The `go` operation requires as a parameter the method that has to be activated after the migration.

The support for migration has taken advantage of Java object serialization. When an agent migrates, it moves also all its private Java objects (they are copied to the new location and then destroyed in the old one). All other objects one agent has references for are left in the current place where they can be later found if the agent comes back; in particular Java non serializable objects maintain their allocation.

The MAMA system, composed of places, domain and gateways is implemented in Java (JDK 1.1.4) [Jav97] with a very limited number of class (175). All the current MAMA security features are based on the JDK version and are going to be changed to adhere to the new security model of the JDK1.2.

4 The Distributed Monitoring Application

Mobile Agents can improve performance in many distributed applications, because they can take advantage of the network bandwidth availability, and they carry on execution in case of disconnected situations [Wal97]. Mobile Agents can be useful also for achieving fast prototyping of applications, if they have been developed in a rapid prototyping environment, available also to applications.

Mobile Agents are commonly indicated as a good solution in the field of electronic commerce, for information-retrieval applications or in network management and in the support for cooperative works.

This section describes our agent implementation of an on-line distributed monitoring system [GoY95] to collect information about the agent application at run-time. The monitoring system is capable of inspecting the configuration of the whole system to provide information for system upgrades, for load balancing policies, and, in general, any dynamic strategy.

The monitoring system architecture is completely distributed. There is no centralisation point and there is no single point of execution: the monitoring

system is not only distributed but also even mobile, being composed of only mobile agents.

Monitoring agents are assigned to specific domains and have the duty of gathering the information collected by moving from node to node inside one domain. We restrict the scope of monitoring agents to one domain in order to provide a light implementation of the agent, avoiding the problems of mobility between domains with different security needs. In addition, it is possible to assign a variable number of monitoring agents to each domain, depending on fault tolerance, time latency of the information to be gathered, etc. The higher the number of agents per domain, the higher the consistency of monitoring information with the current system state.

The monitoring agent looks for information about the configuration of each place (in terms of services available) and the load distribution in the system, in terms of both mobile agents and object-resources present in each place.

We have compared the results obtained by the agent solution with a more traditional one where a single monitoring entity (the master) resides in one node and gathers the monitoring information of the whole domain by message passing with the slaves, one in each controlled node. The agent solution becomes convenient as soon as the operations performed by each agent at any monitored node justify its migration.

Figure 3 shows the results obtained in monitoring a domain composed of a network of 12 SUN workstations (Sparc 4, Sparc 5 and Ultra-1) Ethernet-connected. The figure shows the total time (in msec) to obtain the complete situation over the whole domain. This experiment considers average monitoring information: the master/slave solution requires a number of messages equal to the number of observation (per each slave); the agent solution dispatches a variable number of monitoring agents that compute the average in place. The time measurement shown in Figure 3 is an average measurement over a high number of executions.

The results obtained show that the agent solution can perform the same of traditional one and sometimes can achieve better performance with a limited number of monitoring agents in the given domain.

Figure 3 shows the performance of the monitoring coordination scheme adopted inside a domain (the intra-domain monitoring) and can be extended by taking into account the coordination between different domains. The inter-domain coordination can be achieved by exchanging information between

agents of different domains. To this purpose, we have allocated a monitoring agent on each gateway of the domains, and we measured the time needed for the inter-domain coordination by taking into account also the case in which two gateways are separated by a firewall system (represented by the Java proxy described in section 3.3). The additional overhead imposed by the proxy is around 200 msec.

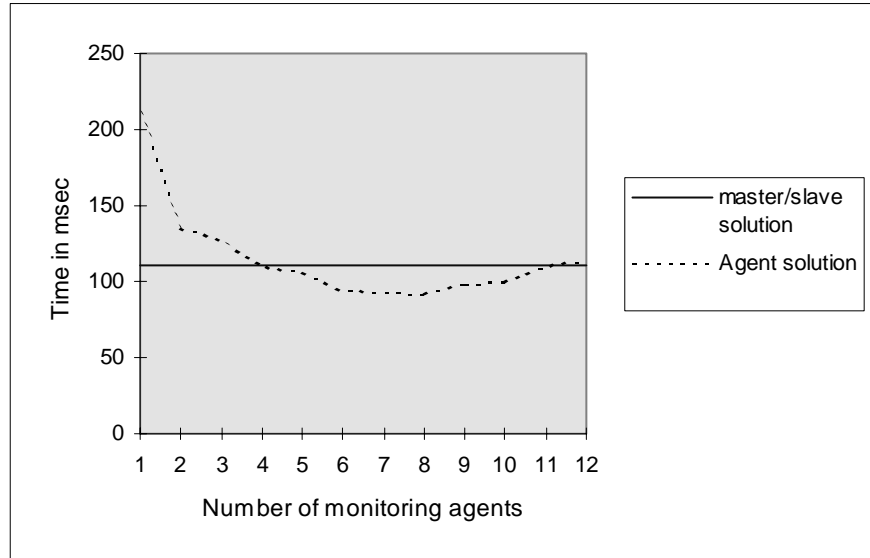


Figure 3. The agent-based monitoring solution compared with a traditional master-slave monitoring approach.

5 Related Work

We have compared our architecture with the different approaches of other MA projects: we consider significant *Agent Tcl* [Gra96], *Aglets* [LaO97] and *Ara* [Pei97], and neglect efforts such as *Sumatra* [ARS96] because they have chosen to change the Java Virtual machine.

The distinguishing feature of MAMA, at the design level, is the hierarchy of places and domains. *Agent Tcl* and *Aglets* do not provide any locality. The MAMA idea of abstraction is present in *Ara*, where only a place is present and no domains are provided. The mechanism of allowances of *ARA* are useful to develop complex security and usage policy in a mobile agent application: at the security level, MAMA can support not only allowances, but also many other different mechanisms, i.e. integrity checks and data privacy.

Aglets has a weak mobility model, because any moving agent must start its execution from the beginning in the new node.

When considering communication, our system supports message-passing between agents that reside in different places (and domains) and a direct access to shared objects in the same places. These two modes are sufficient to match any communication need. We think that the event messages of Agent Tcl for asynchronous notification of particular kind of messages is too peculiar to be provided as primitive.

Aglets provides a specific HTTP-based protocol for agent movement called ATP (Agent Transfer Protocol). This interesting approach becomes significant as soon as it would be accepted as a standard protocol for mobile agent systems: in any case, we can implement it in MAMA.

6 Conclusions

The paper presents an MA model with novel features, introduced with the goal of making easier the design of global and non traditional applications for the Internet scenario. MAMA provides several locality abstractions in a hierarchy in which agents can move depending on their needs and their accesses to different services. MAMA considers fundamental the enforcing of security, for both the agent and the place for execution, to preserve the integrity of all entities.

The Java implementation, apart from the a priori granted interoperability and portability, has been carried out in accord with the possibility rapid prototyping. This approach makes possible to vary the behaviour of several system components and to experiment different policies.

The first experiences, coming also from the implemented managers and applications, exhibit acceptable performance, if taking into account Java efficiency limitations. In any case, we believe that Java will grant higher level of efficiency in a little time.

Bibliography

- [ArG96] K. Arnold, J. Gosling, *The Java Programming Language*, Addison-Wesley, 1996.
- [ARS96] A.Acharya, M.Ranganathan, J.Saltz, *Sumatra: A Language for Resource-Aware Mobile Programs*, in Mobile Objects, Vitek J, Tschudin C., II International Workshop, MOS'96, Springer-Verlag Lecture Notes in Computer Science, 1996.
- [ChB95] W.R. Cheswick, S.M. Bellovin, *Firewalls and Internet Security*, Addison-Wesley, U.S.A., 1995.

- [FPV97] A. Fuggetta, G.P. Picco, G. Vigna, *Understanding Code Mobility*, Politecnico di Milano, Politecnico di Torino, Italy, 1997.
- [GoY95] G. Goldszmidt, Y. Yemini, Distributed Management by Delegation, 15th International Conference on Distributed Computing Systems, IEEE Computer Society, Vancouver, British Columbia, June 1995.
- [Gra96] R. Gray, G. Cybenko, D. Kotz, D. Rus, *Agent Tcl*, Dept. Of Computer Science, Dartmouth College, Hanover, NH, USA, 1996.
- [Jav97] Java Development Kit, Version 1.1.4, Sun Microsystems, 1997. <http://java.sun.com/products/index.html>
- [LaO97] D.Lange , M.Oshima, *Programming Mobile Agents in Java - With the Java Aglet API*, IBM Research, 1997.
- [Pei97] H.Peine, *An Introduction to Mobile Agent Programming and the Ara System*, ZRI-Report 1/97, Dept. Of Computer Science, University of Kaiserslautern, Germany, 1997.
- [StG90] J.W.Stamos, D.K.Gifford, *Remote Evaluation*, ACM Transaction on Programming Languages and Systems, Vol. 12 No. 4, October 1990.
- [ViT97] J. Vitek, C. Tschudin, *Mobile Object Systems: Towards the Programmable Internet*, Vol. 1222 of Lecture Notes on Computer Science, Springer-Verlag, April 1997.
- [Wal97] J.Waldo, G.Wyant, A.Wollrath, S.Kendall, *A Note on Distributed Computing*, in Mobile Objects, Vitek J, Tschudin C., II International Workshop, MOS'96, Springer-Verlag Lecture Notes in Computer Science, 1997.