Instituto Superior de Engenharia do Porto

Mobile Agents

Sistemas Baseados em Agentes

Mestrado em Engenharia Informática

Ramo Tecnologias do Conhecimento e Decisão

1090557 – Diogo Martinho

Abril 2014

Index

1. Abstract 3
2. Agents and Mobility 4
3. Weak and Strong Mobility 5
4. Advantages and disadvantages 6
5. Applications 9
6. Mobile Agents Systems 11
7. Conclusion 18
8. Bibliography 19

1. Abstract

Throughout the last decades, there has been an increasing need for developing intelligent systems capable of solving problems that would help people with their lives and routines. An area in Artificial Intelligence was brought into the programming paradigm as an attempt to deal with that necessity.

Agent-based systems were created, made up of several interacting software agents cooperating in order to solve issues where the opinion from single problem solver would be inadequate.

These agents [1] *"can be classified according to a set of possible characteristics which will be used to describe an agent, such as Mobility, Intelligence, Reactivity, Learning, Flexibility, Autonomy"*, and others.

In this paper, it will be discussed the concept of mobile agents as well as the importance of mobility in software development.

It will be exposed the two types of mobility including their main characteristics and other relevant details.

It will also be presented the advantages and disadvantages for using mobile agents.

It will be shown all the relevant applications for mobile agent technology and then several existing mobile agent systems and their features and architecture.

2. Agents and Mobility

Mobility is one of the most important characteristics found in software agents and software development. It has been used in many technological fields such as software distribution, network management and video conferencing as well as in programming applications.

Agent mobility described as [1] *“the ability to transfer itself to a different computational location”* allows the distribution of resources throughout a network between several machines, relocating them according to the hardware capacity involved in the execution of the necessary operations.

This is possible due to the autonomous nature of the mobile agent, allowing him to schedule its own computation at a specific time at a specific location, according to the execution times and that way avoiding cramming that is often verified in closed environments where these agents are not allowed to operate. They can also decide when to suspend execution and then transfer to another machine, reactivating its function upon arrival.

While a mobile agent is moving across the network, a number of resources is also being transferred, and it is important to assure there is no overuse or waste of those resources. Furthermore according to [2], “*Computational resources exported by mobile-agent hosts are much more difficult to control. The host site generally has no assurance that an arbitrary agent's actions will have any beneficial effects.*”. This means that the host system usually doesn’t know whether sharing resources with an arbitrary agent could be advantageous or not. Many problems can be adverted from the incorrect use of resources such as additional risks from denial-of-service attacks or even additional congestion from incorrect use of mobile agents.

A correct use involves efficient agents that have the ability to evolve in a distributed and heterogeneous environment, allowing them to travel through different machines and operating systems during their lifecycle and having the ability to perform correctly despite the policies and goals inherent to the organizations taking advantage of them.

3. Weak and Strong Mobility

According to [3], agents can be divided into two models according to their type of mobility (weak and strong).

Strong mobility is a process in which the full state of the agent is saved before being transferred and then restored after the agent relocates into the new machine. That state can be defined in the code of the program, as a set of references towards available resources together with their execution state. After saving all the required information the program will be transferred to a different machine and then resume its previous execution. This becomes transparent to the programmer considering that both the stop and resume for the program execution can occur at any given time.

There are a few obstacles towards the implementation of strong mobility agents. First of all, by saving the full state of the program, it will increase the overhead at the moment of the transfer.

Secondly, and because Java has become more and more the preferred pattern when developing mobile agents, it still doesn't offer a mechanism that directly allows the implementation of strong mobility. This is related with the fact, that the Java Virtual Machine, for safety reasons, doesn't provide full information on its applications state and execution.

There have been many workarounds for this issue such as extra insertion and manipulation of the application bytecode in order to retrieve the required information of the application state, however because those alternatives aren't easy to develop, strong mobility is not often used.

In comparison we see more agents relying on **weak mobility** implementations, which do not record the full state of the application, but instead transfer the code of the program as well as the state of the object and any other referenced variables. This type is not so transparent since it is necessary to manually define checkpoints where the mobility can be performed without affecting the integrity of the application. This forces the programmer to set explicitly several areas of the code where the execution can be interrupted, and then reestablished on a different machine.

Unlike strong mobility, there are several mechanisms in Java which allows saving the state of the object as well as its variables, making it easy to send an application from a machine to another.

Therefore weak mobility ends up being the most commonly used type of mobility.

4. Advantages and disadvantages

4.1. Advantages

Danny B. Lange and Mitsuru Oshima [4] believe that the *“interest in mobile agents is not motivated by the technology per se but rather by the benefits agents provide for creating distributed systems. There are seven benefits, or good reasons, to start using mobile agents”* which are:

Network load reduction – When dealing with a specific task, distributed systems usually demand high network traffic due to several communication protocols interacting with each other. This issue can be diminished thanks to the mobile agents’ ability to store a conversation and send it to a host where interactions can be performed locally. Agents also decrease the flow of data by storing it at remote hosts, which allows large volumes of data to be processed locally and not over the entire network.

Network latency reduction – Manufacturing processes involve the use of systems such as robots which should be ready to function correctly in a real-time environment, and any latency can be problematic. Agents deal with this problem by acting locally and that way undertake the controller’s actions directly.

Protocols encapsulation – One of the main issues relate with data transfer in distributed systems is the is the fact that each host deals with the protocol’s implementation in order to send and receive incoming data, whose maintenance is problematic if the protocol keeps evolving and demanding more security or efficiency measures. Agents based on proprietary protocols move to remote hosts accordingly.

Autonomous and asynchronous execution – Another problem is related with the fragile and expensive connections between mobile devices and the network which usually last longer than what is supported by this sort of equipment. This can be avoided by dispatching mobile agents in a network, which will be responsible for the execution of specific tasks. They operate autonomously and asynchronously and are later collected by the mobile device.

Dynamic adaptation – Mobile agents are fully aware of the environment where they operate and have the ability to react according to changes. Besides that multiple agents can distribute, if necessary, among the hosts in a network in order to maintain an optimal configuration when dealing with a specific problem.

Heterogeneity – Network computing is mainly heterogeneous, which includes both hardware and software perspectives and because agents are independent of the computer and the layer they operate in, they are able to offer excellent conditions for unified system integration.

Robust and fault-tolerant – Fault-tolerant distributed systems are easily built thanks to the ability for an agent to react and adapt to an unfavorable situation. An example of this is when a host is forced to shut down and the agents executing on that machine are warned and then transfer themselves into a new host and then proceed with their operation.

Benjamin Pierce [5] has also alerted for other positive aspects of mobile agent computing:

High quality, high performance, economical mobile applications – Mobile agents make the best use of the network and its resources as they travel through all the hosts by taking advantage of their services. By processing the data at the data sources instead of fetching it somewhere else this will contribute for a higher performance and avoid unnecessary data transfer costs.

Portable, low-cost, personal communication devices – All the features related to network support such as security are stored in a lightweight central which manages the movement of the agents in a network. Besides that agents also present a self-contained programming model. This allows recording a small footprint on user devices without compromising the functionality for the application.

Secure Intranet-style communications on public networks – Security over public networks is achieved by using agents that carry user credentials as they travel. Those credentials will be authenticated at every point of the network. Besides that the data transferred with an agent is always encrypted.

4.2. Disadvantages

The first problem related to the use of mobile agents is associated with **security and integrity issues** mainly due to the fact mobile agents is still a relatively new technology [6], meaning that there barely are consistent mobile agent development tools available so far (most of the versions are still in alpha and beta state) and because of that it is common for unknown "bugs" and vulnerabilities to appear that may compromise both the state of the machine and the agents themselves.

The mobile agent management is also problematic. By increasing the number of agents traveling throughout the network, it **involves more control in all communication processes** which will affect the total overhead in the network.

Another issue is related with the fact that traditional security tools are not ready to deal and adapt to mobile agent technology meaning that, even if a mobile agent can provide fault-tolerant properties to these tools, it will still be **vulnerable against security threats and other risks**.

Mobile agents may also **require installing their working environment** in each node of the network on order to operate correctly.

Hervé Paulino [7] has identified other issues in mobile agent computing such as **code portability**, which demands standardization in order for the code, upon an agent's arrival, to be properly interpreted and compiled to native code, despite the mobile agent system installed on the host machine.

He also refers that a mobile agents **are still not able to protect a machine from the attacks of other malicious mobile agents**, without restricting the access rights for an agent to operate under that machine.

He concludes by saying that *"It is unreasonable that any Internet service will be willing to change radically from the client/server paradigm to the mobile agent paradigm. An evolutionary path from current systems to mobile agent based systems must be provided."*

5. Applications

The use of mobile agents has been growing over the years and we see more and more applications taking advantage of the mobile agent paradigm [4]:

E-commerce – Commercial transactions often require real-time access to remote resources that can managed by mobile agents. A good example of this is the agent-to-agent negotiation. Agents negotiate between themselves and use different strategies so that they can complete the tasks they've been assigned with. Having an agent that could act and show the intentions of its creator as well as negotiate on its behalf would be the ideal scenario.

Personal assistance – We've seen already that mobile agents have the ability to execute their tasks at a remote machine on behalf of their creators, and thanks to that they are able to operate despite the state of the network connection. They are able to perform actions such as scheduling meetings with other agents.

Secure brokering – This application happens when it's necessary for several mobile agents to collaborate and there is no certainty that all of them are trustful. Because of that, the parties agree on a specific and secure meeting host where the collaboration will take place, and that way avoid the host taking the side of one of the visiting agents.

Distributed information retrieval – Mobile agents are used in order to access remote information instead of transmitting large number of data throughout the network. They also have the ability to operate during lean hours which helps the overall performance of the entire network.

Telecommunication network services – Mobile agents provide flexibility and efficiency to these systems by controlling their physical size which is an essential requirement so that they can offer user customization and dynamic network reconfigurations as part of the services available.

Workflow applications and groupware – These systems are based on supporting the flow of information between coworkers. Mobile agents provide mobility and autonomy to a workflow item allowing them to move through the organization.

Monitoring and notification – Mobile agents have the ability to monitor the information from a source, despite the system from where that source came from. They can also be dispatched and wait for information before becoming available. This means that a monitoring agent usually lasts longer than the process that created it.

Information dissemination – Mobile agents can disseminate information, such as news or updates, and act accordingly, for example, by installing new software components directly on a remote computer and having the autonomy to keep that machine always updated.

Parallel processing – Mainly used for processing parallel tasks, mobile agents can clone themselves and administer tasks which will be run at the same time through different processors. In a traditional sequential approach this could compromise the performance of a single processor if it didn't have the necessary processing power to handle those tasks.

6. Mobile Agents Systems

Several mobile agent systems have been developed over the past years that take advantage of some of the characteristics mentioned before. In this section it will be described some of those systems and the reasons behind their development.

6.1. Ara

The Ara (Agents for Remote Action) System [8] was developed in the University of Kaiserslautern, and its function is based on integrating mobility with existing programming concepts. For that, the Ara agents travel through different programs and have the ability to understand and use many different programming languages associated with them.

The concept behind the Ara system programming model is rather simple. Like mentioned before, an Ara agent moves between places and use resources such as services in order to complete the task it has been assigned to.

A place is located in a remote machine and it will obviously force security restrictions to the visiting agents before providing them the resources or information they may need. The agents are identical to traditional programs where they use a file system, a user interface and a network interface. This allows separating the Ara architecture from high-level specific concepts and from complex distributed services.

Due to these portability and security issues, most of the systems, like Ara use a virtual area instead of running their agents directly over the real machine, processor or operating system. For that it requires an interpreter and a run-time system, called *core*, which will hide the details of the remote machine and restrain the actions of the agent on that environment. The interpreter isolates language-specific issues and the core collects all the language-independent functions.

Ara does not create a new agent programming language, but it provides an interface instead, which is compatible with many existing languages making it possible to employ several interpreters for each different language on the top of the core.

The structure of the Ara architecture and its components using two interpreters for language A and B is shown above:

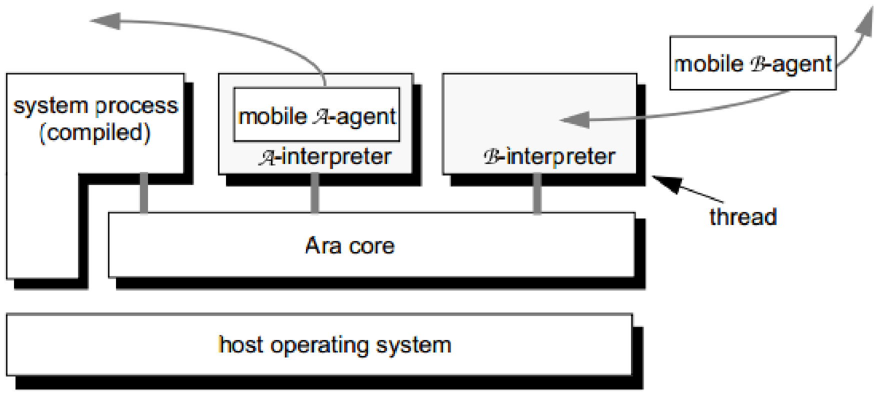

Figure 1. High-level view of the Ara system architecture [8].

Ara agents use a fast thread package and are executed using parallel processes allowing them to become transparent to wherever they decide to move. Besides that the system also requires processes for executing internal tasks (*system-process*) and that way keep the agent management under the control of the Ara core. This will improve the system performance significantly.

6.2. D'Agents

D'Agents [9], also known as Agent Tcl, is a mobile agent system developed by the College of Darthmouth in Hanover, and has been used in many academic and industrial research labs. It focuses on security issues and provides full protection to the machines using this system. The agents are built using Tcl Java and Scheme scripting language and relies on a server which will be the main component ran in the remote machine.

This system architecture is divided into four different layers or levels, and each of those levels will present specific characteristics and responsibilities.

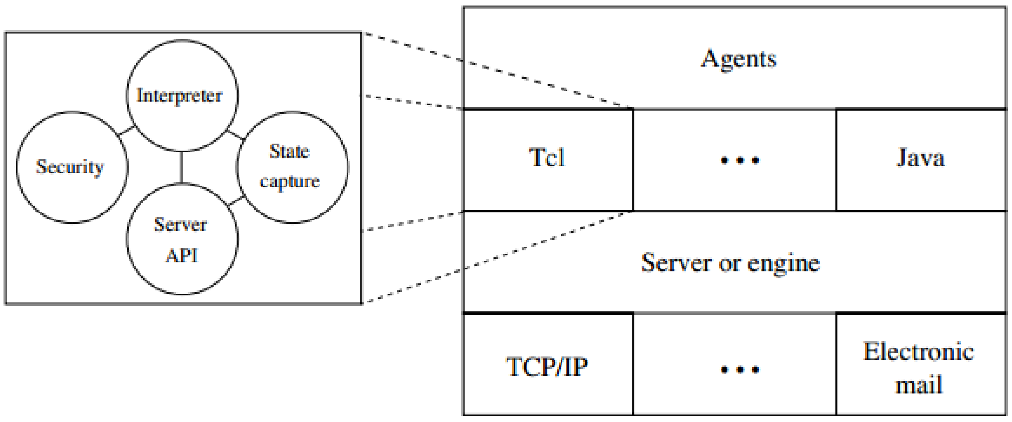

Figure 2. High-level view of the D'Agents system architecture [9].

The first and lowest level is an interface responsible for all the transport mechanisms.

The second level represents the server mentioned before running in the machine. It is responsible for several tasks: tracks all the agents running on its machine; provides communication mechanisms between agents; validate and authenticate incoming agents, restarting them at the right working environment; saves the agents state which can be restored in case of machine failure.

The third level is the interpreter, responsible for the execution of each environment according to the language identified. There are three types of interpreters used for this purpose: Tcl interpreter, Scheme 48 interpreter and Java Virtual Machine. Whenever a new agent arrives, the server starts the right interpreter which will identify that agent execution language and then run it.

The last level is the agents and consists in their execution by the interpreters, and the use of resources provided by the server allowing them to communicate with other agents.

6.3. Aglets

Aglets [10] are mobile agents created using Java programming language. They can be created and executed though the AWB (Aglets Workbench) which has been developed by the IBM in Japan. This workbench includes the ATP (Aglets Transfer Protocol) and AAPI (Aglet Application Programming Interface) components that will be used in order to control all the aglets execution, features and interactions.

An aglet is a serialized object that travels through the network visiting aglet-enabled hosts. Like any other mobile agent, it has the ability to stop, pause and resume its execution in a specific host. When it migrates it will also transport its data and execution code. Because this system is implemented using Java language the code is independent from the platform and can be easily executed in almost any computer. Before being executed, the aglet will create its own execution thread and proceed with the task it was assigned with, and respond to any incoming messages or requests.

The aglet architecture is made up of several mechanisms:

Aglet – The agent itself that travels through the network.

Context – The aglet's workspace. It is a stationary object that controls the environment in where the aglet operates.

Proxy – A proxy is a representation of the aglet. It protects the agent by hiding his true location and controls the access to its public methods.

Message – The message is what is transmitted between the agents. It is usually an object containing data and can be passed both synchronously and asynchronously.

Message Manager – Controls the flow of the transmitted messages as well as the concurrency for several incoming messages.

Itinerary – The map representing the agent's travel plan through the network.

Identifier – Bound to each agent in order to identify it, and is globally unique and immutable.

The following figure represents the aglet lifecycle:

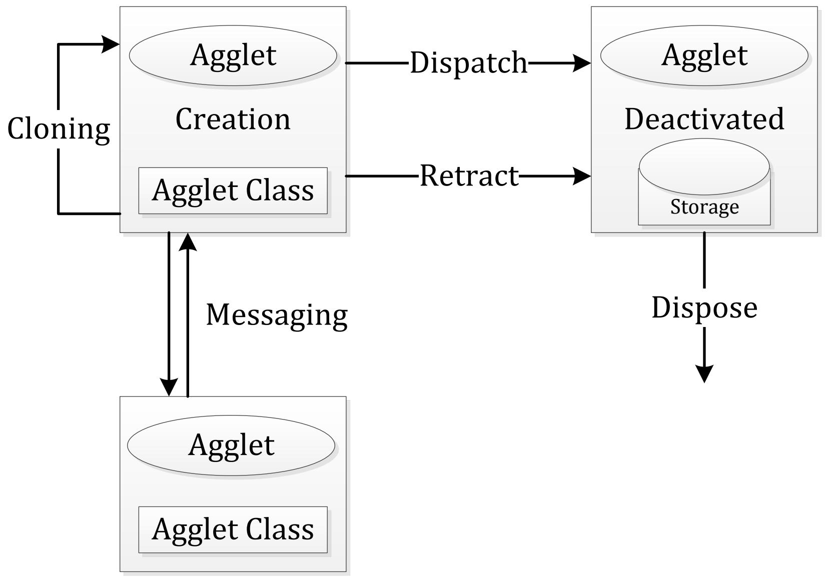

Figure 3. Aglet Lifecycle [11].

Creation – Creates the aglet, which includes assigning it with an identifier, a context and then initializes the object. Only after that, the aglet will be ready to be executed.

Cloning – Copy an aglet and creates an object almost identical only having a different identifier and restarting the execution in that copy.

Dispatching – Process of moving the aglet between two different contexts. It removes the aglet in its original context and moves it into the destination environment where its execution will be restarted.

Retraction – Removes the aglet from a context into another from where the retraction was requested

Deactivation – Removes the aglet temporarily from its context into a storage area.

Activation – Restore the aglet into the context from the storage area.

Disposal – Stops the aglet's execution and removes it from its context.

Messaging – Exchange of messages between aglets.

Naming – The process for binding the identifier to an aglet.

Aglets are also classified in two categories according to their reliance in the system. It is critical to assure that a host doesn't allow an agent to compromise its security, and because of that aglets are divided into **trusted and untrusted** and their actions in the system will be restricted accordingly, for example, to allow an aglet to access a file system, network or other aglets.

The final decision to trust an aglet, however, will always be up to the host.

6.4. Concordia

Concordia [12] is a mobile agent development framework. It has been developed using Java programming language in order to assure the independency and interoperability between different mobile agent applications and its main goal is to provide full coverage and support for agent's features, such as state, security, flexibility, and mobility.

The Concordia system runs under a JVM (Java Virtual Machine) and includes a Concordia Server and at least one Concordia Agent. The JVM allows the system to be run in any machine.

There can be many Concordia servers spread throughout the entire network in different nodes or machines, and they are all aware of each other and connect for every agent's transfer.

In order to successfully transfer an agent to a different host, it first creates an Itinerary, which is an object containing the traveling plan, and then invokes the Concordia server specific methods which will inspect that itinerary and determine the destination. An image of the agent is then saved and transferred to that destination.

After transferring, an agent queues up for restart and then execution in the new host, and only after will be able to execute the method defined in the itinerary.

The agent permissions and access rights to services and information are always under administrative control at all times.

The Concordia system is made up of several components, each having different responsibilities:

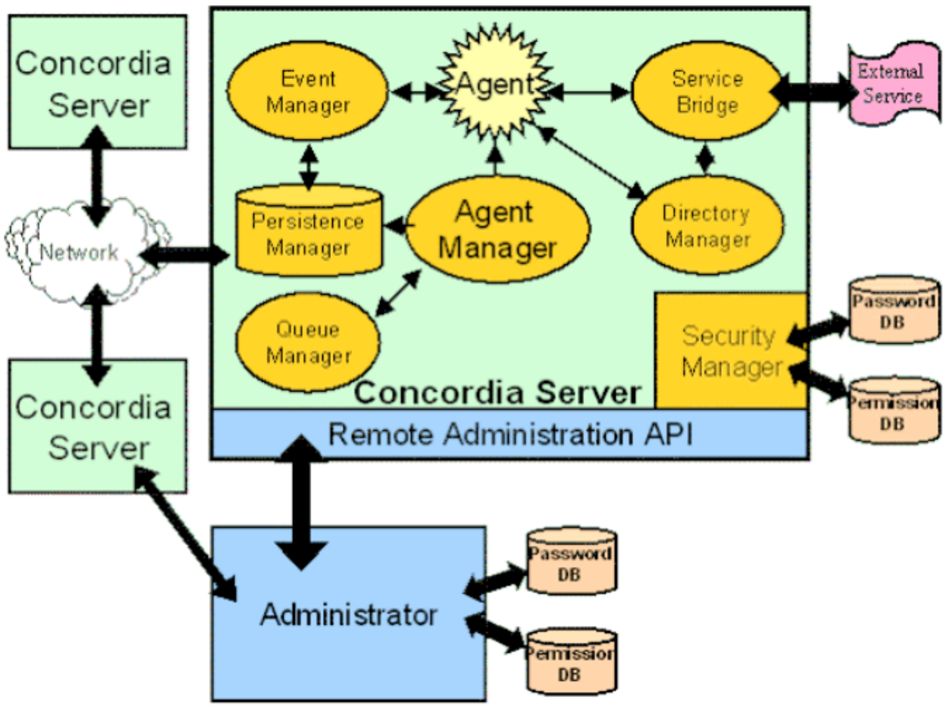

Figure 4. High-level view of the Concordia system architecture [12].

Concordia Server – Installed and running at every node in the network and is responsible for transferring Concordia agents by calling implemented methods

Agent Manager – Provides an infrastructure for allowing agents to be sent and received in a node. It is also responsible for the agent lifecycle, including creation, execution and destruction.

Administrator – Manages all the services accessed by the agent and monitors the agent throughout the network.

Security Manager – Controls all the agent authentications, protects the server resources, and ensures the security and integrity of agents while moving in the system.

Persistence Manager – Checkpoints the state of the agent in the network and in case of system failure, has the ability to restore an agent. It is also transparent because it doesn't need any monitoring or control, in order to perform its functions, but it can still be managed if necessary.

Event Manager – Controls all the events associated with an agent, such as registration, posting or notification.

Queues Manager – Prioritizes agents' execution and schedules them accordingly.

Directory Manager – Responsible for all the naming functions in the Concordia System.

Service Bridge – Provides an agent with services available at several hosts in the network via an Interface.

Agent Tools Library – Responsible for all the classes needed in order to implement Concordia agents.

7. Conclusion

Mobile agent technology has slowly, yet steadily changed the current concept of distributed systems and the way they operate. The introduction of mobile agents in network management brought huge improvements but also new risks.

Security is the major obstacle for believing in this new technology, and many challenges still lie ahead before we are able to overcome this problem, such as code standardization and better development frameworks.

It is my belief that mobile agents are being misunderstood as a dangerous rather than powerful technology because of all the security and development issues mentioned before and not because of all the important advantages they can bring to real systems.

Only when the technological community is able to understand and accept mobile agents, a door will open with many new opportunities and a certainty that mobile agents will be here to stay and succeed.

8. Bibliography

1 - Ramos, C., Silva, A.: Sistemas Baseados em Agentes. DEI-ISEP (2006-2008).

2 - Bredin, J., Kotz, D., Rus, D.: Economic Markets as a Means of Open Mobile-Agent Systems. Dartmouth College (1999).

3 - Lacerra, I., Coraini, T.: Agentes Móveis e migração de processos. Instituto de Matemática e Estatística, Universidade de São Paulo.

4 - Lange, D., Oshima, M.: Dispatch your agents; shut off your machine. General Magic, Inc. Sunnyvale, California.

5 - Pierce, B.: Mobile Agent Computing, a White Paper.

6 – Karygiannis, T.: Network Security Testing Using Mobile Agents. National Institute of Standards and Technology .

7 – Paulino,H.: A Mobile Agents' Overview Departamento de Informática, Faculdade de Ciências e Tecnologia Universidade Nova de Lisboa (2002).

8 - Peine, H., Stolpmann, T.: The Architecture of the Ara Platform for Mobile Agents. University of Kaiserslautern, Germany.

9 - Robert, G., Kotz, D., Cybenko, G., Rus, D.: D'Agents: Security in a multiple-language, mobile agent-system. Darthmouth College.

10 – Lange, D.: Java Aglet Application Programming Interface (J-AAPI) White Paper. IBM Tokyo Research Laboratory (1997).

11 – Karjoth, G., Ohsima M., Lange, D.: A Security Model for Aglets. IBM Research Division, Zurich Research Laboratory (1997).

12 – Peng, J., Li, B.: Mobile Agent in Concordia. Mathematics and Computer Science Department, Kent State University, Ohio, USA.