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
{acorradi, mcremonini, cstefanelli}@deis.unibo.it

Abstract. The Mobile Agent (MA) model seems to provide one of 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 mobile agents for distributed applications is the lack of a thorough model capable of describing the Internet world composed of interconnected networks, each of them with their 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 with Mobile Agents) and its implementation in the Java language. An application for distributed monitoring provides an example of the results achieved within the MAMA system.

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 simple message passing solutions [3].

Instead of traditional client/server programming, emerging models are the Remote Evaluation (REV) [18], the Code On Demand (COD) [3] and the Mobile Agents (MA) [15]. 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 behaviours to its features 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

Work carried out under the financial support of the "Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST)" in the framework of the Project "Design Methodologies and Tools of High Performance Systems for Distributed Applications".

COD application configures the server as a code repository and makes the client load the code by need. This is the model followed by Java Applets.

The distinguished point of the MA model is to allow the mobility of entities, even in execution. The agent is the execution entity - composed of code, data, execution state - that has started on an initial node. Agents are capable of moving autonomously to a node different from the current one and to resume execution there. The MA model can be considered as an extension of the REV one with enlarged functionality.

The MA model could be applied to many areas, such as electronic commerce, network management, information retrieval, CSCW, etc. It can describe distributed applications, but lacks a comprehensive framework which permits to model some common Internet situations, composed of a very large number of interconnected LANs with their peculiar policies for network management, resource administration, and security. We argue that this scenario should be considered in the definition of a general programming model for MA. In addition, MA applications, dealing with insecure interconnected networks and mobile executable code, are forced to take into account the security issue since the first phases of the design.

This paper presents a system called **MAMA** (Melding Abstractions with Mobile Agents) for the development of Mobile Agent 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, and the domain as the locality considered an abstraction for one LAN belonging to a single organisation. Different domains can be connected by gateways that represent the interconnection points for different LANs. We consider this framework suitable to assist the definition of all the policies necessary to develop MA applications over the Internet

MAMA led the design of the MA system architecture, where any abstraction of locality finds its concrete counterpart. Java is the language chosen to implement the system architecture. Java addresses the requirements of typical Internet applications: portability, interoperability, rapid prototyping and 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

2.1 Overview

MAMA permits to face the requirements of a typical Internet distributed application by following a few guidelines:

- the execution model is based on agent 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 at any system level.

MAMA represents everything as either an agent or a resource. The agents can move to a different node, wholly re-establishing there, by migrating their code and copying their execution state [3]. The resources represent the logical and the physical entities available in any node where agents execute: examples of physical resources are printers and local devices, of logical ones are blackboards and tuple spaces.

Agents interact with resources by means of interfaces which assume a fundamental role for security and extensibility of the MA systems. In addition, the MAMA framework has a layered structure composed of places and domains mapping the local environment and the interconnection among nodes of a LAN. These levels of abstraction permit the development of many different security policies in the MAMA model.

2.2 Interface

The concept of interface is a uniform abstraction for handling both physical and logical resources. Agents can not directly access to resources; they make use of resource interfaces.

The interface is very useful from a security viewpoint, since it can control the actions of agents on resources: interfaces provide a safe access to resources, and filter the permitted operations. Agents may be granted/denied access to the local resources depending on the security policy adopted. As an example, consider an interface toward a local database: it is available through the permitted operations to all currently residing agents. Let us note that the same resource can provide different interfaces to different agents, depending on the security policy. As an extreme policy, the resource can become private to one specific agent. In this case, the agent itself becomes the resource manager, and anyone in need of one operation from the resource should issues its request to the manager agent.

2.3 Place and domain

A very important feature of the MAMA model is its structure composed of different levels of locality abstraction that describe the real structure of an internetworked 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, 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.

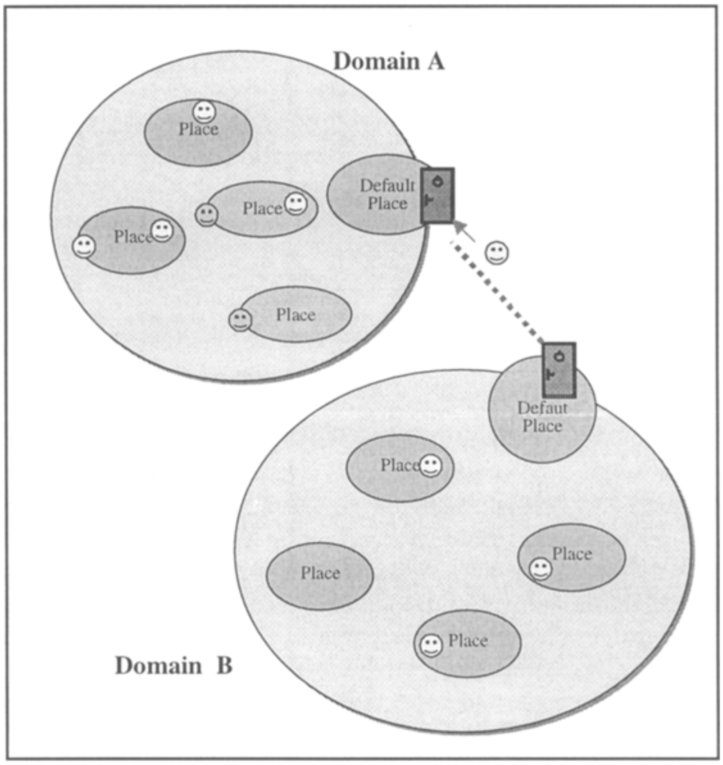

Figure 1. The MA model

The MA model takes into account security both for the agents and for the current locality. Agents should be granted integrity and privacy 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 functionality. 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.

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 an agent execution environment 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 is embodied in a default place containing the gateway for inter-domain communication. The default place is also in charge of dealing with and handling the agents entering or exiting the domain and of enforcing its policies.

3.1 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 node resources. Each agent execution environment provides an abstraction of the physical machine, and it is then constrained into a single node. Agent execution environments can not span over different nodes (whereas there can be several agent execution environments in the same node). An agent execution environment contains the resources that can be accessed by agents. Following the model, each resource is represented by the interface that controls the actions that agents can perform on resources.

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 functionality 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 agents accessing to local objects and 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 which is logically viewed as a unique context composed of places mapped on different physical nodes. In particular, it provides a Domain Name Service and a Directory Service functionality. This module is an application service currently implemented as a set of dedicated agents. However, its functionality could be also implemented in terms of agents interfacing with traditional Directory Services (X.500, NIS, etc.) and Internet DNSs.

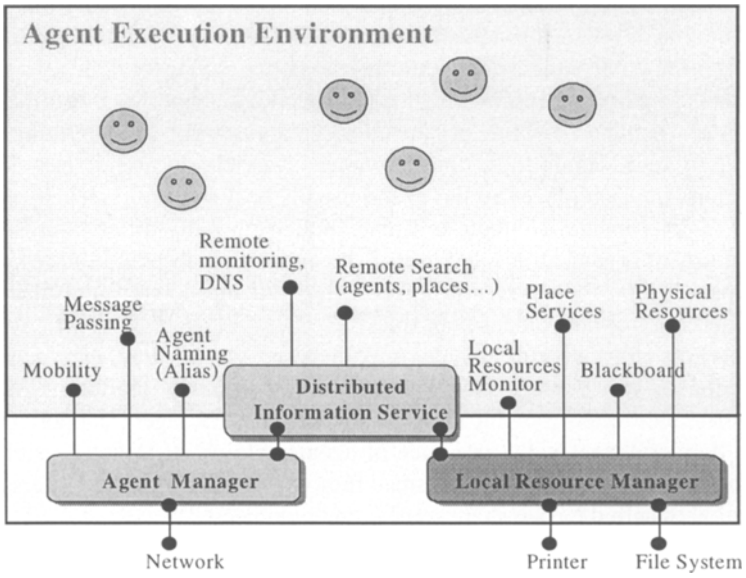

Figure 2. The agent execution environment

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 agents that could cause damage or steal information. On the other hand, it is necessary to protect the agent against possible unauthorised actions by hostile execution environments.

The design of our architecture addresses the security requirements for both the place of execution and the agents. Privacy, integrity and authentication are required for protecting the place of execution. Authentication is necessary in order to authorise agents to interact with local resources of a place. Privacy and integrity aim to protect both the information carried by agents and the ones stored on the host.

In addition, we allow for the definition of layered security policies. Agents entering the domain are authenticated and authorised to interact with places. At the place level, agents are granted a set of permissions to invoke operations on local resources. Domains and places are the structures that permit an efficient definition of policies derived from Access Control models. A general policy for the system can be defined first at the domain level, by controlling the interaction of agents with places on the basis of user identity or role. Then, policies for the specific resources can be defined at the place level by different managers, by refining the domain policies. Coherence between domain and place policies must be granted by the MA system which is in charge of detect all conflicts between the policies at the two levels [16] [11].

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 agent execution environments 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. Domain and place policies usually differ, representing the distinct needs of organisations and users. The agents can access to the resources of a place depending on the restriction imposed by both the domain and place security policies derived from user identity or role.

Agents typically execute 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 guarantees 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.

4 The MAMA Implementation

The MAMA architecture has been implemented by using the Java language [2]. Java provides an intrinsic portability and inter-operability 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.

We use the Java Virtual Machine without modifications and, like other MA systems, we introduce a new operation (go) allowing an agent to move itself during the 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 serialisation. 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 1.1.5) [8] with a very limited number of class (175). All the current MAMA features are based on the current JDK version and some of them are going to be changed to adhere to the new model of the JDK1.2.

5 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 [19]. 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 [17] 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.

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.

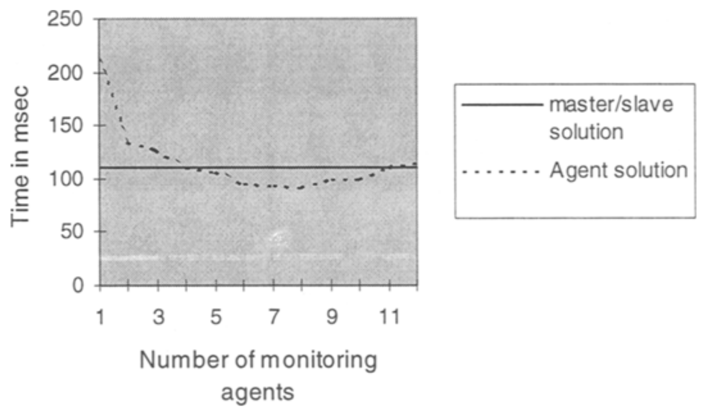

Figure 3. The agent-based monitoring solution compared with a traditional master-slave monitoring approach.

The obtained results show that the agent solution can perform the same of a traditional one and can achieve better performance with a limited number of monitoring agents in the given domain. The presented results refer to the first prototype of the agent support, whose successive versions tend to achieve better performance. Apart from the performance, a mobile agent monitoring application offers a more flexible solution than traditional distributed ones. In particular, when dealing with mobile computing, network bandwidth limitations and unreliable communication scenarios.

6 Related Work

While many proposals of MA systems have been presented recently, we compare only a few of them, chosen because of similarities in goals and application areas. The distinguished features of MAMA, if compared with the other systems, is to present a comprehensive framework, the hierarchy of places and domains, suitable to apply patterns of security policies integrated at the design level.

In the class of Java-based mobile agents projects, we compare our system with *Aglets* [10], *Odyssey* [12] and neglect efforts such as *Sumatra* [1] because they have chosen to change the Java Virtual Machine, loosing its portability.

Aglets uses an event-based programming model, where examples of events are the agent migration, its dispatch and its arrival. It provides only a weak mobility model, because any moving agent must start its execution from the beginning in the new node. Aglets currently has only a primitive form of agent authentication, but a new security architecture for this system has been proposed in [9]. With respect to Aglets, we support a stronger form of mobility.

Odyssey has a simplified structure if compared with our system but also implements a `go` operation. Odyssey `go` operation differs from MAMA `go` because it permits only to specify the destination and the routing of an agent by means of the `Ticket` constructor. For this reason Odyssey agents are restarted at each destination. Otherwise, if a particular method must be invoked at a specific destination, a subclass of the main Agent class must be used, the Worker class that supports only one task per destination.

Ara [14] adopts a different approach compared with us and the other systems, providing a core Java-based module that supports multiple agent languages like Tcl and C++. Agents move between locality called "places". We extend the concept of locality present in Ara.

In the class of mobile agents systems based on script languages, *Agent Tcl* [6] provides a support for agents written in Tcl (in the near future also in Java and Scheme) which communicate mainly by using an RPC mechanism. Our system, instead, supports message-passing between agents and a direct access to shared objects in the same place.

Other works address the issue of standardisation. A group of MA systems developers (Crystaliz, General Magic and IBM among the others) has submitted to OMG a proposal to achieve interoperability for different architectures and implementations [5]. This proposal standardises some aspects of MA technology and its integration with CORBA [13]. A similar effort is the FIPA one [4] where the focus is on the standardisation of communication and management of agents. Another standardisation effort is the Open Distributed Processing (ODP) [7], that is a joint activity of the ISO and ITU and provides a framework to develop architectures with an integrated support for distribution, internetworking, and portability. In addition, it focuses on the use of formal description techniques for specification of the architecture. Our proposal follows the guidelines presented in the different standardisation efforts: we adopt a layered structure with different abstractions, and we agree also on the importance of the integration of MA systems with other technologies. Our paper presents also the realisation of an MA architecture in a Java environment.

7 Conclusions

The paper presents an MA model with locality abstractions (place, domains and gateway), introduced with the goal of making easier the design of global and non traditional applications for the Internet scenario. MAMA provides a layered framework in which agents can move depending on their needs and their accesses to different services. Our system considers fundamental the enforcing of security, for both the agent and the place of execution, to preserve all entities, introducing a structure that permits to apply patterns of policies at different levels and with different scopes.

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

The first experiences, coming from the implemented 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.

References

1. A. Acharya, M. Ranganathan, J. Saltz: Sumatra: A Language for Resource-Aware Mobile Programs. In Mobile Objects, J.Vitek, C.Tschudin (Eds.), Springer-Verlag, Vol. 1222 Lecture Notes in Computer Science, 1997.
2. K. Arnold, J. Gosling: The Java Programming Language. Addison-Wesley, 1996.
3. A. Carzaniga, G.P. Picco, and G. Vigna: Designing Distributed Applications with Mobile Code Paradigms. 19th International Conference on Software Engineering (ICSE'97), 1997.
4. L. Chiariglione: FIPA 97 specification, Foundation for Intelligent Physical Agents. October 1997.
5. Crystaliz Inc., General Magic Inc., GMD Fokus, IBM Corp.: Mobile Agent Facility Specification. Joint Submission. Supported by: The Open Group, OMG TC Document, June 1997.

6. R. Gray, G. Cybenko, D. Kotz, D. Rus: Agent Tcl. In W.R. Cockayne and M. Zyda: Mobile Agents: Explanations and Examples. Manning/Prentice Hall, 1997.
7. ITU Recommendation X.901-904 - ISO/IEC 10746 1-4. Open Distributed Processing - Reference Model, July 1995.
8. Java Development Kit, Version 1.1.5. Sun Microsystems, 1997. http://java.sun.com/products/index.html
9. G. Karjoth, D. Lange and M. Oshima: A Security Model for Aglets. IEEE Internet Computing, Vol. 1, N.4, July/August 1997.
10. D. Lange , M. Oshima: Programming Mobile Agents in Java - With the Java Aglet API. IBM Research, 1997.
11. E. Lupu, M. Sloman: A Policy Based Role Object Model. Proceedings of EDOC'97, IEEE Computer Society, October, 1997.
12. Odyssey, version beta 2, General Magic, 1998, http://www.genmagic.com/agents/odyssey.html
13. Object Management Group: The Common Object Request Broker: Architecture and Specification. Rev 2.0 (OMG Document 96-03-04), 1995.
14. H.Peine: Ara - Agents for Remote Action. In W. R. Cockayne and M. Zyda: Mobile Agents: Explanations and Examples, Manning/Prentice Hall, 1997.
15. K. Rothermel, R. Popescu-Zeletin (Eds.). Proceedings of the First International Workshop on Mobile Agents, Berlin (D), Lecture Notes in Computer Science, Vol. 1219. Springer-Verlag (D), April 1997.
16. R.Sandhu, P.Samarati: Authentication, Access Control, and Intrusion Detection. The Computer Science and Engineering Handbook, 1996.
17. B. Schroeder: On-Line Monitoring: A Tutorial. IEEE Computer, Vol. 28, N. 6, June 1995.
18. J.W.Stamos, D.K.Gifford: Remote Evaluation. ACM Transaction on Programming Languages and Systems, Vol. 12 No. 4, October 1990.
19. J. Waldo, G. Wyant, A. Wollrath, S. Kendall: A Note on Distributed Computing. In Mobile Objects, J.Vitek, C.Tschudin (Eds.), Springer-Verlag, Vol. 1222 Lecture Notes in Computer Science, 1997.