INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Design and Implementation of a Mobile Agent Management and Tracking System

By
Amal Karboubi

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of

M.A.Sc.
in

Electrical Engineering

Ottawa Carleton Institute for Electrical & Computer Engineering
School of Information and Technology Engineering (SITE)
University of Ottawa

May, 2000

Table of contents

List of Figures

Acknowledgements

I am very grateful to Professor Ahmed Karmouch, my academic supervisor for his support, his continuous guidance and the time and effort he has invested in supervising this work.

I would also like to thank the Multimedia and Mobile Agent Research Laboratory members for their help and provision of information.

Another major thanks goes to my parents and friends --all of them, for the assistance and the understanding during all these years of education, without which this work would not have been possible.

ABSTRACT

In the age of information overflow, the mobile agent technology has become an important paradigm. They are supposed to address many problems such as reducing network traffic, surmounting network latencies and improving robustness and fault-tolerant capabilities of distributed systems.

Mobile agents are deployed in different areas such as industrial manufacturing, Internet searching, electronic commerce, on-line shopping, network management, software distribution and many others. However in order to take full advantage from this technology we still need to address and solve some significant problems related to mobile agent namely those in the areas of agent's identification and addressing, communication, security, tracking, location control, resource discovery and scalability.

In the context of this thesis, we propose a global mobile agent management and tracking system that addresses the above problems and also attempts to address the particular issues resulting from the mobile agent paradigm itself with respect to client server paradigm. This framework encourages widespread use of agent technology by providing basic support for control and monitoring agents within the distributed environment through a graphical user interface for better interaction with the mobile agent system. The system also includes a directory assistance service for mobile agents that can help them finding relevant services to achieve the requirements of their mission.

Chapter 1

Introduction

1.1 Overview

The mobile agent paradigm arises to provide distributed applications with more flexibility and gain in performance. It has often been identified and compared with the well-established client-server paradigm. The mobile agent paradigm introduces the notion of location awareness by making the computation explicitly location aware and movable leading to high degree of freedom.

A mobile agent system should provide the necessary services and facilities to help mobile agents to accomplish their mission. These are basic and abstract services, independent of any application or development context and can be implemented in many different ways. Some of theses services or facilities can be easily classified and assigned to appropriate components of the mobile agent system (e.g., agent execution is a service that must be provided by the agent platform), while others can not be easily allocated and may be implemented by different components or even be partitioned over several components that have to cooperate to provide the service (e.g., directory facilitator, location service). The interface that defines the different interactions between a mobile agent and the underlying agent platform is still ambiguous and provides a large amount of flexibility in where and how to design the software component that implements a particular service.

1.2 Objectives

This thesis focuses on the features that are currently inadequately supported by existing mobile agent architectures or even absent namely those in the areas of agent's creation, communication, security, tracking, location control and scalability. All these aspects can be gathered under the area of agent management. The goal of this thesis is to propose a mobile agent management framework that addresses the above problems and also attempts to address the particular issues resulting from the mobile agent paradigm with respect to the client server paradigm. This framework also encourages widespread use of

agent technology by proposing solutions for monitoring and controlling mobile agents. A software delivery application using mobile agents serves as a testbed for this framework. It enables the use of agent technology and provides the user with a simple means to remotely create, launch, and monitor the agent activity.

1.3 Agent Management

This model can be seen as a two distinct parts. One part, the Agent Management System (AMS) is concerned with the low-level functionality such as creating, naming, cloning, locating and control. The other part, the Directory Facilitator (DF) is more concerned with offering directory assistance to other mobile agents; its main task is to provide "yellow pages" services to agents. Agents may register their services with this module or submit queries to find out what services are offered by other agents.

1.4 Agent Management System

The agent management system (AMS) is concerned with the control and identification of mobile agents. For instance, an agent owner may want to know about the progress of a mobile agent's execution or simply its current location. The latter information can then be used to send a message to the mobile agent, or even to issue a command that instructs the mobile agent to change its itinerary, to return to its owner, or even to terminate. The agent management system is far from being optimal in current systems, particularly when dealing with a group of many mobile agents (MA), a management framework describing how different low level facilities should be realized and coordinated to achieve scalable and optimal control is also missing. Currently, this part has being addressed by FIPA as well as the OMG group.

1.5 Directory Facilitator

Mobile agents should be capable to dynamically discover required services and resources before actually migrating to a new agent platform. Service discovery and trading is a fundamental premise of MA. The responsibility of the directory facilitator part is to provide directory assistance and referential services that can aid mobile agents in the discovery of relevant services in the domains which the agent might visit. Current systems include X.500 and CORBA Trader while new and up coming systems are LDAP

and FIPA. Optionally, this module can also supply information about the state of the network (i.e. Network topology information services) and also the host state. This information can help a mobile agent to make planning decisions based on the requirements of its mission.

1.6 Main Contributions

The main contribution of this research consists of defining and building a management framework for mobile agent system, which combines facilities that are realized and coordinated to achieve scalable and optimal control of mobile agents.

The system supports communication transparency and defines a tracking and locating strategy for both agent and services. It also provides basic supports for control and monitoring agents within the distributed environment through a graphical user interface for better interaction with the mobile agent system. The fault tolerance and data management models help handling failure problems ranging from node crash, agent integrity violation or network latencies and protecting the agent's sensitive data. We have also described how all these various services are related to each other and how they fit into the global agent framework in order to meet the global and fundamental requirements of mobile agent paradigm.

1.7 Thesis Outline

This thesis is structured as follows. In the next chapter we introduce the mobile agent paradigm, its deployment areas and identify the particular problem we are concerned about. Chapter 3 is concerned with the current state-of-the-art in mobile agent systems. It reviews a couple of enabling technologies on which a mobile agent system is built. In Chapter 4 we further develop our approach for mobile agent management and tracking. It gives an overview of the framework, the requirements, design principles and architecture. Chapter 5 describes the implementation of the different modules involved in the design section. It goes on to give two example-scenarios for the framework use. Finally, in the Conclusion, we summarize the major results of this work and outline future research and possible developments around the presented technology.

Chapter2

Mobile Agent Paradigm

2.1 Introduction

The word agent is probably one of the most used (and misused) terms in computer science communities and has very different meanings in the area of artificial intelligence, network management, or distributed systems. Since there is no clear consensus on the definition, (recent postings to the software agents' mailing lists prove this), it is very difficult to give a general accepted definition for what an agent is. Nonetheless, when dealing with mobile agents, we can focus on distributed systems research areas, where mobile agents are considered as an interesting and innovative approach to structure distributed applications [CHK95].

Since we are, in the context of this thesis, primarily interested in the management of mobile agents and the deployment of distributed directory assistance to all the community of MA, we do not consider this lack of formal definition as a disadvantage. It is one of our explicit goals to keep the approach of mobile agent management as general as possible so that it can be applied to most systems that fall into the category of mobile agent systems. Therefore, we will keep the definition of what we consider to be a mobile agent system quite broad and simply identify the basic properties and requirements of mobile agent systems that we have to assume in order to apply our approach.

This chapter provides the reader with a brief overview of what an agent is and will explore various technical issues in the context of mobile agents in order to identify these basic properties and requirements.

2.2 General Agent Definition

We are all in one sense or another familiar with the concept of an agent. Indeed, in our societies many organizations are playing this role; the travel and estate agent are a good example that shows the representative role, that is they both act on behalf of other

entities. Many definitions from Maes or Wooldridge claim that acting on behalf of another entity is the first fundamental property of agents. Another fundamental characteristic of agent is autonomy. Autonomy refers to the principle that agents can operate on their own without the need for human guidance, even though this would sometimes be unwanted. Hence agents have individual internal states and goals, and they act in such a manner as to meet their goals on behalf of their users.

A third important aspect of an agent's behavior is the degree of pro-activity and reactivity. Reactivity is related to the fact that agents can perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet or perhaps all of these combined), and respond in a timely fashion to changes that occur in it [JWO 95].

Pro-activeness refers to the principle that agents do not simply act in response to their environment; they are able to exhibit an "unexpected" behavior by taking the initiative so that it looks like if they are "suddenly" doing things with no direct input from their environments. However, the difference between a proactive behavior and reactive behavior is not clear. Indeed, one can see the pro-active behavior as a simple reaction pattern that is so complex that we cannot easily figure out how it came about, i.e., when certain actions cannot easily be deduced to some input, the action is (seemingly) pro-active. Thus, if we consider the passing of time as an input stimuli, any agent "suddenly" doing things (hence appearing to be pro-active) is simply reacting to the fact that time has passed. Had time not passed, and no other input stimuli had been received, nothing would have happened ever.

This argues that each agent is intrinsically reactive, being composed of a collection of lines of code. However, seen from an external viewer not from programmer point of view, the observable behavior of software agent entity can be taken as a pro-active behavior. At a low level, we can consider pro-activeness as a form of reactivity even though it is useful to consider these features differently for systems and applications.

Finally, agents may also exhibit some different level of attributes, the key ones of which are learning, cooperation and mobility.

Cooperation with other agents is an important feature; in order to cooperate agents need to possess the ability to interact with other agents and possibly humans via some communication language such as KQML (Knowledge Query Manipulation Language) [JWO 95]. Moreover, agents are communicating at a higher level of discourse, i.e. that the contents of the communication are meaningful statements about the agents' environment or knowledge. This is one characteristic that differentiates agent communication from, for example, other interactions between strongly encapsulated computational entities such as method invocation in CORBA [OMG98].

Agents might also learn as they react and/or interact with their external environment. This assumes that they embodied bits intelligence. Though, we will not attempt to define what intelligence is, we maintain that a key attribute of any intelligent being is its ability to learn. The learning may also take the form of increased performance over time.

2.3 Environment of Mobile Agents

This section briefly describes the underlying infrastructure that can serve as an environment for mobile agents and also the different computing hardware that can serve as a platform for mobile agents' execution.

2.3.1 Possible Environments for MA

A MA environment is very similar to what currently exists in the Internet and in the World Wide Web (WWW). However, the Internet is not the only existing infrastructure in which mobile agents can be deployed. The current telecommunication networks also provide sufficiently powerful computing technology to serve as an environment for mobile agents. With the telecommunication network playing a pivotal role in present and future information systems, it's easy to understand the growing interest in using mobile agents as part of the possible solutions to implement more flexible and decentralized network architecture. As a matter of fact, it is not obvious to separate these both infrastructures from each other. The Internet is in part realized on top of the telecommunication networks and on the other hand there are considerable efforts to provide traditional telecommunication such as IP telephony and different QoS schemes over the Internet.

The current environment for telecommunication is based on international standards such as Telecommunication Management Network (TMN), Intelligent Network (IN) and Telecommunication Information Networking Architecture (TINA), which is considered as an evolution from TMN and IN in the area of telecommunication and management.

The concept of mobile agents has been already used for the provision of services in future telecommunication networks as illustrated in the TINA effort [DNI95, URL1]. Moreover, the TINA Consortium (TINA-C) also considered this issue in a broadening effort to leverage these new technologies.

The TMN provides a framework for achieving interconnectivity and communication across heterogeneous operating systems and telecommunications networks. It was developed by the International Telecommunications Union (ITU) as an infrastructure to support management and deployment of dynamic telecommunications services. It uses the OSI Management Standards as its framework and can be applied to wireless communications and cable TV as well as to private and public wired networks [URL2].

An intelligent network (IN) is a service-independent telecommunications network. That is, intelligence is taken out of the switch and placed in computer nodes that are distributed throughout the network. This provides the network operator with the means to develop and control services more efficiently. New capabilities can be rapidly introduced into the network. Once introduced, services are easily customized to meet individual customer's needs.

Both IN and TMN rely on the traditional client/server paradigm to offer their services via Services Control Points (SCP). During execution of service the distributed exchanges known as Services Switching points (SSP) will ask the SCP for control services so that the SSP could carry out the actual processing. SCP and SSP communicate via an RPC-based protocol. To be able to use these IN services, another component which is the Service Management System (SMS) will download the appropriate service components into the IN network elements.

Other examples for this convergence between the Internet with its IP-based protocols and more traditional telecommunication networks are the increasing interconnection of services via gateways. An important and very known established gateway is the Internet

Service Providers (ISP), which allow accessing the Internet via conventional telecommunication facilities. We therefore assume that future computing devices will be able to take advantage of any available communication links, which will all provide a connection to the Internet and, thus, to all possible service providers on the Internet such as wireless telecommunication networks, fixed telecommunication networks, and IP or ATM networks.

2.3.2 Agents and Mobile Code:

The actual usage of mobile agents or mobile code in a real system is still under research. There is a big trend for introducing this new technology to provide telecommunication services and management and to help the decentralization of network management services. These services can be implemented using the mobile code paradigm: code on demand (COD) or remote evaluation (REV) or the mobile agent paradigm.

The difference between REV and COD rests on how the process of migration is initiated.

In REV, the sender of the mobile code would initiate the transfer of the mobile code to the receiver. Thus a small code fragment is moved to the devices where it is allowed to invoke other codes to complete the service. The idea of the REV is, for instance, that one can customize the basic low services of a provider by transferring some additional code that composes these low level services into a high level services. It also provides a dynamic configuration change; the manager can pack a series of commands to be sent by REV mechanism to the device where they can invoke and execute built-in functionalities. This can be seen as an extension of RPC paradigm, which only allows calling the basic service of the provider.

COD was originally proposed in [BGP 97]. In COD, it is the receiver of the mobile code who takes the initiative to request the mobile code from the sender. The idea of the COD paradigm is that a computation running on a particular platform can download and link some code from a provider to perform a particular task. This can, for instance, be a user interface, that interacts with the user to seize a certain amount of data, or a computational component that is required to perform a particular task. It allows dynamic configuration and functionality of network devices. This is exactly the paradigm that is supported by

Java applets [SUN 98], which consist of a set of Java classes that can be referenced in an HTML document and are subsequently downloaded and executed when this document is accessed by a remote user.

The two paradigms are of high importance since they are easy to implement in real systems and, thus, more readily available for experimentation. Examples for a straightforward implementation of the COD paradigm are Java applets and ActiveX controls. Furthermore, in Java it is also easy to implement the REV paradigm based on the object serialization that is provided with the Java Remote Method Invocation (RMI) [URL3]. This can be done by implementing a simple server that accepts serialized Java objects and instantiates them locally.

[BGP 97] research aims to test the effectiveness of these paradigms and compare it with the mobile agent approach. The work is promising but still incomplete. The problems raised by mobile code and the solutions proposed to tackle these problems, in particular in the context of scalability, migration and management are very similar to those of mobile agent systems, which are discussed in section 2.4.

2.3.3 Computing hardware

One of the defining trends of the 1990s has been the explosive growth of the Internet. A growing number of people have Internet access at work, at home, and on the road. Meanwhile, other types of networks, such as cell phones and pager networks are proliferating rapidly. In the next decade, more people will expect ubiquitous network access---the ability to communicate with anyone, anywhere, at any time and in any form. This can be confirmed in the rapidly increasing number of subscribers to mobile communication services. Moreover, none can deny the success of new computing devices that are often called personal digital assistants (PDA), such as 3Com's PalmPilot or palm-size devices based on Windows CE. The merging of these end-user equipments will lead to completely new communication and computing devices that will be both a mobile phone and a PDA.

The devices that serve as a host for mobile agents can be classified into two types: The "client" and the "server". The client is concerned with providing a mean for the interaction between the user and the mobile agent (possibly via some special user

interface software). The user will provide the necessary configuration information to the mobile agent and will launch it to accomplish its task. The role of a server is to receive the mobile agent and to provide a platform where the agent can execute. Here, the mobile agent can interact with other agents that are located at this platform and can access resources that are available from this platform. Often it will be necessary for a mobile agent to interact with many servers to accomplish its task or even to visit several of them to interact directly with local resources.

Other important factors for a host is its capabilities (i.e., available resources and connectivity). For a host in the server role we assume that it will have access to the fixed network, very high bandwidth communication facilities, permanent connectivity, and huge processing and storage resources. It will directly interact with end-users only for management purposes, therefore the user interface capabilities are not of immediate importance for its functionality. There will be many different actual systems, but we assume that most of them will satisfy the above requirements.

2.4 The Mobile Agent Paradigm

In this discussion of the mobile agent paradigm, we are primarily concerned with the supporting infrastructure that provides the necessary services for mobile agents. There are many different approaches to implement such an infrastructure; it may be possible to learn from the experience made in the design of distributed systems community to come up with such standard for mobile agent systems [OMG98, RED 97].

As we have already mentioned in the introduction of this chapter, we do not want to focus on a mobile agent system in particular, but we are concerned with the management of mobile agents in general. Therefore, we will not describe a single mobile agent system, but rather identify the issues that need to be resolved for our approach to be applicable and point out possible design alternatives.

In the coming sections, we will analyze the mobile agent system from the viewpoint of a lower abstraction layer and examine the infrastructure components as well as the services they have to provide.

2.4.1 An Example

An often cited example in the domain of agent based electronic commerce is that of a person who wants to buy an airline ticket using the facilities provided by a mobile agent system. These benefits are largely non-functional; i.e. we could built applications without using mobile agents and use instead the client server paradigm. Without using mobile agent, the user is required to write a program that would allow him/her to make reservation by accessing several airline reservation databases. After listing all her preferences (e.g. non-smoking, departure between 7 and 9.30 am from Baltimore, arrival at Austin before noon, no more than one stop-over, and no changes at Chicago O'Hare) the program would need to request all the available databases for flights meeting these requirements which may total more than 200 possibilities and consume large memory. Each of these actions involves filtering through plenty of unnecessary information which could/would reduce the speed or clog up the network. If we consider the alternative of using a mobile agent the scenario will look like the following. The user encapsulates the entire program within an agent, which consumes probably less than 2 kilobytes, that roams the network of airline reservation systems, queries their databases locally, and returns ultimately to the home computer, with different options that might be confirmed or refuted. This alternative overcomes the high communications costs of transferring possibly, large data to a local computer that presumably cannot cope with.

2.4.2 The Mobile Agent

At the center of the mobile agent paradigm is the mobile agent. It represents the entity that migrates between the agent platforms in order to accomplish its tasks.

Mobile agent technology originally emerged from advances made in distributed systems research. MA are also known as itinerant agents [CHK 95], transportable agents [KGR 96] and re-locatable objects. The idea of transporting programs around a network is not a new one: Batch processing or process migration concept although at a different level abstraction can be considered as a primitive form of mobile agent technology, which has been around since the 1960's. However, we believe that the basic ability of a program to transport itself to new locations is not sufficient to describe a mobile agent.

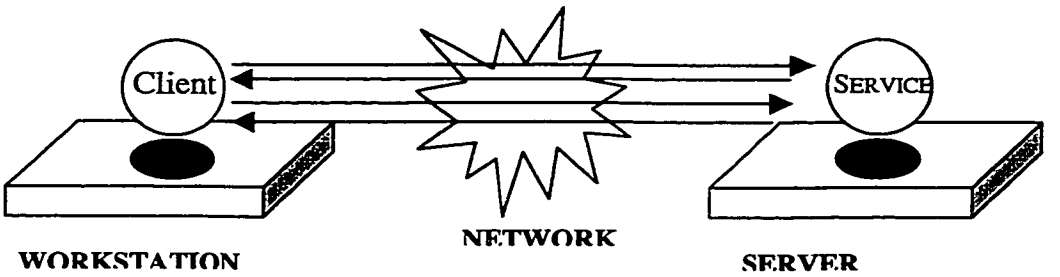


Figure 2.1 Communication Using the Client Server Model

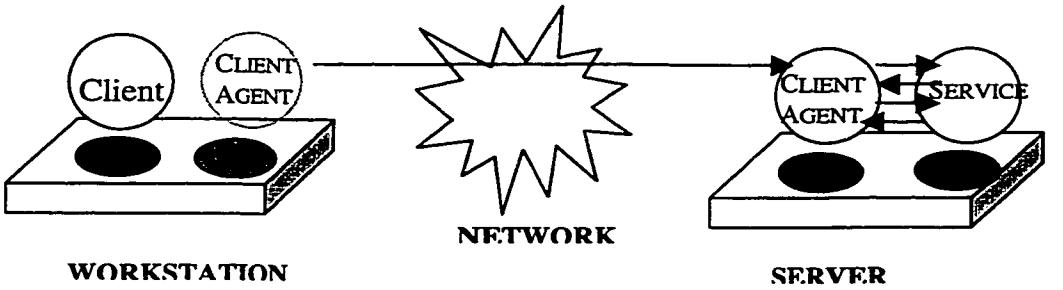


Figure 2.2 Communication using the MA Paradigm

Based on existing literature [CHK95, FIP98, MIT 97] we believe that the following definition characterize the concept of a mobile agent.

Mobile agents are computational software processes, which exist in a software environment that inherit some of the characteristics of an Agent and are capable of roaming wide area networks (WANs) such as the WWW, interacting with foreign hosts to perform the duties set by its user. These duties may range from a flight reservation to managing a telecommunications network.

From an Object Oriented point of view, a mobile agent would be seen as an object (or set of objects) that consists of code and state; in other words, it is active and has an execution thread (or several thread of its own). It can (autonomously) migrate from one agent platform to another where it will interact with local objects, data, or other facilities. It executes a well-defined task on behalf of a user from whom it also obtains its authority and tries to accomplish this task without further intervention from the user (unless unexpected events occur).

Despite the difference in the definitions, most research examples of the mobile agent paradigm as reported in the literatures have two general goals: reduction of network traffic and asynchronous interaction.

In Figure 2.1, a client communicates with a remote service over a network using the client-server paradigm. However in the figure2.2 the client creates a mobile agent to do its bidding, then dispatches it to the remote service where it can interact locally with that service.

Life Cycle of a mobile agent:

This model defines the different execution states in the life cycle of a mobile agent [FIP 97, CHK95], from creation to termination, and the events that cause the movement from one state to another.

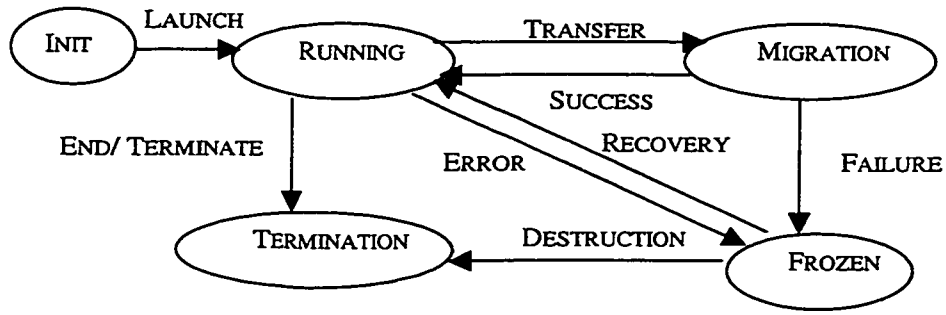


Figure 2.3 Life Cycle of a Mobile Agent

A life cycle of agent starts by initiating the agent, this includes creation, naming and assigning a life span. Afterwards, the agent is launched and enters its 'running' state where a persistent process is executed. When a migration request is granted by the foreign host, the agent transfer is initiated and the agent enters the 'migration' state when the process in the running state is suspended. Next, its context is delivered to the destination node where the process is resumed and re-enters the 'running' state at the point it left off. If any problem occurs during the agent migration, the agent enters a 'frozen' state that leads either back to the running state or termination sate in case of irreversible failure. The agent termination is determined by the user either at the end of

the mission, or by a terminate command during its execution. This life cycle represents a simple, basic and flexible life cycle that can be extended for more complicated scenarios.

2.4.3 The Agent Platform

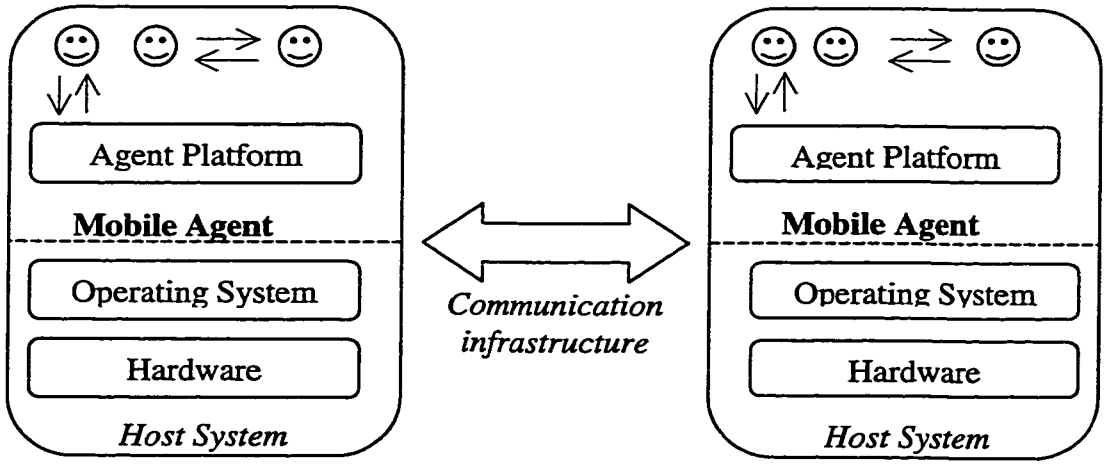


Figure 2.4 Infrastructure for Mobile Agents

The agent platform is the other fundamental part of the infrastructure and represents the static component of a mobile agent system. It can be seen as a distributed software environment that is built on top of the host system to provide an environment where mobile agents can execute.

The main functionality of an agent platform is to serve as docking station for mobile agents, hides the heterogeneity of the underlying hardware, provides the necessary resources and facilities for the agent's execution and defines how the mobile agents can access them. Section 2.4.4 describes in more details the general requirements and services that have to be implemented by the agent platform.

2.4.4 The Services of Mobile Agent Platform

In this section, we will provide an overview of the required services on a conceptual level without technological details [MIT 97, FIP 98, OMG 98b]. These requirements fall into eight general categories:

- Mobility Support

- Security
- Communication
- Resource Discovery
- Resource Management
- Data Management
- Heterogeneity Support
- Agent Execution

2.4.4.1 Mobility

One of the specific features that characterize and distinguish mobile agent systems from other infrastructures for distributed computing and consequently one of the most important services in a mobile agent system is the service that enables agents to move from one platform to another.

In this context, mobility refers to the manner in which a mobile agent is transported; the discovery and resolution of destination hosts are considered as another feature in the agent model that will be discussed later in chapter 4. That is, the following functionalities are required to support agent mobility:

- The ability to move a mobile agent into a 'suspended' life cycle state ready for transportation to a remote host.
- The ability to transport a mobile agent, which is in the 'suspended' state, to a remote mobile agent environment.
- The ability to receive a suspended mobile agent from a remote host and reconstitute it in a new environment.

Roughly, a mobile agent, say M, willing to migrate from the agent platform P1 on which it is currently executing to another agent platform P2, will be packaged in a suitable transport format (e.g. JAR file), and must be sent to P2. Conversely on P2, the "marshaled" agent has to be received, extracted from its transport format, admitted for execution on P2, instantiated, and resumed its execution.

The decision of whether to actually admit a received agent and to launch it on the destination platform is a very difficult problem that depends on various issues including the security issue that will be discussed in Section 2.4.4.2

After we have defined the issues that need to be solved in the context of support for agent mobility, we shall now go to technical problem, i.e. how a flow of execution on P1 can be transferred to P2 and resumed its execution there. Researchers agree that the response time bottleneck when sending an MA from one host to another is the marshalling/demarshalling time rather than the transmission time. As a matter of fact, this depends on the agent and the serialization method used. If the code and state of agent is large in size and the serialization compresses it, the transmission time should be small, while the compression might take longer. The other way around would yield to the opposite effect, given that the compression overhead is often large for small amounts of data.

An agent's state is composed from its data and its execution state. The data of an agent comprises the values of the instance variables of the set of objects that make up the agent. New objects that are dynamically created by the agent have to be added to its set of objects. This data can easily be accessed and packaged during an agent migration. The complex problem is to keep valid the links held by an object O1 to another object O2 after the migration to a new platform.

The data of the agent has to be transferred as well; this requires a complete list of the objects that belong to the agent as well as the access to the instance variables of these objects. This can be realized by implementing special method on the object that returns a stream of bytes, which encodes the values of its instance variables, similar to the object serialization in Java [SUN97].

The execution state of an agent comprises the values of the registers of the virtual machine on which it executes, the thread information, and the execution stacks for each of the threads, which hold the dynamic variables of the active methods and the method invocation history. This information is entirely managed in the execution environment and, thus, not directly accessible to the agent.

The execution state of the agent has to be transferred to P2. This is the most difficult part of the agent migration and can be handled in several ways, depending on the support provided by the agent platform. In the following paragraphs, we will to discuss two extreme cases for this support.

The literature on mobile agents [MIT 97, BGP97] identifies mobile agent systems that provide strong migration where the state, the code, and the execution stack are moved. This way, the execution can continue just after the migration instruction call. In case of light (weak) migration, only the state and the code are moved. The agents have to manage itself the execution history between locations.

- *Strong Migration*

In case of strong migration, P1 suspends the agent's execution, extracts the current Execution State of the agent from its data structures, and marshals it together with the agent's code and data into the transport format. This functionality of the platform is coupled with the actual migration and is accomplished by sending the marshaled agent to the destination platform. When P2 receives the agent and decides to admit it for execution, it instantiates the objects of the agent, integrates the agent's execution state in its data structures, and restarts the agent where it left off.

It is clear that using strong migration reduces programming efforts. This is because a programmer does not have to program his agent in a manner to control the values of the data that have to migrate with the agent when this latter migrates.

- *Weak Migration*

In the case of weak migration, most of the actions of the agent platform described above have to be taken over by the mobile agent itself. However we need to assume that there is a minimal support from the underlying agent platform. This minimal support for agent mobility consists in accepting a message that contains code and data, installing the code with proper access to its data, and to resuming its execution on the local agent platform. The mobile agent must also be capable to analyze its current execution state and it must have access to its own code. Since the agent is capable to read all of its data, it can use

the information stored to create a message that consists of the agent's code and its data, which in turn contains the agent's current execution state.

The message is then sent to P2. Once admitted and instantiated in the new agent platform, the agent must further be capable to restore itself according to the state encoded in its data, so that it can continue its execution where it left off on the previous agent platform. The original agent, which initiated the migration will wait for an acknowledgement from the migrated instance of the agent or from the destination agent platform, which confirms the successful migration of the agent, and can then terminate itself in order to finalize the migration (or it can continue its execution which effectively results in a cloning of the agent).

The real reason to use a strong migration scheme is separation of concerns. With weak migration it is almost impossible to add a management layer which says which agent has to run on which machine. With a strong migration scheme this is much easier to do since the agent is not involved in the migration process. However, this is not the approach taken by a majority of the experimental systems that offer agent mobility.

The reason for this lack of support from the agent platform is that the capture of the Execution State of an agent is quite complex and, not supported by the Java Virtual Machine, which serves as the execution environment in many of the experimental systems (Aglets, Voyager, Jumping Beans, Odyssey, Mole, etc.). In this situation, it might be simpler and more efficient to create a custom solution, which only encodes the relevant state information in the agent's data and otherwise restart the agent in some well-defined initial state on each new agent platform. This is the solution that is proposed for creating agents in the Aglets system.

A closer look at the migration method used shows that its performance heavily influences the performance of the overall mobile agent system application and should therefore be optimized carefully. Despite the importance of efficient code migration, most existing mobile agent systems make no use of this aspect and are mainly concerned with moving the agents with no much consideration to the performance issue. The optimization of code migration is not trivial since for the migration of mobile agents in modular systems, not a single piece of code has to be transported, but a large set of classes that depend on

another in various ways. An efficient implementation of code migration therefore has to exploit the properties of the underlying mobile agent system.

2.4.4.2 Security

Mobile agent security is one of the crucial issues for the acceptance of mobile agent systems. It can be split into two broad areas and is concerned with the protection of the two primary components that we have identified: the agent platform and the mobile agent. The first involves the protection of host nodes from destructive mobile agents while the second involves the protection of mobile agents from destructive hosts.

[FGS 96] identifies three basic security principles that an MA system must realize:

- For the most applications of MA, the different participants cannot be assumed to trust each other.
- An agent critical decision should be made on neutral hosts (trusted environments).
- Unchanging components of the state should be encrypted.

The network or communication subsystem that interconnects the different agent platforms plays also an important role in agent's security. Indeed, a mobile agent has to be protected while traveling over an untrusted network (confidentiality and integrity). A simple method to achieve this is to use the Secure Socket Layer (SSL) [FKK96], provided that the mechanisms used for integrity protection support origin authentication for instance by using appropriate public-key signatures. Hence, the mobile agent can be signed by the agent owner to assert that it has not been tampered while in transit.

The protection of the mobile agent during its execution can be split into two distinct parts: protecting it from other agents on the same agent platform and from the agent platform itself which may want to scan the agent for information, alter the agents state or code, or kill the agent. The critical problem is that in order for the agent to run it must expose its data and code to the host environment, which supplies the means for that agent to run.

Currently most mobile agent systems do not explicitly address this problem, but make the assumption that the agent executor is a trusted principal that behaves correctly [FGS 96,

COL 94]. As we have noted in the beginning of this Section, this assumption is not always justified.

Attacks on host security fall into four main categories [COL94]:

- Leakage: acquisition of data by an unauthorized party.
- Tampering alteration of data by an unauthorized party.
- Resource stealing use of facilities by an unauthorized party.
- Vandalism: malicious interference with a host's data or facilities with no clear profit to the perpetrator.

These attacks can be prevented by using standard techniques such as cryptography, authentication, digital signatures and trust hierarchies.

However a mobile agent is a bit different from the traditional ways of infecting hosts, in that its code is executed by the host. Thus a mobile agent has automatic access to some of a hosts resources. With this level of access mobile agents can cause attacks by altering other local agents, propagating viruses, impersonating other users and mounting denial of service attacks.

Among the solutions to this problem, we can note the approaches used by Java applets, which can be executed in a Web browser. There are four different approaches [RDG98] to address this problem and a fifth approach results from a combination of the first two: code signing, sandboxing, firewalling, and proof carrying code.

In conclusion, although the protection of hosts can be achieved by using conventional and recently introduced mechanisms, a large question remains over the security and protection of the mobile agent from malicious hosts.

2.4.4.3 Communication

A communication model is needed so agents can communicate with the other entities in the computing environment. These entities include users, other agents (static or mobile), the host mobile agent environment and other systems such as CORBA. This service can be used as the foundation for many other services such as mobility, fault-tolerance, directory assistance, or agent management. In order to enable this local and remote communication, the agent platform might offer various communication protocols (e.g.,

TCP/IP, IIOP, SMTP, or HTTP), that can be used to implement different models of interaction, such as message based, event based, RPC, or Remote Method Invocation.

The most complex problem faced in this area is the communication with a receiver that does not have a fixed network address. There are several solutions to this problem, such as a registration server, which is updated whenever the agent moves to a new location and forwards invocations on the agent to its current location, or a proxy, which remains on the agent platform when the mobile agent moves on and forwards invocations to the new location. Many optimizations for this problem can be conceived, such as the ones discussed in [IDM91]. This issue and its optimization will be discussed in more details in the next chapter.

Once the communication problem is solved, the agent platform will require only a minimal support (i.e. encapsulate and make the service accessible to mobile agents under the constraints imposed by resource management) for the communication service since many of the communication services are already provided by the underlying operating system.

Because of the range of protocols used during communication it is probable that mobile agents will require more than one communication model.

A protocol is an implementation of a communication model. There exists a wide variety of protocols which range from the most basic, (email and RPC) to the more general and complex (KQML). Protocols also vary depending on the types of communicating entities (i.e. human speech or interaction through a modern GUI system with humans, KQML with agents).

2.4.4.4 Resource Discovery

Some solutions propose to use a directory facilitator service on a particular agent platform as a registry for mobile agents, which allows locally executing mobile agents to discover the presence of other agents on the same agent platform. However, due to the latency of a remote communication it is difficult to provide this same service to remote agents. This issue will be discussed in more detail in next chapter; our own solution will be exposed in chapter 4.

2.4.4.5 Resource Management

Agent requires access to a certain amount of low level system resources on an agent platform, such as CPU cycles, disk, memory, stable storage, and communication bandwidth. They may also require access to high level services such as querying databases and access to the particular data and facilities of this agent platform. The allocation of these resources must be guided by some policy of the agent platform to ensure fair distribution among requesting agents and to identify what kind of and how much resource should be made available to a mobile agent.

Once certain resources have been allocated to an agent, the system has to make sure that the agent is confined to his access rights. Most of the mechanisms required for this purpose are well-known operating system mechanisms. For instance, a clock interrupt, which limits the amount of time an agent can execute, can be used to regularize the agent's access to the CPU. For other resources not managed by the OS, the execution environment has to implement generic access control mechanisms and to enforce the access control policy defined for the agent platform.

The resource management is not very well covered by current agents systems, most of Java based system relies on low level resource management of the Java virtual machine (JVM). This area will be reviewed in chapter 4.

2.4.4.6 Data Management

A mobile agent can carry with it the data needed to accomplish its task in a given platform. It needs to store itself and its data in a persistent form to deal with system crash or any other failure problem. The support for fault-tolerance in a mobile agent system is interesting, since a mobile agent can be executing for a large amount of time gathering and collecting data and other valuable information. When a crash happens on the host where the MA is currently executing on, this information could be completely lost. Most existing agent platforms do not deal with this issue, which is an important problem for an actually deployed system.

The minimal goal for fault-tolerance in the context of the mobile agent paradigm would be to guarantee the survival of a mobile agent and its data despite the possibility of hosts and the corresponding agent platforms crashing.

Most of the existing systems don't specify how they manage their data and access rights. Data management is important to allow and help the system to be more robust and fault tolerant; it is a factor that would affect the overall performance of the system. This issue will be discussed in more detail in chapter 4 and we will present our own solution to palliate this problem.

2.4.4.7 Heterogeneity Support

Concerning the hardware heterogeneity, the agent platforms is responsible of hiding all the underlying infrastructures and environment from the mobile agents, so that a mobile agent can be executed on any suitable hardware. This can be done using some interpreters for high-level languages, such as Tcl, Perl, or Python, as well as Java virtual machines for agents based on Java byte-code. Most MA systems choose Java as an implementation language due to the platform neutral, ease of network programming and its constantly evolving security model.

The heterogeneity of mobile agent systems was often neglected. It would also be desirable to hide this heterogeneity, in order to enable agents from different mobile agent systems to inter-operate, as it is the case for open systems. Various efforts are underway to standardize the mobile agent paradigm to allow inter-operability between open system such as the specifications emerged from the foundation of intelligent physical agents [FIP98, FIP99, OMG98b].

2.4.4.8 Agent Execution

The basic task of any mobile agent system is the reception and the execution of mobile agents. That is, the code of a mobile agent has to be executed in the context of the data and facilities available at this agent platform, which was the original reason for the migration of the mobile agent. The execution of an agent's code is accomplished in an agent execution environment (AEE) that is an integral part of the agent platform design. It also contributes to reduce the size of mobile agents since they do not need to carry the code or any other facility which they can be sure to find on any visited agent platform.

The execution environment has to execute the code, supervise its access to the allocated resources (this is a very important issue related to resource management), and provide the platform resources. Therefore the agent execution has to be closely coupled with most of the other services such as resource management, heterogeneity support, security, and data

management. The agent execution is a very basic component that must be provided by the agent platform, which in turn may rely on lower layers for additional support.

2.5 The Use of Mobile Agent

In this section, we will first examine whether the mobile agent paradigm provides advantages over the classical client-server paradigm [CHK95] and then illustrate briefly what particular applications can exploit them.

2.5.1 Advantages

The advantages of the mobile agent paradigm as compared to the classical client-server paradigm can roughly be organized into two categories:

1. Reduction of network traffic, and personal assistance (server customization)
2. Decoupling of the interacting principals, which provides support for mobile users, robust handling of partial failures, asynchronous computations, support for heterogeneous environments, and real-time interactions.

The reduction of the overall network traffic is an often-cited reason for promoting mobile agents. This new approach obviates the need for costly network connections between remote computers and overcome low bandwidth limitations [URL4].

The support for personalizing server behavior allows a mobile agent to customize the offered services to his particular needs or even to create new services that previously had to be installed by the service provider. For instance, in the Intelligent Network (IN) domain mobile agents are proposed as a way to personalize the behavior of network entities (e.g. switches) by dynamically supplying new behavior.

The real-time interaction with entities is an obvious consequence of mobility. A mobile agent can interact with remote components directly, without having to suffer from a communication delay due to an intermediate network. An example of this is the control software, for an ATM switch or a safety system in a nuclear installation, which requires immediate responses to changes in their environment [KRA97]. A remote control across an intermediate network will incur significant latencies that are intolerable for critical

situations. Mobile agents can palliate to this problem by moving to the controlled entities and process the directives from the central controller.

It is in general hard to determine where and how to react in order to recover a distributed application in a consistent state in the case of system crash or slow response. The ability of mobile agents to react dynamically makes it easier to build fault tolerant behavior, especially in a distributed client-server applications.

The support of asynchronous and autonomous computing means that tasks can be encoded into mobile agents and then dispatched so that the mobile agent can operate asynchronously and independent of the sending program. Thus, the user's machine will not be loaded with the task of the mobile agent and will be free to work on other tasks. This becomes very interesting in the case of mobile device (PDA or mobile phone), which are rather resources limited computing devices.

The support for mobile clients results from the property of asynchronous interaction and the reduction of overall network traffic. Mobile users who access the network may not be able or willing to remain connected for the duration of a particular service provision due to their resource limited computing devices, and low bandwidth connection. Mobile agents offer the user the possibility of dispatching a mobile search agent onto the fixed network, disconnecting, then reconnecting some time later (possibly upon receiving an alarm from its mobile agent), to collect the results of the search.

Network computing is generally heterogeneous, from both hardware and software perspectives. As mobile agent systems are generally computer and network independent (depending only on their execution environment), they support transparent operation and provide optimal conditions for seamless system integration.

2.5.2 Applications

Searching for the killer application for mobile agents might be a waste of time. There are no mobile agents applications (up to now), that cannot be realized using the traditional methods or concepts (client-server). Nevertheless, there are plenty of applications that can take great advantages and benefits from this new technology.

The first commercial application was Sony's Magic Link PDA or Personal Intelligent Communicator (PIC) using General Magic technology (Telescript) [URL5]. Essentially, it assists in managing a user's e-mail, fax, phone and pager as well as linking the user to Telescript-enabled messaging and communication services such as America Online and AT&T PersonaLink Services. The latter, for example, can carry text, graphics and sound. Magic Link operates through the Magic Cap software platform, and a Magic Cap user can send executable agents (Telescript processes) via e-mail through the network.

France Telecom, who is a member of the General Magic Alliance, has developed some services based on Telescript. In one of their prototypes, they have used mobile Telescript agents to integrate railway ticketing and car renting services, and the prototype is able to propose an optimal solution depending on price and time.

IBM plans to launch their ICS system which uses mobile agents for providing a communication super-service that is capable of routing and translating communications from one service and medium to another, e.g. mobile to desktop, PDA to fax, speech to text, etc [URL3].

The IBM Agent Meeting Point for Mobile Communication is another application that aims to design an Agent Meeting Point (AMP) [CGH 95] where mobile agents can interact with each other, and support various services.

The Perpetuum Mobile Procura Project concerns itself with advanced network management through the use of mobile agent technology. Its goal is to overcome the problems of legacy management systems, to explore new intelligent distributed management techniques with mobile agents and finally to deliver a Java-based platform for both real and simulated networks management. Mobile agents in this system are called Netlets [COL 98]. They are supposed to support service by delegation, installable/extensible/modifiable services, fault management (diagnosis/recovery), plug-and-play networks (auto-provisioning), self-repairing network.

Mobile agents are also proposed for electronic commerce applications [CMA96]. They can be used to build electronic markets where they embody the intentions, desires, and resources of the participants (human users) in such markets [MIT 97]. In this environment MA perform various tasks, such as searching for offers, negotiating terms of

an agreement, or even purchasing goods or services. Electronic commerce applications clearly benefit from the mobile agent paradigm especially in the context of users who conduct routine purchases while they are on the move or who formulate complex requests that require the mobile agent to visit many different service providers and to scan large amounts of data. The InAMoS is another project [URL6] that attempts to bring electronic marketplaces to the mobile users that are represented by mobile agents.

The use of mobile agents in searching for information (which is an application that has been deployed on the WWW although with non-mobile agents) can be seen as a special case of electronic commerce, where the good that is sought for, is information.

StormCast [URL4] is another project that uses the MA technology. The project has been running for many years and it provides critical weather information to the meteorological office in Norway. The key idea is to increase access to the meteorological data, thus, mobile agents are created at the client side (in a WWW browser) and sent to servers, which can take their content and retrieve the necessary information from the vast meteorological data stores.

Another interesting use of this technology is in communication contexts, as a user agent. The roaming users often need a landing port in the network in order to allow others to get in contact with them. Instead of being directly connected to this address, the user can dispatch his agent to this fixed address, which will negotiate with callers on how to proceed. The user agent holds the current location (device address) where the user can be reached and depending on the user preferences decide to forward the call to some asynchronous communication device (e.g. voice mail or mail box) or to route the call to the current registered network address of the user.

Other often-cited applications for mobile agent systems include secure brokering, personal assistance, parallel processing, information dissemination, workflow applications and groupware and network management services.

Chapter 3

Related Work

3.1 Introduction

In the previous chapter, we have introduced the mobile agent paradigm and identified the problems concerning the management of agent, data, resources and communication. Many architectures have been developed which, while not directly addressing the task of agent management, do help to provide a framework within which such a system could be developed. The following sections detail the most prevalent systems currently available.

3.2 Jumping Beans

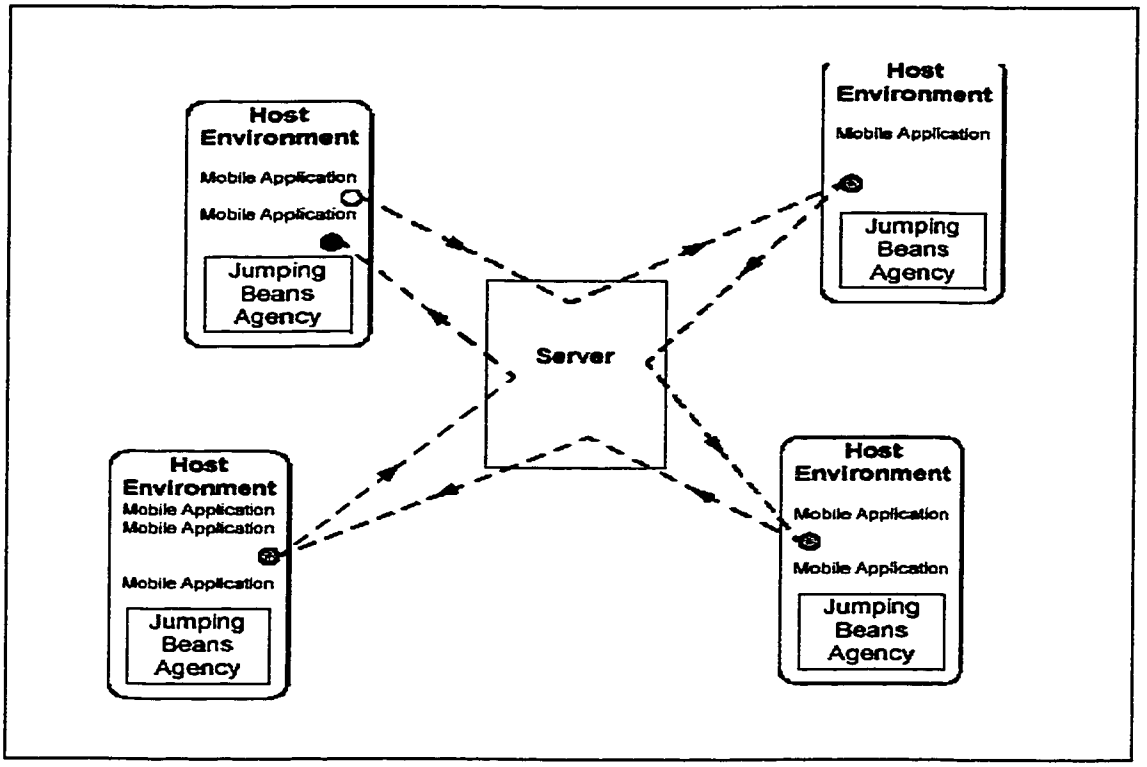


Figure 3.1: Jumping Beans Architecture

Ad Astra Engineering, Inc. of Sunnyvale California, is a Java technology development company with large experience in Java technology-based solutions for enterprise environments. They have been presenting the Jumping Beans technology as a Java framework for developing and deploying mobile applications which can move themselves around a network during their lifetime, and it is considered as the first implementation of Mobile CORBA [JUM99].

Jumping Beans use Client /Server architecture. Each host machine in the environment has client software installed and running, which facilitates the receipt and dispatch of mobile applications, and enforces security. The Agent Platform integrated on each host machine is called an "agency." The management system of Jumping Beans is centralized on the server. A management console allows the administrator to set and edit security privileges, destroy, dispatch, control mobile applications, and monitor all kind of activity in the system.

The Application Programming Interface (API) of Jumping beans offers a framework for moving applications through primary elements and services [URL7].

The client (i.e. agency) resides on the host environment and acts as a "daemon" listening to the TCP-IP port to receive mobile applications (MA). In addition, it enforces security and provides Jumping Beans' services to both MA and to resident host software. The Java interface for MA provides callbacks for certain events including creation, dispatch, arrival, destruction, and others. It should be implemented by all mobile applications wishing to access to Jumping Beans' services that are available trough the Mobile Application Context object (MAC), which is assigned to each MA upon its creation.

The itinerary for the MA is constructed from instances of a predefined class -- ItinerayNode class. Each object forms a node in a tree, and the tree forms the path that will be followed by the MA. The itinerary specifies, in order, the hosts that the mobile application will visit. It is possible that an itinerary have a fork so that the mobile application can dispatch from one host and arrive at multiple hosts. In this case, the MA is cloned to different copies that are distinct and different from the others. The itinerary can be seen as a tree of nodes; each node represents a destination along the itinerary and

can have one parent, and zero or more children. Multiple children coming from a single node represents a fork in the itinerary.

The Jumping Beans itinerary is fully programmable and editable, however this right can be denied according to security privileges so that the MA can "lock down" its own itinerary, to ensure that it will follow a prescribed path; it is also illegal to edit the portion of the itinerary that represents the past itinerary --the hosts already visited.

Jumping Beans security is based on Access Control Lists (ACLs), multi-jump security, user logons, public/private keys, and digital signatures. During its execution, security is enforced by the Jumping Beans client using Java security capabilities. During transport, security is enforced jointly by the server and the clients as the mobile application passes through the server on each jump. Obviously, this is great for security but the authors say that it also helps scalability because in a peer-to-peer solution a mobile object needs to leave a sort of 'follow-me' forwarding address at each site it has passed through. (Ad Astra call it a scent mark). Therefore if object A on host PA want to send a message to object B, which started at host PB but is now at host PF, it will have to launch a chain of messages to reach the target mobile object thus consuming network traffic. However, some development system [BPM96, LWS98] do essentially perform this 'follow-me' approach but once the new destination is discovered the source object/host is essentially updated with the new location thus avoiding any more 'chain searches' (up to host PF).

3.2.1 Creation and Identification

Jumping Beans places few requirements on a mobile application. The primary requirement is that it should be serializable (using Java serialization). Other than that, a mobile application can be composed of any class and resources required by the business logic.

There are three entities that should be identified in the Jumping Beans system: the MA, the agency and the server.

Each Jumping Beans server has a unique ID that contains:

- The server's name.
- The server's IP address.

- The server's port.
- The server's host name.
- The creation time of the server.
- A random ID: this helps to ensure uniqueness of the ServerID.
- The server's global address.
- The version of the server software.

Each agency is assigned an AgencyID when it is created. The agency ID contains this information:

- Agency Name
- Agency Creation Time
- The ServerID of the agency's server.
- The host's name
- The host's IP address
- The agency's IP port
- A random ID: this helps to ensure the uniqueness of the AgencyID.
- The Jumping Beans version of the agency software.

Each MA is assigned a unique MobileAppID at creation. It is composed of the following fields:

- The name of the mobile application.
- The AgencyID of the agency which created the mobile application.
- The time when the mobile application was created.
- A serial number for the mobile application: Each agency maintains a serial number to assign to each mobile application it creates.
- A random ID: This is a random number assigned to the mobile application to help ensure uniqueness.

- A Clone ID: This indicates the number of hops taken by the mobile application, and the itinerary branch taken on each hop.

3.2.2 Management and Tracking

The data management as well as the tracking is straightforward resulting from the centralized approach adopted by the Jumping Beans.

The Jumping Beans server acts as a management console from which the entire Jumping Beans system can be monitored, managed, and controlled. The entire sensitive information about the mobility system is saved on the Jumping Beans server that keeps track of all MA in the system.

Concerning the issue of persistency, a lightweight persistence system and a Java Database Connectivity (JDBC) interface is provided with the package. The current Jumping Beans server JDBC interface was developed using Microsoft SQL Server version 6.5. It is assumed that a typical execution environment will already have in place its own concepts and mechanisms of data storage. The persistence subsystems for Jumping Beans are defined entirely in terms of Java interfaces; any implementation of these interfaces can act as persistence system for Jumping Beans. That is a developer needs to implement certain Java interfaces to integrate a custom persistence system into the Jumping Beans client, and other interfaces to integrate a custom persistence system into the Jumping Beans server. For migration, Jumping Beans packages the mobile application in a canonical form that is independent of the underlying datastore. Therefore different hosts can use different datastores and still exchange mobile applications. Similarly, a server and a client who implement different datastores are fully interoperable.

The easy management of a centralized system was one of the reasons that privilege the adoption of the Client/Server architecture, however there is a lost of flexibility, fault-tolerance and robustness, in other terms all what confronts the centralized systems with the distributed and replicated ones.

3.3 Aglets: IBM Tokyo Research Laboratory

Aglets [URL8, CLA 96] are Java objects that can move from one host on the network to another. That is, an aglet that executes on one host can stop its execution, dispatch to a remote host, and start execution again. When the aglet moves, it takes along its program code as well as the states of all the objects it is carrying. A built-in security mechanism makes it safe to host untrusted aglets. The Aglets Framework is entirely written in Java to ensure maximum portability.

Using the Aglet technology is one among the different ways to build user-defined agents that inherit the default properties and functions of mobile agents. This framework offers the primary and basics services for agents that include a globally unique naming scheme for agents and a travel itinerary for specifying complex travel patterns with multiple destinations and automatic failure handling.

The support for communication and collaboration is done via the white board mechanism (allows multiple agents to collaborate and share information asynchronously) and message-passing scheme that supports loosely coupled asynchronous as well as synchronous peer-to-peer communication.

For migration, an aglet has to suspend its execution, serialize its internal state and Java bytecode into the standard form and then move to the destination. When an aglet is dispatched, cloned, or deactivated, it is marshaled into a byte array, then unmarshaled from it later. The standard Java ObjectSerialization mechanism is used for this purpose. All objects within the aglet object-graph need to implement the java.io.Serializable interface or java.io.Externalizable interface; otherwise they must be referenced as transient. On the receiver side, the Java object is reconstructed according to the data received from the origin, and a new thread is assigned and executed. A network agent class loader allows the agent's Java byte code and state information to travel across the network, and an execution context provides agents with a uniform environment independent of the actual computer system on which they are executing.

Some primitive causes an aglet to be stored in secondary storage and to sleep for a specified number of milliseconds. After the given time has passed or another program has

requested its activation, the aglet is activated within the same context where it was deactivated.

Recently, the Aglets introduce the notion of Pattern-based design and development. These usage patterns describe common dual relationships between agents such as Master-Slave, Messenger-Receiver, and Notifier-Notification and are represented by a number of classes that can be used as templates by the agent developer.

3.3.1- Communication and Management

The Aglets architecture is composed of two basics layers: The Aglet API and the Communication API as shown in figure 3.2. The Aglet runtime layer is the implementation of the Aglet API, and defines the behavior of the API components, such as AgletProxy and AgletContext. The framework provides the basic functions such as creation, management, migration and security features. The communication layer is primarily responsible for transferring a serialized agent to a specific destination and receiving it. It also supports agent-to-agent communication and facilities for agent management.

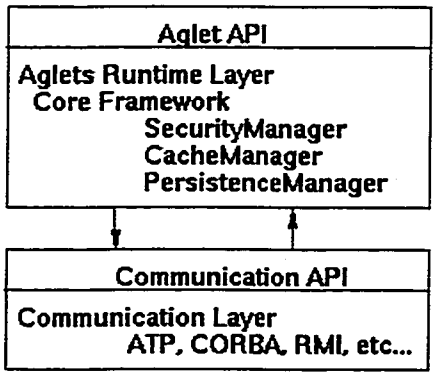


Figure 3.2: The architecture of Aglets Framework

Aglets API

The Aglets runtime layer consists of a core framework and other subcomponents (security manager, cache manager, etc). The core framework provides the following mechanisms fundamental to aglet execution:

- Serialization and deserialization of aglets (using Java mechanisms);

- Class loading and transfer;
- Reference management and garbage collection.

Communication API

The Aglets runtime itself has no communication mechanism for transferring the serialized data of an aglet to destinations. Instead, the Aglets runtime uses the communication API that hides the heterogeneity and the communication between agent systems. This API defines methods for transferring agents, tracking agents, and managing agents regardless to the agent system or the protocol used.

The communication API used by Aglets runtime is derived from the OMG standard, MASIF (Mobile Agent System Interoperability Facility), which allows various agent systems to interoperate. This interface abstracts the communication layer by defining interfaces and providing a common representation in Java that conforms to the IDL defined in the MASIF standard [OMG98b].

Aglets use the Agent Transfer Protocol (ATP) [ATP 97] as the default implementation of the communication layer. ATP offers a simple and platform independent protocol for transferring agents between networked computers. It is modeled on the HTTP protocol, and is an application-level protocol for transmission of mobile agents. While mobile agents may be programmed in many different languages and for a variety of vendor specific agent platforms (consisting of virtual machines and libraries), ATP offers the opportunity to handle agent mobility in a general and uniform way. To enable remote communication between agents, ATP also supports message passing.

An agent system must have a stub class for each protocol it supports. It is the agent system's responsibility to instantiate and manage the stub objects. The latest beta (as of this writing) version of Aglets supports two protocols, ATP and RMI. An application or client willing to send a request to the destination can then get and use the stub object associated with the given protocol. On the server side, an aglet server has an implementation of MAFAgentSystem that actually handles the requests. Furthermore, a server may have one or more daemons to accept requests from a sender and may support multiple protocols by having multiple daemons to handle each protocol.

An aglet object is represented internally as an AgletRef object. Each aglet is associated with a MessageManager to control incoming messages, a ResourceManager to manage resources consumed by the aglet and to manage its related resources such as security information. It implements also the abstract methods and functionality defined in Aglet class.

The Aglets framework has a reference table to store the mapping from the AgletID to the actual aglet instance. An AgletRef object is created and inserted into this table when an aglet is created or received, and removed from it when the aglet is dispatched or disposed.

Concerning resource management, the ResourceManager object, which is allocated for each aglet, is responsible of managing all threads and windows created by this aglet. When an aglet is dispatched, deactivated, or terminated, these two resources are immediately stopped and claimed as available. Other kinds of resources such as file and socket, however, are not handled by this manager. It is left to the garbage collector to decide whether or not such resources will be released, and if so when. Therefore, the programmer must release them manually to ensure their immediate release.

3.3.2- Naming Service for Aglets

The naming services for aglets aims to provide location transparency and message delivery. The integration of these services in Aglets is done by implicit use of different types of proxies. Each aglet is associated with an AgletProxy, which is responsible of delivering the messages to the receiver Aglet. There are three types of proxies: the forwarder, the locator and the plain.

Once an aglet is created, it is assigned a forwarder proxy. Before moving to a new destination, the aglet creates a trail at its current host specifying its future location, this information is used by the forwarder proxy to later forward messages to this aglet.

The locator proxy is used when the aglet registers with a naming server associated with the current aglet server. This locator maintains the address of the naming server that provides the current location of the aglet assuming that this aglet updates its location with this naming server each time it is dispatched to a new destination.

In case where the location of remote aglet is well known then the Plain proxy is used to simply transfer messages to this aglet.

3.3.3 Agent Viewers

A desktop aglet viewer is also integrated with the framework to enable users to mange their agents. Aglets and locations are represented as icons. It allows creation, dispatching and termination by dragging and moving aglets icons to location or trash icons. Aglets would extend this tool to support manipulation and display of itineraries and remote communication.

3.4 D'Agents

D'Agents is a mobile agents system from Dartmouth college. The first release of D'Agents, D'Agents 1.1, is better known as Agent Tcl. Since the system was extended to support Java and Scheme languages, its name was changed from Agent Tcl to D'Agents [URL9]. In this model, a mobile agent is seen as a program that can be written in Tcl, Java or Scheme and accesses features that support mobility via an API implemented on each server (host machine).

The system has four components, transport mechanisms, an interpreter for each supported agent language, the agents and the server that runs on each host machine.

The main component is the server that provides the entire mobile agent specific services such as state capture, transfer, communication and interaction with the machine resources. The agent migration is performed by calling a single instruction, which captures the agent's state, which consists of a global variables table, the names and bodies of procedures call frames and a command stack, and send it to the server on the destination machine. The server starts up an appropriate execution environment (e.g. Tcl interpreter for an agent written in Tcl), and starts the agent execution from the point it left off.

The support for Java and Scheme is still under development, the primary language is Tcl and the primary transport mechanism used is TCP/IP. Since standard Tcl does not support transportability, an extended version was developed to allow scripts migration and

distributed Tcl scripts communication. All transportable agents must run under the extended Tcl shell.

The transport layer, is an interface to each available transport mechanism with support of asynchronous I/O. The server runs on every host machine and implements all the functionalities that the agent needs. It provides the low level inter agent communication via message passing and binary streams, security mechanisms (i.e. authentication of agents coming from other hosts), receiving and running the authenticated agent in its appropriate execution environment.

The third level represents the different execution environments that are supported –one for each language. Since all the languages are interpreted, the execution environments are just interpreters, that is a Tcl interpreter, a Scheme 48 interpreter and the Java virtual machine. All these interpreters shared the same C/C++ library that forms an interface between servers and the execution environments.

Finally, the fourth level of the architecture consists of the agents themselves that run in their appropriate execution environment and use the server facilities for migration and inter agent communication. There are two kinds of agents that co-exist in this layer: the mobile agents and stationary agents; the latter reside on a single host machine and are responsible for providing specific services to either the user or the agents. Such services include navigation, high-level communication and resource management.

3.4.1 Agent Identification

Unlike most mobile agents systems, the mobile agent in D'Agents doesn't have a fixed identifier, that is its name changes while migration depending on the controlling server where it is executing [KGR 96]. The controlling server plays also the role of the agent home. Whenever the agent migrates to a new controlling server it is assigned a new and unique numeric id with the option to add a unique symbolic name if desired. Hence, the mobile agent identifier has 4 fields:

Machine name: the full Internet name of the machine, on which the controlling server is executing e.g. simpson.cs.dartmouth.edu,

Machine IP: IP address of the machine 129.170.192.98,

Symbolic name: the symbolic name of the agent e.g. search_agent,

Numeric id: the numeric id of the agent e.g. 10.

This agent identification has substantial redundancy, given that the machine name and its IP address as well as the symbolic name and the numeric id play the same role.

Concerning the issue of agent hierarchies that occurs when agents create other agents, D'Agents introduces the notion of *root identification*. Thus, the *root agent* is the top-level agent for all the agents in its hierarchical tree. The root server is the controlling server of the root agent and finally the *root identification* is the 4-element identification of the root agent.

All this relevant information is stored in a global read-only array called *agent* that is available and encoded inside the mobile agent. This array is updated automatically as the agent moves through the network and can be accessible via Tcl procedures and commands that create local references to the array *agent*.

The process of registering consists of acquiring a controlling server and assigning a symbolic name according to the following scenario:

The root agent issues a request to acquire a controlling server by specifying the name or the IP address of the machine where the desired server is running, This information is available in the array *agent*. The request specifies also the waiting time for receiving a response from the server. On the reception of the request, the server assigns a numeric id to the agent, (initially the agent has no symbolic name), and updates its internal table of registered agents. As result of a successful registration, the server returns the new 4-element identification of the agent and fills in the root and local sections in the agent array. The controlling server for the child agent is automatically the server to which it was submitted. Afterwards, the agent registers its symbolic name with the server that updates the appropriate sections in the agent array and returns the 4-element identification back to the agent if no error happens (e.g. another agent using the same symbolic name). An agent may use the command *agent-root* to become a root agent, in which case, the root agent information is overwritten with the new information unless the agent wants to keep the old information in a separate variable.

The child agents are automatically removed from the server internal tables upon their termination; however, the root agent uses the explicit *agent-end* command to notify its controlling server that it is finished with its task. The system offers also the possibility to "forcibly" remove an agent from the server internal table and even force its termination. There are no security checks on this command; an agent may use it to remove any other agent from the server tables as long as it knows its 4-element identification.

3.4.2 Communication

Both local and global communication mechanisms use message-passing facility, the format of data can be either arbitrary strings or binary data.

An agent can use the agent-send command to send a message to an agent by specifying the destination which can be either the 4-element identification or any of the 2-element that include either the machine name or the IP address of the machine, and the recipient's name or ID. On the other side, the receiver agent can use a blocking, non-blocking or timed command to receive a message by specifying the type of messages it is waiting for or able to process, this notation must be agreed between all the entities in the system. The system offers also the possibility to use events for asynchronous notification of important occurrences. It has the same syntax, semantics and error messages as message addressing.

D'Agents introduces also the notion of "Meeting", it consists of a direct connection between two agents and it is more efficient than message passing. The agent that asks for a meeting sends a message by specifying the receiver identity, and waits until the acceptance or rejection of the request, when the receiver accepts the meeting, a socket is opened between the two communicating peers and the socket descriptor is returned back to the issuer of the request. Afterwards the communication takes place based on the standard reading, writing and closing mechanisms of sockets and TCP/IP transport layer.

3.4.3 Migration

An agent can migrate to another machine where the server is running, and can eventually clone itself or even create a child agent in the new environment as long as it has the script (i.e. agent code).

The migration is performed through a specific instruction that identifies the destination machine. This leads the agent to stop its execution, capture its internal state and send the internal state to the destination server. The server restores the agent's state and assigns a new local identification to the agent that resumes its execution at the point it left off. The local section in the agent array is also updated with the new identification. The agent lost all the connection upon its migration and cannot regain it unless there is a stationary agent that will take care of these connections and communicate with the transportable agent.

3.4.4 Agent Tracking

The first release of D'Agents (Agent Tcl) did not address the issues of privacy (protecting sensitive information of the agent), neither agent debugging or tracking. As a matter of fact, the communication partner was addressed just by its symbolic name, for instance *clacul_agent*, *search_agent*, in the desired machine. In other words, the agent is communicating with an unknown agent that can perform the specific task but it is enable to continue communicating with a peer who migrate to another machine.

An agent debugger (AGDB) [HKT97] was introduced for Agent Tcl. It is composed of two parts: a debugger agent and a library containing procedures an agent uses to communicate with the debugger agent. The debugger agent can monitor as many agents as the user chooses. The mobile agent must include the library component of AGDB as well as a password and other information that will be used for authentication and processing.

The AGDB uses the agent's machine-name and numeric-id to communicate with a specific agent. For this reason, it is primordial to keep track of an agent's symbolic name. After each successful *agent_name* request, the agent informs the debugger of its new name and the debugger notifies the user of the end result either failure or the new name. AGDB offers the user five options concerning migration: ignore, track, break before, break after, and break before and after. The track option causes the debugger to notify the user of all agent moves, regardless of their outcome. The track option is the default for root agents. The "ignore" option causes the debugger to obtain but suppress (from the user) information about agent moves. Although the user is not informed of the agent's

jumping activity, the debugger must still keep track of the agent's location. The break options are an extension of the track options in that they cause the agent to initiate a breakpoint just before or after (or both) it makes the migration call. The user is able to switch jump options before running an agent or during any breakpoint.

A local tracking was also introduced to enable the tracking to be used in possibly isolated networks. Usually, an agent informs the tracker of every move via message passing over the network. In case of local tracking, the agent will only send data to a tracking daemon on its own machine and no network connectivity is required for the purpose of tracking. All the actions will be logged on their respective machines, and can later be read for display by the tracker.

3.5 Grasshopper

Grasshopper is a mobile agent platform built on top of distributed processing environment. It allows software agents to move between different systems and to execute various tasks in the process. It aims to seamlessly integrate different platforms or organizations to create open systems that fulfil the sophisticated requirements of today's inter-linking world. Grasshopper provides the option to use both agents and client/server computing in one application and is completely implemented in Java. It is also considered as the first mobile agent environment, which is compliant to the industry standard supporting agent mobility and management (OMG MASIF) [OMG98b]. This compliance ensures compatibility with other agent environments or applications based on the same standard.

3.5.1 The Grasshopper Agent Environment

The Grasshopper environment consists of several Agencies and a Region Registry remotely connected via a selectable communication protocol (e.g. RMI-IIOP). Several interfaces are specified to enable remote interactions between the distinguished distributed components. Apart from Grasshopper-specific interfaces, the MASIF-compliant interfaces (i.e. MAFAgentSystem and MAFFinder) [OMG98b] are provided to enable interoperability between the Grasshopper platform and (MASIF-compliant) agent systems from different vendors.

3.5.2 Grasshopper Agencies

Agencies represent the runtime environments (i.e. Agent execution service) for mobile agents. Each agency runs on its own Java virtual machine and is remotely accessible via external interfaces; both the MAFAgentSystem interface and the Grasshopper-specific AgentSystem interface are provided. Agencies can be grouped to one region represented by one region registry. The agency provides the basics capabilities for the execution of agents. Hence, Agents can access the agency for retrieval of information about other agents, agencies or places, or in order to move to another location. Users are able to monitor and control all activities within an agency by accessing the core services, i.e. Communication Service, Registration Service, Transport Service, Security Service, and Management Services. Optionally, a Graphical User Interface can be activated to facilitate user interaction.

3.5.3 Registration Concept

The registration service of each agency is connected to the region registry, which maintains information about agents, agencies and places for an entire region. The region registry is a Java program running on its own Virtual Machine and is responsible for locating other agents, services, places, or agencies while the user (administrator) is able to track agents in the entire distributed environment. A region registry also provides interfaces that enable remote access, such as MAFFinder (MASIF-compliant) and the Grasshopper-specific Region Registration. Therefore, each agency is able to know about all currently hosted agents and places, both for internal and external management purposes as well as for delivering information about registered entities to hosted agents.

3.5.4 Communication Service

The communication service supports both asynchronous and synchronous messaging and method invocation. Agents can contact each other independently of their locations. This is achieved by means of proxy objects i.e. stubs that are directly accessed by the client. The stub forwards the call via the ORB to the remote target object; it is comparable with the client stubs used by CORBA implementation. Currently, the following communication protocols are supported: IIOP, RMI (with/without SSL), and plain socket (with/without SSL).

3.5.5 Management Service

The management services that are developed are derived from the MASIF standard and they enable the user to monitor and control agents and places of an agency. Management services include:

- Creation, removal, suspension and resumption of agents and places
- Information retrieval about specific agents and services
- List of agents residing in a specific place
- List of all places of an agency
- Configuration management for system specification, trace, security, and communication properties.

3.6 MASIF Standard

Among the most important standardization in the area of Mobile agents is the Object management Group (OMG) with its Mobile Agent System Interoperability Facility (MASIF) specification. The MASIF standard has been adopted as a new OMG technology in February 1998. It comprises several important aspects such as agent management, mobility, naming and tracking.

The agent management allows an agent system to control agents of another agent system. MAF addresses this interoperability by defining interfaces for actions such as suspending, resuming, and terminating agents. Agent tracking permits the tracing of agents registered with MAFFinders (i.e. Naming Services) of different agent systems. Agent communication is omitted from MAF since it is extensively addressed by CORBA as object communication. However, the agent transport is addressed by MAF by defining methods for receiving agents and fetching their classes. This interoperability is not simple to achieve and requires a certain amount of cooperation between implementers of different agent systems.

This MAF submission describes two CORBA object interfaces: MAFAgentSystem and MAFFinder. The MAFAgentSystem interface defines agent operations, including receive, create, suspend, and terminate. The MAFFinder interface defines operations for

registering, unregistering, and locating agents, places, and agent systems. Figure 3.3 represents CORBA services that are related to mobile agent's technology.

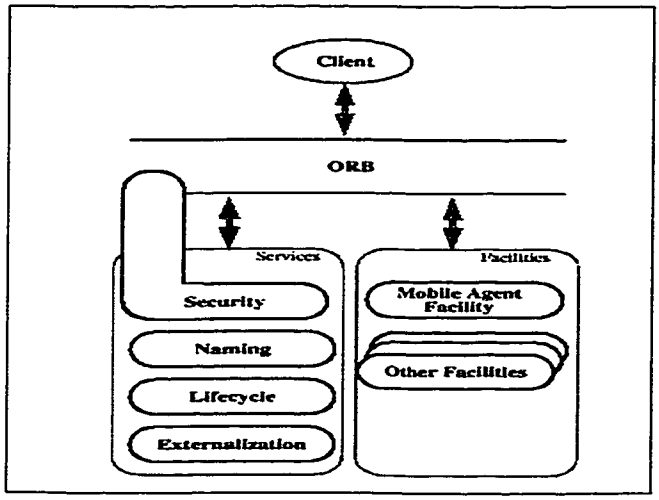


Figure3.3: CORBA Services and Facilities

3.6.1- Naming Service

Before going further and discussing the roles played by the two interfaces, we should define the basic service for naming ensured by CORBA that will be used to identify the different entities in the MA systems.

The CORBA Naming Service binds names to CORBA objects. The resulting name-to-object association is called a *name binding*, which is always related to a *naming context*. A naming context is an object that contains a set of name bindings in which each name is unique.

Each time a mobile agent moves to a new location, a CORBA object reference (IOR) is created. It contains, among others, the name of the host on which an object resides and the corresponding port number. In this case, the IOR that is kept by the accessing application becomes invalid. CORBA standard proposes three different solutions for this problem [OMG98b, OMG98a].

Name Structure:

In order to uniquely identify MAF agent systems and agents, the following components are used:

- Authority: defines the person or organization which the agent or agent system represents. The authority of the agent is equivalent to the principal of the agent's credentials if the CORBA security is used.
- Agent System Type: defines the type of an agent system. In case of agent identification, this component represents the type of that agent system where the agent has been created and that generated the identity of the agent.
- Identity: distinguishes agent systems or agents, respectively, which have the same authority and the same agent system type values.

The identifiers assigned to parameters such as agent system type and authenticator are unique across all implementations of MAF. This is done by defining some global entity to assign and maintain these parameters. In the MAF specification, the OMG is the naming authority for the MAF parameters that require global uniqueness.

3.6.2- MAFFinder

The CORBA services were designed for static objects. For instance, the CORBA naming service might not handle all cases when applied to mobile agents. Therefore, in the context of mobile agent, a MAFFinder interface needs to be declared. The MAFFinder acts as an interface for a dynamic name and location database of agents, places, and agent systems.

The MAFFinder is a naming service that may be shared among regions. Before a client can request the MAFFinder to find an object, the client must obtain the object reference to the MAFFinder. To get the object reference, the client uses either the CORBA Naming Service or the method AgentSystem.get_MAFFinder().

The MAFFinder interface provides ways to locate agents, agent systems, and places that supports a wide range of location techniques:

- Find every agent system in the region, and then send an agent to travel through every agent system to find the agent.

- Whenever an agent leaves an agent system, it leaves a mark that says where it is going. Therefore, an agent system can always follow the logs to locate that agent.
- Every agent registers its current location in a database. This database always has the latest information available about an agent's location
- Register all the stationary places only. An agent's location is registered only when the agent advertises itself.

Although the MAFFinder interface does not restrict implementations to a certain set of agent location schemes, it does assume that each agent system posses an underlying database structure that can support registering, unregistering, and locating agents, agent systems, and places.

3.6.3 MAFAgentSystem

The MAFAgentSystem interface defines methods and objects that support agent management tasks such as fetching an agent system name and receiving an agent. These methods and objects provide a basic set of operations for agent transfer. It includes the agent creation, fetching Agent classes, receiving agent, resuming agent, etc.

Chapter 4

Agent Management Framework: Concepts and Design

4.1 Goal and Motivation

Future management systems need be flexible and open to support communications services, which seamlessly covers different end-users and organizations. Also, end-users will require on-line management services, thereby placing requirements on the system capabilities for remote and efficient communication as well as on the service and network management systems. Such end-user management services are a key to the success of the information society.

Our goal is to build a distributed logical model for agent management and tracking. This model can be seen as a two distinct parts, the Agent Management System (AMS) and the Directory Facilitator (DF).

The Agent Management System is responsible for agent creation, agent tracking, information management, mapping between the logical agent name and the current physical address of the MA. It can be divided also into two main parts: The Interface that enables the user/application to interact with the agent environment or the agent itself and the services that are deployed to ensure the control, the dynamic mapping and to resolve name conflicts...etc.

One can envision system administrator managing mobile agents via standard operations. It should be possible to create an agent given a class name for the agent, suspend execution of an agent's thread, resume its thread, force its termination, track the agent moves, etc. Defining common management functions allows the administrator to manage from its management console the whole agent system.

The responsibility of the directory facilitator is to provide directory and referential services that can aid mobile agents in the discovery of relevant services which the agent

might need. Current systems include X.500 and CORBA Trader. New and up coming systems are LDAP and FIPA. Optionally, this module can also supply information about the state of the network (i.e. Network topology information services) and also the host sate. This information can help a mobile agent to make planning decisions based on the requirements of its mission.

We have adopted a decentralized approach at many levels. We believe the ultimate advantage of decentralization is scalability. A centralized system suffers from the drawbacks of single point of failure and potential long response time. A decentralized system availability and dynamic load sharing. However, the network should be large enough to take benefit and test the scalability advantages.

4.2 Overview of SHiP-MAI Platform

In this section, we present the overall system architecture [SHI 98, SHI 00]. Figure 4.1 represents the agent model.

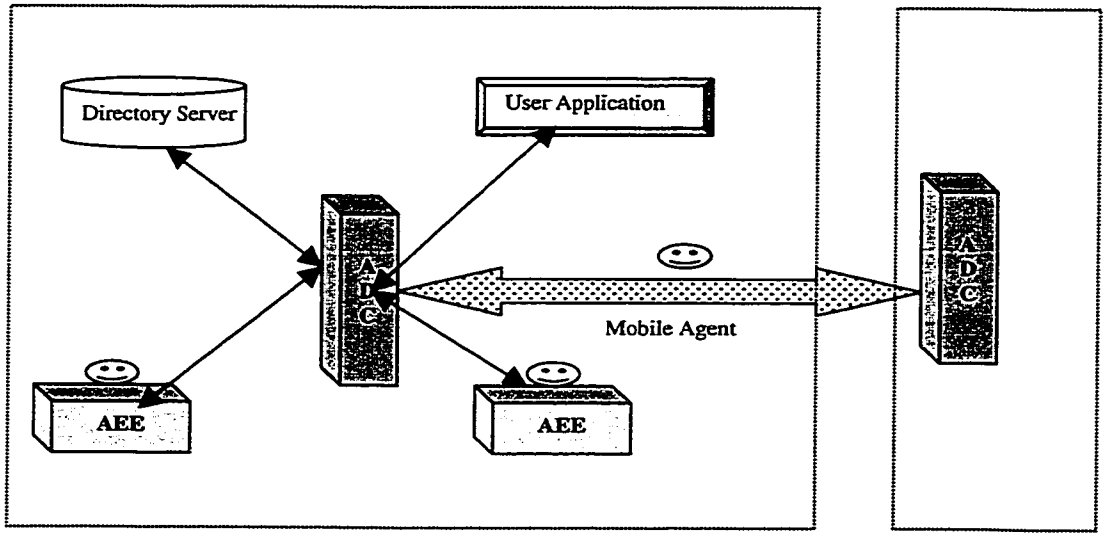


Figure 4.1: The Agent Model: SHIP-MAI

4.2.1 Terms and Definitions

Places: When an agent transfers itself, it travels between agent execution environments called places. A place is a context within an agent system in which an agent can execute.

This context can provide functions such as access control, execution, migration. On the following places and AEE refer to the same.

Serialization: A Java serialization form is used in SHIP-MAI platform.

4.2.2 SHiP-MAI Concept

The model for an agent-based system is mainly based on the concept of agents and locations. Unlike the earlier architectures, which consist of an agent server and agents migrating from one agent server to another, SHIP-MAI introduces hierarchy in the agent architecture. It divides the network into several domains composed from Agent Execution Environments (AEE) and introduces an additional element called Agent Domain Controller (ADC). In this model rather than combining all functions on the agent server, the basic functions for control and security are delegated to domain controller, whereas the AEE is responsible for agent execution, communication and a set of agent services.

Partitioning between control and execution simplifies the design and implementation of the mobile agent infrastructure and makes it more manageable and scalable. The ADC is motivated by two factors: 1) It reduces the amount of an agent charge 2) it provides a fault tolerance functions such as storage of agent's work result and agent's code.

Agent may migrate from one host to another, based on directives from the controlling ADC or on independent initiative. Depending on access rights and domain policies, agents may be restricted to hosts in trusted domain or may be allowed to execute in other foreign domains. There are two different scenarios for migration depending if it is an intra-domain or inter-domain migration.

- *Inter-Domain Agent Transfer*

When an agent wants to migrate to a foreign domain, it creates a travel request and send it to the its current ADC, as part of the travel request, the agent must provide naming and addressing information that identifies the destination place. The ADC verifies the agent's access rights and once verification is completed, it contacts its peer in the foreign domain. Based on policies agreed between these two entities and the destination domain's available resources, the request may be granted or denied. If the destination domain accepts the request it must fulfill the travel request (e.g. agrees to provide a fault

tolerance for the agent) otherwise it returns a failure indication to the agent. At this point, the current ADC responsible of the MA takes a decision based on domain policies and may decide to abort agent transfer and return failure indication to the agent or decide to provide fault tolerance for the agent at its home domain. When the destination domain agrees to the transfer, the mobile agent's state, authority, security credentials, and, if necessary, its code are transferred to the destination agent domain that become responsible for reactivating the agent, and then resuming its execution.

- *Intra-Domain Agent Transfer*

This is more straightforward than the inter-domain migration and deals with transfer of agents from one place to another in the same domain. The agent may migrate to another AEE inside the domain without the intervention of the ADC. The agent issues a travel request to the host, on which it is residing. As part of the travel request, the agent provides naming and addressing information that identifies the destination place. The AEE examines the request and decide to grant the request or not according to security credentials of the agent. If the request is granted, the AEE establishes a connection with its peer and start transferring the agent to its requested destination.

- *Security*

Each domain has a security manager that uses certificate technology to authenticate agents, AEE and ADC. It acts as the Certificate Authority (CA) for the domain and issues certificates to agents created by clients in the domain and to AEE that operate in the same domain. A certificate can be thought of as a digital identity card attesting that particular public key belongs to a particular entity.

4.3 Framework for Mobile Agent Management

Mobile agent management is a full-featured framework for the development and management of Mobile Agents. It consists of multiple components, all written wholly in Java, which combined together provide a complete and robust environment for MA.

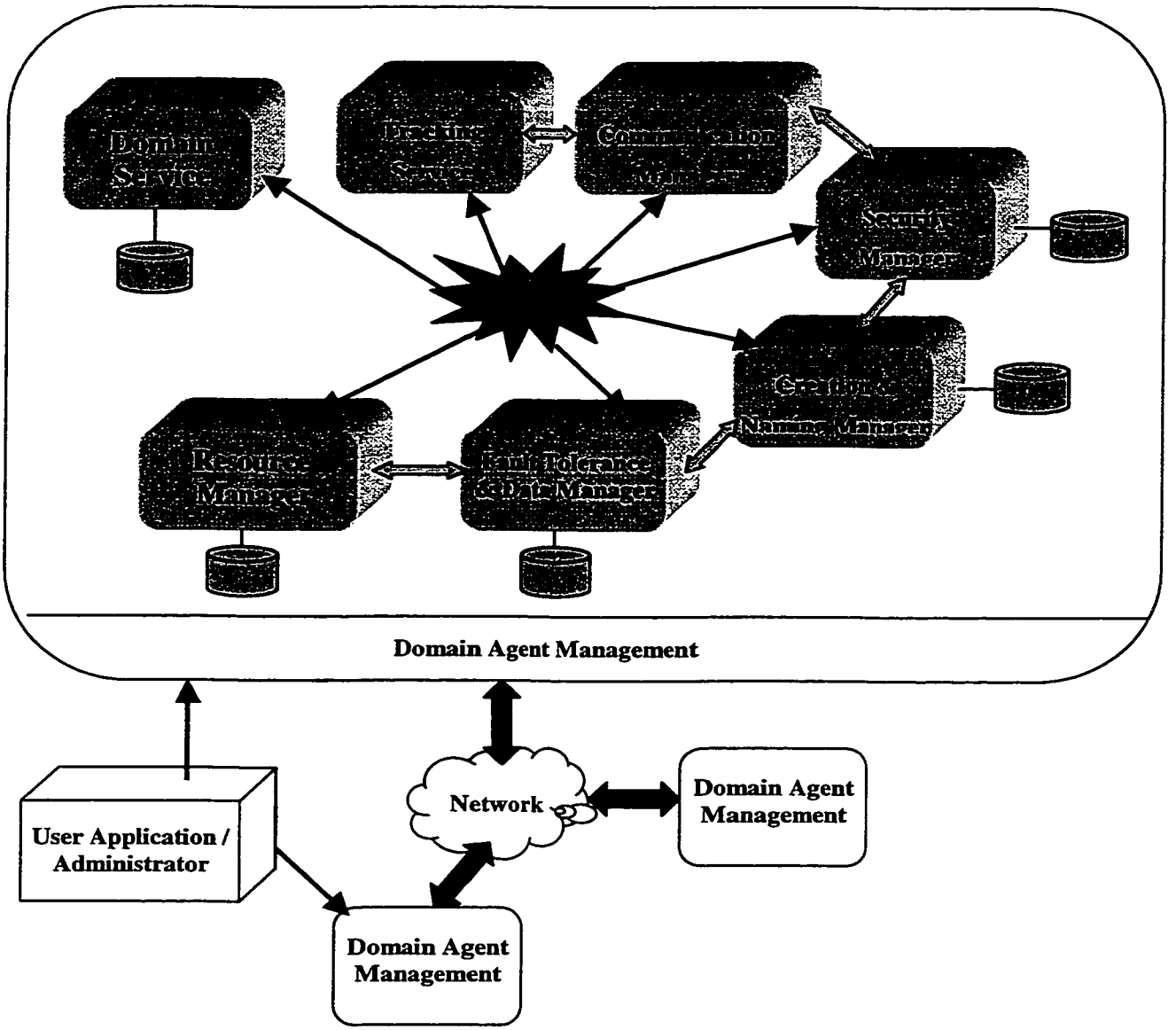


Figure4.2: Mobile Agent Management Framework

The following is a description of each module in the context of the services it realizes.

4.3.1 Agent Management System (AMS)

This part is primarily concerned with agent management services and defining interfaces to access them. An AMS is a mandatory component in the architecture; there is one AMS responsible for each domain. The basic and core services of an AMS are:

- Creating an agent
- Providing globally unique agent names and locations

- Tracking agents
- Providing "White Page" service for mobile agents
- Ensuring a secure environment for agent operations
- Managing user access and agent's data
- Providing User-Agent interaction

4.3.1.1 Creation and Naming

This section will specify a model for supporting agent naming and creation for the SHIP-MAI agent platform. It contains a description of an agent naming mechanism that is suitable for addressing mobile agents.

The use of a locally unique name is simple and feasible for static agent but it does not handle migration. When the agent migrates, name conflicts can arise and then we have either to change the agent's name, which is not recommendable or making the name becomes forwarding reference, which is impractical and inefficient for long chains.

Some proposed naming schemes use an abstract name (e.g. a service name). It therefore assigns a unique identifier for an agent that offers this service. This method can provide a globally unique name and could be useful for stationary agents nonetheless in case of mobile agent; we have to take full advantage from the agent's name in order to achieve an efficient communication with a minimum cost. Also the use of naming services such as X.500 needs a global knowledge. This can be expensive in case of exchanging information and cooperation between servers and will lead to some problems such as system failure in case of using a centralized server.

We believe that agents require names that can be identified in management operations, and can be located via a naming service. An agent's name must be a unique value within the scope of the agent system (Ship-MAI) that identifies a particular agent. Because an agent's name is globally unique and immutable, the name can be used as a key in operations that refer to a particular agent instance.

In our model, an agent name is seen as a container composed of attribute-values pairs that can be extensible in the future to meet other needs and requirement for specific

application and services (social names as in D'Agents system, name resolution, etc.). This allows information within the agent name to apply across multiple services, such as message-send, search-agent, etc.

The only requirements that we place on an agent name are: 1) the agent name must be immutable i.e. location-independent. The fact that the agent name has no relation with the current location ensure location and migration transparency 2) it should be serializable i.e. using Java serialization by implementing the java.io.serializable interface. It is composed of mandatory fields and optional parameters.

The Globally Unique Identifier (GUID) field is the only required parameter in the name composition. In the simplest case the agent name can be reduced solely to the GUID field that is the only field to be processed when comparing agent names for identity. GUID is not dependent upon communication or any other interaction to make them unique. The easiest way to ensure uniqueness is through a concatenation of a name and the domain address. However, an agent name is not necessarily equivalent to a GUID. Table 4.1 shows the different parameters that compose the agent's name.

Parameters	Mandatory	Type
Home agent domain	Yes	String
IP Address	Yes	String
Local Name	Yes	String
Parent Name	Yes	AgentName
Group Name	No	String
Service type	No	byte

Table 4.1 Agent's Name parameters

All parameters that require global uniqueness are assigned and maintained by the AMS of the domain. AMS is the naming authority in our system. Within the same domain, all agents' creation requests are submitted to the local AMS for naming procedure.

The GUID is the composition of the following fields:

- Machine name: the full Internet name address of the HADC (Home Agent Domain Controller) e.g. *spica.genie.uottawa.ca* or
- IP Home Address: the address of the HADC e.g. *137.122.20.74.*
- Local name: locally unique name that can be a string name submitted either by the agent principal or owner (e.g. *printer*) or a numeric identifier generated by a dynamic counter.

The combination of these two fields leads to a global unique identifier for the agent. The ability to derive the agent home location from the GUID is a big advantage that can be exploited in the tracking scheme and also in deploying services using the home location approach. This was inspired from the mechanism used on GSM where the telephone number of the mobile indicates the home location registry that contains all information necessary to reach the owner of the mobile [PLV94]. This was among the reasons to choose a GUID that is decomposable into a valid transport address that is the agent home (HADC).

The final byte is reserved for the Type of Service (ToS); thus 2^8 categories of service could be uniquely represented. This enables agents to query or to look for a certain agent that provides specific services with no need to know the agent's name or location. It is very suitable for agent services that will be viewed in more details in communication section.

The ToS feature adds more flexibility in agent based system. Assume for example, that a group of agents cooperatively perform a user-defined task. Assume further that one agent member wants to request another agent member for a specific task at a particular place for the purpose of co-operation. In this case, the target agent should be reached just through the place identifier and the agent role. The ToS helps handling those application-specific naming schemes for the purpose of co-operation.

In our model, there are basically two ways how agents can be identified: the agent_Ids i.e. agent's name introduced above and the ToS.

For each agent, there is a class from which the agent system instantiates an agent. This class is defined as source code of the agent. To create an agent, an agent system creates an instance of the Agent Class within a default place or a place the client application specifies. To create an agent, an agent system should instantiated the Agent class, generate (if necessary) and assign a globally unique agent name, generate a X.509 certificate that can be authenticated and finally start the execution of the agent (this could be delayed till a specific time and triggered by time or a user command.). Upon creation of an agent, the Agent Management System of the ADC home has to ensure that the GUID is unique. If the agent name presented to the AMS for validation contains just a GUID parameters, then the AMS ensures that the GUID is unique by comparing it against its local database and adds the information about the agent to its local database with no other specifications; the service type or group agent can be added further.

When an agent clone itself, it creates a copy of itself, duplicating its internal execution state. The same procedure for naming is followed, and the identifier of *Parent name* is also submitted with the creation request. Thus, by going up to the top-level agent, we can identify a hierarchical tree of agent generations. The clone agent is an independent agent that may continue to exist even after the parent termination. It runs on its own thread of control and inherits all the characteristics of the parent agent. A dynamic counter is added to distinguish between generations and to give information about the tree depth.

Using this naming scheme, the agent's name indicates the name of its HAD and thus its AMS, and also the information about its antecedents. The fact that the agent name has no relation with the current location ensures location and migration transparency thus allowing agents to transparently communicate and co-operate with each other. Figure 4.3 illustrates this naming scheme.

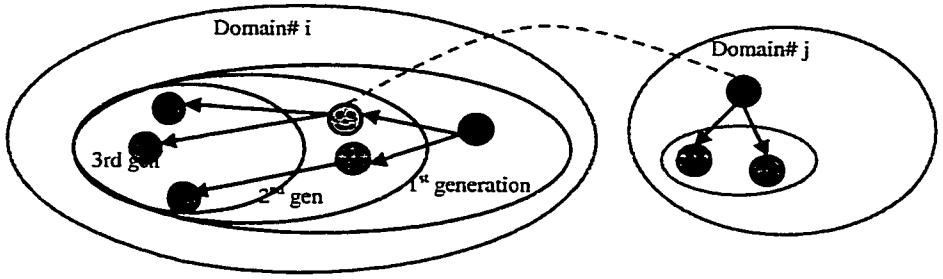


Figure 4.3: Naming Scheme for mobile Agents

User-applications use the AMS to publish named agents and declare them, or to find an agent given only the name (white page service). The resolution of an agent's GUID to its **current** ADC and hence to current transport address is achieved through the *locate_agent* action of the AMS. This action returns the address at which the agent can be physically contacted, more details about this subject is presented in section 4.3.1.5

- *Remote Agent Creation*

In remote agent creation, the client program interacts with the destination agent domain to request the creation of an agent of a particular class. This client program might be:

- A program in a non-agent system
- An agent in an agent domain different from the destination agent domain
- An agent in the same agent domain as the destination agent.

The client authenticates itself to the AMS of the destination agent domain, establishing the authority and credentials that the new agent will possess. The client supplies initialization arguments and, the class needed to instantiate and execute the agent.

4.3.1.2 Security

Because a mobile agent is a computer program that can travel among agent systems, a mobile agent can be compared to a virus. So, it is imperative for agent systems to identify and authenticate incoming agents. An agent system must protect its resources including the operating system, file system, disks, CPU, memory, other agents, and access to local programs. The agent system must also know what access the authority is allowed to perform. The ability to identify the authority of an agent enables access control and agent authentication within an agent system.

Unlike most agent systems, our system provides security based on the rights of the user of the applications, not on the permissions given by the developer of the application. This provides more personal control of the files, databases, resources, etc. that are available to a specific end user or agent.

Access control denotes the process of controlling access of an entity to a system or to resources within that system. It is closely linked to authentication i.e. the actual access

decision is based upon the profile of the identity performing the access. An Agent Execution Environment can either accept or reject anonymous agents (with no security certificates). If it accepts an anonymous agent, it would for instance gives the agent an extremely restrictive access to its resources.

Protecting the host machine is done via the authentication, i.e. verification of the identity and integrity of the incoming agent and its authority and via the given authorization i.e. assigning resource limits to the agent based on security credentials and enforcing those resource limits.

4.3.1.2.1Access Control List

An ACL is composed of "Authorizations", "Interdictions" and "limits". An ACL will have a list of authorization, a list of limits, and a list of interdictions. This contains all of the rights and privileges of one mobile agent. A list of authorizations or interdictions specifies the access rights to a list of items. For instance, a manager decides which agents are allowed to directly access low-level TCP/IP and UDP network services. It either grants complete access to the network or no access at all.

Example of authorizations:

- Allow read access to a specific list of files,
- Allow connect access to a specific list of TCP-IP addresses
- Allow write access to agent's register to report mission result
- Allow access to Print service

Example of Interdictions:

- Deny write access to a specific list of files,
- Deny connect access outside the Intranet domain

If all the items in the list have authorized access or denied access, the list can be set up to an Allow_All or Deny_ All, in which case there will be no explicit entries in the list and the associated privilege to each item will be always allowed or denied.

A limit specifies a maximum or minimum value that a certain parameter may take. Examples are:

- Maximum number of children a mobile agent will generate inside a specific domain during its life span,
- Maximum size of a mobile application,
- Maximum number of active channels,
- Duration time of remote network communication

These permissions and limits have to be granted by the hosted environment, and they allow easy management, the control of offered resources. While cloning, this ACL is inherited by the child agents. Its security credentials are transferred with it automatically and its access to services is under local administrative control at all times. Table 4.2 shows an example of ACL. It helps protecting access to system resources and data.

Type	Item Description	Action	Permission
File	Pathname(i.e. File name + location) :	Read	N
		Write	N
		Execute	Y
		Delete	N
Socket	IP address or Hostname	Accept	Y
		Connect	Y
		Daemon listen	N
Agent	Full agent name	Terminate	N
		Edit itinerary	N
		Dispatch	Y

Table 4.2: Example of permissions given to a mobile agent

While protecting the system resources is easy, protecting the mobile agent's sensitive data or state is the most difficult to achieve. To do this, the agent's owner set an access list that determines the privileges each entity has on the mobile agent. The agent's owner

has full control over its agent, but the agent's services or commands can be allowed or denied to other system participants (i.e. other agents, users, host machine etc). Upon its creation, the user can give full or partial access to agent's data to other system users. The host machine has to make sure the other system participants would respect these limits. For instance, the agent register should be manipulated only by the trusted stationary agent that resides on each AMS. Only the agent it self, the agent's owner and the accounting department have a write access to their respective register classes. All the other user application or agents are denied the write access and are allowed to query the register for information about the agent after their authentication and verification of their credentials.

4.3.1.2.2 Authentication of Clients for Remote Agent Creation and Administration

Security services provide for the authentication of system client applications. This authentication is done using passwords. After registering, users are entered into a security database under the control of a security manager. Each user has an Agent Control List that contains the list of all agents and the respective action he is able to perform on them. Users/applications are allowed to create and launch their agents and perform actions concerning agents on their ACL. The security manager is responsible for identifying users, authenticating their agents, controlling user permission, and ensuring the security and integrity of agents and their accumulated data objects as the agent moves among systems. The security manager has a user interface component, in order to configure and monitor the security attributes of the various users and services of the system. This user interface function is integrated into the Administration Manager.

The Administration Manager provides remote administration of the system. It supports simultaneous, central administration of multiple entities. The Administration Manager has a user interface component and can monitor the activity of the entire system from any Management Access Point. This allows the administrator to detect and control unwanted behavior, and is therefore an important part of security. Because the management interface also allows control over the mobile agents on the network, the administrator can respond quickly to security events, without having to physically visit any hosts. Using password logons, we ensure that the user has proper authorization, for starting, creating and dispatching a mobile agent. Any operation on the administrator interface, which

would alter the configuration of any entity or the configuration of a place or server system, requires the user password. Basically, this means that every dialog box in the UI on the administration manager requires the administrator password.

Through the migration history embedded inside the mobile agent's register, we can detect any rerouting attacks, particularly if an agent is following a fixed itinerary. This migration history could be examined and verified remotely by the administrator to prevent any unwanted behavior or travel to an untrusted domain.

4.3.1.3 Fault Tolerance and Data Management

Each transportable agent has an *AMS agent register* to provide fault tolerance that can be thought of as a container for all the relevant information about the agent including its history, its mission, consumed resources, policies that control its execution, etc. The agent acquires this register once it is created and it is updated whenever the agent migrates or achieves a part of its mission. The AMS is responsible for managing the agent's register and enforcing security concerning the access rights. As a matter of fact, each AMS will permanently manage its own database that is its native's agents and all information concerning the agent's owner, and temporarily the visitor agent's information.

There are only two ways in which an agent can be managed and concerned as "registered" with the AMS:

- The agent was created on the domain: When an agent is created on a specific domain it is automatically registered with the AMS of the domain.
- The agent migrated to the domain: When an agent is transferred to a new domain with its register, it is explicitly registered with the AMS of the actual domain.

In a given domain this register is represented according to the LDAP model and transferred as an XML file when the agent migrates to a new domain. This register is read-write and is updated automatically as the agent moves through the network and follows the agent from one domain to another. To access an agent register, one should get the reference to the AMS of the domain where the agent is currently executing. After authentication, the AMS decides whether to grant or reject the request.

In our model, we have adopted a distributed management information base. This choice is justified by the fact that a centralized resource can possibly cause a bottleneck for access and search. We have opted for a directory server using Lightweight Database Access Protocol (LDAP) model to ensure distribution and X.500 compatibility.

The agent's register is to used to facilitate searching for an agent either by knowing the mission, the creation date or any other significant field. It contains information about its actual location, its mission progress and optionally information about the user billing based on the resources accounting, i.e. the resources used by the agent during its mission. Although this register provides very interesting benefits, it also adds new security threats to agent management; additional security features should be applied in order to provide secure register operations.

Transferring the register with the agent to the new domain has several benefits, including:

- The register is readily available to the system administrator.
- The register is physically separated from the mobile agents, and therefore it is immune against attacks by any other hostile entity.
- The mobile agent does not grow indefinitely with each migration.

According to this model, where a relative central fault tolerance is adopted inside the domain, agents do not need to carry their work from one host to another while travelling inside the same domain. All the data and results are stored at the Agent Domain Controller under the supervision of the AMS. It acts as a repository for agent's work and data and hence provides a central fault tolerance for mobile agents.

The agent's register is composed of different classes of information. The main classes are:

- History: describes the agent's information that will not change in the course of time such as creation date, home agent domain, owner, supported language, etc.
- Mission Goals: describe the mission of the agent,
- Mission Results: reported by the agent, they contain the agent progress and the result of the mission,

- Itinerary: describes all the places that have been visited by the agent,
- Resource Accounting: this is primordial for financial and business applications,
- User Billing: according to the resource accounting and the charging cost formula, the user bill is established.

Table 4.3 describes in more details the content of the register.

Elements	Contents
Agent Name	The full agent identification
HAD	The name of agent Domain
Home IP	IP address of HAD
Creation date	The time stamp of agent's creation
Life Span	The duration of agent's lifetime
Owner	Identity of the agent's principal
Creator	Developer of the agent
Language	The language supported for agent development
Agent Group	In case of task distribution, the multicast agent group
Parent Name	The full parent agent identification
Agent Family	Different cloned agent
Delegation	Puts restrictions to agent mission (delimiters)
Goal	Key words describing agent mission
Result	Result / progress reported by the agent
Place Name	The internet name of the AEE
Region	The region identifier

Memory	The amount of memory used by the agents and its threads
CPU Time	The time slots of CPU consumed
Bandwidth	The amount of consumed bandwidth
Total Time	The total time he agent spend in the specified place
Charged Price	The total price charged for a personal service
Current Location	The current AEE where the agent is executing
State	One among the different agent's states
In Migration	1 if agent is in transit in the network, 0 otherwise
Default Contact Address	The default address where the agent may contact the user application if needed
Failure	Failure or report

Table 4.3 Contents of the mobile agent's register

- *Life Span:*

During its life an agent roams through the network, needs resources, CPU time, memory, I/O, and uses services provided by the system. The life span was introduced as an additional restriction on agent autonomy to prevent agent's looping and infinite roaming. It acquires more accuracy when embedded in a specific context of an application [BFK 98]. It can be thought of as an energy that the agent gets at the beginning of its life, and whenever it consumes resources or uses a service, this energy decrease according to the unit energy price of the resource or the service. Or it can be thought of simply as duration of time during which the agent has to perform the given task no matter how much resources or migrations he performs. When the life span is consumed, the agent is defined as an orphan, it stops its execution and according to policies agreed between the agent's

owner and the host machine, the agent can be either terminated or sent back to its HAD. The user application can choose to ignore the life span of the agent.

This provides a simple mean of determining orphans agents (similar to orphan process in distributed systems but more complex to determine) with the advantage that the activity (i.e. the use of resources, time spent) of agents determines their life span. The disadvantage of this simple approach is that the user application has to know in advance the life span to give to the mobile agent to enable it to finish its task [BAU 98].

If the agent determines that its life span is not sufficient to finish its task, it has the option to issue a request to its user application asking for "extended" life. It's up to the user application to decide whether to grant the request or to reject it. If the request is granted, the life extension is sent to the current AMS that performs this augmentation.

When creating another agent, the agent child gets the same amount of the current life span as its parent. This might be dangerous in case of malicious agents creating each other and thus getting infinite life; however, the maximum number of clones an agent might have could be set by the user application and restricted by the host domain. Another safe solution for this problem would be to get life span of the child agent from the parent agent. The creating agent should decide on the amount of life it should give to the clone agent according to its task [BAU 98].

The actual AMS hosting the agent is responsible of managing its life span. It checks the life span of the agent at regular intervals and decreases it according to the policy agreed (time basis or resources basis) to ensure a finite lifetime. If the decrease results in the agent having no life span left, the AMS will either terminate the agent or stop its execution, serialize its state and send it back with its register to its HAD.

When the life span of the agent is getting below a specific threshold, the AMS sends a message to the agent to inform it of its current situation. The agent may send a message to its user application/ HAD asking for augmenting its life span and report to its register the last results of its mission. The user application or the HAD receiving the message can reject the request or send an amount of energy if it judges it necessary (e.g. by consulting the mission results of the agent and its progress or the resource accounting) to the host

domain. The actual AMS is then responsible of increasing the life span of the agent to enable it finishing its task.

Enabling this option of controlling the agent's life span makes the agent dependent on its launching application concerning the additional life span to be sent. Another disadvantage consists on the time spent for checking on every agent's life to decide whether it should be terminated or not. However this concept makes it possible to determine orphan agent by using just the local information with no need for additional communication or supervision of HAD [BAU 98].

- *Itinerary:*

The Itinerary specifies where a mobile agent travels. It could be fixed by the agent's owner in advance, in which case the itinerary will contain the past and futures location the mobile agent has visited [BFK 98]. This itinerary forms the path that will be followed by the mobile agent. It specifies, in order, the hosts that the MA will visit.

In case where the mobile agent should decide where to go, the itinerary is updated dynamically to inform the user application about the locations that have been visited so far.

The itinerary would be beneficial in cases when it is the duty of the ADC to make decisions about the agent's next hop or to make sure that the agent follows the prescribed itinerary fixed by the user application. Also, the user application can detect any misbehavior of a visited host that tries to violate the agent integrity or doesn't respect the security agreement.

4.3.1.4 Resource Management

One of the important tasks in agent management is the support of resources control and accounting. Accounting is a prerequisite for commercial applications with agents and resource control is necessary for making sure that the resources are used properly.

Resource management can be achieved by computing some parameters that show the amount of resources used by the mobile agent in a specific place. This information can be useful for charging and giving permission to access some critical resources...etc. These

parameters are computed for each mobile agent in each visited place (AEE) and reported into its register, among them are:

- CPU time: this value can be obtained by incrementing a counter each time a time slice is allocated to threads launched by the agent. This can be done using the Java Scheduler that manages such resources on the Java Virtual Machine (JVM). Some researches are currently investigating in resource accounting using the Java platform since most of the mobile environments are Java based [URL10,URL11].
- Total time in the place: By enabling a counter at the arrival of the agent in a specific place (AEE) and its departure we can easily compute the time the agent stay in a specific place.
- Communication with remote places: it includes number of opened channels and remote communication time.
- *User Billing*

Prices could be service-dependent and not according to a flat rate. These price values are often set by the market and competition, however it would be important for the system managers and owners to know the relative costs of providing various services.

A simple formula to compute the charging price could be based on service charges and residence charges.

Total_Charges = Residence_Charges + Service_Charges,

Services_Charges = $\sum$ unitprice(Si)*unit

Residence_Charges = TotalTime*a + CPU_Units*b+memory*c, where a, b and c are the unit prices.

Residence_Charges states for charging the agent for the amount of time it spent in a specific place. An agent system has finite resources and limited capacity concerning the number of agents it would receive, and execute. Thus the presence of the mobile agent prevents other agents from being accepted in this environment and benefit from its services: as a matter of fact, the Admission Control Agent running during the migration negotiation phase may reject a request if the available amount of resource required cannot

accommodate the new request. It is therefore fair to charge even for the resources used by agent (memory, CPU, Storage); this charge has to be proportional to the duration of the residence, the amount of memory and CPU consumed. The service's charges are concerned with the host specific service and depend on the service unit price.

4.3.1.5 Communication Service

A communication infrastructure provides communications transport services, tracking, and security services for an agent system. This infrastructure enables all remote interactions between the different system entities such as location transparency inter-agent communication, agent and register transfer, localization of agents by means of the tracking service, and inter domain and regions communication.

The low-level interactions are achieved using the existing communication infrastructure via Java Remote Method Invocation (RMI) or plain socket. These connections can be protected for a secure communication by using the Secure Socket Layer (SSL) which is the standard Internet security protocol. This service supports synchronous, asynchronous communication and multicast communication

4.3.1.5.1 Protocol Support

Remote communication is generally achieved via generic existing protocols. The communication service supports both Java's RMI and plain socket connections. To enhance security, both RMI and plain socket connections can be protected with the secure socket layer (SSL).

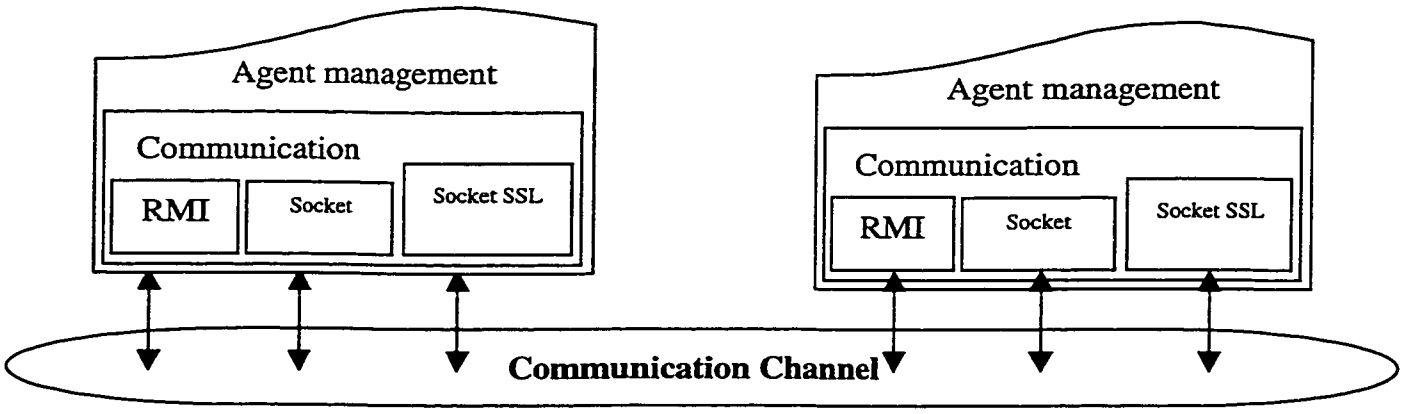


Figure 4.4 Protocol Support for Communication

RMI was introduced in early Java versions (JDK1.1). It enables Java objects to invoke methods on other Java objects running on any Java Virtual Machine (JVM). It is supported by default, and need no further installation or configuration since it is included in every JDK1.1 or later versions [URL3]. The option of using RMI with SSL consists of running RMI over protected socket by means of SSL protocol [FKK96] that provides secure transport of all data. Refer to SSL section for more details.

The fastest way to achieve remote connections is by means of plain socket connecting to a specific port of the target host server. Sockets are a ubiquitous operating system capability, and provide good performance and response time. However, plain sockets has low level of abstraction than the desired ability to make remote object invocation as transparent as invoking a function on a local object. To fill this gap, we must address many error, messaging and location issues. The protocol and message formats for the data to be passed through sockets need to be defined. This implies also writing and maintaining our own socket-level programming to determine the node and port addresses of the destination object, initialize the socket to connect to that destination, and subsequently marshal, transmit, and demarshal the data in the appropriate formats. However, once these issues are resolved, any application-based agent can use and benefit from this service with a very high level of abstraction. Refer to the AMS business scenario as example in next chapter. For secure connections we use the plain socket with SSL for communication over untrusted domains.

Within the same region or domain, the system is able to detect the most suitable protocol for the remote interaction with the desired communication peer. New communication protocols can be easily integrated to be supported by the communication service (e.g. CORBA IIOP) by writing API to interface the new protocol. All the supported communication protocols are realized via their appropriate interface.

The figure 4.5 shows an example for network and protocol configuration. Usually, an Intranet is connected to the Internet via a router and protected via a firewall running on this router. Here we can view a domain as an Intranet connected to the other domains and outside world (e.g. Internet) via the ADC. Only this host server is visible to the outside users. This technique is useful for protecting users and agent-services from malicious

host or untrusted clients and authorities by providing one single access point i.e. ADC allowing only authenticated authorities and agents to have interaction with the agent services and the AMS. Inside the domain/region, interactions are achieved with no use of SSL (i.e. fast communication).

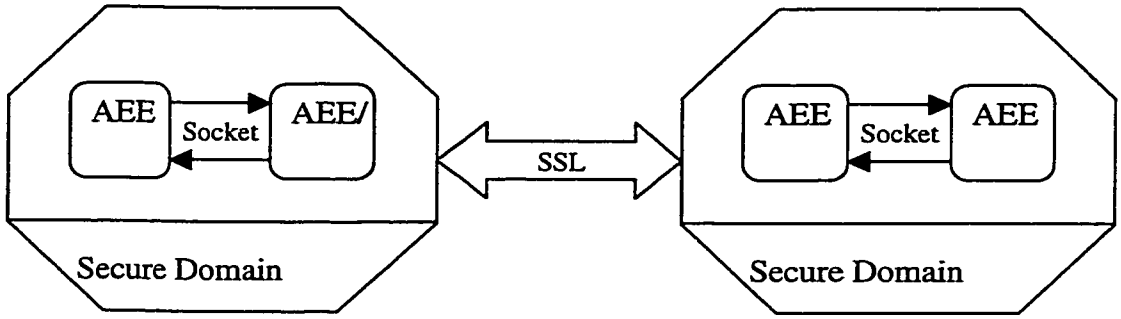


Figure 4.5: A Possible Configuration for Secure Network Protocols

For Intra-domain communication, hosts connect directly to each other using plain socket connections; this allows fast communication between hosts in the same domain. The network access point represented by the ADC accepts all communication from the exterior hosts using SSL protocol. Based on security credentials and domain policies a communication request can be accepted or denied.

4.3.1.5.2 Location Transparency

The communication service allows location-transparent interaction between agents, domain and non-agent based entities (e.g. human users, applications). These entities should not be concerned with the actual physical location of the target objects they are communicating with (i.e., whether they are in the same process, on the same processor, or on the Internet). For remote communication, the application should be isolated from the details of finding objects. In achieving location transparency, the objects become insensitive to the physical architecture (where they are located) and the underlying transport mechanisms (how communication with remote objects is supported). The application design decisions become a separate concern from the physical distribution and migration decisions. Applications built to exploit location transparency are thus more adaptable to changes and have a longer useful design life. In addition, the application

objects components are much more reusable, as they all exploit a common distribution infrastructure and their design is de-coupled from the physical architecture. A high level design for a white page service that could be used to provide location transparency is shown in figure 4.6.

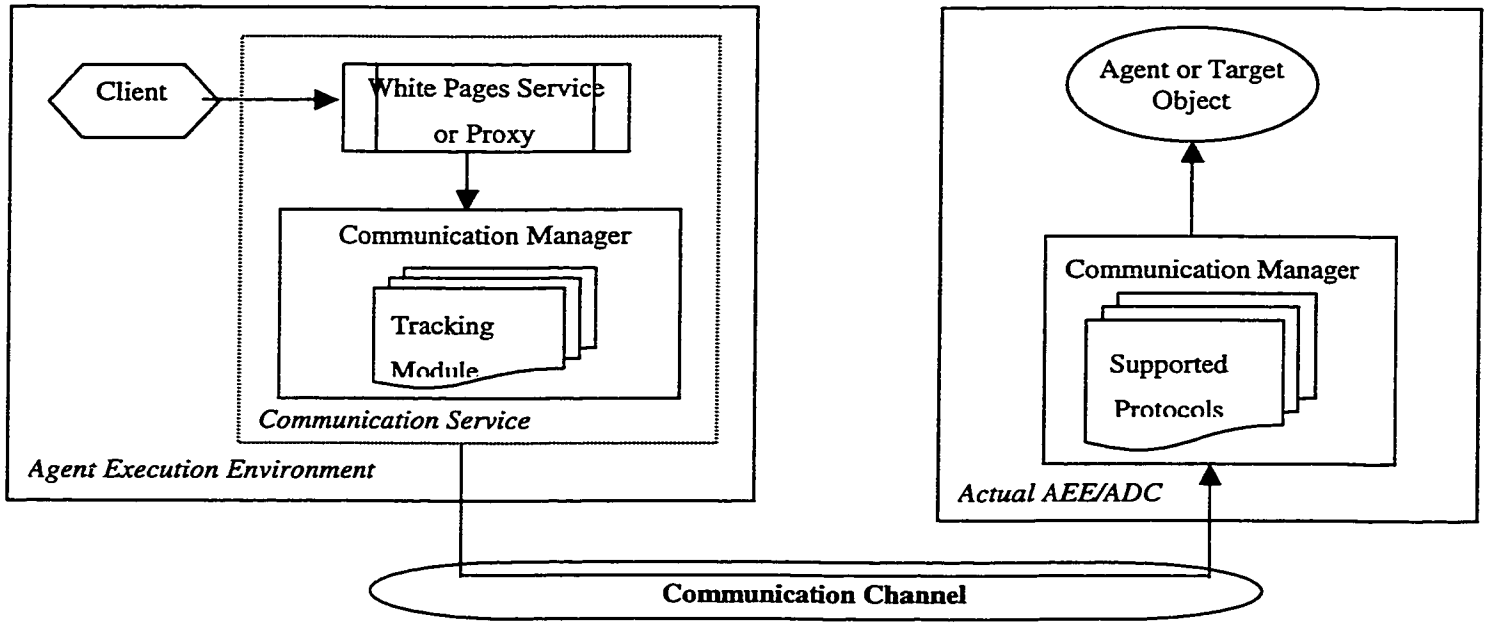


Figure 4.6: Location Transparent Communication

The white page service is a local service on each AEE; it offers a simple API to agent-based applications developers and hides all internal/external requests and negotiations. The location transparent communication works in coupled with the tracking module and is not restricted to a specific domain, it covers all the regions and domains included into the system. The white page service is responsible for finding the actual location of the target agent or object (e.g. agent's register), and contact the responsible AMS for further interactions or queries.

This service is responsible for name resolution from the agent's GUID to its current ADC. Upon reception of a request the white page service executes the following steps:

- Extract the address of the HADC (the GUID is resolvable into a valid transport address of its HADC),
- Check if the specific agent belongs to the actual domain by comparing the HADC with the current domain,

- If so ask the local AMS for a reference to the current ADC (The HADC keeps always a reference i.e. kind of proxy to the actual domain (refer to tracking section) and return the address,
- Otherwise use a remote method invocation on the White pages service at the remote AMS on HADC and get the actual ADC.

Figure 4.7 shows the flow chart for the look-up mechanism of the white page service.

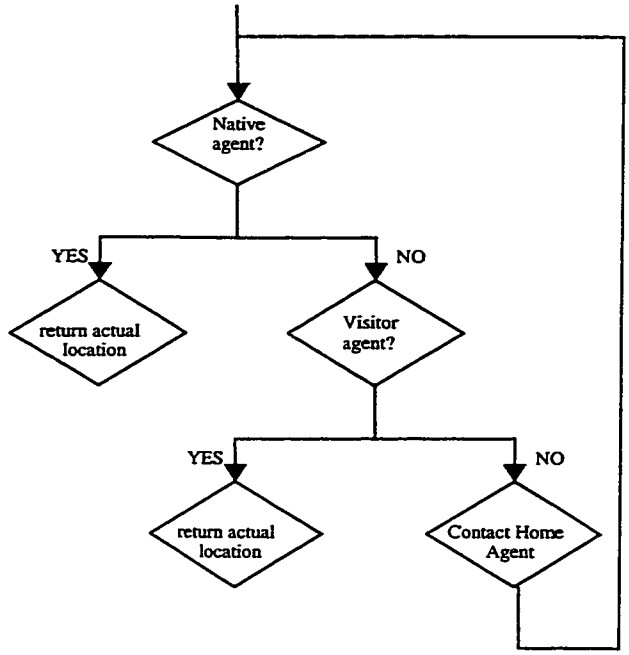


Figure 4.7: Lookup Mechanism for Mobile Agents

4.3.1.5.3 Communication Modes

In the context of agents and agents platforms interaction, communication could be performed either synchronously or asynchronously.

- *Synchronous communication:*

This is the most common way of communication. It follows the same concept as method invocation in any programming language. The client sends a message or invokes a method on a server, waits for the receiver to compute the result that is sent back to the client, which then can continue its execution. The only difference when dealing with agent's messaging is the issue of locating the receiver agent and forwards the message to it. This problem is largely based on effective communication manager service. When the

communication partner has been located using the lookup mechanism, the message can be sent to the actual location of the target object, at which point the receiver can process the message and return a result i.e. response or failure or denial.

- *Asynchronous messaging*

In some case, it is useful to send a request for processing and be able to proceed with other tasks while waiting for the response. The result might be returned later at any point of time, on or not all. When the result message arrives, the client is notified via a call back method about the arrival of result. In case of human user-interaction, the result can be sent via emails or displayed on the user interface.

- *Multicast communication*

Multicast communication allows a client to send a request or invoke a method on different target objects (e.g. Agents, DFs or AMSs) in parallel. Those target objects should belong to a specific group. A group offers an option for realizing partitioned concurrent behavior and can be constructed dynamically at runtime upon the client request or it may already exist in the system. It consists of an arbitrary number of members identified either by their transport address or their names i.e. GUD of agents. Multicasting is useful for an AMS to contact a group of agents in the context of multi-agent group task, or for agents to contact each other and coordinate their actions. This group encapsulates the thread-based mechanisms needed to achieve parallel behavior that hide all the interactions and address translation from the sender. At the low level, this concurrent communication will be achieved using a multithreaded approach i.e. thread group, where each thread will be performing a specific communication task. The sender should also specify if the threads performing the different requests should be synchronized or depend on each other result in which case the methods implemented by the thread group should be synchronized and their results gathered and passed back to the sender. In case of agent addressing, the first task is to find out the actual corresponding transport address by mean of the communication manager service and is reported on the group transport address list so that the receiver can be contacted directly.

Whenever a thread is unable to deliver a message to one of the addresses on the list either because the specified location or place is unavailable or invalid agent name etc, the

thread stops its execution with an exception failure that will be reported back to the sender. If the communication manager finds that the specified receiver has left a forwarding address (see tracking section) this forwarding information is added to the current list of group addresses. That is the forwarding information left behind by the destination agent takes priority over the information originally given by the sending agent or by the previous AMS. If no forwarding address is left, the communication manager white pages service is solicited again to find the actual agent's location, so that the list of the groups transport address is always updated with the last available information. Failure to send message to an AMS or DF is not addressed since these services have a fixed transport address.

To be compliant with the multicast concept, a join and leave options should also be available to the group members. However, they should be manipulated very carefully to avert any temptation of a spy agent to join another agent group or any agent to leave its own group. A minimum of security mechanisms e.g. authentication, should be observed while using these requests by agents.

4.3.1.6 Tracking

This module provides methods for maintaining a dynamic name and location database of agents. There are many possible ways of locating an agent [HST96, ADL98]. Here are four basic possibilities:

- Brute force search: In the Brute Force mechanism, an agent is located by searching for him in multiple destinations. This search can be done in parallel. This scheme is suitable when the itinerary of agents is short and already known. It becomes impractical for destinations not known in advance and for large number of hosts due to communication overhead related to the broadcast.
- Registration: The agent updates its location in a predefined directory name server. This server provides the following services for agents: register, unregister, and locate. These directory servers can be organized in a hierarchical manner in case of large area. But they can be overloaded or temporarily unavailable. This mechanism is similar to DNS.

- Updating Home Location: Whenever the agent migrates from one place to another it sends an update message to its home. This can be very expensive if the frequency of migration is over a specific threshold and if the hosts are far from Agent Home.
- Forwarding: The idea of the path concept is that each agent will leave a trail behind. Each location will keep the information that the agent had been in this place. By following the trail to its end, the agent can be found. This is suitable for a small number of migrations. It is not appropriate for long chains. The fact that the paths stay after the agent's migration and maybe termination requires some distributed garbage collector algorithm to remove obsolete forwarding information.

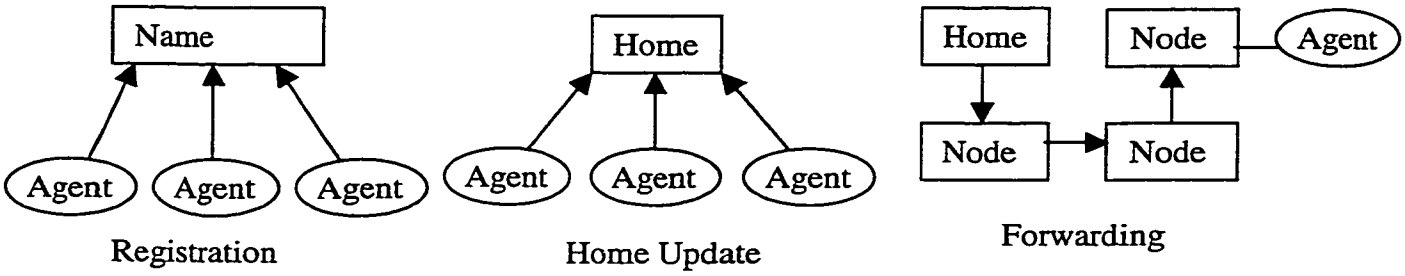


Figure 4.8: Different Locating Schemes for Mobile Agents

In practice the selection of a specific agent tracking mechanism is subject to:

- Destinations: this includes the number of migrations, the number of nodes inside the network, intra or inter-domain migration, etc.
- Performance requirements: the cost of the tracking must be justified depending on the type of application, frequency and type of migrations.
- Robustness or fault tolerance: we mean by this how to deal with failures and network latencies. This includes network failure, agent lost or integrity violation etc.

We propose a hybrid scheme that combines the benefits of registration and involves the home node by using the mechanism of home updating. We distinguish between two kinds of updating location procedure depending whether it is Inter or Intra-Domain migration. The update procedures could be done by the agent systems, which are

involved in the migration process or by the migrating agent itself. Our tracking model is based on messages exchanging without involving the agent, this gives the agent more autonomy in migration and liberates it from updating his location each time he migrates. The basic messages exchanging are illustrated in figure 4.9 and 4.10.

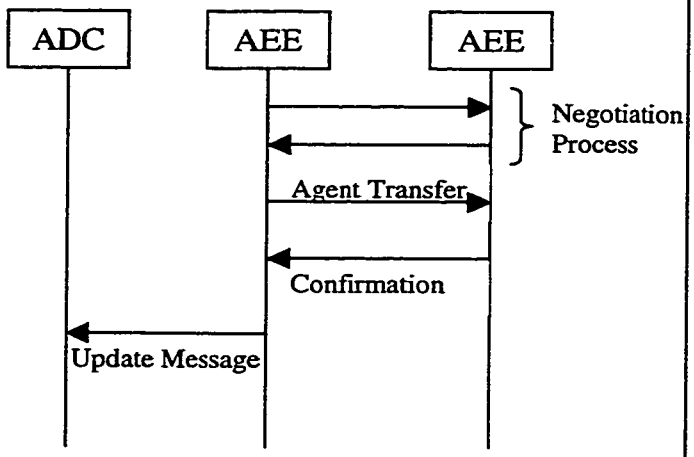


Figure 4.9: Message Diagram for Intra domain migration

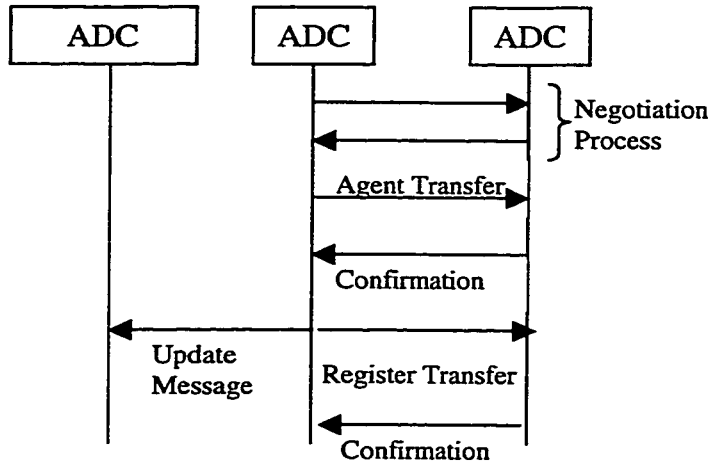


Figure 4.10 Message Diagram for Inter domain migration

- **Inter-Domain**: Let's assume that agent X migrates inside the same domain from AEE#1 to AEE#2. After negotiation between the two AEEs, the source AEE (AEE#1) sends an update message to its ADC after receiving a message from the destination location (AEE#2) that the agent has been successfully transmitted.
- **Intra-Domain**: If the agent migrates from the home domain to a foreign one, The register is delivered to the visited ADC after the ACK message concerning the agent migration has arrived (see register transfer in chapter 5). In the Home ADC only a link to the visiting ADC is kept. When the Inter-Domain migration is performed from a foreign domain to another one, the ADC that carry out the migration send an update message to the ADC Home indicating the new domain of the agent and sends the register to the current ADC where the agent has migrated. After confirmation of register reception, all information about the agent is deleted from the old AMS. The current AMS becomes responsible for handling all requests and messages to the mobile agent and registers the visitor agent with its agent management database.

In many cases, even when the agent is successfully located, it might migrate by the time the location is reported to the requesting agent or user/App. So all communications or requests sent to the old location will be just ignored and lost. This can be critical in case of control messages or synchronization request. The optimization scenario proposed below aims to solve this problem.

Optimization Scenario

The optimization that we propose is to use the forwarding mechanism during agent's transfer and installation in its current place and home update to make sure that the client will get the exact reference from the HADC. This is achieved by keeping a forwarding proxy until reception of the update ACK from the HADC of the migrating agent.

We distinguish here again between inter domain and intra domain migration.

- **Intra domain Migration:** In this case, when the agent's migration request is granted by the new place and the transfer is initiated, the AEE responsible for the migration sends a message to its ADC indicating the future location of the agent. The ADC will have information about the agent's state, the "current" location and the future one.
- **Inter Domain Migration:** in this case there will be no extra messages, The ADC will simply update the state of the agent (in transit) and the forwarding proxy with the future location.

Using this hybrid approach, whenever there is a message addressed to the agent at the past location, it will be forwarded to the future one, and the agent's migration will not be delayed anymore waiting for some critical messages to be delivered. All messages received during agent's transit will be kept until agent's resumption on its actual AEE and forwarded to it, the forwarding proxy concerning this agent is terminated.

Another problem faced with mobile agents is failure issues. It can be either integrity violation or any other reason for which the agent will not be able or allowed continuing its mission. A simple solution is to send back the register to the HADC with a failure notice, and according to the policy agreed between the agent's owner and the host domain, the actual AMS will either choose to terminate the agent or stop its execution, serialize its state and send it back to its home. Thus, the agent's owner will be aware of

the problem and the results that his agent has accomplished so far (included into the register).

The reproach to the most tracking mechanisms such as forwarding or registration is the cost and failures. However these problems are overcame as we explained above. In our model, adopting the idea of domains reduces the communication overhead. In fact the communication inside the domain is less expensive than the Inter-Communication; the communication between domains is needed only in the case of Inter-Migration. This is not the case used in the home update or the registration mechanisms when the message exchanging is performed regardless to the current agent location.

4.3.1.7 Messaging

Our system provides a high level of abstraction by adopting a messaging approach built on top of JVM support for sockets.

We've defined different types of messages depending on the type of request. The default parameters for all messages are the message type, the sender identity and the destination. The sender identity and/or the destination can be either a specific location (host and port) or a logical name (e.g. agent name). In case of a logical name as a destination, our system is able to resolve the actual agent location with the help of the white page service.

An agent has two options when it wishes to contact another agent:

- It can request the communication manager of the AEE on which it currently resides to route the message to the target agent.
- It can contact the white page service of the actual AEE to get the actual transport address of the target agent and send the message directly to the AEE that will be responsible for routing the message to the agent.

The communication service will route messages on an agent's or any entity behalf where possible. The transport protocol is completely hidden from the senders; that is an agent is not requested to support any additional protocol apart the one agreed with its environment. All the entities and other services on the system supports message based communication while the communicator supports stream based communication.

The remote communication involves both the communication managers of the hosts where the sender and receiver may reside; it is then the responsibility of this service to route the message to the receiver agent according to agreed protocol. More details concerning the message object syntax will be given on the chapter 5.

The message delivery service will detect and report errors, such as: bad formed message, unknown host, unreachable agent, etc, to the sender. Depending on the error condition, this may be returned either as a return value from the message sending interface, or through the delivery of an appropriate error message. The message delivery assumes that each agent has a name which will allow the message to reach the correct destination.

When an ADC or AEE receives a request message concerning a specific agent, it will check with the local AMS to see if the agent identified by the GUID in the destination parameter of the message is actually registered. If the destination agent is not registered then the AMS returns an AgentNotFound Exception to the sender.

Agents running on mobile devices and getting disconnected from the network need to update their knowledge and be aware of different messages that could have been sent, when reconnecting to the network. To handle this device disconnection, the mobile agent might specify another fix and valid transport address (HAD for instance) as a contact address in case of message delivery failure due to unavailable host. While forwarding the message to the contact address, the sender should specify in the type of message that this message will be for storage and not delivery. The contact address should be able to store these messages and communicate them in a FIFO order once the mobile agent asks for them.

- *How the communication partner can be addressed?*

In the case of mobile agents the concept of agent names is not always sufficient. Assume for example, that an agent wants to communicate with other agent participating in the same task at a given place. If only agent_Ids option is available, both agents must know each other's names. Rather, for identification it would be possible to say [At place P1, I would like to contact an agent performing the task T1]. This type of identification is supported by the concept of ToS defined in the section 4.3.1.1. This ToS is an application-generated identifier that is assigned to agents upon their creation and is

hierarchically ordered. For example all agents participating in achieving a specific task would have the ToS indicating the overall task followed by its proper subtask.

Using this feature, an agent can be identified by a place name and ToS pair values or even just by using the ToS using the DF (see section 4.3.2). Note that at most one agent qualifies when specifying the full agent name while several agents may qualify for the same ToS. In that case a randomly picked agent is chosen.

4.3.1.8 Summary of AMS Benefits

This section specifies features of the proposed AMS architecture. It highlights inherent weakness and proposes in some cases potential solutions.

- *Fault Tolerance system and Agent's Register*

In some agent-based applications we have noticed that the mobile agents need to consult results performed in previous sites or consult other agent's results in the context of agent's cooperation. That is, the task to be accomplished in a particular site would depend on the results already performed elsewhere.

On one hand, the problem of embedding the agent work's results and other sensitive data within the agent may be sensitive in case where the agent get lost, captures or corrupted leading to the lost of all valuable work accomplished so far. On the other hand, carrying this record log i.e. register, from one node to another would increase the network load and security overhead while transferring these data. In our model, these issues are overcame by providing a repository model on each domain that will be managed by the AMS.

Data and results storage is provided by the AMS on the ADC on which the mobile agent resides. By keeping temporarily cached copies of agent's register, state and maybe source code, it enables the agent to consult its own register and optionally other agents to consult their each other's register according to the domain policies and their access control list. This register will be kept only until the agent successfully migrates to another domain, in which case its register is transferred between AMS as an XML exchange document. This may help significantly reducing network traffic since the register consultation will be done inside the domain using fast protocol communication with no need for security overhead which will not be the case if the data is kept at the HAD.

- *Protecting sensitive data:*

In the proposed architecture, the AMS is responsible of all the sensitive information needed to monitor mobile agents within the domain and provide access to these data according to the security credentials and access control list of other entities. Cracking an AMS would give access to all agents running on the domain, it is, thus, primordial to provide elaborate security measures to protect the AMS and hence the domain data. A possible alternative may be to duplicate and keep updates the agent's register at its HAD or on the user home that created the agent. However this will generate a considerable amount of network traffic and will put additional requirements on the user device concerning storage facility.

- *Communication overhead:*

In order to evaluate the communication overhead or reduction introduced by our model, we will enumerate the different interaction between different entities in the system during agent creation, migration and tracking.

AMS-AMS Interaction:

A. When transferring the agent's register from one domain to another; or control data (such as acknowledgement that are reported to the application level so that the user application will make appropriate decisions in case of failure or corrupted data),

B. Location-Update messages exchange between HAD and the hosting domain introduced by the agent tracking module,

C. Distributed look-up mechanism concerning mobile agents that is achieved at the maximum of two level, involving the HAD and the hosting domain.

B and C might not be considered as an overhead since it provides means for agent tracking and localization in a more efficient manner than the basics schemes introduced earlier. Moreover these interactions took place only in case of inter-agent migration that could be dramatically reduced by exploiting the locality and enforcing stability with respect to a specific domain. (For more details, refer to DF configuration in section 4.3.2).

A could present a communication overhead in case where the agent doesn't need to consult its results and when there is no need for resource accounting and billing, however it helps reduce communication costs and present a "recovery" solution when agent are lost, damaged or corrupted.

AMS-AEE interaction

A. Creation (or cloning) requests and security credentials setting upon the agent's creation,

B. Communicating status information regarding residing agents,

C. Communicating resource accounting and usage concerning each residing mobile agent,

D. Updating internal agent location within the domain,

A, B and C derives from the central agent management system, however this approach has the benefit that the AEE is not required to provide storage facilities or to possess additional resources to support data management and fault tolerance. D is introduced by the tracking module and is doesn't represent a big communication cost since it is concerned with intra-domain message exchange.

Agent-AMS interaction

A. Register manipulation requests: either for writing the results or reading data,

B. Sending Look-up requests to the white page service,

A is primordial to ensure results reporting and knowledge sharing or update, while B is a basic interaction that exists in most agent systems.

4.3.2 Directory Facilitator

We have adopted a logical hierarchy that benefits from the geographical hierarchy that already exists. We define a logical root node that comprises the whole network, and we represent the entire agent universe as the set of all regions and their domains in the distributed agent enviroment.

4.3.2.1 Regions Concept

A region will be responsible of offering a set of electronic services that are somehow related to each other. Regions allow scalability because one can distribute the load across different agent domains and regions. It also provides also a level of abstraction to clients communicating from other regions.

An example of regions concerning electronic markets will be:

- Travelling Options.
- Shopping center (Flowers, Clothes...etc).
- Software Delivery.

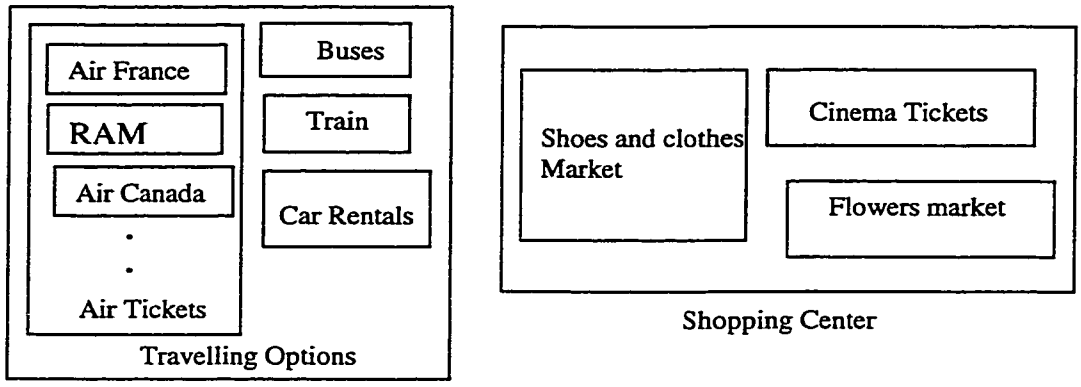


Figure 4.11Example of Regions Concerning Electronic Markets

Each region is managed by a Service Agent Access Manager (SAAM) that is responsible for a set of domains. The directory facilitator is seen as logical distributed tree (composed of SAAM) that covers the distributed agent environment. SAAMs exchange information between them and advertises their domain services to other domains via XML based documents.

The primary task of a DF is to provide "yellow pages" service to other agents. Domains may register their services with the DF while agents may query the DF to find out what services are offered by other agents on specific domains. The DF is a trusted component in the system architecture in the sense that it must keep an accurate, complete and timely list of domain/places and provide the most current information about agents and services in its directory to all authorized agents. At least one DF must be resident on each region. The DF may restrict access to information in its directory, and will verify all access

permissions for agents, however, it does not control and has no authority on the life cycle or states of any Agent. It just plays the role of a proxy between the clients (i.e. agents, users) and agent-services.

It will also help the agent choosing when and where to move. An agent should jump when the local environment doesn't fit its requirements anymore (e.g. slow performance, end of task) or when this actual local environment will cease to exist. An agent will jump to a new host machine (AEE or ADC) M, if it has unique resources required by the agent or it has the best environment for the agent's next task(s).

The decision when to jump and the decision where to jump can be thought of as an optimization problem. This is generically NP-Hard because it involves uncertainty about future environments, decisions about whether to clone or not, etc. Where to go problem depends on:

- Locations of candidate services;
- Network latencies;
- Network bandwidths;
- Machine loads.

4.3.2.2 DF Configuration

This part deals with the building of the logical search tree. The SAAM is a software component responsible for one region; it doesn't have to know about the entire network configuration, it cares just about its adjacent region managers. The configuration data is manipulated only by the system administrator.

A DF configuration might be affected by several factors such as security, performance, flexibility, etc. The above configuration and distribution example for domains and regions was motivated by the following properties:

- Ensuring Scalability and flexibility. Every physical node is able to handle its workload independently of other physical nodes, thus facilitating the transfer and redistribution of the workload in the future.

- Forcing stability and exploiting locality. An agent is said to be stable with respect to a domain, if its migration took place inside that domain. Since a buyer agent for instance may need to visit and compare offers from all services providers within a specific geographical region, it will only need to perform intra-domain migration.
- Reducing search cost, migration cost and tracking cost. If an object is stable with respect to a domain D, we believe that migration overhead or updating the agent's location in D is cheaper than when the agent is not stable with respect to D. That is, exploiting locality in location updates can reduce tracking costs.

4.3.2.3 Domain Service Manager (DSM)

A Domain Service Manager is responsible for managing the services residing on the domain, it also keeps track of the system load and domain state to help the mobile agent taking its decisions about where to go in case of multiple service providers choices.

There are 3 basics functions for manipulating the DSM:

- *Register*: this action is implicitly performed upon the creation of a new agent-service. Thus all agents-services on a specific place are registered with their respective DSM unless the owner of the agent specify not,
- *Unregister*: this action retrieves the agent from the directory service and the service becomes no more available although it might be always present on the domain.
- *Search:* this action look for the appropriate domains and services that might help the mobile agent achieving its mission.
- *Modify:* this action deals with any changes concerning the service agent's description in a particular service directory.

4.3.2.4 Service Agents

Service agents cannot migrate and are therefore considered trustworthy; hence, they are not subject to the tight security restrictions in place for mobile agents. This is an important feature, as it is necessary for certain agents to have full access to the host

machine and resources. They can be started only by the local administrator of the domain, and are automatically registered with the DSM upon creation. Services can be clients for local database access, mail server or a selling agent. They can consist of simple classes, but are not restricted to them. The service directory is a central repository for all available services and contains the service description of all the service agents. This directory could be organized as an X.500/LDAP directory service server or any other database.

Agents registered with the DMS are considered as static agents, table 5.1 and 5.2 show the different attributes for such agents.

Attributes	Content
agent-name	The agent full name
Services	The set of services the agent can provide
Address	The local internet address where the agent resides
language	The agent supported language
ownership	The agent owner/authority
public	1 if the service is public, o if the agent was developed for private purpose

Tab5.1: Agent description registered with Domain Service Manager (DSM)

The service attribute denotes the service(s) the agent can provide. This includes a description of the characteristics of the service and is composed of the following fields:

Attribute	Mandatory
Service-name	Yes

Service description	Yes
Keywords list	No
Service-type	Yes

Tab5.2: Services description registered with DSM

The service type is hierarchically ordered keywords from the list of keys specifying the type of the service. If no proper keyword describing the service exists a new keyword can be introduced. The service description is a description of the permanent characteristics of the service that could be a complex structure using a particular predefined ontology or XML ontology.

In case of duplicating or transferring the service inside the same domain from one place to another, the DSM updates its resources tables with the actual network address of the service. If the service is transferred to another domain (e.g. as a result of load balancing decision), the DSM retrieves and deletes the entry from its resources tables. The DSM is always informed about the service transfer, duplicating or removal using the 3 basics actions that will be described in more details later.

4.3.2.4 Domain State

To provide the client with more information concerning the system performance, DSM has another component that is aware of the domain load and network state (e.g. bandwidth, delay, etc). This feature will help load balancing and take part in the agent's decision about where to move in case of different service provider's offers.

4.3.2.5 Service Mapping:

The agent's world is a decentralized collection of regions. Each region contains a set of local resources (i.e. services) that encapsulate the offered capabilities, it can be a mail server or a digital camera or any software service in case of a market place. The service mapping is concerned with mapping service name to the real network location(s).

By using a flexible location and communication management protocol in association with a resource discovery protocol, it is possible to access the most appropriate resource provider and thus to achieve a better performance.

By analogy to a conventional operating system, a region can be thought of as a kernel, services as device drivers, and agents are like processes. To discover resources, we use a mechanism similar to UNIX resource retrieval, but which is able to discover dynamically resources in unknown domains and other regions that form the distributed agent environment.

The mechanism therefore uses the resource tables (services are considered as system resources) that are modified dynamically according to service availability. These resource tables contain pointers to the available host services and applications. It ensures the matching of service names with real network transport address and agent name that provides the service.

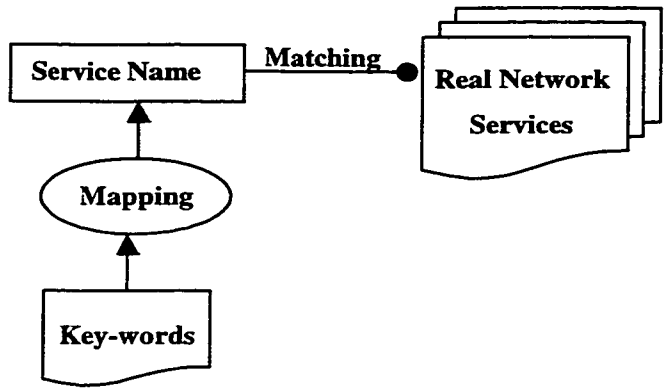


Figure 4.12:Mapping Services Names to Real Network Locations

The DSM is responsible for finding appropriate services that match specific key words. The key word can be the service name itself if possible or its type, otherwise a set of pre-defined words can be used to characterize the service. In the latter case an additional processing should be performed to match the key words with the most appropriate service(s). Once we get the service name, another search on the local agent directory service is performed to find the agent responsible for the service. Note that it might be more than one agent responsible for offering the same service, a list of all these network services location and optionally information about the domain state is the result of such search. This module is the mediator between the mobile agent's request and information about the service agents stored in the service directory of a particular domain.

4.3.2.6 Search Mechanism

Agents will make use of search methods for finding a solution to a specific graph problem. Many issues or problems may fall under this area, such as the traveling salesman or finding the closest and best place for execution.

The first step to solve such problem is the tree construction, after what, the agent has to choose how to explore the tree, it could be either:

- Complete Search, in which case, the agent make sure to get the best solution, however this can be very costly in term of time and space, or an
- Incomplete Search, in which case the agent can find a possible solution in a limited time, and sometimes, this search can also leads to the best solution under certain criteria.

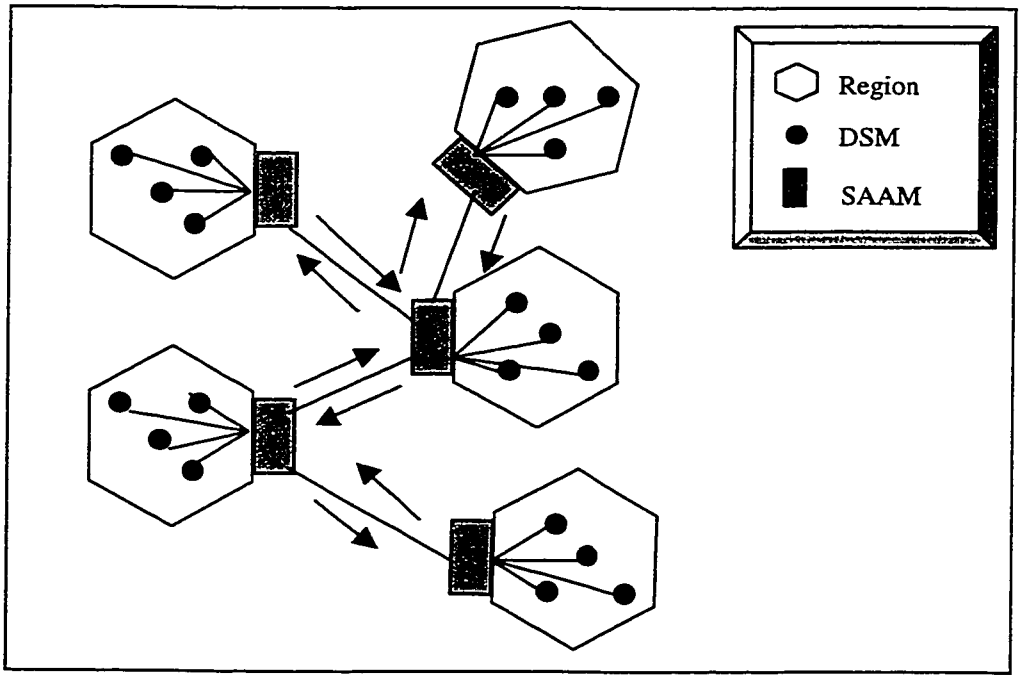


Figure 4.13 Search Mechanism adopted by the DF

By default, the search mechanism adopted for the DF is done in two steps:

a. Scattering the search:

For each SSAM:

- Forward the search request to all DSMs in its region (i.e. local region search), if no satisfying results then
- Create a sending queue and add to it its adjacent SAAM neighbors as specified in the configuration file,
- For each entry on the queue different from the sender, forward the search request and remove the node from the queue,
- When the queue is empty stop.

For each DSM:

- Perform the local search as specified in the previous section for the mapping.

b. Gathering the result:

Each component (DSM or SAAM) involved on this graph search, keep the address of the client that initiated the search, the result is passed back, from child to parent until it reaches the initiator of the search request (i.e. SAAM or DSM on behalf of another entity for instance an agent).

An agent can ask about the services a domain offers by issuing a basic search to its actual DSM. The agent shall also ask for the domain state (i.e. load, performance) to decide whether it is worth to migrate to such environment. Requests for services are described with keywords, service name or service type. The basic procedure is to build a query that describes the requested service. The DSM will be responsible for finding a list of agents that offers the requested service.

While sending this search request, few constraints and options are defined to determine when the search should be stopped.

- Duration time: The default and basic search request has a time-out parameter. The default time-out can be set in the DSM upon the service registration or it can be specified by the sender. Once a search request is received, the DSM initiates the search mechanism and assign to the query a start time and its time out; these values are checked at regular intervals, if the duration time = current time- start time > time-

out the search initiator send back the search result and discard any other results beyond the time-out.

- Maximum: The request can specify the maximum number of answers. The search stops when the result list length reaches this max. When forwarding search requests to other regions, the current SAAM has to decrement the maximum value by the number of answers it has found.
- Hops: This parameter indicates the tract where the search request will be forwarded. Thus, the mobile agent can choose to restrict the search inside the domain, the region, the adjacent region, etc.
- AddressFilter: the mobile agent can specify the region in which the search can be performed by supplying the exact transport address of the requested SSAM. In such case, the default search mechanism doesn't apply; the search query is directly forwarded to the concerned SAAM.

4.3.2.7 DF Services

The following commands are defined to interface and manipulate the DF services:

- *Registration*

An agent is registered with the DF in order to publicize its services to other agents or non-agent based entities. The DF is the mean by which this service can be available to other entities; it plays the role of broker between the client and the registered service.

This action adds the named agent to the list of agents registered with the DSM where the agent resides. The query should provide the agent description required for this purpose, (see service description in table 5.2) with all the mandatory attributes. It may also supply optional additional private attributes to be included in the agent description and stored on the agent's service directory.

Example of use

The client can send a register-request for an agent responsible for selling specific items. This agent is responsible for handling all requests for products and maintains a persistent and updated stock control database. It must also be able to generate bills for the buyer in case where the request id satisfied or generate a stock update when the ordered products

are not currently in stock, this can be achieved by the help of other static agents working on the same group. For such an agent the request will be:

- Local Agent Name: sales_agent _widget
- Language: Java
- Type: Selling agent
- Agent group: widget_selling
- Address: machinename.domain
- Ownership: GoodsforSale Company
- Agent State: Active
- Public: yes
- Service description: selling the widget
- Service type: GoodsSales.Widget
- Keywords: widget
- Properties list:
    - Store location: postal address,
    - City: City name,
    - Phone: phone number

Exceptions and failure reasons

Errors occur when a mandatory attribute is missing for the agent or service description, or when the DSM is unable to accomplish its task due to any database manipulation exception.

- *Searching*

The search service requests information from the DF. The DF relies on its distributed service directory to provide the accurate information in response to a search request. The search is satisfied when the SAAM find an agent service entry in its directory that satisfies the query constraints. This processing is done locally first, but could be extended

to other SAAMs if no local agent service match the query content according to the default search mechanism or the query specifications. A successful search can return one or more agent descriptions that satisfy the search criteria, when no agent entries is found, a null result is returned.

Example of use

The client that initiates the search may ask for all stores that are responsible for selling specific items. The search query may contain for instance the service_keyword: widget, in which case all services that have this keyword on their keywords field are added to the result list with the full description. Combined queries are also possible and useful for limiting the search action to specific regions, for instance the above query can specify also that the stores should be located in Dallas city. A combination of all above fields in the service agent description could be used as filters for the search query. An abstract form of this query would be:

DF_Service_Search

(KeyWord (widget))

(Filters (StoreLocation:Dallas) (language (Java))

(Maximum (3))

This request ask for information about agent services representing any store in Dallas and are responsible for selling widgets and that supports Java language, the maximum number of answers is fixed at 3.

The result for this query would be a list of services-agents description matching the above criteria.

Exceptions and Failures:

The search request fails due to:

- Invalid syntax for an attribute value,
- When the SSAM is overloaded and reject the request due to lack of resources,
- Directory Service access failure.

These exceptions are returned back to the sender.

- *Modification*

This command action deals with any changes concerning the service agent's description in a particular service directory. All attributes that need to be modified are supplied in the modify query. Unlike the other services, this service is restricted to the agent's owner or a similar entity that have the same authority. A successful modification returns an acknowledge while an error exception is returned in case of failure.

The client should submit the agent-service name as well as the attributes to be changed and their values.

Example of use:

The agent's owner may want to change the service description to cope with the current situation or may suspend the agent state meaning that the service is no more available for the time being.

Exception and errors

The modify request fails due to:

- Authorization for performing the modify action failed,
- The specified agent was not found,
- Invalid syntax for an attribute value,
- The request was rejected due to system overload.

These exceptions are returned back to the sender.

- *Removal*

An agent is un-registered in order to remove any related information form the service directory of its HAD. The service agent become no more available to other agent even if it still exist, the DF could no more broker information relating to that agent and any information concerning this agent will raise an AgentNotFound Exception. This action is also restricted to the agent's owner or a similar authority.

Example of use

Only the name of the agent is submitted for this query, the agent's owner may want to stop providing the service or register the service with another domain.

Exception and Errors

The Unregister request fails due to:

- Authorization for performing the modify action failed,
- The specified agent was not found,
- The request was rejected due to system overload.

These exceptions are returned back to the sender.

Chapter 5

Framework Implementation

5.1 Introduction

This section describes the implementation of various components that constitute the agent management system. The project was developed mainly on Windows NT, PC-based machines. We were using the bare JDK for development, Swing for the interface, in addition to other tools such as XML and LDAP Directory.

The developed system focuses on the management features namely that of creation, tracking, agent look-up and scalability. The system design and implementation should be easily extensible. This allows new features to be added, for example, extended security and other communication models. The development of the framework from scratch has a number of advantages: more control over the system and its implementation and also the ability to tailor the framework closely to the problem being solved.

Concerning the user interface, the User Management Interface (UMI) provides access to agent management services (e.g. creation, termination, queries, debugging) and provides a graphical user interface to the distributed MA community. The UMI provides a view of the state of the system and a control interface by communicating with the launched mobile agents and the system. The UMI is not just for information; it is also for control. New agents can be created from the drop-down menus or buttons. Agents can be moved, tracked or killed by sending appropriate commands. The UMI makes it easy for users to experiment with agent concept and have control over the mobile agent' behavior.

5.2 Java and Mobile Agent Development

The object-oriented languages that compile to a platform-independent byte code or an interpreted language hold the most promise for the future development and implementation of framework. Java, developed by Sun Microsystems Incorporated, is an object-oriented language that is very reminiscent of C++ but removes some of the more contentious issues, (e.g. pointer arithmetic). The Java code is compiled to platform-

independent byte code for portability, but migration and dynamic extensibility of byte code are not explicitly supported.

It can be argued that Java has become the most prominent language for writing mobile agent environments and applications [DAV96, GMG96]. Systems using it include JumpingBeans, Aglets, Voyager, and Concordia. The use of this platform neutral language provides the portability requested by the mobile agent paradigm. Any agent system written in Java will have the capacity to take advantage of the facilities or legacy systems on that platform as long as there is a Java Virtual Machine in the system.

5.3 Communication

The communication manager offers basics services for message routing and delivery, hiding the low level system capabilities and handling errors and network failure problems.

5.3.1 Messaging System

The communication service provides an efficient mechanism for entities communication via a messaging system. Each system entity (e.g. User, AMS, ADC, DSM, etc) is presented by an *identity* object. The senders and recipients involved in a communication are identified by this object that contains the following data:

- Location Address: It contains the IP address of the host where the entity can be reached. Since mobile agents don't have a fixed address, this information is not static (e.g. it changes dynamically in course of time) and contains the actual location where the agent resides. This is used to send messages to the target entity.
- Entity Distinguished Name: provides the name of the entity followed by its type, e.g. agent name
- Entity type: describes the entity type, it could be a User, Agent, AMS, ADC, AEE, DSM, SSAM, AgentGroup or Undefined.
- Certificate: contains the public key assigned to the entity.
- Content: contains the information or the request, it could be a KQML format or any Agent Communication Language.

Note that for agent's addressing, the receiver name is sufficient to find the address for the receiver, either by looking up the name in the local AMS or querying the "White Page" service.

The message object is the key element in the communication and implements the Java serialization interface. It encapsulates the data to be transferred, the destination and the contact address for a response. The basic parameters for a message are the message name, message type, sender's and recipient' identity. Other parameters may include requests for delivery receipts, error report handling, message buffering, how the message content has been encoded, priority of the message etc. For instance to support remote exception handling, the raised exception at the destination machine will be encapsulated into the message object itself and sent back to the entity that initiated the communication. The size of messages is flexible and depends upon the data and the message content.

Message could be declared as secure in which case the message will be delivered over a secure channel, using SSL. In case of synchronous communication, the send method of the message, blocks and returns the reply of the recipient's at the other side of the communication link, thus enabling both entities to establish a dialogue channel. In case of asynchronous communication, this method returns immediately after the message transfer enabling asynchronous processing.

5.3.2. Messages Routing

The communication manager will read the information contained in the recipient' parameter for message routing and delivery purposes. This manager is not required to read the contents of the other message parameters and actually might not be able to read encrypted messages.

For each message type, an appropriate constructor is declared to fit the request requirement. In addition to its content, messages are signed by the source and the signature is verified by the destination entity by extracting the public key that is embedded in the sender's identity.

- *Multithreaded approach:*

The overall communication performance depends on many factors, including the threading, the communication mechanisms the applications can exploit, the underlying transport mechanisms and operating systems, and the processor hardware. Entities that are initiating communication should have the option of running on their own thread so that they do not block an entire process (and thus all the other objects in that process) if they themselves are blocked in a synchronous communication. Likewise, entities that are the destination of messages often fulfill server roles and if the server is not multi-threaded each request must wait for the previous request to be completed. This can result in clients being unduly blocked. For message handling, we have opted for a multithreading approach to prevent application blocking and improving overall system responsiveness.

The AMS listens on a particular port for incoming messages and dispatch these messages to the appropriate handler (e.g. look-up request) or recipient (e.g. agent residing on its host) based on message type and recipient's identity.

All system entities communicate with each other via message-passing mechanism. After dispatching, messages are stored on the appropriate receiver (or handler) message queue. The message handling is dynamically performed based on the message name and/or type. Thus additional services and features could be added to the system without affecting or changing the whole system: the new service as well as the appropriate message or request that invoke the service (i.e. handler) will be developed and registered with the other services offered by the AMS.

5.4 Data Repository Model

The data managed by each AMS is of two types; actually this component is responsible for keeping information about the system users and administrators as well as the mobile agent. These data is kept on an LDAP directory. This directory consists of entries containing descriptive information. For example, a directory might contain entries describing users, network resources, AEE, ADC, etc.

The descriptive information is stored in the attributes of the entry. Each attribute describes a specific type of information. For example, attributes describing a person (i.e. user) might include the person's common name, telephone number, and email address.

The entry for an administrator might have the following attributes:

Commom_name: Admin

ID: John Smith

Mail: Admin@company.com

TelephoneNumber: 565-2215

OfficeNumber: A302

An attribute can have more than one value. For example, a person might have two common names (a formal name and a nickname) or two telephone numbers (home and work).

Attributes can also contain binary data. For example, attributes concerning an agent mission result might include a JPEG photo describing a particular item or a song sample recorded in an audio file format.

5.4.1 Organizing the Directory:

Since LDAP is intended to be a global directory service, data is organized hierarchically, starting at a root and branching down into individual entries. Figure 5.1 illustrates an example of a hierarchy of entries in an LDAP directory service concerning agent data.

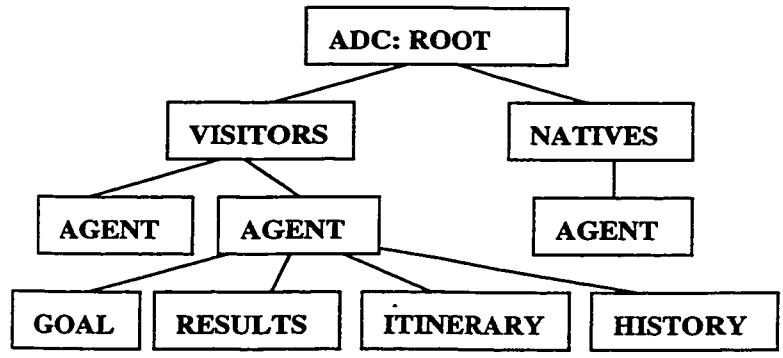


Figure5.1: Hierarchy of Entries in the Directory

The data stored in a directory is distributed among several LDAP servers (one per domain). For example, one LDAP server at Domain#1 might contain entries representing mobile agents residing on its places while another LDAP server at Domain#2 might

contain entries representing the domain users and administrator. The AMS of each domain is responsible of managing these LDAP servers, the request forwarding and AMS interaction defined in the previous chapter allows to refer requests from one server to another one. For example, if the AMS at Domain#1 receives a request for information about a mobile agent in a Domain#2, that AMS can refer the request to the AMS at Domain#2, in this way, LDAP servers can appear to be a single source of directory information. Even if an LDAP server does not contain the information a client request, the AMS can refer to another server that does contain the information.

5.4.2 Overview of LDAP:

The directory service is a distributed database application designed to manage the entries and attributes in a directory. It also makes the entries and attributes available to users and other applications [LDA98]. The Netscape Directory Server is an example of a directory service that manages and provides information about users and organizational structures of users, such as groups and departments. Examples of LDAP clients might include the HTTP gateway to the Netscape Directory Server, Netscape Navigator, and Netscape Communicator. The gateway uses the directory service to find, update, and add information about users. For example, a user might use the directory service to look up someone's telephone number. Another application might use the directory service to retrieve a list of email addresses.

LDAP is a protocol defining a directory service and access to that service. LDAP is based on a client-server model. LDAP servers provide the directory service, and LDAP clients use the directory service to access entries and attributes.

Many LDAP servers support version 2 of the LDAP protocol. This version of the protocol is specified in [RFC95]. The most recent proposed standard is version 3 of the LDAP protocol, which is specified in [RFC97]. Some LDAP servers, such as the Netscape Directory Server 3.0 and its future versions, support this new version of the protocol.

The LDAP v3 protocol includes these new features:

- Specifying controls (both on the server and on the client) that extend the functionality of an LDAP operation.
- Performing extended operations (beyond the standard LDAP operations).
- Using Simple Authentication and Security Layer (SASL) mechanisms to authenticate to the directory.
- Introducing servers DSEs (DSA-specific entries, where a DSA is a directory server) that provide information including the versions of the LDAP protocol supported, a list of the controls, extended operations, SASL mechanisms supported by the server, and the naming contexts of the server (specifying the portion of the directory tree managed by this server).
- Making Servers' schemas available to clients. (possibility to get a directory server's schema from the root DSE.)
- Supporting data in UTF-8 format in Both client and severs side; clients can now request and receive data that is tagged with language information.

5.4.3 Data Tools

- *Connection Manager*

The connection manager tool is a Java program that is manipulated by the internal entities (static agents). It is aware of the relevant parameters for connection to a data source (URL, id and password). The validity of these parameters are initially tested, and if need be, changed and updated. Once a satisfactory connection has been made (after 5 temptations) a reference to a LDAPConnection is returned back to the client (local agent). One particular user can open up to 10 parallel connections. This connection manager is considered as an LDAP client and the interaction with the server is defined by the LDAP protocol.

- *Authentication*

The LDAP protocol also defines a simple method for authentication. LDAP servers can be set up to restrict permissions to the directory. Before a client can perform an operation on an LDAP server, the client must authenticate itself to the server by supplying a

distinguished name and password. If the user identified by the distinguished name does not have permission to perform the operation, the server does not execute the operation.

The option to use encryption for server communication and over the connection is also defined. The encryption is performed using the Secure Sockets Layer (SSL). Encrypted LDAP communications are referred to as LDAPS connections. This SSL is also used to perform other security activities such as message integrity checks, digital signatures, and mutual authentication between servers as well as between servers and clients.

- *Directory Manipulation*

The following defines operations that authenticated clients use to search and update the directory via the static database agent. A client can perform these operations, among others:

- Searching for and retrieving entries from the directory (i.e. looking up the actual agent' location)
- Adding new entries to the directory (i.e. adding users, adding agent result or other proprietary information)
- Updating entries in the directory (i.e. location update, itinerary update, etc)
- Deleting entries from the directory (i.e. remove a service from the DF, remove agent's data, etc)
- Renaming entries in the directory (i.e. renaming a service in the DF, etc)

To perform any of these operations, the client needs to establish a connection with an LDAP server via the connection manager that hides the connection establishment aspect.

5.5. Register Transfer and Services Advertising

Both the Register transfer and Services advertising fall into the problem of transfer and exchange of electronic documents between different system entities (AMS-AMS or SAAM-SAAM). The document form should be machine-understandable and allows automatic and easy processing of data. One solution that fit these requirements could be the use of XML.

- *XML Overview*

XML [GRA99] is primarily intended to meet the requirements of large-scale Web content providers and the processing of Web documents by intelligent clients. It is also expected to find use in certain MetaData applications. XML is fully internationalized for both European and Asian languages, with all conforming processors required to support the Unicode character set in both its UTF-8 and UTF-16 encoding [XML98].

Among the benefits of XML use and deployment:

- Allow industries to define platform independent protocols for the exchange of data, especially the data of electronic commerce,
- Deliver information to software entities in a form that allows automatic processing after receipt by processing data using inexpensive software,
- Allow people to display information in their appropriate way,
- Provide MetaData (i.e. data about information) that will help finding appropriate information and help information producers and consumers find each other, etc
- *Simple example of XML Usage:*

The following is part of the agent's register represented as an XML document. It represents the fixed properties of the mobile agent mission description.

```
<MISSION-GOALS>
  <DELEGATION> PROSPECT, BARGAIN, REPORT EVENTS </DELEGATION>
  <ITEM-CATEGORY> MUSIC </ITEM-CATEGORY>
  <ITEM-DESCRIPTION AUTHOR="A. VIVALDI">
        <TITLE> THE FOUR SEASONS </TITLE>
        <GROUP> VIENNA PHILHARMONIC ORCHESTRA </GROUP>
  </ITEM-DESCRIPTION>
  <PRICE-INTERVAL>
        <LOW> 13 </LOW>
        <HIGH>20 </HIGH>
  <DEVISE> AMERICAN DOLLAR </DEVISE>
  </PRICE-INTERVAL>
</MISSION-GOALS>
```

The XML syntax uses matching start and end tags, such as <Title> and </Title>, to mark up information. The data between tags is called an element; elements may be further enriched by attaching name-value pairs called attributes (for example, Author =

"Vivaldi"). This simple syntax makes the document easy to process by machine (software programs), and remains understandable by humans. XML is based on SGML, and is familiar in look and feel to HTML.

Another example of using the XML ontology would be in describing a service or an item to sell to another entity. For example, the data element tagged "price" would equal the retail selling price and "discount price" would be the price offered to those who make volume purchases exceeding amount. Buyers, using catalogs based on XML tags, would know that the dollar amount returned or "price" is conform to the same definition and refer to the same context.

5.6 Agent Interfacing System (AIS)

This interface provides methods for maintaining a dynamic name and location database of agents. It provides means to locate agents according to the tracking scenario described in chapter 4. It defines also methods for agent management, creation, control, etc. The AIS is an implementation of the AMS services and has to fulfill the following requirements:

- Provide the authentication and logging interface to the user.
- Launch mobile agents on behalf of the user and track their progress and position.
- Organize and preprocess information returned into a form that is suitable to the user. This may involve filtering information, presenting urgent information to the user quickly and rendering information using the tools that the user is most familiar with, etc.

We've adopted a layered architecture for the AIS model. We divide the generic layered architecture into three levels; namely the user/application interface, system and transport level as shown in figure 5.2.

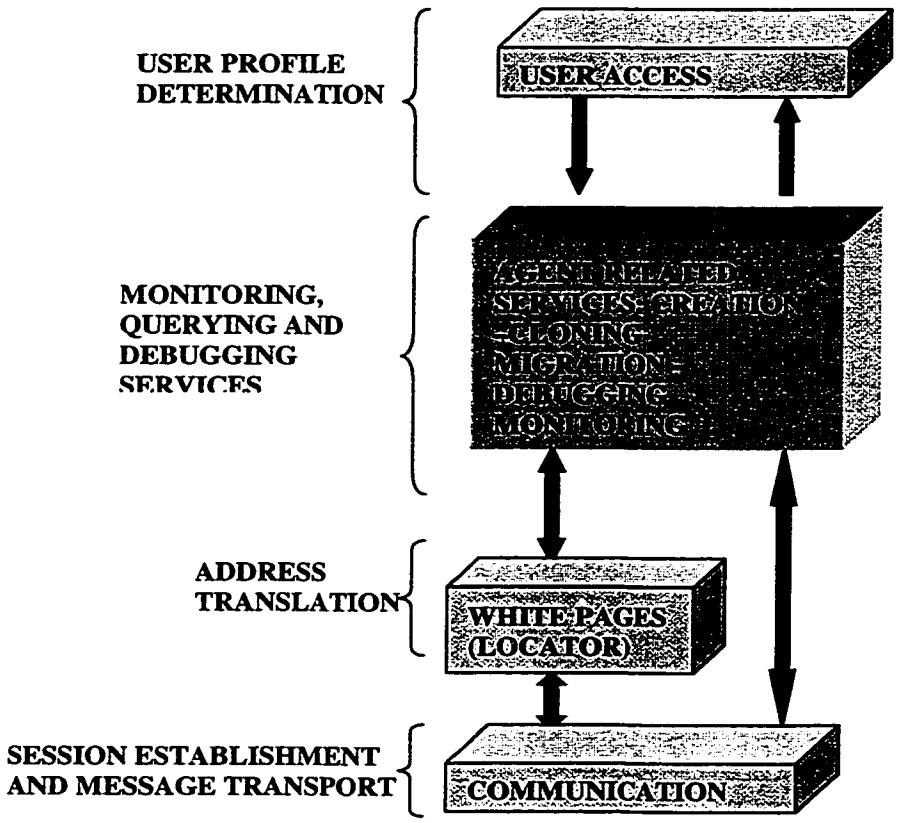


Figure5.2 Layered architecture presenting different agent ' services

The following is a description of the basic services and operations:

- *Creation procedure*

An agent system performs the create_agent operation to create an agent according to a remote client's (i.e. user or agent) request. The result of this procedure consists of returning the full actual name of the created agent if the creation and naming procedure was successful and adding the created agent on the owner's ACL. Using this method, a person can launch his/her mobile agent from a handheld device, because there is no restriction or requirement on the end user device.

Parameters

The client has to submit the following parameters:

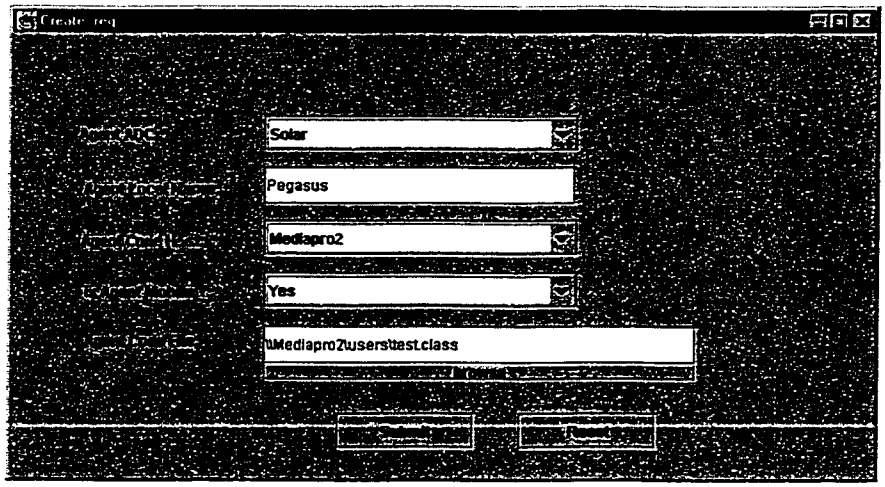
- Local name of the agent, otherwise a numeric id will be assigned as local name,
- The Home Agent Domain: specify the home domain of the agent,

- Class List: specify the list and path of classes necessary to instantiate the agent. It is a reference to the code base containing the necessary class definitions.
- Place Name: specify the Agent Execution environment where the agent will reside. If this parameter is not specified, the AMS creates the agent in a default place within the domain,
- Type: specify if the agent is mobile or stationary i.e. represents a service,
- Others: This parameter can contain anything that the sender needs to inform the remote agent system and may contain proprietary information that is unique to agents of a particular profile. The AMS should be able to decode the meaning of the information in this parameter.

Exceptions and Errors

The creation procedure fails due to:

- The AMS failed to find class definition in the specific path.
- The domain or place name is incorrect,
- The agent is already registered in its HAD
- Missing mandatory attributes for agent creation,
- Invalid syntax for an attribute value,



- *Cloning procedure:*

The only difference with the creation procedure is that the agent's creator has to supply just the Parent Name for the agent and a local specific name, since the agent child will inherits from most of characteristics of its parent. Refer to creation and naming section in chapter 4.

- *Mission, History and Goal Settings:*

After a successful creation, the agent's owner has to set the values for the different attributes in mandatory classes of agent's register. Refer to fault tolerance and data and management section in chapter 4.

Set_Agent_Goal is a method that enables the agent's owner to set the goal description and fills the parameters concerning mission goals as described in the register (application specific). The agent's owner has to specify the full agent name and supply all attributes values for the goal setting.

Set_Agent_History is a method that enables the agent's owner to set the history description (fixed parameters) and fills the parameters concerning this class as described in the register. The basics parameters are life span, owner and creation date.

Set_Agent_Mission is a method that enables the agent's owner to set the mission description and fills the parameters concerning this class as described in the register.

All the above parameters can be remotely changed during the agent lifetime.

- *Agent's Information:*

The user-application may want to check the agent status, get a feedback concerning the agent mission or trace the agent itinerary. These operations are useful in management applications and allow the system manager to monitor the status and behavior of an agent.

Get_Agent_Status is a method that returns the status of the specified agent that could be either Migrating, Active or Suspended. It has one parameter, which is the name of the agent whose status the caller wants to know. It raises an AgentNotFound Exception in case where the AMS could not find the specified agent.

Get_agents_ACL is a method that returned all agents full name listed within the user ACL, as parameter it has the user identifier. This operation might be used by the system administrator to track and monitor the agents within the distributed agent environment.

Terminate_agent is a method that stops execution of the specified agent. It takes as parameter the agent's name and raises either an AgentNotFound Exception or TerminationFailure in case where the system cannot stop the agent's thread control. This method provides a forceful termination of an agent.

Find_agent is a method that returns the actual agent location, it takes as parameter the agent's name that the client want to look-up and raises an AgentNotFound exception. This operation is a key element in the system and is basically used by all other control operations. An agent can use this method to find another agent with which it wants to communicate. This method does not guarantee that the agent will be in the location the method returned for any specified time interval, however the forwarding mechanism used by the tracking system overcame this problem.

Trace_Itineray is a method that returns a list of all past locations that the mobile agent has already visited. It takes as parameter the agent's name and raises an AgentNotFound Exception. This method may be used in management operations to ensure that the agent is respecting the prescribed itinerary in case of fixed one or to detect the malicious host responsible for violating or corrupting agent's integrity.

Get_Agent_Report is method that informs the client of the agent progress. It takes as parameter the agent's name and raises an AgentNotFound Exception. This method helps the user to get feedback from the agent that would be useful for taking decision about aborting or changing parameters.

User-System Interaction

Automated agents are not people; they make decisions and act on them at a vastly greater speed. They are less sophisticated, less flexible, less able to learn, and notoriously lacking in "common sense" than the human brains. Given these differences, it is entirely possible that agent-based applications or transactions will behave in very strange and unfamiliar ways.

The User Management Interface (UMI) provides a level of abstraction for the user away from the details of the mobile agent framework. It is essentially a graphical interface that is linked to the actual Java Classes that represent the real services.

UMI deals with the human-agent interaction part of our agent system and provides the user with a window onto their agents, their status, their results and the mobile agent management system.

5.7 Scenario for manipulating the AMS: Software Delivery Application

This section presents a realistic scenario for agent deployment concerning an agent-based software delivery application. It incorporates most of the services supported by the AMS.

5.7.1 Overview

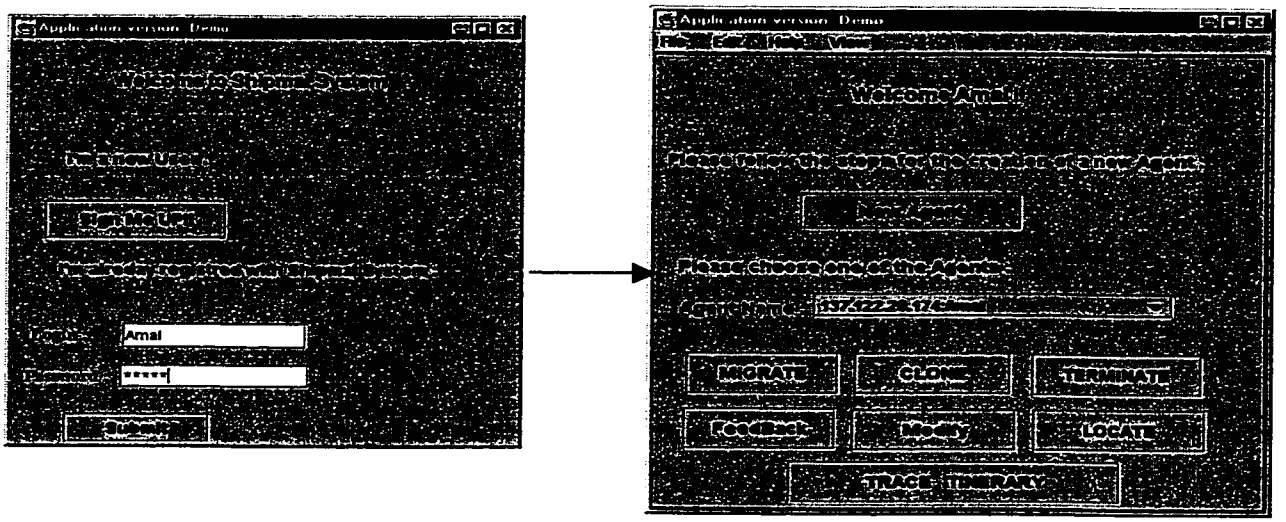
let's assume that C&G is a company that produces a range of innovative software solutions for professional software developers and offers a wide range of products and services for home and office. The company web site provides information about the products and their prices. Registered clients receive notification about any purchased product update or new release.

Let's assume that the development department of a particular company is developing tools using the C&G software, and want to get the new version for all department employees. According to the traditional solution, each employee will go and visit the C&G web site, download the new software version and install it on its machine, the same procedure will be followed by all employees. To release users from this task, C&G decides to adopt an agent-based application to deliver its products.

5.7.2 Example Scenario

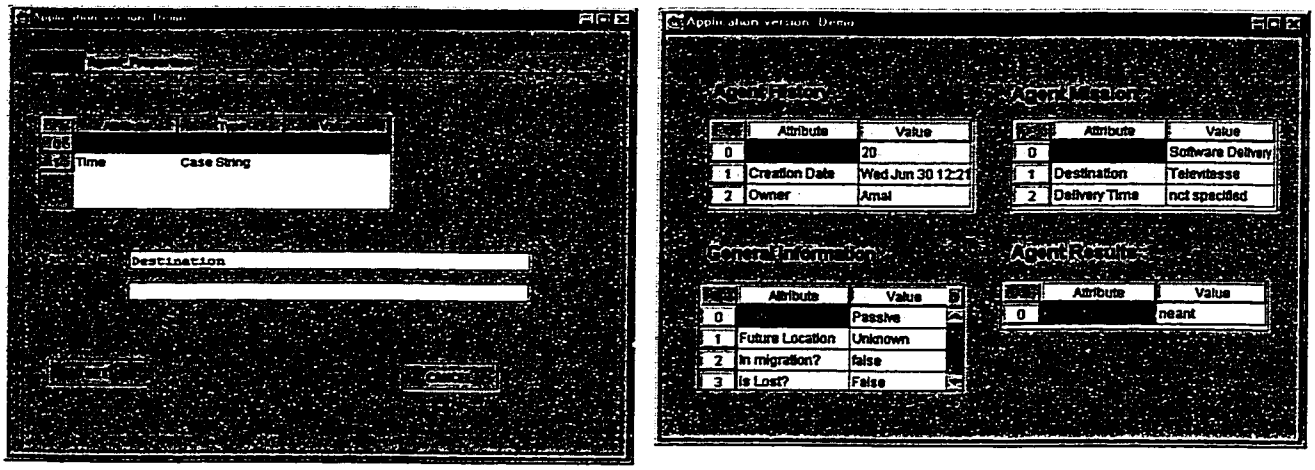
The manager on the development department accesses the C&G web site and orders the delivery of the specific software to different machine hosts at a particular time and leaves. On C&G Company, John the application administrator uses a script-based interface (UMI) that allows him to create, monitor, debug and query his mobile agents. John creates an agent for each specific software or package on a selected host system by choosing NewAgent option on the graphical interface and provides information that will

be used while the invocation of the creation procedure (i.e. determination of agent profile). The HADC uses this agent profile data to create a new instance of the agent and registers it with the AMS in order to benefit from the platform services. (e.g. allow other entities to be aware of it and locate it, using agent characteristics for filtering during a lookup operation , etc). Once this agent is created, John has access to the different interfaces available to control the activity and movement of the agent throughout its life.



Whenever a delivery request for this package is submitted to the agent system, the main agent is cloned, and the proprietary parameters (application-based) that are in our case the destination and the time of delivery are passed to the child agent. This child agent inherits all the basic properties from its parent and is included on the owner ACL. A scheduler is responsible for automatically launching the agent at the specified time. The agent travels to the specified host, performs installation and terminates gracefully.

John can log into the system from any remote node; after authentication, a list of all agents issued from the ACL is returned as result of a successful logging. The administrator can change any of the agents's related parameters according to the user preferences or network's state (i.e. change delivery time due to node crash or bad link).



He can also monitor and debug the mobile agent after its launch by tracing its itinerary (agent's tracking), consulting its work progress, or even terminate the agent in case of misbehavior or integrity violation. All these services collaborate with the tracking module to determine the agent's or the register's actual place. Once the agent achieves its task (successful software installation, or failed installation), it reports the result into its register and is terminated gracefully (no need for forceful termination).

5.8 Scenario for Manipulating the DF

This section presents a realistic scenario for agent deployment in the area of business and commerce application.

5.8.1 Overview

The Quality, price, and the service are a key to the success of the business companies. Also, the ability to effectively manipulate, distribute and access information is a key requirement for common existence within the global market places. However this could be hard to achieve due to communication links, network overload adding the problem of continuing and fast growth companies.

5.8.2 Example Scenario

The following scenario involves most of the mobile agent services that has been introduced in the previous chapter namely:

- Creating an agent to start looking for the items requested by the user (using the UMI),

- Finding places (where to go) that contain the appropriate services (using DF search),
- Migrating from place to place to look for the requested items,
- Tracking agent' moves so that others can find it (Tracking service),
- Gathering and reporting the results back to the user,
- Tracking Agent' progress
- Changing mission parameters (e.g. price interval using UMI)
- Suspending or terminating the agent (using the UMI),

The main actors are shown in Figure 5.3. The user application interacting with the client at his/her terminal is the UMI through which the client can control and monitor all agent activity in this scenario.

Let's assume the following scenario: John wants to buy a flight ticket from Ottawa to Jakarta. As there is no direct flight, he'll have to choose from a large number of options varying in Airlines, prices, flight time and stopovers. Instead of doing this manually, he creates a mobile agent (similar procedure as in the above scenario) through the user interface. The agent mission will be to find the most appropriate flight that match John's preferences who fixes the departure day to be any time on 12/04/00 morning and the arrival date on 120/04/00 on the afternoon. The price interval as well as the flight class is also set in the agent mission parameters. John launches his mobile agent, continue his work and waits for the result.

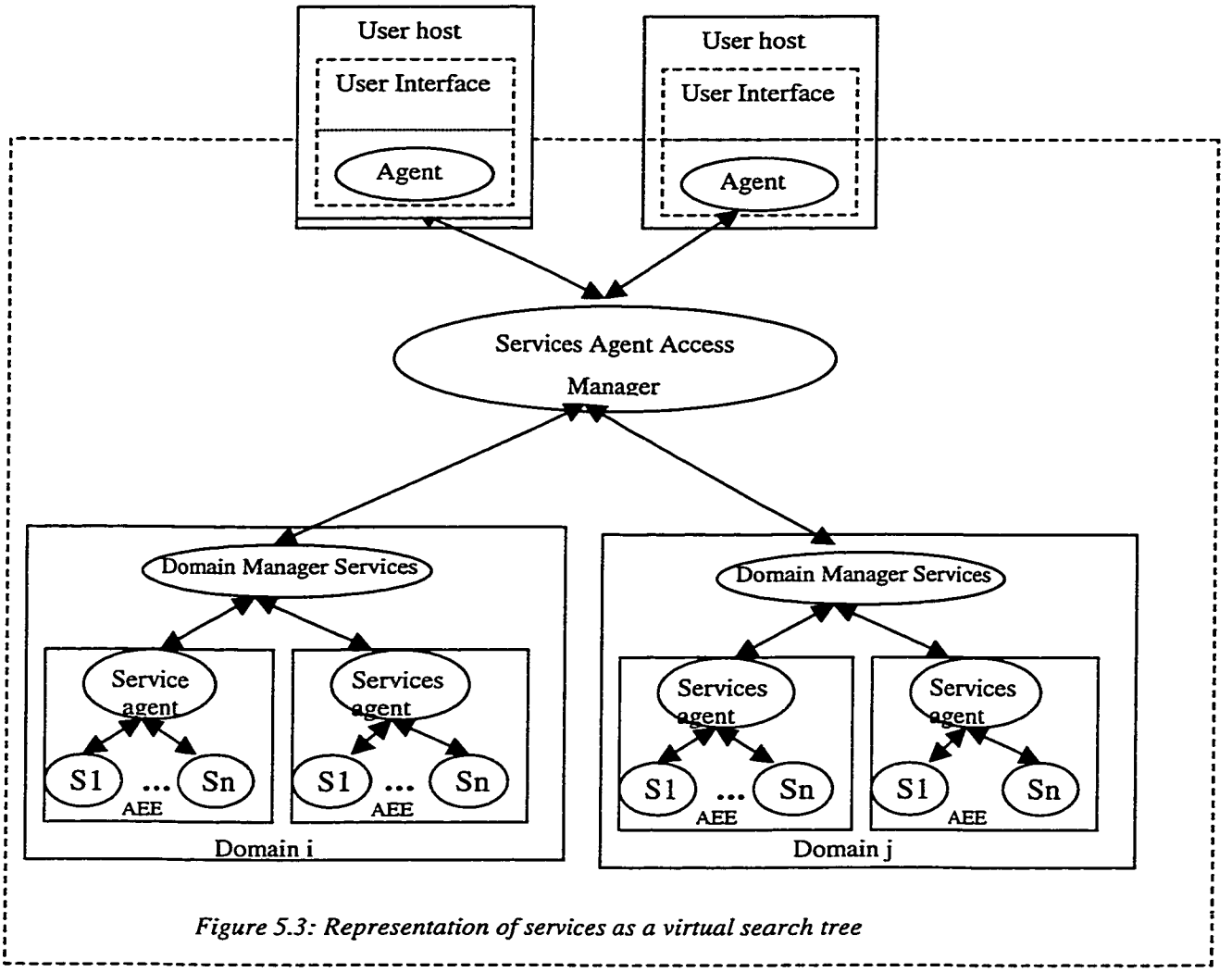


Figure 5.3: Representation of services as a virtual search tree

John does not specify any itinerary to be followed; it is the task of the agent to figure appropriate places that may contain the relevant information. The agent begins by obtaining a reference to the local DSM and send a search request that takes as parameters the keywords that John already set for the mission attributes (i.e. Flight Ticket, Airlines travel). The DSM uses the default search mechanism to find the appropriate sites that may help the agent achieving its mission and forms a list of all the possible service providers sorted according to their regions and best matching. This list forms the preliminary agent itinerary.

The agent starts its migration to the first host on the list, and interacts with the airline service-agent. The agent finds a flight that fit the specific departure and arrival date, but the price was not satisfying. When the agent completes its work at the first place and based on the information found at that site, it migrates to the next host on its list. Note that it is the agent itself that initiates migration, not the user or the agent system.

The agent has visited several sites so far, and gathered relevant information from those sites about the particular flight. All these quite complex searches that requires processing a lot of data from several databases on interconnected systems were locally performed and the agent keeps just the summary of the requested information for later transmission to the user.

After a set period of time, John wants to retrieve the information the agent has gathered so far. John must find and interrogate the agent for its status and progress, and also obtain any relevant data that the agent wants to send back. John uses the UMI to locate its agent, and ask for a feedback concerning the mission progress. The agent system uses the white pages service to locate the actual agent' AMS and interacts with the agent to get back the results that will be displayed at the user interface.

John sees the information on his terminal: he could get an interesting price if he accepts to return back late on the evening. John decides to modify the arrival date time, always through the UMI and waits. The agent receives the new value for the flight arrival time and continues its search using this new criterion. After a set period of time, the agent finds the right flight that fit all John's preferences. As the agent was not allowed to buy the flight ticket, it reports the result to its register and asks to be transferred to its HADC. This was the end of the agent lifecycle (graceful termination). Th report is sent back to John and the information is displayed on his terminal.

5.9 Summary

In this chapter we discuss the implementation issues and solutions concerning the framework for mobile agent management and tracking. The system was developed using Java platform, the data repository model and directory organization for user and agent using LDAP model were also presented. The chapter then continues with two-example

scenario for the deployment of mobile agent in the area of software delivery and business application. These scenarios focus on the use and manipulation of AMS as well as the DF interfaces by the user and the agent.

Chapter 6

Conclusion

6.1 Summary

Mobile agents are a relatively new technology that is fueling a new industry. Due to its important and useful properties, this technology has received a rapidly growing attention over the last few years. Many platforms of MA are under development and form a new direction to software engineering. In addition, there are already various efforts from OMG group (MASIF standard) and FIPA to standardize mobile agent concepts and architectures for open communication and interoperability.

There are many alternatives to mobile agents such as queued RPC, proxy servers and client-server computing, however the problem with these alternative techniques is that each one focuses on a specific problem and is only suitable for particular applications. A mobile agent system on the other hand is a single, unified framework in which a wide range of distributed applications can be implemented efficiently and easily. Agent technology is a step away from direct and interactive tools; agents are developed to perform tasks that would normally be carried out by the user; they make a significant contribution to accessing distributed information, for building applications and then integrating them into a distributed environment.

However, the deployment of mobile agent needs a supporting infrastructure that provides the necessary services for mobile agents. The functionality of such a mobile agent system is to allow the creation of a society of mobile agents and can be compared to that of a middleware for distributed applications. To achieve this, the platform has to supply services for mobile agents as well as the user-applications specify how all these system entities interact with each other to achieve a particular goal.

In the context of this thesis, we presented an approach for building a model for mobile agent management and tracking framework within SHIP-MAI architecture. A lower stratum dealing with low-level concepts (i.e. naming, creation, communication, security,

etc) and an upper stratum dealing with high-level services (i.e. data persistency, fault tolerance, directory assistance and organization, user-system interaction, etc) are presented as the main components of this framework that supply distributed management services. We have also presented the design and implementation of the framework. In particular we presented the model for creation, communication mechanism, agent tracking, fault tolerance and data management and discuss the advantages, drawbacks as well as the cost of such mechanisms.

6.2 Suggestions for future work

The future directions and ongoing work for this research may be summarized as follows:

- Security: Security threats may happen throughout agent management (registration, agent's communication, agent's configuration, human-agent interaction and agent mobility). We have presented the key security risks in agent management and defined some basic solutions to such problem. The mobile agent security is one of the crucial issues for the acceptance of mobile agent systems: although the protection of hosts can be achieved by using conventional and recently introduced mechanisms, a large question remains over the security and protection of the mobile agent from malicious hosts.

- User-Agent Interaction: The part concerning the human interaction with an agent system is either missing or incomplete in most current agent systems. We have presented a model for interfacing the agent system and the human user via a graphical user interface that supports remote agent monitoring and control and it does not put any requirements on the final user device. This part can be extended to support a personal system that can maintain user models and supports their customization either by accepting explicit information about the user or by learning through observing the user behavior.

- Interoperability: The differences among mobile agent systems prevent interoperability and interaction between agents issued from different systems. To promote both interoperability and system diversity, some aspects of mobile agent technology must be standardized. The MASIF and FIPA standards offer basic and common conceptual

model notably in the areas of agent interaction, agent transfer, communication and security.

Given that mobile agent technology has been warmly adopted by developers, and considering the big promises it holds, we believe that this technology would play a key role in telecommunication systems and business applications on the future.

References

[ADL 98] Yariv Aridor and Danny B. Lange. "Agent design patterns: Elements of agent application design" In Katia P. Sycara and Michael Wooldridge, editors, Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98), pages 108-115, New York, May 9-13, 1998. ACM Press

[ATP 97] "ATP draft" Danny B. Lange and Yariv Aridor http://www.trl.ibm.co.jp/aglets/atp/atp.htm

[BAU 98] J. Baumann, F. Hohl, K. Rothermel, M. Schwehm and M. Straßer (1998): "Mole 3.0: A Middleware for Java-based Mobile Software Agents". In: N. Davies, K. Raymond and J. Seitz, 'Proceedings Middleware 98', Springer Verlag London, pp. 355 - 370

[BFK 98] B. Falchuk, A.Karmouch, "Visual Modeling for Agent-Based Applications", IEEE Computer, Vol.31, No.12, pp.31-38, December 1998

[BGP 97] Baldi, M.Gai, Picco " Exploiting Code Mobility in Decentralized and Flexible Network Management", in Rothermel, K. and Popescu-Zeletin, Lecture Notes in Computer Science series 1219, p. 13-26, Springer 97.

[BPM 96] Aline Baggio, Ian Piumarta "Mobile Host Tracking and Resource Discovery". 7th ACM SIGOPS European Workshop, Connemara (Ireland), p. 9-12 Sept 1996.

[CGH 95] Chess D., B. Grosof, C. Harrison, D. Levine, C. Parris and G. Tsudik, "Itinerant Agents for Mobile Computing". Technical Report, October 1995, IBM T.J. Watson Research Center, NY.

[CHK 95] Chess D., C. Harrison and A. Kershenbaum, "Mobile Agents: Are they a good idea". Technical Report, March 1995, IBM T.J. Watson Research Center, NY.

[CLA 96] Chang D. and D. Lange, "Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW". 1996, OOPSLA'96 Workshop.

[CMA 96] Chavez, A. and Maes, P., Kasbah: An Agent Marketplace for Buying and Selling Goods, Proceedings of the First International. Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, (1996).

[COL 98] M. Collier, "Netlets: the future of networking?," presented at the First IEEE Conference on Open Architectures and Network Programming, San Francisco, 3-4 April 1998.

[DAV 96] Stephen R.Davis "Learn Java, the Easy and Quick Way with Microsoft Visual J++".

[DNI 95] F. Dupuy, G. Nilsson, and Y. Inoue. The TINA Consortium: Towards Networking Telecommunications Information Services. - Proc. XV. Intl. Switching Symposium, ISS'95, Berlin, Germany, Apr. 1995, Vol.2/B5.2 pp. 207-211

[FIP 98] FIPA: "FIPA 98 Specification". Foundation for Intelligent Physical Agents, 1998. http://www.fipa.org/.

[FGS 96] Farmer, W.M. Guttman, J.D. Swarup, "Security for Mobile Agents: issues and Requirements", Proceedings of the 19th National Information Systems Security Conference, p. 591-597, Baltimore, October 1996

[FKK 96] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol version 3.0. Technical Report <draft-freier-ssl-version3-02.txt>, IETF, November 1996.

[GMG 96] J. Gosling and H. McGilton. The Java language environment. White paper, Sun Microsystems, Inc., 1996. http://www.java.sun.com

[GRA 99] Ian S. Graham and Liam Quin. XML Specification Guide. New York, NY: John Wiley & Sons, 1999

[HKT 97] Melissa Hirschl and David Kotz. AGDB: A debugger for Agent Tcl. Technical Report, Dept. of Computer Science, Dartmouth College, Hanover, NH, February 1997.

[HST 96] M. van Steen and F.J. Hauck and A.S. Tanenbaum: "A Model for Worldwide Tracking of Distributed Objects" , Vrije Universiteit, TINA '96, Conference, Heidelberg, Germany, September 1996, Pages 203-212,

[IDM 91] J. Ioannidis, D. Duchamp, and G. Q. Maguire. IP-based protocols for mobile internetworking. In Proceedings SIGCOMM'91, pages 235-245, Zuerich, Switzerland, 1991. ACM Press.

[JUM 99] Jumping Beans White Paper, Ad AstraEnginnering Corporated, 1999

[JWO 95] R. Jennings and M. Wooldridge N.R. "Applying Agent Technology", Applied Artificial Intelligence: An International Journal, Taylor & Francis London, 9 (4) 1995, 351-361.

[KGR 96] Kotz D., R. Gray and D. Rus, "Transportable Agents Support Worldwide Applications". In Proceedings of the 7th ACM SIGOPS European Workshop, in September 1996, Connemara, Ireland.

[KRA 97] Krause S., MAGNA-A DPE-based Platform for Mobile Agents in Electronic Service Markets. ISADA '97 The Third International Symposium on Autonomous Decentralized Systems, 9-11 April 1997, Berlin, Germany

[LDA 98] Netscape Directory SDK 3.0 for C Programmer's Guide http://curiac.acomp.usf.edu/db/ldap/sdk/contents.htm

[LWS 98] S. Lazar, I. Weerakoon and D. Sidhu, "A Scalable Location Tracking and Message Delivery Scheme for Mobile Agents," Proceedings of the 7th IEEE WETICE Conference, Stanford, June, 1998.

[MIT 97] "Software Agents", Jeffrey M. Bradshaw (ed.), MIT press.

[OMG 98a] OMG: CORBA Services: Common Object Services Specification. Object Management Group, December 1998.

[OMG 98b] OMG: Mobile Agent System Interoperability Facilities Specification (MASIF). March 1998.

[RDG 98] A. D. Rubin and Jr. D. E. Geer. Mobile code security. IEEE Internet Computing, 2(6):30-34, 1998.

[PLV 94] Pahlahvan and Levesque, Wireless Data Communications, Proc. of the IEEE, September 1994, pp 1398-1430

[RED 97] F.E. Redmond DCOM: Microsoft Distributed Component Object Model. IDG Books Worldwide, Foster City, CA, 1997.

[RFC 95] Lightweight Directory Access Protocol http://info.internet.isi.edu:80/in-otes/rfc/files/rfc1777.txt

[RFC 97] LDAP Specification http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2251.txt.

[SHI 98a] V.A Pham, and A.Karmouch, "Mobile Software Agents: An overview", IEEE Communications, p. 26-37, July 1998.

[SHI 00] Masc. Thesis. R. Tkito and A.Karmouch, "A Secure High Performance Mobile Agent Infrastructure". Electrical Engineering , University of Ottawa

[SUN 97] Sun Microsystems: Object Serialization Specification, JDK online documentation edition, 1996,1997.

[URL1] http://www.tinac.com/about/principles_of_tinac.htm

[URL2] http://www.webproforum.com/tmn/

[URL3] http://www.java.sun.com/marketing/collateral/javarmi.html

[URL4] http://www.cs.uit.no/DOS/StormCast/

[URL5] http://www.genmagic.com/MagicCap/

[URL6] http://www.cs.tuberlin.de/ cs/research/english/projects/inamos.html

[URL7] http://www.jumpingbeans.com

[URL8] " The Aglets Workbench" http://www.trl.ibm.co.jp/aglets/

[URL9] http://agent.cs.dartmouth.edu/network/

[URL10] Java Virtual Machine Profiler Interface (JVMPI).
URL: http://www.javasoft.com/products/jdk/1.2/docs/guide/jvmpi/jvmpi.html

[URL11] Proposed Java Real-Time Extension Specification.
URL: http://www.sdct.itl.nist.gov/~carnahan/real-time/sun/index.html

[XML 98] Extensible Markup Language (XML) 1.0. W3C Recommendation 10-February-1998. Editors: Tim Bray (Textuality and Netscape), Jean Paoli (Microsoft), and C. M. Sperberg-McQueen (University of Illinois at Chicago).