MACE:
A MOBILE AGENT CARRIER ENVIRONMENT

Tzone I. Wang

Laboratory of Intelligent Network Applications
Department of Engineering Science
National Cheng Kung University
Taiwan
wti535@mail.ncku.edu.tw

Abstract

This paper presents a Mobile Agent Carrier Environment -MACE. MACE is designed in allusion to a new computation notion called the *service-on-demand* that makes its design philosophy quite different. A mobile agent carrier envoriment instead of a mobile agent development system is constructed. A specially designed protocol, the Service Framework, provides an easy and flexible way of building service-oriented applications. In the framework, a service is abstracted and realized by two closely related components, an Agentlet and a Serverlet; and the mobile agent carrier plays the role in associating them together. While special in its architecture, MACE retains the flexibility of a general mobile agent system. It may as well support many other domains of applications without any difficulty.

1 Introduction

Mobile agents are recognized as software modules that move from host to host. They may interact with each other and access distributed resources in a heterogeneous network [KK1]. Mobile agents consume fewer network resources and, therefore, are particularly useful for developing distributed applications. There have been studies, implementations, and prototype applications of mobile agents [CTB1], [KGN+1], [KLO1]. Most of them use a model combing multi-agent computation and mobile code technology, and are for systems with permanent network connection.

In contrast to the client/server model in distributed computing paradigm, which ships data to remote application codes, the computation model of mobile agent paradigm migrates application codes to remote data. The motivation is to reduce network traffic by reducing interactions and data transfers between distributed components. The aim is to gain more efficient bandwidth utilization and higher system availability. This code mobility greatly benefits those distributed applications that compute with simple logic and huge remote data.

Along with the advance of network technology, a new concept of computation is also gradually taking its shape. This new notion is referred to as *service-on-demand*, and the core of it is a service base. A service base resides in a service station and is a collection of various services supplied by different service providers. Users may choose their necessary and favorite services in a base, and pay for the use of them afterwards. The same scenario has been happening in our daily life. Customers are using yellow page service, consultant service etc. and there are so many such service providers.

Mobile agents, combining multi-agents, mobile code and the client/server technologies, present a new paradigm for distributed computation. They are inherently suitable for seamlessly realizing the *service-on-demand* notion, and for easily building service-oriented applications. At the first impression, it might seem natural to implement services as mobile agents, and agents migrate to the site from which requests are made. After a second thought, it turns out that implementing a service as a mobile agent might not be meaningful in such situations when a computation happens at the same site on which the data it consumes resides. Unfortunately, most of the *service-on-demand* applications belong to this category. So, what is the point in using mobile agents to build such kind of systems?

Besides reducing network traffics, perhaps a very good answer to the question is being able to support the so-called "fire-and-forget" functionality. Users, after launching the agents, are free from engaging in any interactions with the code that actually performing computations. They just collect the results later in a convenient time and space. For example, before taking a teatime break, a user can launch agents to visit flight and ticket information (service) sites at three airlines. And, after the break, without being attended, the agents would have chosen a ticket of right time and good value for the user.

This paper describes first the design and then the implementation of a Mobile Agent Carrier Environment-MACE to support *service-on-demand* applications. The rest of this paper is organized as follows: in section 2, the design philosophy behind the MACE is advocated. This philosophy gives the reasons for designing a carrier system instead of a complete mobile agent developing system. Section 3 describes the details of the MACE architecture, including the functionality of each component. In section 4, some security issues on a mobile agent system and on the MACE are discussed. Section 5 presents the use of MACE in different applications. In section 6, comparisons between MACE and other mobile agent platforms are made. Finally, section 7 makes a conclusion and describes the possible future works.

2 The MACE's Approach - The Service Framework

Many prototype or commercialized mobile agent systems have been proposed and developed in recent years [Wh1],[Gr1], [KGN+1]. The development of JAVA further accelerated the design and implementation of mobile agent systems [LO1], [Ag1],[Ge1]. Most of these systems offer an integrated environment for agent programming, transportation, communication, execution, and more. In spite of offering a total solution, they may not be appropriate for service providers in a *service-on-demand* application. From a service provider's point of view, a mechanism for delivering a service easily will be much more appreciated. Service providers may be willing to program their services using their own business logic in a language that they are already familiar with. Or, they would like to just quickly plug in some already running services. They may not be pleased to do the programming job with a whole new mobile agent developing system, let alone the tedious debugging works in a complex application in which agents interact heavily.

On the other hand, giving an entire mobile agent developing system to a service user seems also inadequate in a *service-on-demand* application. When providing a service, the service provider cannot expect a user to code an agent to interact correctly with the service program at the service station. What the service provider most likely does is to offer the user an input program with an attractive user interface. When the user calls for the service, this input program is run to collect from the user only the necessary arguments for instantiating the service program. The business logic of the service is totally soft-coded in the input and the service programs by the service provider. In this way, it is not the entire input program but only the list of arguments that has to migrate to instantiate the service program. This observation establishes the important principles behind the design of the **Service Framework** in the **MACE** system.

MACE adopts the weak migration notion, in which the execution state of an agent is not transferred [PTV1]. In fact, it uses an even weaker migration mechanism. The **Service Framework** is established for realizing the *service-on-demand* concept, in which distributed resource accesses and management are abstracted as services. In the Service Framework, a service is fulfilled by the execution of two closely related components, an **Agentlet**, and a **Serverlet**. In general, both of them are developed by a same service provider. The Agentlet is distributed to some dedicated servers that provide directory services to all the users. The Serverlet is stored at a service station where the service will be actually carried out. Agentlets of frequently used services are downloaded and cached locally in a user's local directory. In fact, service providers can run a directory service of their own and users may be authorized to download Agentlets to their interest. Agentlets may be renewed regularly by its service provider and MACE's directory service mechanism guarantees that the newest version of an Agentlet is always invoked. Users invoke Agentlets via an **Agent Creator**. An Agentlet, after invoked, may produce one or more **service items**, each of which corresponds to an invocation of a Serverlet at a service station. Thus, an agent carries service items instead of codes, and a service item is actually the tight connection between an Agentlet and a Serverlet.

The beauty of a service item is that its content can range from as simple as a list of arguments to as complex as a segment of code written in a customer-defined scripting language. In the latter case, the associated Serverlet of such a service item becomes an interpreter. To the extreme, the Agentlet becomes a compiler and the Serverlet serves as a virtual Machine. In such a case, MACE is in reality as well in name a Mobile **Agent** Carrier Environment. Therefore, the Service Framework is simple yet powerful, confined mainly in the rules for transferring information among three parties - Agentlet, Agent Creator, and Serverlet.

The design of the Service Framework frees users from programming agents by themselves. Service provides may use their own familiar language for developing Agentlet and Serverlet as long as they are confined to a user-defined information exchange protocol. Though designed in allusion to *service-on-demand* applications, MACE retains the flexibility of a general mobile agent system. It may as well support many other domains of applications without any difficulty.

3 MACE: the Mobile Agent Carrier Environment

The whole MACE system is divided into four major parts as shown in Figure 1. The **Agent Launch Mod-**

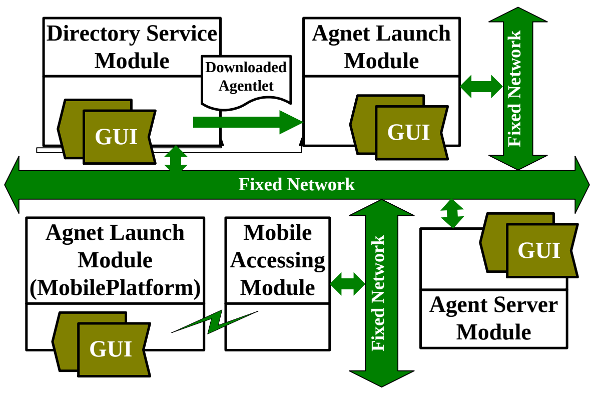

Figure 1: The MACE System

ule, as the name suggests, creates and launches agents into the network. Launched agents will roam through the network and reach some **Agent Server Module** where services are carried out. Agent Launch Module is the implementation of **Agent Creator** in the Service Framework. While creating agents, users can consult the **Directory Service Module** to find out preferable or favorite services and download the associated Agentlets. The **Mobile Accessing Module** is a bridge for mobile platforms to launch and collect agents to and from the network.

3.1 Agent Launch Module

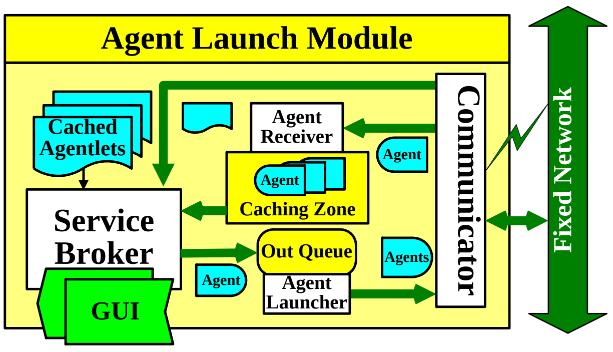

Figure 2: Functionality of Agent Launch Module

There are two types of Agent Launch Module, one is for hosts with permanent connection to the network, and the other is for mobile platforms. They have almost the same components except the physical links to the permanent network. The main functionality of this module can be shown as in Figure 2.

A user interacts with the **Service Broker** component via a GUI. The Service Broker displays to the user the services whose associated Agentlets are cached locally. Alternatively, when requested, it may consult a remote Directory Service Module for a wider range of services. The associated Agentlet of a service that is found in a remote Directory Service Module is downloaded if the user chooses that service. Thus, through the Service Broker a user can easily find out the desired services and use them. To use a service, the associated Agentlet is invoked first to collect from the user the necessary information for carrying out that service. This information may be the key attribute of a distributed database table to be sorted, or it may be the business logic of a transaction for some electronic commerce, and so on. Several Agentlets may be activated in the course of creating an agent. In other words, many service items may be packed into a single agent, and each of them with its own service information. A service item normally starts with a migration command followed by some services. In addition to the service items, every agent also has an ID field, which uniquely identifies the agent and the user who creates the agent. The content of a simplified agent carrier is shown as in Figure 3.

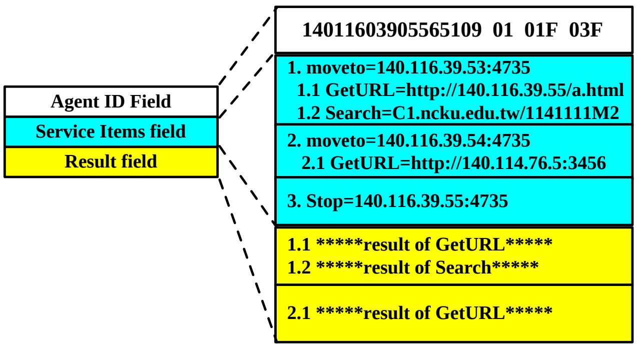
Agent ID Field
Service Items field
Result field
14011603905565109 01 01F 03F
1. moveto=140.116.39.53:4735
1.1 GetURL=http://140.116.39.55/a.html
1.2 Search=C1.ncku.edu.tw/1141111M2
2. moveto=140.116.39.54:4735
2.1 GetURL=http://140.114.76.5:3456
3. Stop=140.116.39.55:4735
1.1 *****result of GetURL*****
1.2 *****result of Search*****
2.1 *****result of GetURL*****

Figure 3: A Simplified Agent Carrier Content

After an agent is created, the Service Broker puts it into the Out Queue. The Agent Launcher will launch all the agents in the Out Queue into the network to start their journeys. For mobile platforms, the Out Queue is replaced with a Temporarily Waiting Lounge. Agents created in a mobile platform will stay in the lounge until a proper connection to the permanent network is made. An agent, after roaming through the network and having all its service items served, does not necessarily return to it birthplace. Users can specify the final location for an agent to stop. This capability allows a mobile platform to release agents to work and later collect them at another location, a great support for mobile computing. In any case, agents are received and put into a Caching Zone by the Agent Receiver. Users can then inspect the results of all the services carried by an agent via the Service Broker. Other housekeeping works, such as terminating an agent and saving the results, are also the responsibility of the Service Broker.

3.2 Agent Server Module

The Agent Server Module plays the major role in serving the service items in a received agent. It resides at a service station that provides different kind of services. Figure 4 shows the functionality and components of an Agent Server Module.

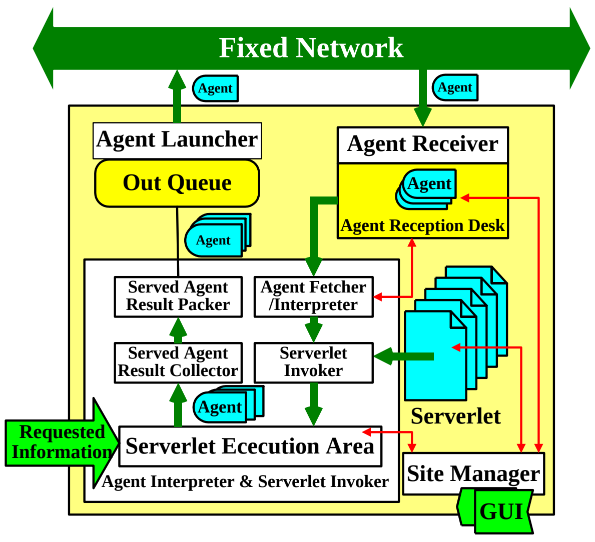

Figure 4: Functionality of Agent Server Module

An incoming agent is received by the Agent Receiver, and is put into a waiting area called the Agent Reception Desk. Later, the Agent Fetcher/Interpreter fetches the agent, picks out the service items targeted for this service station, and extracts service information from these items. The extracted information from a service item is then handed to the Serverlet Invoker to initiate and instantiate the corresponding Serverlet. After instantiated, a Serverlet enters the Serverlet Execution Area and executes there. When the service is finished, the execution result is passed to the Served Agent Result Collector that finds out the agent owning the result. It then gives both the agent and the result to the Served Agent Result Packer. This component packs the result together with some indexing information into the agent and puts it into the Out Queue. Finally, the Agent Launcher fires the agent into the network to continue its unfinished journey.

For the time being, MACE is targeted at applications in which agents execute to produce only simple short results and agents carry these results while they are travelling. For those applications that might transfer voluminous results between distributed sites, a logistics and delivery system is being investigated.

3.3 Directory Service Module

Figure 5 shows the functionality of a Directory Service Module. This Module, as its name suggests, helps a user to find specific services. It is an important part of the implementation of Service Framework in MACE. Agentlets of a variety of services are managed by the Directory Server in the module and entries for these services are kept in the Service Directory. The Service Framework defines how a service provider can register a certain kind of services with the Service Directory. In the mean time, associated Agentlets of these recorded

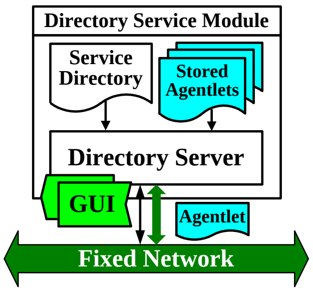

Figure 5: Functionality of Directory Service Module

services must be uploaded to the service station. A service provider can renew the Agentlet of a service when it is necessary. The Directory Server automatically manipulates version information of Agentlets.

3.4 Mobile Accessing Module

To support mobile computing, MACE includes a component named Mobile Accessing Module. In addition to the operations of normal wireless network connections, this module supports the so-called *disconnected operations* for mobile platforms [GKN+]. The functionality of the module is shown as in Figure 6.

The Mobile Accessing Module resides in a docking host that operates a physical device to accept wireless connection from mobile platforms. The Accessing Point is a bridge between wireless and permanent network. As described in section 3.1, there is a Temporarily Waiting Lounge in the Agent Launch Module of a mobile platform. Agents created in the mobile platform wait there until a proper wireless connection is established between the mobile platform and the docking host. They then quickly jump off to the docking host, from which they are launched into the permanent network to carry out their missions. This allows the mobile platform to disconnect from the network as soon as possible, and to stay disconnected while the computation is going on.

On the other hand, mobile agents, after being properly served, may return to the original or a pre-specified docking host to find that the mobile platform is still disconnected. They then enter another Temporarily Waiting Lounge to wait for the reconnection. Once the reconnection happens, mobile agents will quickly jump back to the mobile platform and report their achievements.

4 Security Notes

Security is one of the most concerned issues in the mobile agent paradigm [FGS1]. To be widely accepted, especially for electronic commerce, a mobile agent system must secure the safety in every perspective. Allowing a foreign code segment to access directly the CPU

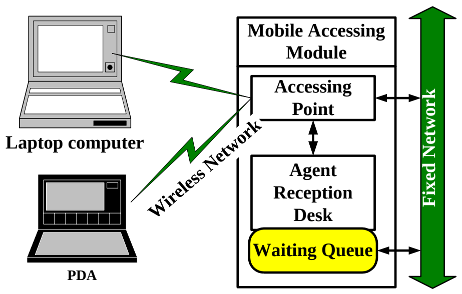

Figure 6: Functionality of Mobile Accessing Module

and other system resources in a server is always worrying. Virus attacks are the most obvious examples. There are so many methods by which malicious or illegitimate mobile agents can destruct an agent execution environment. With no mechanism to authenticate incoming agents, servers are easily exposed to security threats. On the other hand, mobile agents are also vulnerable to malicious hosts. For example, a malicious host can implant its own tasks into an agent or modify the agent's state. This may lead to the theft of resources from the agent or from other sites.

The argumentation of this paper is that, from security's perspective, passive agents that are interpreted are preferable to active agents that execute on their own. This can be proven from the acceptance of JAVA applets and Aglets [LO1], [Ag1]. However, security threats may still exist even though one can guarantee the safety and the creditability of the agent execution environment of a service station. Travelling agents are particularly vulnerable to interception, modification, and cloning. Most of the mobile agent systems and agent construction languages include security management architecture and security model [KLO1], [TFM1], [TV1].

4.1 Security in MACE

Limited to the paragraphs, this paper focuses mainly on the details of functionality and the Service Framework of MACE. In fact, security issues in MACE have also being thoroughly studied. Security threats, arising from mobile agent's migration, execution, and communication, were identified and classified. In general, they fall into three categories, i.e. threats from agent execution environments or agent interpreters, from agents themselves, and from transportation media. What have also been investigated in MACE to solve these problems are:

- How and what the identity should be carried in a mobile agent for authentication performed at all the service station,
- How the mobile agents should be encrypted and decrypted before and after migration, and

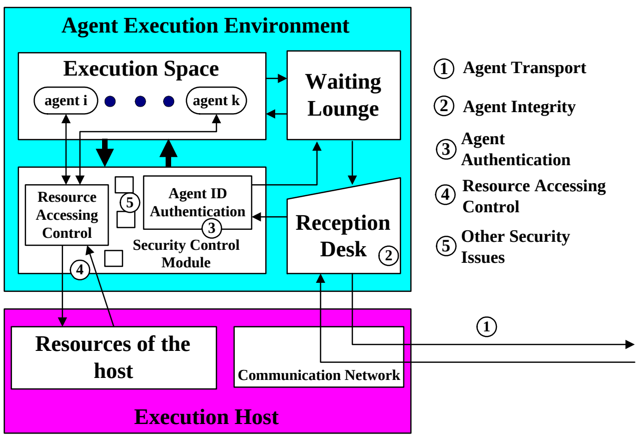

Figure 7: Extra Modules For Security Control

- How the public keys and private keys should be transferred through secret channel between service stations?

Several security management modules have been added to the Agent Server Module to answer these questions. These modules are shown in figure 7. They perform several security controls, three important ones of them are:

- **Reception Desk Integrity Test:** This integrity test is an additional task of the Reception Desk. It detects transmission errors that occurs in agent's transportation. A complete set of rules were made for this module to detect whether the identity, the service items, and the collected data of an agent were modified or even completely replaced by a cloned agent at a previous service station.
- **Agent Identity and Content Security Control:** This is done by the Security Control Module. The first is to authenticate the identity of a mobile agent and to assign different service rights to the agent according to the privilege of the agent's owner, i.e. its creator or legal user. The second is to encrypt and decrypt an agent when it comes and goes, and to exchange proper keys with other service stations involved in the agent's mission.
- **Resource Access Control:** This control gives an agent the proper right to access to resources of the execution host.

5 Application Notes

A prototype of MACE has been operational. It has been used to build a simple information retrieving system, in which mobile agents autonomously roam through a network (ultimately the Internet) to find information resources and collect desired information. Another going on project is to build a distributed database system, using MACE as its infrastructure. Agents carrying SQL queries migrate among distributed database sites to consult tables and to perform on these tables operations such as projection, production, and more. Database operations might result in large quantity of intermediate data that is not suitable for agents to carry while traveling. This project thus initiates the design of a logistics and delivery system.

In Recent years, mobile information systems have been attracting more and more researchers as they discover the benefits of being able to connect to distributed information resources without any spatial and temporal constrain [Gr1]. However, mobile information retrievers such as PDAs or laptops also raise new issues in the context of communication. Drawbacks like low bandwidth and high latency of a present wireless network make it preferable to connect a mobile platform to a wireless network for a period as short as possible. Another well-marked problem is that a mobile platform may get different network address assignments in different connections to a wireless network. This makes a mobile platform more appropriate as an information sender than as a receiver if it keeps moving. These mobility-related difficulties are yet to be eliminated and autonomous mobile agent seems to be one of the promising technologies to respond to these questions. A planed project is to use MACE to support mobile information retrieval. MACE in itself is effective enough to be the infrastructure of a distributed information system. To support effective on-the-go information access, a component that interacts with the local information processing system such as the DBMS will be added; another addition will be a staging post that serves as a temporary stopping place for mobile agents with retrieved data. The former can be simply built as a Serverlet, and the latter is adapted from a docking host. The resulting framework can easily support diverse plugged-in value-added services too.

Finally, unlike other mobile agent systems, there is not yet a mechanism designed for the communication between agents in MACE. The primary reason for not designing such a mechanism is as simple as to cut down the messages to prevent them from being intercepted. Nevertheless, it is observed that communication between agents is not quite necessary in most MACE prototype applications, as it might seem to be.

6 Other Related Works

Several prototype or commercialized systems have been proposed and developed in recent years. A well-known mobile agent system is Telescript developed at General Magic [Wh1]. Telescript uses a proprietary script language to create agents. It also supports mobile platforms and has been successfully used on Personal Digital Assistants (PDA). However, neither the detail of how mobile agents jumping between mobile platforms nor the handling techniques of *disconnected operation* is released. Security issues have been studied, though [TV1]. Another script-based mobile agent system is Agent TCL [Gr1], [KGN+1], developed at Dartmouth College. A set of special commands was added to an existing high-level scripting language Tcl, developed in 1987, to create Agent Tcl. An agent uses these commands to migrate from host to host and to communicate with other agents. Security issues have been studied and implemented, too.

Like MACE, these two systems use a proprietary script language to create passive agents. But, unlike MACE, users of these systems have to create these agents by themselves.

The development of JAVA accelerated the design and implementation of several mobile agent systems. Most of these systems depend heavily on JAVA Virtual Machine, JAVA class loading model, and other features, such as serialization, remote method invocation (RMI), multi-threading, and reflection. The greatest advantage of using JAVA to develop a mobile agent system is its ability to operate in heterogeneous platforms. Among these JAVA-based systems, Aglets [LO1], [Ag1], developed at IBM's Tokyo Research Laboratory, has been very popular for some time. It is praised for its GUI that makes the world of agents accessible to the JAVA novice. A distinguished feature is to provide the security preference that allows the owner to specify the degree of trust of aglets. Another JAVA-based mobile agent system is Odyssey [Ge1], also by General Magic. Though implemented purely in JAVA, it still incorporates some of the concepts previously developed for Telescript. One unique feature of the Odyssey system is its audit trail mechanism to help programmers debugging their agents.

Unlike MACE, theses systems offer an integrated environment for agent programming, transportation, communication, execution, and more. In spite of offering a total solution, they may not be appropriate for an application such as the *service-on-demand*. MACE will compensate this by using its unique Service Framework.

7 Conclusion and Future Work

This paper has presented the design and the implementation of a Mobile Agent Carrier Environment-MACE for distributed *service-on-demand* applications. A simple yet powerful Service Framework is described, which makes service providers free from being confined to a specific developing environment. Owing to it flexibility, one can also use MACE to develop an entire distributed system. For the time being, an agent tracking mechanism is being planned. It will be used to trace the existence and the location of all previously created mobile agents in order to direct the dangling agents to the right places. A logistics and delivery system, as mentioned before, is also critical to the practicality of MACE. It will make MACE suitable for more application fields. In the long run, a coordinator, which masterminds the cooperation between agents, will be investigated. Hopefully, it will include a GUI for MACE users to precisely specify the relationship between service items and between agents. Other issues such as resource discovery tools and a better navigation system are also important. They all are also the important subjects of future researches.

References

[Ag1] Aglets: Mobile Java Agents, IBM Tokyo Research Lab, URL = http://www.ibm.co.jp/*trl*/*projects*/*aglets*

[CTB1] S. Covaci,; Zhang Tianning; I. Busse, "Java-based intelligent mobile agents for open system management", In Proceedings of Ninth IEEE International Conference on Tools with Artificial Intelligence, Page(s): 492 -501, 1997.

[FGS1] W.M. Farmer, J.D. Guttman and V. Swarup, " Security for Mobile Agents: Issues and Requirements, ". Proceedings 19th National System Security Conf. (NISSC 96), 1996, pp.591-597.1997.

[GKN+] Robert Gray, David Kotz, Saurab Nog, Daniela Rus and Geoge Cybenko. "Mobile Agents for mobile computing.", Technical Report PCS-TR96-285, Department of Computer Science, Dartmouth College, May 1996.

[Gr1] R.S. Gray. "Agent Tcl: A Flexible and Secure Mobile-Agent System," in Proc. Fourth Annual Tcl/Tk Workshop (TCL 96), 1996.

[Ge1] General Magic, "Agent Technology: General
Magic's Odyssey", http://www.genmagic.com/html/agent_overview.html, 1997.

[KLO1] G. Karjoth; D. B. Lange; M. Oshima, "A security model for Aglets", IEEE Internet Computing, Volume: 1 4 , Page(s): 68 -77, July-Aug. 1997.

[KK1] Keith D. Kotay and David Kotz, "Transportable Agents ". In Proceedings of the CIKM Workshop on Intelligent Information Agents, Third International Conference on Information and Knowledge Management, Gaithersburg, Maryland, December 1994

[KGN+1] D. Kotz; R. Gray; S. Nog; D. Rus; S. Chawla; G. Cybenko, "AGENT TCL: targeting the needs of mobile computers", IEEE Internet Computing Volume: 1 4 , Page(s): 58 -67, 1997.

[LO1] D.B. Lange and M. Oshima, Java Agent API: Programming and Deploying Aglets with Java, published by Addison-Wesley, Fall 1997

[Or1] J.J. Ordille, "When Agents Roam, Who Can You Trust?" Proc. Fitst Conf. on Emerging Technologies and Applications in Communications(etaCOM), May 1996.

[PTV1] A. Puliafito; O. Tomarchio; L. Vita, "MAP: Design and implementation of a mobile agents' platform", In Journal Of Systems Architecture, Vol 46, Issue: 2, pp. 145-162, 2000

[TV1] J. Tardo and L. Valente. "Mobile Agent Security and Telescript," Proc. IEEE CompCon 96, IEEE Computer Society Press, Los Alamitos, Calif., 1996

[TFM1] C. Thirunavukkarasu, T. Finin, and J. Mayfield. "Secret agent: A Security Architecture for the KQML Agent Communication Language," Proc. Intelligent Information Agents Workshop held in conjunction with Fourth Int'l Conf. Information and Knowledge Management CIKM 95, Baltimore, Dec. 1995.

[Wh1] J. E. White, "Telescript technology: The foundation for the electronic marketplace." General Magic White Paper, 1994.