Survey and Analysis of Intelligent Mobile Agents

Nagi Nabil Basha

School of Computer Science
McGill University, Montreal

March 2002

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

Abstract

The notion of mobile agent, a software component that can move autonomously between the different nodes of a network is gaining wide popularity in business and in academia. The term mobile agent was first introduced in 1994. Since then, lots of research has been carried out in various aspects of the newly introduced paradigm. It might even be surprising to know that a recent census reports the existence of more than 70 mobile agent systems. Therefore, there is a need to gather and analyze what has been done so far in this new area.

This survey reviews the field of mobile agents by summarizing the key concepts and giving an overview of the most important implementations. Design and implementation issues of mobile agents are analyzed in general. Some of the most important mobile agent systems are presented and discussed. Java's support for mobile agent development is thoroughly examined. In addition, the role of the Common Object Request Broker Architecture (CORBA) as a broker between mobile agents and their environment is also analyzed. Most importantly, a survey of the major security concerns is provided followed by an analysis of the currently available techniques to address these concerns. Last but not least, a detailed analysis of the Foundation for Intelligent Physical Agents (FIPA) standards for interoperability between heterogeneous agents and their hosts is included. This survey will help in understanding the potentials of mobile agents and why they have not caught on. Once progress is made in the areas of security, programming language support for specific mobile agent requirements, and standards for coordination between heterogeneous agents, it is expected that the mobile agent paradigm will dramatically revolutionize the way the Internet is being used now.

Résumé

Le concept d'agent mobile, une pièce de logiciel capable de se déplacer indépendamment à travers le réseau de relais en relais, est en pleine expansion dans l'industrie et l'Académie. Le terme agent mobile a été introduit en 1994. Depuis, différents travaux de recherche touchant les différents aspects de ce paradigme sont lancés. Au fait, un récent recensement a établi l'existence de plus de 70 systèmes d'agents mobiles, d'où le besoin de réunir et d'analyser la recherche établie dans ce domaine.

Cette étude passera en revue le domaine d'agents mobiles en établissant les concepts clés et en présentant une vue d'ensemble des plus importantes mis en œuvre. Nous analyserons en général les issues majeures de conception et de mis en œuvre d'agents mobiles. Nous présenterons certains des plus importants systèmes d'agents mobiles. Nous examinerons l'utilisation du langage Java pour le développement d'agents mobiles. Ensuite, nous analyserons l'interopérabilité entre les agents mobiles et leur environnement grâce à CORBA (Common Object Request Broker Architecture). L'étude des soucis majeurs de sécurité sera suivie d'une analyse des techniques disponibles pour les contre-carrer. Enfin, nous analyserons en détail les standards d'interopérabilité FIPA (Foundation for Intelligent Physical Agents) entre les systèmes a base d'agents. Cette étude éclaircira le potentiel des agents mobiles et la raison de leur retard. Le progrès dans les domaines de la sécurité, et des langages de programmation soutenant les besoins d'agents mobiles, et des standards d'interopérabilité parmi les agents hétérogènes permettrait au

paradigme d'agents mobiles de révolutionner l'Internet comme nous le connaissons.

Acknowledgements

I am most grateful to God for His support and His indescribable guidance throughout the course of collecting and analyzing the material presented in this thesis.

Many thanks go to Professor Newborn for his immeasurable commitment to helping see this survey through to its final completion, and his equally generous and wise guidance during its development. His contribution to this survey was of utmost value.

Finally, I would like to thank my family for all their support and encouragement as I followed the twisting path. I dedicate my thesis to the soul of my father who departed to the heavenly kingdom on October 9th 2001.

Table of Contents

List of Figures

List of Tables

Chapter 1:
Introduction

The field of intelligent mobile agents is emerging as a promising paradigm for the design and development of e-commerce applications. This survey reviews the field of mobile agents by summarizing the key concepts and giving an overview of the most important implementations. The key concepts presented in this survey address issues related to security, mobility, and interoperability. Chapter 1 provides the basic definitions and technical background needed for the rest of the thesis. An analysis of some of the programming languages that have been used in the development of mobile agent systems is covered in chapter 2. Chapter 3 focuses on presenting a survey of some of the frameworks available up to the date of this thesis. A comparison between the surveyed frameworks is also covered in Chapter 3. Chapter 4 discusses the issue of security in the mobile agent paradigm. Without a firm understanding of the various security aspects involved in the deployment of mobile agents, a researcher cannot realize why mobile agents are not widely spread in the market. Chapter 5 illustrates the various migration techniques used so far and the latest proposed techniques to achieve strong migration. Chapter 6 deals with the issues related to interaction between heterogeneous agents in multi-agent systems. A thorough analysis of FIPA – one of the most important standards available so far for mobile agent systems – is also given in chapter 6. FIPA's main objective was to achieve interoperability between different agents.

1.1 What is an Intelligent Mobile Agent?

It is quite hard to define what is an intelligent mobile agent in a clear and unambiguous way. The definition presented by [16] and [17] is adopted throughout this thesis. A mobile agent is

> *"A computational entity, which acts on behalf of others, is autonomous, pro-active, and exhibits the capability to learn, cooperate, and move in a heterogeneous network."*

The term intelligent mobile agent contains three separate and distinctive concepts: intelligence, mobility, and agency. An intelligent mobile agent is a self-contained software program that has some degree of intelligence programmed in it [2], can move within the network autonomously, and act on behalf of the user who has dispatched it [4] [5]. The agent chooses where to move on the network. Agents may communicate together to achieve a common goal. An agent marketplace is a computational system that has the ability to host agents coming from different sources allowing them to interact together. An example of a mobile agent marketplace is having a buyer agent and a selling agent. Each one of them has its own negotiation skills and techniques. Both of them are communicating together to reach the best possible outcomes for their owners [2].

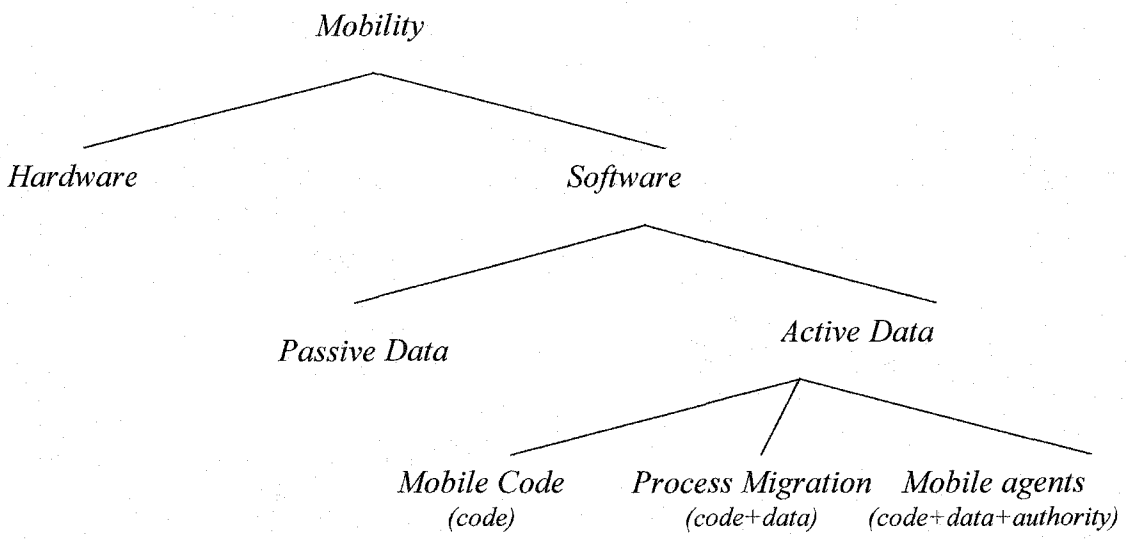

Figure 1: Mobility and the evolution of mobile agents

Mobility can be classified as hardware and software. Hardware mobility deals with mobile IP's, mobile networks, mobile devices, etc.. Hardware mobility is out of the scoop of this thesis. Software mobility can be viewed as passive data and active data. Passive data is the classical movement of raw data like moving files from one node to another. Active data can further be divided into mobile code, process migration and finally mobile agents. These three classes represent the evolution of Mobile Agents [11]. Mobile code is defined as transferring only code between nodes. Java applets are an example of mobile code. Process migration deals with transferring the code of a certain process that is being executed along with its data. This requires transferring the state of the

process as well. This transfer is controlled by a single centralized administrative domain [11]. Finally come mobile agents where the code, data and especially authority are transferred between nodes. There is no central administration to control movements of mobile agents. They decide to move whenever they feel it is appropriate to do so and they have the authority to act on behalf of their owners within the entire network.

1.2 What Is the Difference between Mobile Agents and Distributed Objects?

For many years CORBA has utilized the concept of objects and distributed object applications. Microsoft too has introduced its own objects (OLE, ActiveX, COM, and DCOM) [2]. To know the differences between a distributed object application and a multi-agent system, let's first examine a distributed object application. An object is a mere encapsulation of data and a set of functions that operate on these data. Objects interact together through invoking a very well defined set of interfacing functions. An object can use another object to achieve a certain task. Objects do not initiate actions by themselves [2].

Where in a multi-agent system, we have N number of agents. Each one of them has its own goal. Each agent initiates an action to achieve its goal. They communicate together by conversation not by method invocation. Each agent decides what to do and when to do it. An agent can say no when requested to perform a certain action. Objects have fixed roles where agents can change their roles dynamically [2].

1.3 A Comparison between Mobile Agent and Client-Server Paradigms

The client server paradigm requires the client to be connected to the server during the whole course of service. The client sends a request to the server and waits for a response. If a response is not received after a predetermined period of time, the client has to resend the request again and wait for a response. This scenario is very costly if we consider a mobile user who is connected to a network (say the Internet) through a wireless connection. The client server scenario is also very time consuming. The user has to spend time to connect to servers, request services, and wait for responses. When responses are received from servers, the client has to analyze them.

Considering the intelligent mobile agent paradigm, the intelligence needed to analyze the data received from servers can be embedded inside the agent. Therefore, analyzing the data is done locally on the servers while the agent is physically running on these servers. Connection time is needed only to dispatch the agent and recollect it. For wireless users who are on the go most of their times, this feature is quite beneficial. All the network traffic needed to exchange messages between the client and the servers in case of the client-server paradigm is replaced by just the traffic caused by the mobile agent to move from one server to another [15].

1.4 Advantages of Using Mobile Agents

According to [6] there are at least seven good reasons to use mobile agents. These seven reasons are lists here in addition to some extra ones.

1. Reduction of network load

Mobile agents reduce the network load by reducing the amount of data to be transferred over the network. Classically, data are moved from one computer system to another where the data are processed. Mobile agents move to the data and process them locally. Also distributed systems depend on remote procedure calls (RPC) which are messages exchanged back and force between two or more computing systems. By using mobile agents, conversations between the computing systems can be packaged, sent, and processed locally [6].

2. Overcoming network latency

Critical real time systems can face a serious latency if controlled through a network with a substantial size. For example, a robot in a factory needs to respond in real time to changes in its environment. A mobile agent can be dispatched from a central controller to the robot. Once it reaches the robot, the mobile agent can execute the central controller's directions locally on the robot [6].

3. Encapsulation of protocols

Each host on a network has its protocol for encoding outgoing data and decoding incoming data [6]. There are now a few number of protocols available in the market. For example, there are TCP/IP, NFS, etc.. Each time a new protocol is introduced, testing its compatibility with the ports of the existing operating system is a tedious work [7]. Instead of extending the capability of a network by defining a new protocol, mobile agents opened the door to encapsulate the protocol in an agent and send it all over the network [7].

4. Asynchronous execution

Once a mobile agent is dispatched, it can work asynchronously from the process that has dispatched it. For example, a user may only get connected to the Internet for few seconds to dispatch an agent. The agent will execute autonomously and asynchronously from the process that has dispatched it. Latter, the owner of the agent may connect again to the net to collect the agent or to receive messages from it [6].

5. Dynamic adaptation

Mobile agents can sense their environment and adapt dynamically [6]. An agent can be programmed to sense and react to a certain stimuli in its environment [2].

6. Mobile agents are naturally heterogeneous

The Internet is a heterogeneous computing system. It has all kinds of different hardware and software systems. Mobile agents are ideal for integrating these different systems together since mobile agents are generally system independent.

7. Mobile agents are robust and fault tolerant

Due to the mobility attribute, agents running on a host that is going to be shut down are given a warning and adequate time to dispatch themselves to another host and resume executing on the new host [84].

8. Mobile agents provide dynamic load distribution

Mobile agents have the ability to migrate from overloaded nodes to less loaded ones. By doing that the, the overall load on a certain network can be distributed [11].

9. Mobile agents result in better software quality

There are some software engineering benefits in using mobile agents. The mobile agent paradigm helps software developers and designers to conceptualize solutions better for certain problems. This will result in better code modularity and reusability [15].

1.5 Applications That Can Benefit from the Mobile Agent Paradigm

The mobile agent paradigm can particularly benefit some fields. Some of these fields are listed below.

1. E-commerce

E-commerce applications are at the top of the list of applications that can benefit from the mobile agent paradigm. A selling agent, a buying agent, an auction bidder agent, and an agent that can search for information and analyze it are just very few examples of how mobile agents can be used in e-commerce.

2. Personal assistance

Mobile agents can be very effective as personal assistants. For example a mobile agent can be an online shopper's assistant [9]. Frustration of customers at an online store is primary due to a failed search or the retrieval of so many items. In [9], a mobile agent that was developed to function as a shopper's personal assistant is proposed. This personal assistance is capable of moving to certain online stores like Amazon and Vstore. When the agent is physically inside the server hosting these online stores, it queries the stores catalogs, processes the data collected, and returns a result or some suggestions that can narrow down the number of items returned.

Mobile agents opened the door wide for so many applications as personal assistants. However, it is worth mentioning that designing a good helpful

assistant requires studying the behavior people follow to do certain tasks. For example, before launching an agent for purchasing an airline ticket, one should study the consumer's behavior when shopping for a ticket [10]. What are the factors that affect his or her decisions? What are his or her preferences?

3. Secure brokering

In secure brokering, an agent can work as the broker of his or her owner. For example, a mobile agent can be developed to carry out the tasks performed by a stockbroker or insurance broker. Utilizing all the security mechanisms provided by encryption, special hardware devices, digital signatures, digital certificates, and secure communication mechanisms can lead to the development of a secure broker that can carry sensitive information about its clients.

4. Distributed information retrieval

The autonomous mobility characteristic of mobile agents provides the vehicle to achieve distributed information retrieval. An agent can be programmed with a list of itineraries where it can go to and retrieve information from all the hosts in its itineraries and process them.

5. Telecommunication networks services

Mobile agents can perform certain tasks very efficiently in the field of telecommunication. Some of these tasks are: load balancing, network management, and protocol encapsulation [18]. Mobile agents are already in use as part of the Telecommunications Information Networking Architecture (TINA).

In [60], telecommunication services are formally specified as a set of cooperating mobile agents. The work done in [60] is in conformance with the Reference Model of Open Distributed Processing (RM-ODP) standards developed by ISO.

An efficiency evaluation of a mobile agent based network management system is presented in [61]. The results presented in [61] showed effected use of mobile agents for network management. The use of mobile agents in network management leads to less need for computing resources in network equipments since mobile agents do not need computing resources in the network equipment permanently. Also by using mobile agents, some of the management functions can be executed directly on the network equipment. Therefore, the computing load of the network management is reduced.

6. Monitoring and notification

A mobile agent can go and reside on a certain host waiting for a certain event to happen. This event could be a sale on a certain item, the availability of a certain server, etc.. Once this event happens, the agent is triggered to take an

action. This action could be as simple as notifying its owner or as complex as finalizing a stock purchase deal and paying the money.

7. Information dissemination

A mobile agent can be programmed to go to certain hosts and disseminate certain messages. For example, a mobile agent can go to all customers of a certain company to install new updates to their systems.

8. Parallel processing

Through coordination between mobile agents, an application can be developed as a set of mobile agents. Each one of them has a certain task to achieve and report its result back to a central coordination point. This coordination point (can also be an agent) receives the partial results of all agents and computes the final output.

1.6 Issues to be Considered When Designing a Mobile Agent

There are so many issues involved in the design and implementation of a mobile agent system. Some of these issues are presented in [4]. For example, what is the mechanism that the agent will use to move from one host to the other? Once a mobile agent is dispatched, how can its owner locate it again? How can an owner recollect the agent? An agent system should have a well-defined naming mechanism as well as a reliable technique to locate an agent.

By definition, a mobile agent moves autonomously. This indicates that an agent must be able to suspend its execution state, move to a new host, and resume execution from the point it has stopped at. How can this be done? Do the currently available programming languages provide ways to do so? What about data? A mobile agent has internal data. These data need also to be transferred to the new host as well. Therefore, a mobile agent system must provide a mechanism for data transfer.

Ideally, an agent developed under any mobile agent framework should be able to communicate with any other mobile agent developed under any other mobile agent framework. In order to achieve this high level of interoperability between heterogeneous mobile agents, standards have to be developed and respected by all mobile agent systems.

Another very important issue that has to be taken into consideration is the issue of security. A whole chapter in this thesis is dedicated to the issue of security and how it is handled in various mobile agent systems.

Stability and performance are also important issues to be considered when developing a mobile agent system.

As for platform dependency, of course, a mobile agent system must be platform independent. By definition a mobile agent moves freely between hosts. These hosts could probably be running different operating systems.

Last but not least, the issues related to ethics, legality, and society should also be considered. If an agent causes some sort of damage to a host, who will be responsible for that? Who will be held accountable? How will the social and

business conduct be respect [12]? In this thesis, some of these issues are surveyed and analyzed along with a presentation of the currently available technology that handles some of these issues.

Chapter 2:

Analysis of Some Programming Languages Used in Developing Mobile Agent Systems

Since mobile agent paradigm is a new field, it is no surprise that the languages that have support for it are also fairly new. Some of these languages are Java, CORBA, Telescript, and Tcl [11].

2.1 Java Support for Mobile Agents

Java is a programming language that has been developed to be used in web-based applications. It provides some unique features for mobile agent development. Most of the currently available mobile agent systems are Java-based. In this section the pros and cons of using Java in mobile agent development are discussed.

2.1.1 Pros of Using Java in Developing Mobile Agents

2.1.1.1 Java is platform independent [1]

Thanks to the Java Virtual Machine (JVM), a Java program is processor independent and operating system independent. For example, a Java program developed on a Pentium II PC running Windows 95, can be executed on a UNIX or Sun workstation. That is because a Java program is a set of machine independent byte codes. These byte codes are then interpreted (or compiled) by the JVM to a specific machine code. This unique characteristic (Byte code and

JVM) is an excellent way to develop an agent that can run on any machine running any operating system [14].

2.1.1.2 Java provides secure execution [1].

Since Java was built for use on the Internet, security issues were emphasized in its design. For example, the Java security manager can check for potentially unsafe operations such as file access or network connection [77]. Java security manager checks if a running Java program has the right to perform certain operations or not [1]. For example, a certain agent can be granted the right to read and write files. Another agent can be granted the right to read only files. A third agent may not be granted any file access rights [14]. The security features provided by Java make it safer for hosts to accept potentially not trusted agents since agents will not be able to access private information on the host [1].

2.1.1.3 Java provides dynamic class loading [1]

This is Java's way to achieve mobility. A piece of running Java code and its state can migrate by Java's class-loading mechanism. Java's class loaders can dynamically load an application's classes either locally from the class path directory or from a web server [14].

2.1.1.4 Java provides multithreading programming

By definition, mobile agents are autonomous. This indicates that each mobile agent should be able to execute independently from the other agents.

Java gives each mobile agent a "thread of execution" which enables the agent to work independently in its space without having to depend on other agents. Java also provides a set of synchronization primitives that enable interaction between agents.

2.1.1.5 Java provides object serialization

By the definition of mobility, an agent should be able to move from one node on the network to another and resume its execution from the point at which it has stopped. A Java-based agent uses Java's serializing / de-serializing feature which provides a means for translating a graph of objects into a byte-stream and achieves migration at a coarse granularity. This implies that an agent restarts execution from the beginning each time it moves to another host. Java provides a very strong built-in serialization mechanism that accomplishes this conversion and reconstruction almost transparently [14].

2.1.1.6 Java's other aspects that enable mobility [2]

The JavaBean delegation event model allows an agent to plug itself in an already running server and unplug itself when it wants to leave. The Java.net package allows a mobile agent to communicate with servers and send serialized Java code and process state data over sockets. The Java Remote Method Invocation (java.rmi) allows an agent to call methods on other objects across a network. Java's rmi-registry (remote method invocation registry) can be used to

locate remote objects by name. This enables the communication between agents who move freely on the network.

2.1.2 Cons of Using Java in Developing Mobile Agents

Even though Java is one of the best programming languages to be used in developing mobile agent systems, there exist some weak points in Java that we have to be aware of.

2.1.2.1 Inadequate support for resource control [1]

Java does not provide a way to limit the resources consumed by a Java object. Once an agent is running on a host, the agent can abuse the host's resources. For example, the agent can keep consuming the host's processing cycles, memory resources, or any other available resources. By doing that, a mobile agent can render the host from being able to perform any other tasks. A related issue is the ability of an agent to allocate resources external to the program. For example, an agent can create a window as its user interface on one host. The agent may latter decide to move to another host leaving behind its window. Since Java does not provide a mean to bind a Java object with an externally created resource, this window will never be garbage collected.

2.1.2.2 No protection of references [1]

The public methods of a Java object are accessible to any other object that has references to these methods. The concept of protected references does

not exist in Java. This means a mobile agent can never know who is accessing its methods or why.

2.1.2.3 No object ownership of references

In JDK 1.1, there is no ownership for references. This means the execution thread of an agent can be taken away from it by another object. The Java garbage collection mechanism will not free any Java object as long as it is being referenced by another object. This causes the agent to be kept alive against its will [1]. Weak references in JDK 1.2 provide a solution for this problem.

2.1.2.4 No support for preservation and resumption of the execution state [1]

Java does not provide a way to keep the exact state of the execution counter and the stack frame. Therefore, [1] claims that this will render the agent from being able to resume execution from exactly where it has stopped. But if Java provides such capabilities, it will no longer be machine independent. Using the built in serialization mechanism of Java, the state of a Java object can be packed on one platform and latter be unpacked on a different one transparently. There is no need to know neither the execution counter nor the stack frame. The serialization mechanism provides a reasonably sufficient mean to migrate a running Java program from one platform to another.

2.2 CORBA Support for Mobile Agents

CORBA stands for Common Object Request Broker Architecture [58]. The main goal of introducing CORBA to the market by the Object Management Group (OMG) was to provide a foundation for making diverse applications work together in heterogeneous environments [59]. Recently, a Mobile Agent Facility (MAF) is added to the CORBA definition in order to support communication between heterogeneous agents. CORBA is best suited as a means of interoperability between different agent frameworks. For example, the MASIF standard (see chapter 6 for more details) is implemented in CORBA.

The unique interoperability characteristic of CORBA has played a major role in allowing many applications to communicate together in a transparent way. For example, [54] proposes the architecture of a prototype system that integrates SOMA which is a mobile agent framework and a distributed multimedia application. By using CORBA, [54] claims that their system achieves large accessibility and interoperability.

The Object Request Broker (ORB) component of CORBA makes it possible to exchange messages between different applications without having to dramatically change the implementation of these applications [57]. Therefore, interoperability between the existing mobile agent systems can be achieved by adding a CORBA-based interoperability component to every mobile agent systems. The CORBA-based interoperability component can be integrated with the existing systems without having to dramatically change their internal implementations.

From the points presented in this section and the previous one, it is believed that combining the benefits of both Java and CORBA can lead to the development of a mature mobile agent framework.

2.3 Tcl Support for Mobile Agents

Tcl is a high level scripting language that has several useful features for mobile agent development. According to [56], Tcl is easy to learn, highly portable, embeddable, and extendible. It is highly portable because it is an interpreted language. Therefore, a mobile agent that is developed using Tcl can run on various platforms. Since Tcl is embeddable in other applications, those applications can implement part of their functionality as a mobile Tcl agent [56]. Tcl can be extended by some user-defined commands. This allows an easy way to integrate agent-related functions to the rest of the language. This extendibility feature allows the host to add some Tcl functions as service functions. The Tcl interpreter is a C library package that can be incorporated in a variety of applications. Figure 2 from [75] shows the structure of the Tcl language and how it can be extended and embedded into other applications.

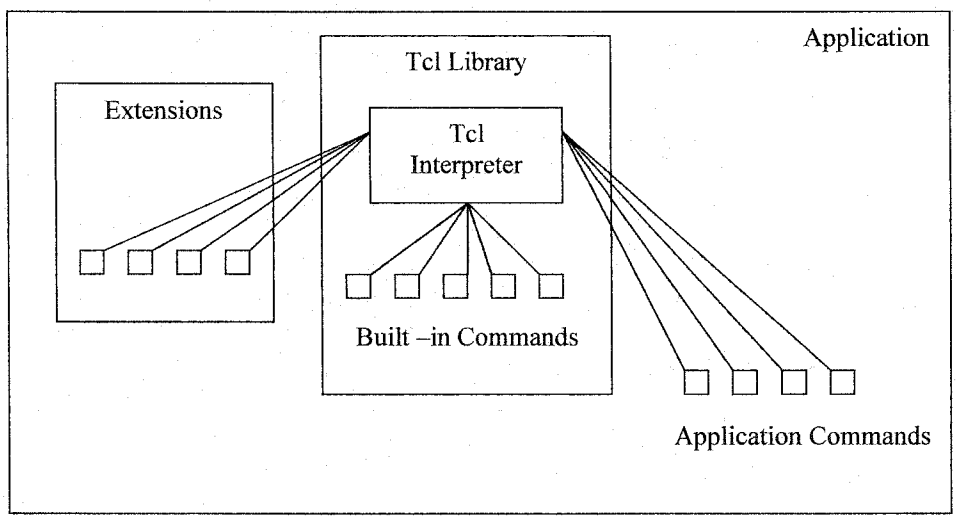

Figure 2: Tcl extendibility and embed-ability

The main disadvantage of Tcl is that it does not provide code modularization. This makes it difficult to write and debug large scripts [56]. Another disadvantage of Tcl is the absence of a mechanism to fully capture the complete internal state of a running script. Therefore, it is not possible for Tcl-based mobile agents to migrate to a new host and resume execution from the point where they have exactly stopped on the former host.

According to [75], Tcl is good choice to develop a safe mobile agent system due to some specific properties. The first one is that Tcl is an interpreted language. Since each line of code of a suspicious mobile agent is interpreted by the host, it is quite feasible to add security checks and controls at any point of time. This property makes it safer for the host to execute any agent regardless of its source. A second property is that Tcl is safe with respect to memory usage. Tcl does not support pointers, provides an automatic management for storage,

and checks the boundaries of array references. This property prevents visiting mobile agents from accessing the host's storage in an unauthorized way. Another property is that Tcl can assign an interpreter to each executing script. This property makes it possible to isolate scripts from one another allowing a host to give different security privileges to different agents.

Even though Tcl is a very simple language to learn and use, it is not considered to be a practical development language for a real life mobile agent system. It does not provide built in facilities for migration, serialization, or interoperability. The developer has to develop the handling for all these aspects almost from scratch. Having said that, one can conclude that Tcl is more appropriate for developing prototypes for mobile agent systems rather than real life complex ones.

2.4 Agent Oriented Programming

Agent Oriented Programming (AOP) is still a premature paradigm. An agent is viewed as a computation entity that has a set of mental states. These mental states include beliefs, capabilities, choices, desires, intentions and commitments. The rationale behind this approach in representing agents is that humans use such concepts to represent complex systems [55]. AGENT-0 is an example of an agent oriented programming language [49] [55]. An agent in AGENT-0 language is represented by a set of capabilities that indicate what an agent can do and what it cannot do. An agent has also a set of beliefs,

commitments and commitment rules. Figure 3 from [55] shows the flow of control in AGENT-0.

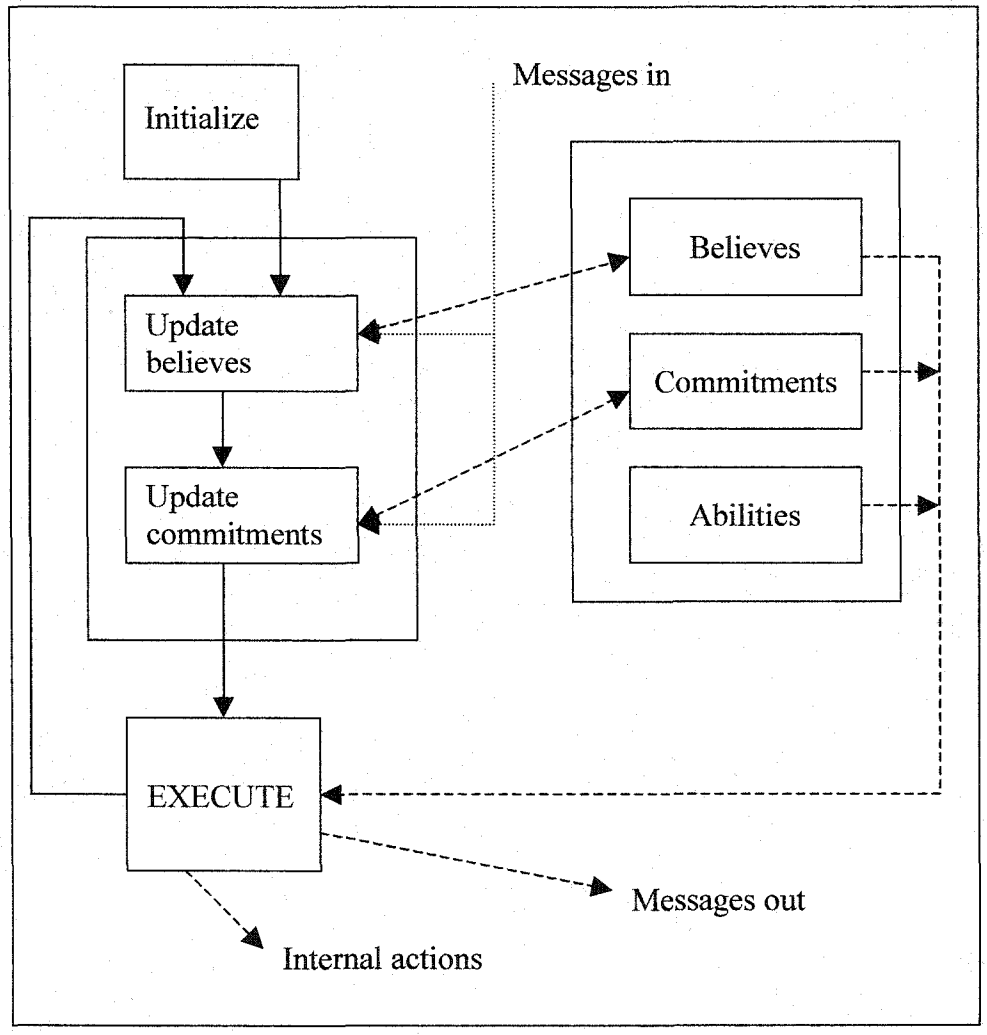

Figure 3: The flow of control in Agent-0

The idea of having mental attributes associated with a computation entity is quite suitable for mobile agent development. Having a programming language with these primitives, a developer can represent his or her agent in a way that

truly simulates the real world. The field of Agent Orient Programming is very promising. However, the lack of real life agent oriented programming languages (either in academia or industry) makes the idea of agent oriented programming a premature topic that needs lots of research before it is ready for real life applications. it is believed that when the new paradigm of agent oriented programming reaches a reasonable degree of maturity, the field of mobile agents will be dramatically boosted.

Chapter 3:
Mobile Agents Frameworks

Even though the mobile agent paradigm is quite new, there are plenty of mobile agent systems that have been developed both in academia and in the industry. A list of them can be found in [8]. Most of the newly developed frameworks are Java based. In this chapter a selection of these platforms are presented and analyzed. The analysis of the selected frameworks covers the techniques used for run-time support, migration, and collaboration.

3.1 Concordia

Concordia is a Java based commercial framework that was developed by Mitsubishi. For more information about Concordia, the interested reader may refer to Concordia's web site at [36]. Concordia was introduced to the market in 1998. According to [37] (who is working for Mitsubishi as an assistant laboratory director), Concordia was "designed for complex, secure, reliable, real-world, enterprise applications." The design goals of Concordia as outlined [62] are:

- Flexible agent mobility
- Support for agent collaboration
- Reliable agent transmission
- Persistence agent state
- Agent security
- Ability to add intelligence

3.1.1 Run-time support

The Concordia system consists of several integrated components. The main component is the Concordia server. The Concordia server is the component that is responsible for providing the communication infrastructure for mobile agents. It manages the life cycles of visiting agents as well as enables them to be transmitted and received by nodes on the network [62]. Each server has a set of "managers" responsible for providing support for agent mobility, security, communication and persistence [34].

3.1.2 Migration

Concordia's agents migrate only by having a predefined set of itineraries [34]. Each itinerary contains a host name and the function name to be executed at this host (host name, function name). This makes it possible to decouple mobility code from the application code. When an agent decides to migrate, it invokes a migration request method in the agent manager. The agent manger will then suspend the execution of the running agent and create a persistent image of it. Then, the agent manager will check the agent's itinerary object to determine the host to which the agent wants to migrate. The persistent image that has already been created will be sent to the new host. The new host will save the received image of the agent and then send an acknowledgment to the old host. The new host will then unpack the image to reconstruct the agent and give it a thread of execution to carry out its tasks.

3.1.3 Collaboration

The interaction between Concordia's agents is achieved by asynchronous distributed events and collaboration [34]. Events provide a direct mean of communication between agents. Collaboration is provided by Concordia's framework by utilizing a distributed Java object as the collaboration point between agents. The JAVA RMI system allows object running on one Java Virtual Machine (JVM) to invoke methods on an object running on another JVM. Each agent submits its result to the collaboration point by invoking appropriate methods. The collaboration point will then collaborate all the results together and compute a final result. The final result is communicated back to the agents by invoking the appropriate methods in the agents. Concordia uses the SSLv3 (Secure Socket Layer) to ensure security for all the transmitted data [62].

According to [34], Concordia is "not sufficiently adequate to deal with complex coordination patterns." The only way of communication provided by Concordia is using Java RMI. Concordia does not support a universal agent communication language. Therefore, Concordia's agents cannot communicate with other agents that have been developed under any other mobile agent framework. Concordia's agents cannot communicate complex forms of knowledge. For example, Concordia does not support exchanging knowledge in the Knowledge Interchange Format (KIF) or any other similar knowledge-based format. KIF is a special format designed to represent knowledge in a standard format that can be shared by different applications (see chapter 6 for more details).

3.2 Aglets

Aglet is IBM's Java-based framework for mobile agents [1][2][34][62]. Aglet is developed in IBM's Japan laboratories. It was first released in 1998. Each aglet has a set of methods that handle predefined events like creation, cloning, dispatching, retraction, activation, deactivation, disposal, and messaging. The programmer of an aglet can override these methods to provide his or her own code for handling these events [34]. Each aglet has a "proxy" object. The proxy object provides a shield that protects an aglet's internal public methods from being abused by the host. The proxy also provides a secure mechanism to communicate with the other agents running on the host [1].

3.2.1 Runtime support

The run time support for aglets is provided by having a "context" assigned to each aglet upon its arrival to the host. The context is assigned by a preinstalled Tahiti server which is an aglet server program. The Tahiti server listens for incoming aglets and accepts them into the host. The context is the execution environment for aglets. It also plays the role of a security shield for hosts from malicious aglets [34]. Each context has a graphical user interface to monitor and control a running aglet. The context provides an interface for aglets to get access to the necessary computing facilities available on the host such as CPU cycles, memory, and file access rights.

3.2.2 Migration

An aglet migrates from one host to another by invoking its dispatch method. On completion of executing the dispatch method, all threads created by the aglet are killed. The dispatch method takes a URL as its input [1]. For example,

Agent.dispatch (new URL ("atp://somehost.com"))

An aglet programmer has to code handles for some events to achieve successful migration of the aglet. For example, the aglet programmer needs to implement methods to be invoked when the onDispatching and onArrival events occur. Handling onDispatching allows the programmer to make the aglet finish what it is supposed to do right before migration to a new host [1]. Coding a method to be invoked onArrival, provides a way to initialize the aglet on its new host. The onArrival method is also the single entry point of execution of the aglet on the new host [1]. The protocol used for the transportation of aglets is the Aglet Transfer Protocol (ATP). The ATP is developed by IBM as an application-level protocol for distributed agent-based systems [62].

The ATP provides a mechanism for retracting an agent by invoking the agent's "retract" method. An aglet programmer can implement a method to be invoked when the aglet receives the onRetracting event. In this method, the developer can implement whatever the aglet needs to do before it returns back to its owner [1].

Currently, the ATP transfers all the aglet's classes when the aglet is dispatched. The ATP does not check if certain classes are in the destination cache or not. If some of the aglet classes are already on the destination, then there is no need to transfer these classes with the aglet. The aglet can use the classes that are already on the destination host. Optimizing the transfer of classes can result in improved transformation and downloading time [62].

3.2.3 Collaboration

Aglets interact with each other through messages. A message is a Java object that can be sent in synchronous or asynchronous mode [34]. The aglets communication mechanism is based on a simple callback scheme. Each aglet has to have handlers for the kinds of messages it can understand. Messages are sent to aglets through proxies. One of the benefits of using proxies is that it provides a location independent interface for sending messages to aglets.

Multicasting is supported by Aglet's framework by a technique called message subscribing. The basic principle is that aglets subscribe to one or more multicast messages and implement handlers for these messages [1].

3.3 Voyager

Voyager is an Object Request Broker (ORB) framework for developing mobile agents [34]. Voyager is implemented in Java based on CORBA. Therefore, it supports both CORBA and Java RMI. The ORB provides the capability to create objects on remote systems and to invoke methods on those

objects. Voyager adds agent capabilities to the traditional ORB. Voyager is a commercial framework package developed by Recursion Software, Inc. One of the main features of voyager is that the target object cannot only be a host but also a program or a Java object. Once the agent is on the host, it can obtain a reference to the target object and start communicating with it locally [34].

Another unique feature of voyager is that it allows a transparent method invocation in mobile agent. That means if you are an owner of a mobile agent and you have dispatched it from your PC, you can continue invoking functions on the agent as if the agent is still residing in your machine. The voyage mobile agent leaves a forwarder in each location it visits. An invocation of one of the agent's methods will be transparently forwarded to the new location.

3.3.1 Runtime support

Voyager system includes a set of mangers that control the execution of its mobile agents. One of these managers is the security manager [2]. The role of the security manager is to protect the host from malicious agents. It can restrict the operations that a visiting agent can perform on the host. Voyager also includes other managers that control the serialization and migration processes of agents.

3.3.2 Migration

Voyager's agents migrate by invoking a moveTo method. The moveTo method takes the destination and a method to be invoked once the agent

reaches its destination. Voyager supports a multiple entry point execution for its agents by associating a method to be invoked with every host an agent plans to visit.

3.3.3 Collaboration

Interaction between voyager's agents can be either by synchronous or asynchronous messages. Voyagers agents can also communicate between themselves and their environment by remote method invocations. According to [34], Voyager communication support is very strong but it is somewhat limited in open distributed systems because it is not possible to know the interface of an agent in advance. This pre-knowledge of an agent's interface is required in order to support direct communication between agents.

For the latest information about Voyager, the interested reader may check [38] for up to date details.

3.4 Odyssey

General Magic [39] was the first company to work on mobile agents using the Telescript language [34]. When Java was introduced to the market, General Magic decided to develop Odyssey which is a Java-based mobile agent system.

3.4.1 Runtime support

The Odyssey framework is characterized by its support for Microsoft DCOM and CORBA IIOP in addition to Java RMI. An Odyssey's agent can move

to a "place". The "place" is defined by an Odyssey's developer. This makes the Odyssey framework very flexible.

3.4.2 Migration

For migration, Odyssey supports itineraries as well as a go() method that takes a place to go to. Initially, an agent may start its trip with a predefined list of places to visit. At any point of time, an agent has the ability to decide to go to a particular place that was not originally in its itineraries. This feature gave the agent the ability to adapt according to particular conditions in its environment.

3.4.3 Collaboration

Agents interact between themselves and their places by remote method invocation and also by exchanging references to objects. Odyssey does not provide any means of synchronization between the participants [34]. This could be considered as a limitation in the collaboration technique used by Odyssey. A sophisticated distributed system would probably require a more powerful collaboration mechanism.

3.5 Mole

Mole is a research mobile agent system developed by the University of Stuttgart. The Mole project started in 1994 and is still going under the supervision of Professor Rothermel [40]. One of the interesting features of Mole is the detection of Orphan agents. An Aglet will never be automatically garbage

collected unless it destroys itself. This is because an Aglet has its own thread of execution. Mole implemented an algorithm to detect an "orphan" agent and remove it from the system.

3.5.1 Runtime support

Mole's runtime support is attained by having stationary agents on Mole's hosts. A stationary agent on a host provides the interfacing functions that a visiting mobile agent may need to access the resources available on that host. These stationary agents can be viewed as managers. Each one of them has a particular role and set of responsibilities.

3.5.2 Migration

The migration mechanism supported by Mole does not support itineraries. A go like method is used to initiate a migration request to a particular host. Like an Aglet, a Mole mobile agent can only have a single point entry [34]. Therefore, a Mole mobile agent cannot specify a certain method to be invoked on a particular host.

3.5.3 Collaboration

The interaction between agents relies on exchanging messages. These messages are of the form of objects. The stationary agents publish their services on a common dictionary. The visiting mobile agents can then look up the offered

services in order to determine how they will achieve their tasks. Mole does not support any advanced support for coordination between agents [34].

For the latest news, publications, talks, and thesis (in German) about Mole, the interested reader may visit Mole's home page at [40].

3.6 Macondo

Macondo is a mobile agent framework developed in the University of Bologna, Italy [34]. The main design goals of Macondo were simplicity and flexibility. Macondo stands for Mobile Agents and Coordination for Distributed applications [41]. Macondo is developed using both Java and MJada programming languages. MJada is based on Java but it includes extra mechanisms that provide enhanced support for mobility and coordination.

3.6.1 Runtime support

Macodo utilizes the notation of "agent server" to provide runtime support for its agents. The agent server receives a visiting agent upon its arrival to the host. The agent server is responsible for providing the necessary resources requested by an agent. Once an agent decides to migrate, it is the role of the agent server to insure a safe and secured migration for the agent. Therefore, Macondo's mobile agents move from one agent server to another [34].

3.6.2 Migration

Macondo's support for migration relies on a go() method that takes the name of a host, the host's port number and a method name to be invoked when reaching the host. By doing that, Macondo supports a multiple entry code for execution for its mobile agents. Itineraries are also supported giving an agent the flexibility of having a fixed list of hosts to visit sequentially.

3.6.3 Collaboration

Macondo framework uses the notation of "tuple space server" as a mean of interaction between agents and their hosts. The "tuple space" as a concept is defined as a feature of the Jada programming language. A tuple space server is a container of nested tuples that can be manipulated by agents to communicate with each other.

For the latest information and news about Macondo, the interested reader can check [41].

Feature	Aglets	Concordia	Voyager
Multicast support	No	No	Yes
Agent persistence support	No	Yes	Yes
Remote agent creation	No	No	Yes
Proxy update on MA Migration	Yes	No	Yes
Messaging modes between MA	Synchronous Asynchronous	Methods	Synchronous Asynchronous
Java messages to MA	Transparent	No	No
Garbage collection	No	No	Yes

Table 1: Qualitative comparison between different agent frameworks [63]

Chapter 4:
Mobile Agents and Security

By definition, the basic characteristic of a mobile agent is its ability to roam freely on a heterogeneous network such as the Internet. On one hand, there is a possibility that an innocent agent would visit some malicious hosts. On the other hand, an innocent host could be visited by a malicious agent. This dictates the need to have new security measures for both agents and hosts. The mobile agent paradigm has introduced new concerns that were never thought of before in other distributed system environments like the client/server paradigm. Before having mobile agents, no one has ever thought of how to protect a running piece of code from being tampered by its execution environment. The security concerns that arose from deploying the client/server paradigm were totally different from the ones introduced by mobile agents. In case of a client/server application, there are at least two separate computation entities: a client and a server. In most real life cases the client would be running on another machine than that used by the server. They communicate together by exchanging messages over the network. The client and the server, each of them is running on a trusted computing environment.

In this chapter, a thorough survey of the principles and techniques used to achieve a reasonable level of security for mobile agent systems are discussed.

4.1 Security Principles in Designing a Mobile Agent System

How can one evaluate whether a particular mobile agent system provides a good security system or not? If a new mobile agent system is to be developed, what are the security principles that need to be addressed in the system? What are the questions to be asked to determine the level of security provided by a mobile agent system? In this section, a comprehensive list of security principles is surveyed from the literature up to the date of this thesis. Each security principle has a rationale behind it. The goal of respecting these principles is to prohibit some security threats such as masquerading, unauthorized access, unauthorized disclosure, and unauthorized modification. It is quite challenging to meet all the principles listed below in one mobile agent framework.

Principle 1:

Mobile agents and hosts cannot be assumed trust worthy [4] [20] [73].

This principle is quite obvious. Neither the agent nor the host should trust one another. The agent should expect that the host will try to access and reveal its private information. The host should also expect that the agent would try to abuse the system resources or gain access to private information that the agent is not allowed to access. This principle is vital to achieve a practical foundation for mobile agents over a wide area network like the Internet. Some limited applications may not consider this principle. For example, consider a mobile agent whose task is to visit some network nodes and display a message reminding the person who is sitting at this node of attending a meeting or paying

a bill, or something similar. This agent may just ignore the fact that the host it is visiting might be malicious since in this particular case the agent would not be carrying sensitive or important information.

Therefore, a good mobile agent system should be flexible enough to allow a user to create mobile agents with different levels of security. In the case where an agent is not carrying sensitive information, there is no need to overload the code of an agent with tight security algorithms and techniques. However, in case of a buying mobile agent that carries credit card numbers and bank account information of its creator, a very tight and sophisticated security techniques would be a must.

Principle 2:

All critical decisions that need to be made by an agent should be done on a trusted host [4] [20] [23] [73]

At the time of writing this thesis, there was no practical secure solution for protecting an agent from a malicious host [67]. Therefore, this principle can be viewed as a work around solution for the lack of confidence in making a decision on an un-trusted host without worrying that this host may affect the process of decision-making in an unauthorized way [23]. Hopefully, as research proceeds in the direction of developing flawless security techniques, this principle would eventually be removed for good.

Principle 3:

An agent must be able to seal some of its critical state cryptographically [4] [20] [24].

Consider a bidding agent who goes to auctions and bid on behalf of its owner. This agent has the authority to buy an item for its owner. This implies that the agent is carrying very sensitive information about its owner. It may be carrying his or her credit card numbers, bank account numbers, etc. A practical mobile agent should be able to hide and seal such information so that it can never be accessed by anyone who does not have the authority to access it.

Unfortunately, almost all cryptographic techniques known so far have serious flaws. In [67], some of the known cryptographic protocols were analyzed and shown how vulnerable they were to malicious attacks. For example, the multi-hops protocol proposed by [68], the chained digital signature protocol proposed by [69], and even a protocol based on using a trusted secure co-processor proposed by [70] were all broken by some attacks. No doubt, cryptography is and will be an integral part of any mobile agent system. But at the time of writing this thesis, a flawless cryptographic protocol is still an unachieved goal.

Principle 4:

A mobile agent should be able to protect itself from malicious hosts [23] [24] [71].

As mentioned in the previous principle, an agent should be able to hide some sensitive information from being accessed by unauthorized parties. The agent must also be able to protect itself from being manipulated by the host. Let's consider an intelligent bidding agent. This agent bases its decisions on its programmed intelligence. A malicious host may try to manipulate it by changing its intelligence in a way that forces the agent to make a decision that is beneficial to the host rather than to the creator of the agent. Achieving this principle is quite difficult since the host provides and controls the computational environment in which the agent operates [23].

Principle 5:

A host should be able to protect itself from malicious agents [23] [24] [73].

A mobile agent-enabled host must always have security system that is capable of protecting it against attacks from malicious agents. A typical host security system can be subdivided into three basic subsystems whose tasks are authentication, authorization, and accounting [21].

Authentication can be defined as the ability to rank the credibility of a visiting agent. Based on this rank, the host may accept or reject the agent. In [72], a technique called proof-carrying code can be used at the authentication

stage to make sure that the agent is a safe one. Basically, the technique ensures that the code of the agent will not harm the host in any way.

Authorization can be defined as granting a visiting mobile agent access rights to some resources on the host. The host should be able to give different access rights to different agents. Some agents may be granted full access to all the resources of the system. Some other agents may only be granted very limited access rights like having a read only right for few files.

Accounting is the ability to define different quotas for different agents and being able to charge them if applicable. For example, a host should be able to assign a certain CPU quota to a certain agent. Assigning quotas is not only important for billing purposes, but it also guarantees that a malicious agent will not be able to abuse the hosts by totally consuming its resources rendering it from being able to carry out other tasks.

In [23], a model of a security-enhanced agent is proposed. In this model, an agent is carrying a passport. This passport specifies the security characteristics of the agent. The passport is checked by the host and according to the information inside the passport, the host can determine the type of actions that the agent can perform on its computational environment and what actions the host can perform on the agent. The information stored in the passport includes: a unique identifier for the agent, an authentication certificate from its creator, time stamp of the agent creation, lifetime span, and list of privileges given to the agent by its creator. The idea of having "passports" is quite interesting. It gives mobile agents an identity on the network. This identity is very

useful since it will allow the host to know whether the agent is a high risk one or not. For example, If I run a mobile agent-enabled host, I would not be afraid to accept mobile agents who have been created by IBM, McAfee, Microsoft, TD Bank, Royal Bank, Canadian Government etc.. etc.. But, I will be careful when I receive an agent from a no name source or a source that has already been flagged as high risk.

Principle 6:

An agent should be able to protect itself from other agents [24].

In a typical mobile agent marketplace, there will be more than one agent sharing the same computation environment. Each agent needs to communicate with other agents in order to achieve its goals. Therefore, an agent should be able to protect itself from other malicious agents. This principle is very similar to the principle where the need for an agent to protect itself from its host was addressed. The difference is that the host should also play a role in providing a secure environment for agents to communicate and work together. It is like providing law enforcement personnel in a marketplace. So, to realize this principle, an agent must be equipped with techniques to protect itself from being tampered by other agents and a host must also be able to provide some degree of protection to all agents running on it.

Principle 7:

The migration of an agent from one host to the other should be protected against unauthorized disclosure or modification [23].

The migration of an agent can be considered as the most vulnerable state in the agent life cycle. During this state, the agent suspends its execution and transforms itself into a shippable form. This shippable form should be able to seal whatever information it carries. A key solution to this problem is the use of sophisticated flawless cryptographic techniques.

Principle 8:

A host should be able to make sure that the agent is really who it is claimed to be [23] [73].

The access rights granted to an agent depend on whom the agent is. For example, let's assume that an agent is created to apply for a credit card on behalf of its owner. To achieve this task, the agent will have to migrate to a bank, fill in a credit card application form, and sign it on behalf of its creator. The bank, which is the host in this case, should be able to determine for sure whether this agent does truly represent the individual that it claims to belong to or not. The bank should be able to verify that the agent was not created by someone who is trying to steal the identity of someone else.

Principle 9:

An agent creator should be able to kill or revoke an agent.

In some cases, the creator of an agent may need to terminate or call back the agent that has been dispatched. For example, assume that an agent was created and dispatched to book a flight ticket to a certain destination on a particular date. There is a possibility that the creator of the agent may decide to cancel the task of booking a ticket. A mobile agent system must provide a way to revoke the agent or at least to inform the agent that the task it has been dispatched to achieve is no longer wanted. Revoking an agent could mean ceasing it to represent its owner. In this case, when a host tries to authenticate the agent, it will figure out that this agent is no longer a representative of its owner. In [23], it is stated "there is no easy solutions when it comes to distribution of revocation state to agent bases (hosts) in a large network environment. It is believed that it is impossible to distribute revocation state to hosts." At the very end of this thesis, an authentication idea is proposed. The implementation of this idea will be done in the future.

Principle 10:

A host should be able to detect whether a visiting agent has been maliciously modified by another host or not [23].

One of the principles mentioned earlier was the need for an agent to protect itself from malicious hosts. But what if a host managed to modify the code of an agent? The agent is still capable of migrating to hosts and it is still carrying

the true identity of its creator. A good agent could turn into a malicious one just by tampering its code. The problem here is that changing the code of an agent will change its behavior but will not change its trustworthiness. Therefore, a host must be able to verify that the code of an agent was not modified in a host that the agent has visited previously. In the model proposed by [23], the checksum method is used to ensure the code integrity of an agent. There are also some other techniques that can be used to achieve the same goal. For example, the proof-carrying code proposed by [72] can also be used for that same purpose.

Principle 11:

Communication between agents should be secured from being maliciously manipulated by the host [29].

Basically, the whole idea behind the mobile agents paradigm is to have computation entities that are capable of communicating together to achieve certain goals. If the messages between agents were modified by the some other entities, the agents would not be able to achieve their goals in the best possible way since they would make decisions based on tampered information. Therefore, a reliable mobile agent system should provide secure communication channels between the agents. Agents also should be able to detect whether a message received from another agent has been tampered with or not.

Principle 12:

Avoid risky hosts and risky agents.

When it comes to security, the best policy ever is the one used in real life; as much as possible, if you are an agent, do not go to untrusted hosts. If you are a host, do not receive suspicious agents. But this policy is not very practical and cannot be followed blindly. But still, It has to be considered and thought of when designing and deploying a mobile agent or when creating security policies for a mobile agent-enabled host. For a mobile agent, it would help very much if the agent could have list of known malicious hosts so that it avoids going there. Also for hosts, it could be very useful to have a watch list of unwanted agents or agent sources so that a host can reject an agent just by knowing its identity or its creator.

4.2 Security Techniques to Protect Hosts

In this section, a survey of the currently available security techniques used to protect hosts from malicious agents is presented. Even though the literature has plenty of these techniques, securing a host still needs a lot of research. Some of the efforts done in this area can be found in [21] [23] [24] [27] [71] [72] [73].

4.2.1 Authenticating an agent by the use of digital signature

Digital signatures can be used to verify the identities of the creator and the sender of a mobile agent. They can provide information on where and when the

agent was sent. There are a number of algorithms for digital signature. The most widely used one is a public key signature algorithm [24]. Java and Agent-Tcl provide a number of algorithms for digital signature.

Digital signature does not guarantee that that agent is harmless. It only authenticates the identity of its creator and its sender. Digital signature can also be used to make sure that the code of an agent has not been tampered with previously. Therefore, digital signature can be considered as a first line of defense. Some other techniques should be used in conjunction with digital signature in order to protect a host in case the authenticated digitally signed agent attempts to harm the host in some way.

4.2.2 Access-level monitoring and control [24]

In this technique, a mobile agent-enabled host uses a monitoring manager that has the authority to grant access rights to mobile agents as well as to keep monitoring them from accessing any resources that they are not allowed to access. One of Java's components is called the "security manager". Its equivalent in Safe-Tcl is the "master interpreter" [75]. These components can be used to monitor the running agents. If one of the running mobile agent attempts to read a file that it is not authorized to access, it will be stopped by the monitoring manager. The monitoring manager has the right to control access to files, communication ports, peripheral devices, and so on. IBM Aglet's framework also provides a restriction on the number of times a resource can be accessed.

4.2.3 Code verification [24]

In this technique, the binary image of the mobile agent code is scanned. If the code verifier finds illegal instructions in the code of the agent, the agent will be rejected and will not be allowed to execute on the host. The byte-code verifier in Java can perform this task. It can detect illegal instructions such as writing out of the agent's memory space. Since most mobile agents are Java based, this technique is quite useful. In the contrary, the Safe-Tcl execution layer does not require code verification [75].

4.2.4 Time limits

In this technique, the host decides how much time to be given to an agent to execute on its environment. This time could depend on the identity of the agent or the task it needs to achieve.

A host could also determine whether an agent is allowed to execute after its lifetime has expired or no. Some agents have a lifetime span and expiry time. When a host detects that an agent has exceeded its time span, the host may terminate the agent or send it back to its creator.

4.2.5 Range limits

In this technique, a host determines the maximum number of destinations or network hubs a mobile agent can visit before it arrives to the current host. The more hosts a mobile agent visits, the higher the possibility that its code has been

tampered with. A host may also decide to reject a mobile agent if the agent has visited a high-risk host in its way to the current one.

4.2.6 Duplication limits

A host can control the number of times an agent is allowed to duplicate itself (clone itself). Cloning is a technique used in mobile agents to divide a big task into a set of smaller ones. Each smaller task is assigned to a cloned agent to achieve. Cloning is also used when an agent wants to access a particular and this resource is not available on the current host. But the agent may not have finished its task on the current host and it may not want to migrate to another host yet to access the needed resource. A clean solution would be the creation of a new cloned agent. This cloned agent is then sent to the host where the needed resource can be found [76]. Cloning is also used to achieve persistence in agent migration. So, the cloning technique is quite good and very useful to divide and conquer a task. But what if a malicious agent keeps on cloning itself endlessly? It could easily swamp the hosts on a network. Therefore, hosts need to put a limit for agent cloning. The limit should be set wisely so that a good agent should be able to clone itself to achieve its task. A bad agent will be forced to stop cloning itself when it reached the limit.

4.2.7 Audit Logging

In this technique the host keeps log files of all the activities carried out by all the visiting agents. Whenever an attack on a host occurs, these log files can

be used to detect which agent is responsible for the attack and how exactly it did it. This technique does not protect the host but it allows it to determine who does what. The next step that should be established in the mobile agent communities is to have governmental or even better international laws that govern the behavior of mobile agents and their hosts. For example, there is a need to have a law that clearly indicates a punishment for a creator of an agent if his or her agent attempts to attack a host's security system. There is a need to have a set of agreed upon rules. It will make a huge difference in the development of mobile agents if a legal system to control the actions of agents and hosts is put in place. The owners of malicious entities should be held accountable for their actions. In conclusion, even though the audit logging technique does not provide a direct protection to hosts, it is believed that audit logging is one of the most important techniques that should exist in any mobile agent system.

Table 2 is a modified version of the one presented in [24]. It summarizes the different techniques that can be applied to mobile agents at each phase of their life cycles on a host. Some of the techniques are more suitable to be applied at the moment an agent is received by a host. For example, authenticating the credentials of the agent, verifying its code, checking its life time expiry date, checking how many host it has visited before, and logging its arrival in the audit log are all very suitable techniques to be applied to an agent upon receiving it. When an agent is in its execution phase, the host may keep an eye on it and monitor its action to prevent it from accessing any resources that

the agent is not entitled to access. The host may also keep track of the expiry date of an agent as well as how many times it has cloned itself. All activities done by an agent may always be logged in the audit log file. When an agent decides to depart, the host may just check the expiry date of the agent and log its departure.

	Arrival	Execution	Departure
Authenticating credentials	Yes		
Code verification	Yes		
Access-level monitoring and control		Yes	
Audit logging	Yes	Yes	Yes
Time limits	Yes	Yes	Yes
Range limit	Yes		
Cloning limits		Yes	

Table 2: Host protection techniques during an agent's life cycle

4.3 Security Techniques to Protect Agents

Protecting an agent is not only concerned with malicious hosts but also unreliable ones. In addition to that, an agent should also be protected from unreliable network connections. Fault tolerance techniques are used in order to protect agents from unreliable hosts and unreliable network connections. When it

comes to protecting agents from malicious hosts, encryption techniques are used. In this section, an analysis of the currently available techniques is provides.

4.3.1 Fault tolerant techniques

Fault tolerant techniques are used to make sure that the agent will survive a hardware problem or a network problem. A hardware problem could be something like a crashed host where a network problem could be a damaged network path.

4.3.1.1 Persistence

In this technique, once a host receives a mobile agent, it keeps an image of the agent's execution state saved on a persistent storage device (a hard drive). If the host crashes, the mobile agent will not be lost. When the host comes back to service, it will restart all mobile agents from its persistent storage [24]. This technique is used in the Concordia framework to ensure the persistence of its mobile agents [37].

It could happen that a host was down for quite a long period of time. When it comes back to life, it will restart all agents in its persistence storage device. If these agents do not have expiry dates, they will resume their executions. But since the host was down for a long period of time, then most probably these agents are too old to be of any valuable use to their creators. Their owners would have probably lost hope to get them back. Therefore, a mobile agent system should provide a way to revoke an agent. Otherwise, a lost and forgotten agent

could suddenly come back to life and start acting on behalf of its original creator in an unwanted manner.

4.3.1.2 Redirection

Practically, one cannot assume that all parts of a network will always be up and running. Therefore, an agent needs to be programmed to try to find alternative paths or ways of communication channels to reach its destination. Sometimes, parts of the network are down. Some other times an agent may fail to establish a certain type of communication channel like Ethernet. In such cases, the agent should be able to find out other alternatives. It should be able to redirect its path to avoid the damaged part of the network. It should also be able to establish another communication channel such as packet radio instead of Ethernet [24].

4.3.2 Encryption techniques

The encryption techniques are basically used to protect an agent from being tampered with by a malicious host. Practically, an agent carries sensitive information that needs to be hidden from hosts as well as from other agents. The area of cryptography has been a very active research field in the past few years due to the need for secure transactions over the Internet. All the work done in the field of cryptography can be used and applied to protecting agents' code and data from hosts and other agents. In this section, a survey of the commonly used encryption techniques that have been used in mobile agents is presented.

4.3.2.1 Sliding data encryption [24] [25] [79]

In this technique a mobile agent uses a public key to encrypt the data it has acquired from hosts. The encryption key is public which means it is known to hosts as well as other agents. Once the data is encrypted using the public key, they can only be decrypted by another private key that corresponds to the public key. The Concordia framework uses this technique to protect the data acquired by its agents [37].

4.3.2.2 Function encryption

Some functions of a mobile agent code are required to be hidden from hosts and other agents. For example, if the mobile agent's role is to buy an item for its creator, then the code of the agent must include a signing function that will be executed to sign documents to purchase the required item. This function and the private key of the agent's creator must be protected. In [28], a special class of polynomial and rational functions are proposed to be used together with encryption schemes to hide a function inside a mobile agent code. This hidden function can then be executed in a non-interactive protocol. This technique protects the signing function from being abused by a malicious host.

Figure 4 from [28] explains how encrypting functions works in mobile agents.

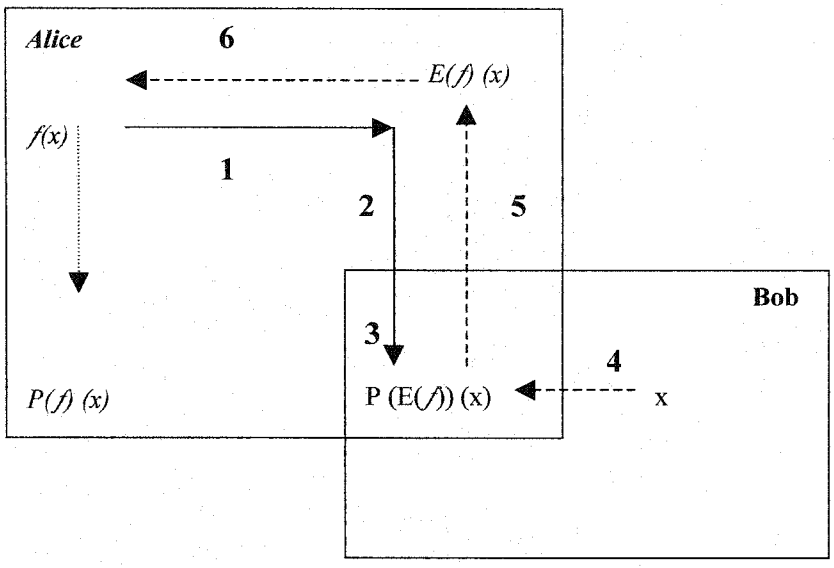

Figure 4: Computing with encrypted functions

In this example Alice represents a mobile agent and Bob represent a host. The protocol for non-interactive computing with encrypted functions looks like this:

1. Alice encrypts f
2. Alice creates a program P (E(f)) which implements E(f)
3. Alice sends P(E(f)) to Bob
4. Bob executes P(E(f)) at x
5. Bob sends P(E(f)) (x) to Alice
6. Alice decrypts P(E(f)) (x) and obtains f(x)

4.3.2.3 Trail obscuring [24] [25] [78]

Sometimes it is important for a mobile agent to hide its identity and its path throughout the network. For example, a bidding agent may want to hide where it has been before. It may also like to visit the same host more than once without being detected that it is the same agent. In [78], a technique to hide the identity of the agent by changing its own binary image is presented. This technique can only be used if the host is willing to accept anonymous agents. By changing its binary image, the mobile agent makes it harder for the host to track it by simple pattern matching since each time the same agent visits the host, the agent will have a different binary image. This technique is just making it harder for hosts to track an agent.

4.3.2.4 Code obfuscation [24] [79]

On one hand, a host needs to verify the code of an agent in order to make sure that the agent will do no harm to the host. On the other hand, an agent needs to protect its code from being exposed to a host. An agent needs to have privacy on the tasks it is doing. Code obfuscation is a technique that can be used by mobile agents to make it harder for a host to reverse engineer its code. There are many forms of code obfuscation. Blackbox security [26] [29] is one implementation of code obfuscation. In the balckbox security mechanism the code of an agent is encrypted. The agent roams the network with a special execution layer attached to it. This execution layer acts as an interface between

the encrypted agent and the host. The host has to execute the mobile agent in an exhaustive loop with different input parameters in order to be able to exploit the functionality of the agent's internal code. To totally eliminate the possibility that a host can reverse engineer and agent, [29] proposes an enhanced version of the blackbox security. He named it time limited blackbox security. In this version, the agent is given a predefined period of time during which the agent may function. When this period of time is over, the agent stops to communicate with any other entities in its environment. In addition to that, the agent does not migrate to any other host. Despite the fact that the agent will not complete its task, it will keep its internal data and code safe and secured from being exposed to a host or to other agents.

4.3.3 Other techniques for agent protection

In the last two sections, agent protection by fault tolerance techniques and encryption techniques was presented and analyzed. There are some other techniques that can also be used to protect agents from being maliciously modified by hosts or other agents.

4.3.3.1 Detection and proof of modification attacks

This approach does not focus on hiding or encrypting information. Its main concern is to have a legitimate framework for mobile agents and hosts. If a mobile agent can prove that a host has maliciously modified its data, code, flow

of control or communication with other agents, the owner of the agent can use legal or organizational ways to get the damage refunded [30].

4.3.3.2 Tamper proof hardware

In this technique, a special piece of hardware is attached to every mobile agent-enabled host. This piece of hardware is used by the agent to prove whether or not it has been modified [27] [31]. In [33], a set of processors that can be added to hosts to execute mobile agents in a totally sealed environment from the host is proposed. This technique is so powerful and so secure but it requires dedicated and expensive pieces of hardware. The associated high cost renders this technique from being applicable to a wide range of hosts. When cost is not an issue and a very degree of security is needed, this will definitely be the technique to be used.

4.3.3.3 Execution tracing

An execution trace is a log file of the history of the agent's execution on a host. These execution traces are collected, analyzed, and compared to a "supposed history of execution". In case, that the code, data, or flow of execution of a mobile agent is tampered, the agent's owner can prove that his or her agent could have never performed the actions it performed [27] [32]. Since it is believed that having some sort of a regulatory system that governs the relationship between agents and hosts will boost the deployment of mobile agents, having

such a technique in place will facilitate the construction of a mobile agents court system.

Chapter 5:
Mobile Agents and Migration

Being autonomous is definitely the most attractive feature of mobile agents. Their ability to move freely from one node to another on a network is what makes them very interesting and very appealing. In this chapter, a survey of the mobile entities in computers and computer networks is presented. Followed by a discussion of the different types of mobility control. The advantages and disadvantages of the different mobility control types are analyzed. Finally, the chapter is concluded by an up to date survey of the techniques proposed to achieve strong migration. Strong migration is defined as the ability to resume execution from exactly the point where an agent has stopped on a previous host.

5.1 Different Kinds of Mobile Entities

5.1.1 Mobile data

Data are the most commonly known mobile entity. Downloading a file from a certain computer is actually a form of data mobility. The file or at least copy of it is moved from one location to the other. Receiving an html page from a server is also a form of data mobility [34]. Mobile data are probably the most commonly known mobile entities.

5.1.2 Mobile reference

When surfing the Internet, moving from one web page to another can be viewed as a movement of a reference from one page to another. It is like having a pointer that is pointing to a particular location. When this pointer moves, all what happens is that it starts to point to another location. This kind of movement is analogous to passage by reference in a conventional programming language like C++.

5.1.3 Mobile code

The most commonly known form of mobile code is a Java applet. A Java applet is a piece of code that can be downloaded on demand from a remote server. Once the applet is downloaded to a user's computer, it starts to run without having to preinstall itself [34].

5.1.4 Mobile code and store

In this form of mobility, the mobile entity consists of code, stored values of internal data variables, and possibly an execution entry point. In other words, it is a running program in a particular state. Having the state of execution transferred along with the code allows the running program to resume its execution once it reaches its destination. This form of mobility is the one offered by most Java based mobile agent systems [34].

5.1.5 Mobile closure

Closure is defined by [34] as "the complete run time description of a computation which includes code representation and mapping between language identifiers and resource values." This form of mobility does not lose a network connection during migration because a network connection is usually represented by an identifier. A mapping of the connection identifier will be moved with the closure. Therefore, the network connection will be restored once the closure reaches its destination. Obliq [35] is one of the languages that support closure migration.

5.2 Types of Mobility Control for Mobile Agents

According to [34], there are three basic categories of mobility control for mobile agents. The mobility of an agent could be preplanned. So, before the agent is dispatched, its owner decides exactly where the agent will go. Another form of mobility control is to have no control at all. The agent decides where to go based on the information it gets from the servers it visits. So, an agent may start by going to a directory site. From the directory site, the agent can collect some information about the services provided by some other sites. Then it will decide where to go from there in order to achieve its task. The third form of control is to kick an agent from one host to another. It is simply forcing the agent to migrate to a particular host. In this section, a description of these forms of mobility control is provided.

5.2.1 Planned mobility

In this type of mobility control, the agent is provided by an itinerary before it starts its trip. The itinerary lists all the sites that the agent will need to visit. In some cases, the itinerary also associates a particular function with each site an agent will visit. Having a predefined set of hosts to visit, an agent will not worry to compute its targeted sites. Therefore, the mobility code can be decoupled from the application code. Concordia, Aglet, Odyssey, and Macondo support this type of mobility control.

5.2.2 Spontaneous mobility

In this type of mobility, the agent calculates what hosts it needs to visit at run time. This type of mobility control facilitates developing an intelligent mobile agent. Assume that the task of a mobile agent is to get the best possible deal for booking an air ticket to Paris. As an agent owner, it should not be my responsibility to tell the agent where to go to get me the best possible deal. I am willing to provide it with a couple of directory sites but not a comprehensive list of all relevant sites. My agent should be intelligent enough to go to these directory sites, query about travel agents' sites, airlines sites, and any other sites that may be useful. Then it should go to these sites and fetch me the best possible deal.

Furthermore, in order to achieve network load balancing, this kind of mobility control has to be supported. If an agent is running on an overloaded node on a network, the agent should be smart enough to detect a less loaded node, migrate to it, and resume its work on the new node.

All Aglet, Voyager, Odyssey, Mole, and Macondo frameworks support spontaneous mobility control. It is worth noting that for certain type of applications; the preplanned itinerary is more efficient than the spontaneous mobility. Therefore, a practical mobile agent framework should sport both types of mobility control.

5.2.3 Controllable mobility

In some cases, an owner of an agent or the host that is running an agent needs to force the agent to move to another site. The agent does not migrate voluntarily but it is forced to move as per a request from some authority. For example, collecting an agent back is a form of controllable mobility. The owner of an agent forces its agent to come back to its home.

Table 3 from [34] provides a comparison between the different mobile agents frameworks and their support to the different types of mobility control.

Type Of Mobility	Concordia	Aglet	Voyager	Odyssey	Mole	Macondo
Planned	Yes	Yes	No	Yes	No	Yes
Spontaneous	No	Yes	Yes	Yes	Yes	Yes
Controllable	No	Yes	Yes	No	No	Yes

Table 3: Mobility control in different mobile agent frameworks

5.3 Mobile Agent Migration Techniques

In this section, a survey of the latest migration techniques for mobile agents is presented. Transparent and reliable migration has always been a goal to realize in any commercial or academic mobile agent framework. In particular, strong migration is definitely the most challenging goal to achieve. As mentioned in chapter 3, most mobile agent frameworks are Java-based. Java does not directly support strong migration. Basically, this is why strong migration was not an easy task to fulfill. The content of this section is an attempt to gather some of the latest and most promising migration mechanisms.

5.3.1 Transparent stack and program counter migration in Java

In [80], a mechanism is proposed for transparent stack and program counter migration of a Java thread. All mobile agents in all Java-based frameworks are Java programs that have their own threads. Using the Java Platform Debugger Architecture (JPDA), [80] could access and capture the stack frames, the local variables, and the program counter of a Java program. Once captured, these pieces of information could be packed and shipped with the migrating agent. When the agent reaches its destination, can set back the program counter along with all the other information in the new host computation environment using the JPDA. Setting this information is done either at the byte code level or at the portable parts of the virtual machine. Since the JPDA is part of Virtual Machine Specification of Java, the approach proposed by [80] is portable to different operating systems and different version of JDK.

5.3.2 Strong mobility by adding markers in an inherently weak mobility languages

Strong mobility is defined by [81] as "the movement of code and of the execution state of a thread to a different site and the resumption of its execution on arrival." Where weak mobility is defined as "the dynamic linking of code arriving from a different site". Strong mobility is better than weak mobility because the agent will be able to resume its execution from exactly the same point it has stopped at on its previous host. In [81], a generic algorithm is proposed to traverse the code of a running agent and insert a unique marker at every location where a migration call may be executed. These markers are shipped along with the code and data state of a migrating agent. The agent will resume its execution at the marker where the migration call was executed. By doing that, the agent will resume its execution from exactly the point it has stopped. This technique is language independent. It can be applied to any programming language that supports weak mobility.

5.3.3 Strong mobility by using Java's VM and the exception/error handling mechanism

In [82], a very interesting technique is used to capture and transfer the execution state of a Java program by just using a preprocessor and Java's exception/error handling mechanism. When a migration call is executed by a Java-based agent, its internal execution state can be captures as follows:

- The preprocessor inserts a special try/catch statement for each method that might initiate a migration request in order to save its local variable.
- When a migration request is initiated, the method that has initiated it will save its local variables and then throw an error
- The error will keep on propagating through out all methods in the stack allowing them to save their local variables.

5.3.4 Strong mobility by implementing a Prolog interpreter on the Java VM

In [83], a Prolog interpreter is implemented on the Java VM in order to capture the agent's current execution stack image. The prolog interpreter allows capturing the execution state of a running Java program by using the serialization techniques for Java objects. The technique presented in [83] enables agents to reactively and autonomously migrate from one location to another. In addition to that, the MiLog, implements strong migration of agents in prolog.

Chapter 6:
Mobile Agents and Coordination

It is clear that in most real life cases, heterogeneous mobile agents need to communicate together to achieve their tasks. Most current mobile agent systems are not compatible with each other. A mobile agent that has been developed on one framework cannot coordinate with another mobile agent that has been developed on another framework [42]. Solving this problem can be achieved by having standards for mobile agents' communication. MASIF and FIPA are two successful attempts to standardize the architecture of mobile agent systems. FIPA was developed by a collaborative work exerted by various organizations. The main goal of setting standards is to have interoperable architectures of different mobile agent systems. In this chapter both standards are analyzed and examined. It is believed that setting standards is vital to increase the usability and usefulness of the mobile agent paradigm.

The development of agent specific communication languages was also a very positive step towards interoperability between different agents. The second part of this chapter is dedicated to survey and analyze some of the very promising agent communication languages that are available in the market up to the date of this thesis.

6.1 Standards for Mobile Agents Architectures

6.1.1 MASIF (Mobile Agent System Interoperability Facility)

The OMG (Object Management Group, Inc) proposed the MASIF [43] specification for mobile agent architectures. MASIF defines a set of interfaces that facilitate the interoperability between mobile agents developed on different frameworks. MASIF is a CORBA based standard. The goal of MASIF is to achieve interoperability between existing mobile agent frameworks without forcing any radical changes to the implementation of the existing systems [45]. Basically, MASIF proposes a standard for agent names, agent system names, agent system types, and location syntax [45]. MASIF consists of two main interfaces: MAFAgentSystem and MAFFinder. The MAFAgentSystem provides operations for the management and transfer of agents where MAFFinder provides operations for the localization of agents. The MASIF MAFAgentSystem interacts internally with the mobile agent system service functions and provides the visiting agent with the equivalent CORBA interface. In other words, the mobile agent framework specific functions are wrapped into a standard CORBA interfaces. Therefore, an agent developed using any MASIF–compliant framework can operate on and interact with any other MASIF–compliant mobile agent system.

IBM Aglets have recently announced their intention to be a fully MASIF compliant in the near future. Grasshopper framework [46] is also MASIF compliant. Therefore, in the near future, Aglets will be supported by any Grasshopper host and vise versa.

MASIF proposes a very comprehensive set of standards for agent management, agent tracking, and agent transport. But when it comes to communication between heterogeneous agents, MASIF does not indicate how this kind of communication can take place [45]. The lack of standards in agent-to-agent communication makes MASIF a limited standard to address the issue of interoperability in a complete way. MASIF sets some rules to make a host handle different agents coming from different platforms. But there is no way for these agents to communicate together. Standards to establish an agent community or a marketplace are not provided by MASIF. The basic advantage of MASIF is that being MASIF compliant does not require dramatic changes to the current implementations of the frameworks. That easiness to be MASIF compliant makes it feasible for current commercial frameworks like Aglets to announce that they will be MASIF compliant in the near future.

6.1.2 FIPA (Foundation for Intelligent Physical Agents) [44]

FIPA was formed in 1996 as a forum of international companies with a strong interest in telecommunication [47]. The official mission statement of FIPA is: "The promotion of technologies and interoperability specifications that facilitate the end-to-end inter-working of intelligent agent systems in modern commercial and industrial settings" [47]. FIPA provides a set of standards that can be realized in a number of commercially available software development environments. Figure 5 from [48] shows the abstract architecture and its mapping to concrete realizations.

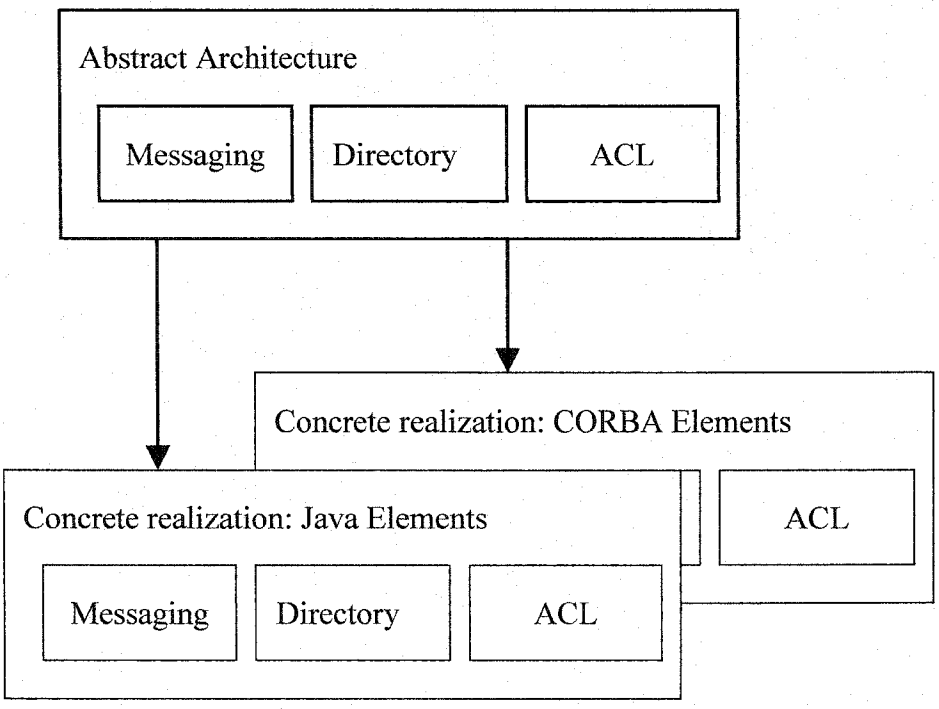

Figure 5: Abstract FIPA architecture mapped to various concrete realizations

FIPA focuses on the definition of general standard languages and protocols for communication, coordination, and management of heterogeneous agents [45]. FIPA proposes an Agent Communication Language (ACL) as part of its standard. This language is used as a mean of communication between FIPA-compliant agents. FIPA also focuses on how agents can communicate with other entities such as humans, non-agent software, and the physical world [45]. FIPA has three basic components: the Agent Management System (AMS), the Directory Facilitator (DF), and the Agent Communication Channel (ACC). AMS provides operations for mobile agent management. The DF is used by mobile agents to advertise their own services and search for available ones. The ACC

enables communication between agents. Each message exchanged between two FIPA-compliant agents has to have a certain structure. This structure is illustrated in Figure 6 [48].

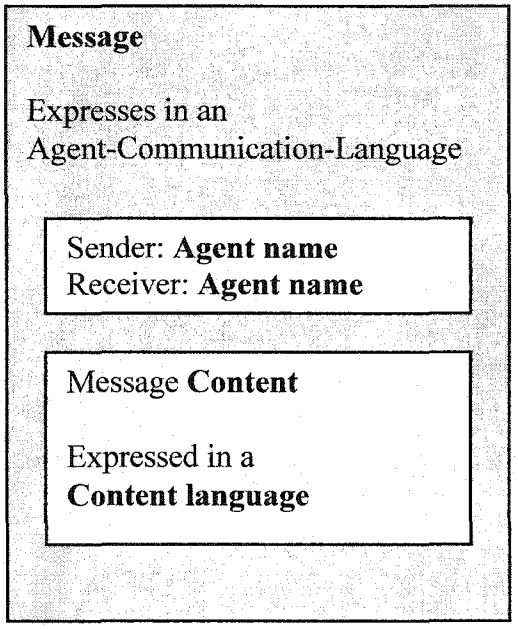

Figure 6: FIPA message structure

The content of each message has to be expressed in a content language. FIPA approved the following languages as content languages: SL (Semantic language), RDF, KIF (Knowledge Interchange Format), and CCL.

The Directory Facilitator contains a list of entries; one entry for each agent. Each agent has a unique name, a set of locators, and a set of attributes. Locators are used by other agents to send messages to the agent. These locators are particular forms of message transport such as IIOP, SMTP, or

HTTP. For example, Agent ABC can specify the following locators as means of communication with it:

Locators:

Type	**Specific address**
HTTP	http://www.AgentABC.ca/
SMTP	AgentABC@SomeAddress.ca

Figure 7 from [48] shows how locators can be used in communication between agents

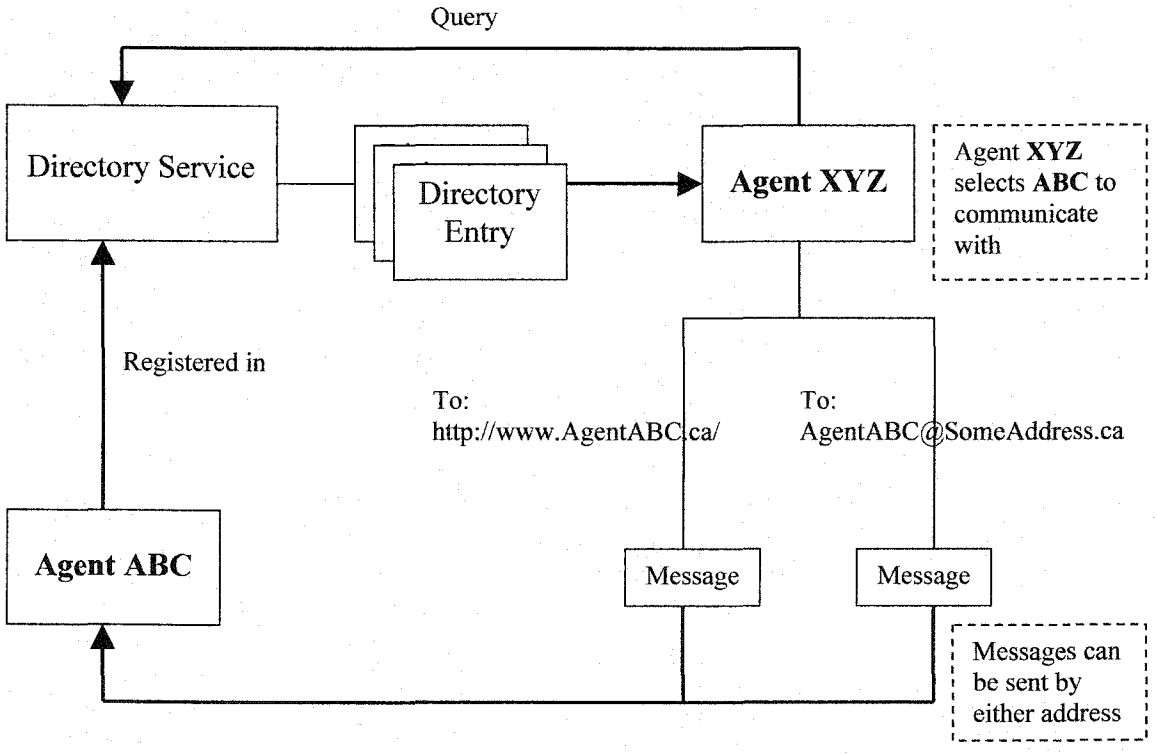

Figure 7: Communication between agents using locators

Few FIPA compliant frameworks have been developed so far. JADE and FIPA-OS are examples of FIPA compliant frameworks. JADE (Java Agent Development Framework) [50] is an academic agent framework development environment that is designed and implemented having FIPA standards in mind. JADE is an open source project that can be downloaded from [51]. As its name indicates, JADE is developed using Java.

FIPA-OS is another open source system originally developed by Nortel Networks [2]. FIPA-OS platform supports communication by using the FIPA agent communication language standards. It provides a set of services that are specified by FIPA agent standards such as an agent management system for life-cycle management, a directory facilitator, and an agent communication channel for FIPA-compliant messaging and interaction protocol. The latest information about FIPA-OS and the latest code can be found at [65] and [66].

6.2 Agent Communication Languages

Ideally, all agents - regardless of their implementation environment - should be able to communicate using a single universally understood language. But this statement is like saying all people on the universe should speak the same language. Realistically, it will never happen that all mobile agents will adopt the same language. Basically, there are two types of agent communication: direct and mediated [49]. The direct type is achieved by using one of the sets of standardized languages such as KQML, ARCOL, FIPA-ACL, and ICL. All these are well-defined languages that can be adopted (or supported) by any mobile

agent framework. The second type of communication relies on having a facilitator layer between the heterogeneous agents. This facilitator layer acts as a translator (mediator) between the heterogeneous agents. It communicates with one agent using one language, translates the agent's message to another language, and sends it to the other agent. CORBA with all its brokering facilities is considered one of the most promising languages to implement the facilitator layer. In this section some of the most widely known agent communication languages are presented.

6.2.1 KQML

The Knowledge Query and Manipulation Language (KQML) is a general-purpose language that facilitates the communication between several agents [49] [53]. KQML is the result of a research done by the Knowledge Sharing Effort (KSE). This research group designed the Knowledge Interchange Format (KIF) as a common language to represent the content of the messages exchanged between agents [49]. The KQML language has a set of reserved words that can be used to share meaning between agents. For example, *tell* is one of the reserved keywords. When an agent A send a *tell* message to agent B, then the meaning is:

> Agent A wants to tell agent B that the content of the message is in agent A's knowledge base.

Another reserved word is *error*. When agent A sends an *error* message to agent B, then agent B can understand that:

Agent A considered agent B's previous message as ill-formed

6.2.2 ARCOL

ARCOL is the agent language developed by the France Telecom for their ARTIMIS agent technology framework [49]. ARCOL uses the Semantic Language (SL) to represent the content of its messages. ARCOL has a set of primitives for communication. For example, *Inform*, *Request*, and *Confirmation* are all predefined primitives that are used to exchange messages between agents.

6.2.3 FIPA-ACL

FIPA-ACL language is inspired by both ARCOL and KQML. As mentioned earlier, the content of the message can be represented by a number of well-defined languages such as KIF or SL. FIPA's ACL has a very precise and formal set of primitives. These primitives are categorized into three main sections: action primitives, information primitives, and negotiation primitives. *Agree, Refuse, Cancel,* and *Propagate* are examples of action primitives defined by FIPA ACL. Where *Inform, Inform-if,* and *Disconfirm* are some of the information primitives. Negotiation primitives include *Propose, Accept-proposal, Reject-proposal,* and *Query-if*. Since FIPA is an organization that is composed of

several parties in industry and academia, its standard language has a very well defined formal semantics foundation [49] [53].

6.2.4 ICL

The Open Agent Architecture (OAA) is designed by SRI International [52] in order to integrate heterogeneous agents in a distributed system. To achieve this goal, SRI developed a logic-based declarative language called InterAgent Communication Language (ICL) [49]. This language uses facilitators to achieve communication between different agents as well as between software and agents and humans. Requests (from software agents or humans) are processed by an ICL kernel and represented in ICL expression format. This means, the ICL language has a natural language processing component that is capable of transforming a natural language request like "Send a Merry Christmas message to all my friends" to a formal ICL expression like send_message (email, 'all', [subject (Merry Christmas)]). ICL uses a prolog-like language as a language to represent the content of its messages. The main drawback of ICL is that it is tied to one particular agent architecture; this architecture is the OAA [49].

Chapter 7:
Future Work

In this chapter, a new idea for revoking an agent is proposed. The implementation of the revoking mechanism presented in this section will be done in the future. The need to regulate the relationships between agents and hosts, setting rules, enforcing laws, tracking and catching malicious agents or hosts is emphasized. The existence of a legal system that governs and sets the rules for both agents and hosts will make it safer and easier for mobile agents to widely spread without having to implement very tight security mechanisms for hosts and agents. Lots of research in various areas such as sociology, law, and of course computer science is needed to realistically establish a legal system for the new paradigm of mobile agents.

7.1 A Revocation Mechanism for Mobile Agents

In chapter 4, one of the security challenges was how to revoke a mobile agent. Revoking a mobile agent means ceasing it to represent its owner. The development of a mobile agent registry server is believed to be a possible solution for this problem. The mechanism may go as follows: When a mobile agent is created and dispatched to the network, its owner has to register it on a known and trusted mobile agent registry server. The creator of an agent can un-register it at any point of time. The agent should check its registration every now and then. The owner of an agent sets the frequency according to which an agent will check the registry server. If the agent finds out that it has been unregistered,

it must report back to its owner and kill itself immediately. The authentication process that a host performs on an agent to verify that the agent is really what it has claimed to be may include checking the mobile agent registry server. If the agent's registration is confirmed with the registry server, the rest of the authentication process may take place. Otherwise, the agent has to be rejected and sent back to its owner.

7.2 Law Enforcement Agents

It is believed that one of the best ways to establish a secure environment for both hosts and agents is to have some law enforcement agents. These very special agents will play the role of e-policemen in the new world of e-agents and e-markets. Even though it sounds like a science fiction idea, most of the ingredients to establish such an environment are already there. Hosts can keep logs of the activities performed by agents (please revise chapter 4). These logs can be used as documents to prove or disprove that a particular agent has or has not committed a wrong doing act. These logs can be digitally protected from being forged or modified by an unauthorized party. Having such a tracking system, the notion of trust between agents and hosts can then be introduced. An agent will always be considered friendly unless proven otherwise. Once an agent is proven to be malicious, its owner will be persecuted and will be held responsible for whatever damages his or her agent may have caused.

Chapter 8: Conclusion

The contribution of this thesis to the research community is to provide an up to date survey of the currently available technologies for mobile agents in different aspects along with an analysis to show the pros and cons of the currently available techniques. No doubt, the mobile agent paradigm is an extremely promising field. It is expected to revolutionize the Internet some day. However, there exist some barriers for this developing paradigm to be overcome. Security comes at the very top of the barriers list. Followed by the need to have standards for collaboration so that agents coming from different frameworks can communicate and cooperate together. Then the need to have programming languages support for some specific mobile agent aspects such as the need for strong migration support, the need to have constructs to adequately represent the agent's believes, the need to have strong security mechanisms for both the running code and its data, and the need to have a high degree of interoperability. Last but surely not least, people need to be socially prepared to accept and trust electronic intelligent mobile agents that can make decisions on their behalf. Until progress is made in these areas, the field of mobile agents will remain somewhat limited.

References:

[1] Danny B. Lange and Mitsuru Oshima: *Programming and Deploying JAVA Mobile Agents with Aglets.* Addison-Wesley, 1998.

[2] Joseph P. Bigus and Jennifer Bigus: *Constructing Intelligent Agents Using JAVA: Second Edition.* John Wiley & Sons, Inc. 2001.

[3] Xiabo Fan: *Applying Mobile Agents to Implement to Authentication in Large Scale Networks.* McGill University, 1999.

[4] Vu Anh Pham and Ahmed Karmouch: "Mobile Software Agents: An Overview." IEEE Communications Magazine, July 1998. P. 26-37.

[5] Ahmed Karmouch: "Mobile Software Agents for Telecommunication." IEEE Communications Magazine, July 1998. P. 24-25.

[6] Danny B. Lange and Mitsuru Oshima: "Seven Good Reasons for Mobile Agents." Communications of the ACM, March 1999. Vol. 42, No. 3. P. 88-89.

[7] Bill Joy: "Shift from Protocols to Agents." IEEE Internet Computing, January-February 2000. P. 63-64.

[8] http://www.agent.org

[9] Partha Sarathi Dutta, Sandip Debnath, and Sandep Sen: "A Shopper's Assistant." In Proceedings of the 5th ACM International Conference on Autonomous Agents, Montreal, Quebec, June 2001. P59-60.

[10] Joan Morris, and Paul P. Maglio: "When Buying On-line, Does Price *Really* Matter?" MIT Press 2000.

[11] Dejan S. Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou: "Process Migration." ACM Computing Surveys, Vol. 32, No. 3, September 2000. P241-299.

[12] Detlef Schoder and Torsten Eymann: "The Real Challenges of Mobile Agents." Communication of the ACM, Vol. 43, No. 6, June 2000. P111-112.

[13] David Chess, Colin Harrison, and Aaron Kershenbaum: "Mobile Agents: Are They a Good Idea?" Research Report, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY. Mar., 1995; www.research.ibm.com/iagents/publications.html.

[14] David Wong, Noemi Paciorek, and Dana Moore: "Java-based Mobile Agents." Communications of the ACM, Vol. 42, No. 3, March 1999.

[15] Ravi Jain, Farooq Anjum, and Amjad Umar: "A comparison of Mobile Agent and Client-Server Paradigms for Information Retrieval Tasks in Virtual Enterprises." IEEE Proceedings on the Academia/ Industry Working conference, 2000. P209-213.

[16] Do van Thanh: "Using Mobile Agents in Telecommunication." IEEE Proceedings of the 12th International Workshop on Database and Expert Systems Applications, 2001. P685-688.

[17] B. Pagurek, A, Bieszczad, and T. White: "Mobile Agents for Network Management." IEEE Communication Surveys, September 1998.

[18] Ahmed Karmouch: "Mobile Software Agents for Telecommunications." IEEE Communications Magazine, July 1998. P24-25.

[19] Y. Yemini: "The OSI Network Management Model." IEEE Communication Magazine, May 1993, P20-29.

[20] W. M. Farmer, J. D. Guttman, and V. Swarup: "Security for Mobile Agents: Issues and Requirements." Proceedings of the 19th International Conference on Information System Security, Baltimore, MD, October 1996. P591-597.

[21] Sebastian Fischmeister, Giovanni Vigna, and Richard A. Kemmerer: "Evaluating the Security of Three Java-Based Mobile Agent Systems." Lecture Notes in Computer Science 2240. Springer-Verlag, May 2001, P31-41.

[22] Moses Ma: "Agents in E-commerce." Communications of the ACM, Vol. 42, No. 3, March 1999. P79-80.

[23] Vijay Varadharajan: "Security Enhanced Mobile Agents." Proceedings of the 7th ACM Conference on Computer and Communication Security 2000. P200-209.

[24] Michael S. Greenberg, Jennifer C. Byington, and David G. Harper: "Mobile Agents and Security." IEEE Communications Magazine, July 1998, P76-85.

[25] A. Young and M. Yung: "Sliding Encryption: A Cryptographic Tool for Mobile Agents." Proceedings of the 4th International Workshop on Fast Software Encryption, 1997.

[26] Fritz Hohl: "Protecting Mobile Agents with Blackbox security." Proceedings of Mobile Agents and Security Workshop, University of Maryland, October, 1997.

[27] Hock Kim Tan, and Luc Moreau: "Trust Relationships in a Mobile Agent System." Lecture Notes in Computer Science 2240. Springer-Verlag, May 2001, P15-30.

[28] Tomas Sander, and Christian F. Tschudin: "Protecting Mobile Agents Against Malicious Hosts." Lecture Notes in Computer Science 1419. Springer-Verlag, February 1998, P44-60.

[29] Fritz Hohl: "Time Limited Blackbox: Protecting Mobile Agents From Malicious Hosts." Lecture Notes in Computer Science 1419. Springer-Verlag, February 1998, P92-113.

[30] Giovanni Vigna: "Protecting Mobile Agents Through Tracing." In Proceedings of the third ECOOP Workshop on Operating System Support for Mobile Object Systems, 1997.

[31] U. G. Wilhelm, S. Staamann, and L. Buttyan: "Introducing Trusted Third Parties to the Mobile Agent Paradigm." Lecture Notes in Computer Science 1603. Springer-Verlag, February 1999.

[32] Giovanni Vigna: "Cryptographic Traces for Mobile Agenst." Lecture Notes in Computer Science 1419. Springer-Verlag, February 1998, P137-153.

[33] U. G. Wilhelm. "Cryptographically Protected Objects." Technical Report, Ecole Polytechnique Federale de Lausanne, Switzerland 1997.

[34] Paolo Ciancarini, Andrea Giovannini, and Davide Rossi: "Mobility and Coordination for Distributed Java Applications." Advances in Distributed Systems, Lecture Notes in Computer Science 1752. Springer-Verlag, 2000, P402-425.

[35] L. Cardelli: "A Language with Distributed Scope." Proceedings of the 22nd ACM Symposium on Principles of Programming Languages, 1995. P286-298

[36] Concordia framework: http://www.concordiaagents.com/

[37] Reuven Koblick: "Concordia." Communications of the ACM, Vol. 42, No. 3, March 1999. P96-97.

[38] Voyager home page: http://www.recursionsw.com/products/voyager/orb.asp

[39] Odyssey home page: http://www.genmagic.com/

[40] Mole home page: http://mole.informatik.uni-stuttgart.de/

[41] Macondo home page: http://www.cs.unibo.it/~cianca/wwwpages/macondo/

[42] Myeong-Jae Yi, Yang-Woo Yu, Jin-Hong Kim, Yang-Soo Park, and Myung-Joon Lee: "SMART: A CORBA Based Mobile Agent System Supporting the OMG Specification." IEEE Communication Magazine, 2000. P70-74.

[43] Dejan Milojicic, Markus Breugst, Ingo Busse, John Campbell, Stefan Covaci, Barry Friedman, Kazuya Kosaka, Danny Lange, Kouichi Ono, Mitsuru Oshima, Cynthia Tham, Sankar Virdhagriswaran, and Jim White:

"MASIF: The OMG Mobile Agent System Interoperability Facility." Lecture Notes in Computer Science 1477. Springer-Verlag, 1998, P50-67.

[44] Foundation for Intelligent Physical Agents – FIPA '99 version 0.2 http://www.fipa.org/

[45] Paolo Bellavista, and Cesare Stefanelli: "CORBA Solutions for Interoperability in Mobile Agent Environments." IEEE Proceedings on the International Symposium on Distributed Objects and Applications, 2000. P283-292.

[46] IKV++ - Grasshopper 2, http://www.ikv.de/products.

[47] Stefan Poslad and Patricia Charlton: "Standardizing Agent Interoperability; The FIPA Approach." Lecture Notes in Computer Science 2086. Springer-Verlag, 2001, P98-117.

[48] FIPA: "FIPA Abstract Architecture Specification." http://www.fipa.org/, Document number XC00001J. August 2001.

[49] Mamadou Tadiou Kone, Akira Shimazu, and Tatsuo Nakajima: "Critical Review: The State of the Art in Agent Communication Languages." Springer-Verlag, Knowledge and Information Systems, Vol. 2 Issue 3, 2000. P259-284.

[50] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa: "JADE: A FIPA2000 Compliant Agent Development Environment." ACM Proceedings of the Fifth International Conference on Autonomous Agents, June 2001, Montreal, Quebec, Canada. P216-217.

[51] JADE project home page: http://sharon.cselt.it/projects/jade.

[52] Open Agent Architecture by SRI International home page: http://www.ai.sri.com/~oaa

[53] Yannis Labrou: "Standardizing Agent Communication." Lecture Notes in Artificial Intelligence 2086. Springer-Verlag, 2001, P74-97.

[54] Paolo Bellavista, Antonio Corradi, Domenico Cotroneo, and Stefano Russo: "Integrating Mobile Agent Infrastructures with CORBA-based Multimedia Applications." IEEE Proceedings of the 9th Euromicro Workshop on Parallel and Distributed Processing, 2001. P121-128.

[55] Michael Wooldridge: "Intelligent Agent." *Multiagent Systems: A Modern Approach to Ditributed Artificial intelligence. Edited by Gerhard Weiss.* The MIT Press, London. 1999.

[56] Robert S. Gray: "Agent TCL." Dr. Dobb's Journal March 1997.

[57] Herald Kosch: "CORBA, Web and Databases." IEEE Proceedings on the International Symposium on Distributed Objects and Applications, 2000. P674.

[58] Steve Vinoski: "Introduction to CORBA.", ACM Proceedings of the 2000 International Conference on Software Engineering, 2000.P822.

[59] Object Management Group 1999. *The Common Object Request Broker: Architecture and Specification.* Revision 2.3.1. FTP://ftp.omg.org/pub/docs/formal/99-10-07.pdf. Framingham, MA: Object Management Group.

[60] Marie-Pierre Gervais, and Alioune Diagne: "Enhancing Telecommunications Service Engineering with Mobile Agent Technology

and Formal Methods." IEEE Communications Magazine, July 1998. P. 38-43.

[61] Kiminori Sugauchi, Satoshi Miyazaki, Stefan Covaco, and Tianning Zhang: "Efficiency Evaluation of a Mobile Agent Based Network Management System." Lecture Notes in Computer Science 1597, Springer-Verlag, 1999. P527-535.

[62] George Samaras, Marios D. Dikaiakos, Constantinos Spyrou, and Andreas Liverdos: "Mobile Agent Platforms for Web Databases: A qualitative and Quantitative Assessment." The IEEE Proceedings of the Third International Symposium on Mobile Agents, 1999. P50-64.

[63] Rahul Jha and Sridhar Iyer: "Performance Evaluation of Mobile Agents for E-commerce Applications." Proceedings of 8th International Conference on High Performance Computing, Hyderabad, India, Lecture Notes in Computer Science 2228. Springer-Verlag, December, 2001. P331-340.

[64] Marios Dikaiakos, Melinos Kyriakou, and George Samaras: "Performance Evaluation of Mobile-Agent Middleware: A Hierarchical Approach." Proceedings of the 5th International Conference on Mobile Agents, Atlanta, GA, USA, Lecture Notes in Computer Science 2240. Springer-Verlag, December, 2001. P244-259.

[65] FIPA-OS: http://fipa-os.sourceforge.net/

[66] Nortel Networks: FIPA-OS: http://www.nortelnetworks.com/products/announcements/fipa/

[67] Volker Roth: "On the Robustness of Some Cryptographic Protocols for Mobile Agent Protection." Lecture Notes in Computer Science 2240. Springer-Verlag, December, 2001. P1-14.

[68] A. Corradi, R. Montanari, and C. Stefanilli: "Mobile Agents Protection in the Internet Environment." Proceedings of the 23rd Annual International Computer Software and Applications Conference, 1999. P80-85.

[69] G. Karjoth, N. Asokan, and C. Gulcu: "Protecting the Computation Results of Free-Roaming agents." Proceedings of the Second International Workshop on Mobile Agents, Lecture Notes in Computer Science 1477, Springer-Verlag, September 1998. P195-207.

[70] G. Karjoth: "Secure Mobile Agent-based Merchant Brokering in Distributed Marketplaces." Lecture Notes in Computer Science 1882. Springer-Verlag, 2000. P44-56.

[71] Ciaran Bryce, and Jan Vitek: "The JavaSeal Mobile Agent Kernel." The IEEE Proceedings of the Third International Symposium on Mobile Agents, 1999. P103-116.

[72] George C. Necula and Peter Lee: "Safe, Untrusted Agents Using Proof-Carrying Code." Lecture Notes in Computer Science 1419. Springer-Verlag, February 1998, P61-91.

[73] Shimshon Berkovits, Joshua D. Guttman, and Vipin Swarup: "Authentication for Mobile Agents." Lecture Notes in Computer Science 1419. Springer-Verlag, February 1998, P114-136.

[74] Gunter Karjoth, Danny B. Lange, and Mitsuru Oshima: "A Security Model for Aglets." Lecture Notes in Computer Science 1419. Springer-Verlag, February 1998, P188-205.

[75] John K. Ousterhout, Jacob Y. Levy, and Brent B. Welch: "The Safe-TCL Security Model." Lecture Notes in Computer Science 1419. Springer-Verlag, February 1998, P218-234.

[76] Onn Shehory, Katia Sycara, Prasad Chalasani, and Somesh Jha: "Agent Cloning: An Approach to Agent Mobility and Resource Allocation." IEEE Communications Magazine, July 1998. P58-67.

[77] Dennis Volpano and Geoffrey Smith: "Languages Issues in Mobile Program Security." Lecture Notes in Computer Science 1419. Springer-Verlag, February 1998, P25-43.

[78] A. Young and M. Yung: "Sliding Encryption: A cryptographic Tool for Mobile Agent." In Proceedings of the Fourth International Workshop on Fast Software Encryption, 1997.

[79] James Riordan and Bruce Schneier: "Environmntal Key Generation Towards Clueless Agents." Lecture Notes in Computer Science 1419. Springer-Verlag, February 1998, P15-24.

[80] Torsten Illmann, Tilman Krueger, Frank Kargl, and Michael Weber: "Transparent Migration of Mobile Agents Using the Java Platform Debugger Architecture." Lecture Notes in Computer Science 2240. Springer-Verlag, May 2001, P198-212.

[81] Lorenzo Bettini, and Rocco De Nicola: "Translating Strong Mobility into Weak Mobility." Lecture Notes in Computer Science 2240. Springer-Verlag, May 2001, P182-197.

[82] Hong Wang, Guangzhou Zeng, and Shouxun Lin: "A strong Migration Method of Mobile Agents Based on Java." In Proceedings of the Sixth International Conference on Computer Supported Cooperative Work in Design, 2001. P313-318.

[83] Naoki Fukuta, Takayuki Ito, and Toramatsu Shintani: "An approach to Building Mobile Intelligent Agents Based on Anytime Migration." Lecture Notes in Artificial Intelligence 2112. Springer-Verlag, 2001, P219-228.

[84] Gian Pietro Picco: "Mobile Agents: State of the Art and Research Opportunities." Lecture Notes in Artificial Intelligence 2182. Springer-Verlag, 2001, P247.