A Secure Mobile Agents Platform

Leila Ismail
College of IT
United Arab Emirates University
P.O.Box 17551, Al-Ain, United Arab Emirates
Email: leila@uaeu.ac.ae

***Abstract*—Mobile Agents is a new paradigm for distributed computing where security is very essential to the acceptance of this paradigm in a large scale distributed environment. In this paper, we propose protection mechanisms for mobile agents. In these mechanisms, the authentication of mobile agents and the access control to the system resources are controlled by the mobile-agents platform. Each agent defines its own access control policy with regard to other agents using an Interface Definition Language (IDL), thus enforcing modularity and easing programming task. An evaluation of these mechanisms has been conducted. The measurements give the overhead involved by the proposed protection mechanisms to the performance of mobile agents. An important advantage of our protection mechanisms are transparency to agents and the portability of non-secure applications onto a secure environment. A mobile agent system and the protection mechanisms have been implemented. Our experiments have shown the feasibility and the advantages of our mechanisms.**

***Index Terms*—Mobile agents, security, authentication, access control**

I. Introduction

Mobile-agents technology has emerged to build distributed computing over the Internet [1]. A mobile agent is a process with its own code and data that can migrate in the network from one node (called agent server) to another to perform a specific task on behalf of their users. Mobile agents representing different users on a global network can meet and interact with other agents while migrating in the network. A mobile-agents platform is a distributed middleware that is responsible to create, execute, migrate, send, receive and destroy mobile agents. It also provides communication facilities between mobile agents ([2], [3]).

As mobile agents are intended to be used over large-scale distributed systems [4], security becomes an essential issue to resolve. When received over the network by host servers, a mobile agent must not access resources which it does not have authorization to. Receiving hosts need to have the assurance that a received mobile agent is not malicious. Also, other mobile agents running on the host servers need to have the assurance of whom they are communicating with and consequently give appropriate access rights.

Received by host servers, a mobile agent can invoke objects exported either by the servers, or by other agents running on these servers. In this context, protection has become an extremely important issue: nobody will use the mobile-agent paradigm if there are no protection mechanisms which assure the host server and other agents running on this server that the mobile agent will not damage information of the server and of the other agents. Java [5] is probably the best known runtime environment which provides facilities for implementing mobile-code based applications and protection mechanisms. The Java compiler generates a bytecode which is interpreted by the Java virtual machine, thus enabling code transfer between heterogeneous hosts. From protection perspectives, the main advantage of Java is the implementation of a sandbox [6] which limits the instructions and consequently the resources used by a mobile code. The Java sandbox is responsible for protecting a number of resources at a number of levels: memory, file system and disk. The memory is protected because the java language is type-safe, which means that the Java language does not allow the use of virtual addresses. The file system and the disk are protected through the use of an access control mechanism based on a policy file. However, the language does not provide mechanisms for mobile agents and servers to define different rights for different mobile agents based on authentication, and to allow these rights to evolve dynamically during communication and execution, and to move with agents during migration.

To allow dynamic exchange of access rights in a flexible way between cooperating agents, we have proposed that agents use capabilities for access control ([7], [9]). In this initial proposal of protection, the mobility characteristic of an agent was not considered. In particular, when moving from one server host to another, an agent needs to carry its capabilities with it and to export them for other agents running on the destination server. An evaluation of feasibility and advantages in a real mobile-agents system was not done. Furthermore, to exchange initial capabilities, mobile agents must authenticate each others. An initial capability could include the minimum access rights to be given to the agent. More capabilities can be granted dynamically while agents are communicating. Our recent work on *network* authentication for a mobile-agents system [10] authenticates the sender network node of the mobile agent and does not include fine-grained authentication of mobile agents so that different capabilities can be granted to different authenticated agents coming from the same network node.

In this paper, we propose protection mechanisms, where mobile agents can define and grant different access

policies to different authenticated or un-authenticated agents in a transparent, dynamic, modular, portable and efficient way. In particular, we propose authentication mechanisms for mobile agents, based on digital signature, along with access control mechanisms based on capabilities. In our mechanisms capabilities are then granted dynamically to agents upon authentication by an authentication service. Mutually suspicious agents will then control the access to their resources based on authentication. To be able to grant capabilities while moving, agents should be able to move along with their capabilities. Agents should be able to export these capabilities and grant them to eligible users.

The generation of capabilities and their use by mobile agents, as well as well as the authentication of agents should be transparent to mobile agents' programmer. Indeed, we do not want to include security features inside the mobile agents' application code, thus easing programming. The programmer would then concentrate on the application's logic rather than the security issues.

In this paper we conduct an evaluation of these mechanisms through a concrete implementation within a mobile-agent system that we have implemented and a performance measurement. The protection mechanisms have been implemented on top of Java and integrated into a simple mobile agent-based distributed system that we have implemented and that we know the implementations details. This will allow us to test the feasibility of our mechanisms on one hand, and to measure the performance of the security functions used for protection in a basic mobile-agent platform on the other hand.

The rest of the paper is structured as follows. In section II we provide a background for this work. In section III, we present the authentication and access control problems for mobile agents. Requirements for secure mobile agents cooperation are discussed in section IV. In section V, we describe the system architecture supporting our authentication and access control mechanisms. After an evaluation of our mechanisms in section section VI, we conclude in section VII.

II. BACKGROUND

Due to the popular use of Internet applications, new classes of distributed applications have emerged, such as information search on the web and electronic commerce applications ([11], [12]). The use of the Internet as a platform to execute distributed applications introduced two major problems: the heterogeneity of machines and the lack of efficiency of the applications. First, we need to write programs that execute everywhere on the Internet. Second, applications slow performance becomes a major problem, as applications components are distributed over a large-scale network, and may run on a relatively slow distributed environment.

Mobile agent based programming [1] is an emerging paradigm for structuring distributed applications over the Internet. An agent is a process that can move with its code and execution context from site to site to perform its task. Mobile agents are mainly intended to address autonomy and efficiency problems of applications distributed over large scale and slow networks. They reduce communication costs by moving computation to or close to the host on which the target data reside [4]. Consequently mobile agents can move to meet and communicate with each others. In these scenarios, protection becomes an urgent need.

The research in the area of mobile agents programming is highly active. Several projects, such as Telescript [13], D'Agents [14], Aglets ([15], [16]), Mole [17], MOA [18], Voyager [19], JADE [20] have developed execution environments for mobile agents. In these platforms, the security has been only partially addressed. In particular a modular, transparent, dynamic and portable protection mechanisms were not handled by these systems.

Throughout this paper, we call a client agent the agent that initiates a communication with another agent, called the server agent. As an example of protection mechanisms of mobile-agents-based systems, if a mobile client agent A wishes to obtain a service from a mobile server agent B, then agent B should ensure that agent A is not malicious i.e. agent A is the one who is claiming to be so that access rights can be granted. The protection mechanisms here consist then of two major security services:

- Authentication. Authentication refers to the ability to associate an identity with each agent and to verify that this identity belongs to the one claiming to own this identity. The identity of an agent may be the identity of its creator (the program writer or the vendor of the software agents), or its users or its group of users (the agent's owner). The agent's dispatcher node is a network node which has ran the agent and dispatches it to its next hop of its itinerary. This dispatching node should be also made responsible for the agent's behavior on its receiving node. Once an agent is executed by an agent server, it could have been corrupted by a malicious agent server. An agent server executing an agent has full control over the agent's code and state. Many research works are being done in the domain of protecting a mobile agent from a malicious agent server ([21]–[26]). However, no satisfactory solution is found till the moment we wrote the paper. An agent must not be able to masquerade another agent by steeling its identity. Solutions have been introduced to detect tampering afterward.
- Access Control. Access Control is the definition of the policy that determines the rights given by an agent to other agents. For this purpose, an agent must be able to export object references while controlling the access rights it provides. The protection policy determines when and how these rights are exported. An incoming agent may be provided with an initial protection environment based on its authentication by the receiving host server. This protection environment can evolve dynamically according to the agent's execution.

In Telescript, an agent is authenticated using its authority. An authority is associated to an agent on its creation. It is an encrypted message that is associated to an agent, and can be the agent's signature which is built using the agent's owner information. In the public version of Telescript that we have tested, the encrypted message is generated automatically by the telescript engine. An agent has to deliver its authority and proves that it knows the private key which generated that signature based on RSA public key authentication. The protection mechanisms do not authenticate the agent's writer, neither its dispatching. Furthermore, there are few experiments with measurements and evaluations that have been conducted for these mechanisms. In Telescript, access control mechanisms are divided into 3 categories: securing agents' objects, context switching during communication, and the permit-based access control. Exporting a reference to an object means that all the public methods for that object can be called, including the method which can modify the object. By declaring an object reference as *protected*, the object can be accessed on read only. As an example, a mobile agent can move to get a list of flights for its owner to a specific destination. The server agent providing the list of flights grant read-only access to that list.

*getFlight: op() Dictionary[String, Flight, Equal] = if sponsor.name == *.name.authorityflights else flights.protect()@ ;*

In the above code, the server agent (called place by telescript) verifies whether the client agent has the same authority as the place, i.e. the client agent and the place belong to the same owner. In that case, a reference *flights* to the list of flights is granted, otherwise, a *protected* reference to the list of flights is granted, by calling the method *protect()* on the reference *flights*. Context switching allows a server agent to implement its own access control for file access for instance. It also allows to define the ownership of objects created during cooperation. By declaring a method as *sponsored*, the server agent will own all the created objects during cooperation. If the client agent move, the objects created during cooperation will not move with it. The third category of access control mechanisms protect the host resources, such as CPU and memory. It is also used to grant some agent the rights to execute system operations. Telescript, access control is managed by the agent's application code, which makes the programming of agents difficult. We argue that programmers would rather concentrate on the application logic rather than on the programming efforts of security services that ought to be provided by the system. There are no mechanisms to define different policies for read or write accesses on objects. In D'Agents [32], the authentication is based on PGP (Pretty Good Privacy) [19]. However, the digital signature is also used as follows. On creation, mobile agents should register to a running agent server. Before sending a registration request *agent_begin*, the agent encrypts that request with its owner private key, and then encrypts the result with the agent server public key. On reception of this registration message, the agent server can authenticate the agent's owner. On migration, the agent is signed by its dispatcher node. The authentication of the agent is then based on the authentication of its dispatcher node. This is limitative, as it is sufficient for an agent to go through by an unknown dispatcher node to be considered untrusted. Different agents applications may want to consider different access policies depending on the authentication of the agents' writer or owners as well. PGP is known for its performance drawback. In addition, D'Agents does not include indications about digital certificates and keys distribution and manipulation. Concerning access control, D'Agents does not allow sharing data and relies on message passing for communication between agents. On reception of a message by an agent, access control should be coded by the agent's application code. Each message passed between authenticated agents is signed and authenticated which introduces a major performance problem. Aglets provides basic authentication approach where agents are separated into 2 types: trusted and untrusted . For an agent server, trusted agents are those which are dispatched from agents servers within the same domain of the receiver. Agents servers are within the same domain if they share a secret key. In this approach, there must be a setup of the domain before any exchange between agent servers. In addition, each user of an agent server within a domain must obtain this key signed with the user password. This approach has scalability and usability problems, whereas each new connecting agent server over the Internet must communicate with a system administrator and obtain the secret key to join the domain and that each user must obtain its encrypted secret key. Concerning access control, agents can define in a policy file read/write access rights to local files and the rights to execute system operations. Aglets, like D'Agents does not allow objects sharing and relies on message passing for communication between agents. Furthermore, to our knowledge, very little work has been done to investigate the effectiveness of security components and their usability in a distributed environment. [27], as well as JADE [20] uses digital signature for authenticating users. In JADE, all agents are owned by an authenticated user. It uses a policy file to protect host resources from an agent by specifying permissions to host resources and system operations in the same way as Aglets do.

When designing our agent protection mechanisms, we consider the following requirements:

- Transparency in achieving protection via transparent implementation of security services. The mobile agents' authentication as well as the enforcement of access control must be transparent to mobile-agents' applications.
- Dynamic exchange and evolution of access rights. Access rights must be dynamically exchanged between cooperating agents. Rights must dynamically evolve during execution and cooperation.
- Modularity. The implementation of protection for mobile agents must be separate from the agents' ap-

plications code. Agents must not implement security services in their application code.
- Portability of the agents' applications. The implementation of the protection mechanisms must not use language-specific features. It must be ported to any mobile-agents platform.
- Non-repudiation by agents. The agents must not be able to deny an action afterward. The mechanisms must provide assurance of origin or delivery of objects exchanged between a sender and a receiver.
- Efficiency. The authentication and access control mechanisms for mobile agents must be devised in a way to minimize the protection overhead.

Existing mobile agents systems propose partial solutions to the above problems. To ensure communication among mobile agents while controlling access to agents resources, the system must include proper authentication, naming service, and access control mechanisms. As mobile-agents systems are interesting in large-scale distributed environments like the Internet, efficiency should be considered while designing these mechanisms.

The mechanisms we propose take into consideration the above requirements. These mechanisms provide agents in a transparent fashion the ability to cooperate and enforce protection.

III. MOBILE AGENTS AUTHENTICATION AND ACCESS CONTROL PROBLEMS

The following issues motivate our design choices for the authentication and access control mechanisms of mobile agents:

- Transparency. The authentication and access control mechanisms should be transparent to agents' applications. The applications should not include security-related instructions. Transparency enhances the usability issues of nowadays user-level complex security solutions.
- Usability. The security mechanisms should not be too much time-consuming or difficult to use. This decreases the productivity of the users who want to use protection and the protection mechanisms end up by being abandoned or used haphazardly introducing security holes to systems.
- Assurance. The authentication mechanisms should meet the non-repudiation security requirements. The sender and the receiver of the mobile agents should not be able to deny sending or receiving agents. Digital-signature-based authentication mechanisms offer non-repudiation services to both cooperating agents.
- Modularity. The authentication and access control mechanisms should be implemented in a separate layer from the agent's application code. Therefore, the programmer would concentrate on the application programming logic rather than on protection issues.
- Portability. The authentication and access control mechanisms should be general and applicable to all mobile agents systems. The mechanisms should not include features that are operating-system-dependent or language-dependent so that they can be ported to any mobile agent system.
- Efficiency. The authentication and protection mechanisms should be efficient. For instance, the mechanisms should not introduce significant overhead on the mobile agent messaging system.
- Mutual suspiciousness. The agents must be equal with regard to protection. The authentication and access control mechanisms should impose no hierarchy among cooperating agents.
- Autonomy. One of the important characteristics of mobile agents is autonomy. At the moment of writing the agent's application code, the programmer does not know which agent servers would make part of the agent's itinerary. Therefore, it impossible for the programmer to register access control policies in a centralized protection server. Each agent should be responsible to define its own access control policy which should move with the agent during its execution.

IV. REQUIREMENTS FOR SECURE MOBILE AGENTS COOPERATION

When designing our mechanisms for secure agent cooperation, the following issues have been considered.

- Authentication of local cooperating agents. Cooperating agents within the same site must be authenticated before communication takes place. A server agent granting access rights must have the assurance that the client agent requesting access is not malicious. We distinguish two possible scenarios.
  - Authentication of a client agent created by the local agent server. In this scenario, a client agent has been created locally within an agent server and wishes to communicate with another server agent within the same site.
  - Authentication of a visitor client agent. In this scenario, a client agent agent has not been created locally by the agent server. It is a visitor agent and wishes to communicate with a server agent within the same site. This visitor agent represents its owner, has been sent by a dispatching agent server and its code/behavior has been written by a third party programmer.

  To solve the first problem, two techniques have been used in the literature of mobile agents: the trust model technique, and the explicit request technique. In the trust model technique, every client agent created by the local agent server is considered trusted and therefore no authentication is taking place by the server. Aglets [16] uses this technique by considering agents created within the same domain as trusted and therefore an authentication is not needed. However, in Aglets an explicit setup of the domain is necessary by a system administrator for all users and agent servers in the same domain to share a

secret key. The main drawback is in the manual domain set up by a system administrator making the system difficult to administer. The authentication technique, used by D'Agents [28], consists of authenticating the client agent's owner during agent's registration with the local agent server. The local agent server then intercepts every message exchanged between the client agent and the server agent to verify that the message belongs to a registered communication channel by an authenticated agent. This algorithm has a performance drawback as every message has to be verified. In the explicit request technique, a server agent application has to explicitly requests the agent server whether the agent has been created locally. The main drawback of this algorithm, used by Telescript, is the lack of transparency to agents applications.

To solve the second problem, two techniques have been used in the literature of mobile agents: the trust model technique, and the explicit request technique. The trust model technique, used by Aglets, limits the agent itinerary to a trusted domain, and therefore no authentication is needed. The main drawback of this technique is the limitation imposed on agents itinerary for mobility. In the explicit technique request, agents applications should include a code which verifies whether visiting agents have been authenticated. D'Agents and Telescript use this technique. The main drawback of this technique is the lack of transparency to agents applications. As stated earlier, this is restrictive in a dynamic environment where communication with potential, a priori unknown agents, could take place.

- Authentication of remote cooperating agents. In the literature of mobile agents, the client agent is authenticated on every message exchanged between the client and the server. This algorithm has performance drawback as client is authenticated on every message passing.
- Specifications and mobility of access rights. When an agent moves, the access rights which are used for access control must move with the agent. This gives the agent the possibility to cooperate with other agents on the visiting sites by granting access rights. In the literature of mobile agents, access control lists are used to specify access rights. Two approaches are used to specify and use the lists: in the first approach, as in Aglets and D'Agents, access control list takes the form of a policy file on a destination. The policy file specifies access policies for mobile agents on the local host resources. As stated earlier, this approach is not appropriate for a dynamic environment like the mobile agents. We argue that an agent server cannot predict all the possible cooperations which would take place among cooperating visiting agents. In the second approach, as in Telescript, access control lists are coded in the agents applications. The main problem of this approach is that agents applications are forced to handle access control mechanisms at the application level.

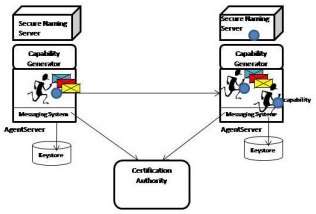


Fig. 1. Overall Architecture of a Secure Mobile Agent Platform

V. ARCHITECTURE

In this section, we present our proposed architecture to protect mobile agents in a mobile agent system.

A. Overall Architecture

A secure mobile agent platform includes the following components as shown in Figure 1:

- Agent. An agent is a mobile object which can move from site to site under its own control to achieve tasks on these sites. In general, in order to move to a site, an agent must explicitly invoke a *move(site)* message. An agent is composed of its code, its execution thread, and its data which correspond to the values of the agent global variables. When an agent moves from one site to another, it continues its execution on the destination site at the instruction which immediately follows the invocation of the *move* operation. To communicate with other agents, agents invoke a method which are translated into a message by the underlying messaging system.
- Agent Server. Each site, as part of the mobile agent platform, runs an execution environment, the *agent server*. This execution environment implements facilities for creating agents, executing them concurrently, suspending them, destroying them, etc.
- Messaging System. A messaging system is part of an agent execution environment. It provides facilities for agents to communicate both locally and remotely. It establishes communication links between communicating agents.
- Certification Authority (CA). It is a trusted third party which provides a pair of private/public keys and digital certificates for mobile agents owners, writers, and agents servers. All digital certificates are digitally signed by the CA for further verification of their authenticity and validity. Each agent server joining the system will obtain its pair of keys automatically by contacting the CA. We assume to have a centralized CA. The distribution of public keys among agent servers is not our focus in this work.
- Keystore. Each agent server has a keystore [15]. The keystore is used to store and manage private keys

along with their corresponding digital certificates. The agent server's private key and its corresponding digital certificate are created by the agent server's administrator by sending requests to a trusted certification authority (CA). A keystore is automatically created for each agent server in the system. The agent owner, writer, and agent server digital certificates are digitally signed by the CA.

- Secure Naming Server. A secure naming server is associated to each agent server. It exports an interface to mobile agents to communicate by using symbolic names. Server agents can export initial capabilities to client agents using the secure naming server. An initial capability is transparently obtained when the client agent consults a naming server which associates an agent human-friendly symbolic name to access capabilities which are to be granted to different users. A capability will then gives authorization for access to the server agent's objects through authorized methods calls. Before grating an initial capability, the naming server verifies whether the client agent has been authenticated. It enforces the access control policies defined by mobile agents. The naming server can be called locally or remotely. For a client agent calling the naming server locally, authentication is not required as the client agent has already been authenticated upon reception by the agent server. However, a remote call to the naming server requires authentication.
- Capability generator. Our access control mechanisms are based on capabilities. An agent specifies the capabilities to be granted to other agents using an Interface Definition Language (IDL) in a transparent way to the agent application code. A capability generator is associated to each agent server. It generates capabilities as specified in the IDL. The capabilities then move with the agent for granting access to other agents at runtime upon authentication.

B. Authentication Mechanisms

In mobile agents systems, agents must present proper identities to the agent server that receives them. The agent server will have then the assurance that the received agent is not malicious and can be trusted. When agents are authenticated, then different access rights can be granted to different agents by the servers and other agents exporting services within the agent server. Therefore, authentication is an essential step toward the protection of the agent server resources including other agents and servers running within the agent server. The authentication algorithm that we have implemented and integrated within our mobile-agents platform is based on digital signature ([29]–[31]). The digital signature is an electronic signature that is used to authenticate the identity of an object sender and to ensure that the object has not been modified since it has been signed. The digital signature involves the public key cryptography which relies on a pair of keys (a public and a private key) associated with an entity. The advantage of digital signatures is that they allow the non-repudiation which is one of our objectives.

The execution of the digital signature within the platform is transparent to the agent. A mobile agent has to be authenticated in the following cases: when received by an agent server after migration and when requesting cooperation with another agent via the secure naming server.

C. Authentication Algorithm on Mobility

MOVE($agent$)
1 $state \leftarrow$serialize(agent)
2 $\triangleright$ Get stored owner signature
3 $sig_o \leftarrow \mathrm{getSig_o}()$
4 $PrivateK_s \leftarrow$retrieve(keystore)
5 $sig_s \leftarrow \mathrm{sign(state, code, PrivateK_s)}$
6 $sig_w \leftarrow \mathrm{getSig_w}()$
7 $\mathrm{send(code, state, sig_o, sig_s, sig_w, cert_o, cert_s, cert_w)}$

When a mobile agent calls a method to move to a destination agent server, then the sender agent server, on which the agent is running, signs the agent digitally. The sender agent server produces a signature object which uses a hash algorithm and the private key of the sender agent server. The sender agent server retrieves the private key from the keystore. The agent (state and code) along with its signature by the agent server, its owner identity and signature, as well as the writer(vendor) identity and signature are sent to the destination site. The owner signature is produced only once at the home agent server as the private key will be no more available when the agent leaves its home. The writer signature is downloaded by the home agent server when downloading an agent from a third party agents writer. On reception of the signed agent, the receiver agent server (the visited host) verifies the identity the mobile agent platform which sends the agent. The receiver agent server then decides whether to accept or reject the reception of the agent based on the level of trust that the receiver has in the sender, owner, and writer.

RECAG($code, state, sig_o, sig_s, sig_w, cert_o, cert_s, cert_w$)

```
1  ▷ Retrieve Public Key of CA from keystore
2  PK_C A ←retrieve(keystore)
3  ▷ Retrieve Public Key of authorities from keystore
4  PK_o ←retrieve(keystore)
5  PK_s ←retrieve(keystore)
6  PK_w ←retrieve(keystore)
7  v_o ←isValidSign(cert_o, PK_C A, PK_o, sig_o)
8  v_s ←isValidSign(cert_s, PK_C A, PK_s, sig_s)
9  v_w ←isValidSign(cert_w, PK_C A, PK_w, sig_w)
10 ▷ If all authorities must be authenticated
11 if enableHostAll
12    then if v_o falseor v_s falseor v_w false
13            then sendAuthInvalid(client)
14    else ▷ Can run the agent
15         adClient(client, cert_o, v_o, cert_s, v_s, cert_w, v_w)
16 elseif
17 ▷ Owner authentication required?
18 hostOwnerAuth
19    then
20 if v_o false
21    then sendAuthInvalid(client)
22 elseif
23 hostSenderAuth
24    then
25 if v_s false
26    then sendAuthInvalid(client)
27 elseif
28 hostWAuth
29    then
30 if v_w false
31    then sendAuthInvalid(client)
32    else addClient(client, cert_o, v_o, cert_s, v_s, cert_w, v_w)
```

On reception of the message (see algorithm above) by the destination agent server, the receiver agent server retrieves the digital certificates from the message, and verifies whether the certificates are valid (expired or not) and issued by a trusted CA. The verification procedure needs the CA's public key which is retrieved from the receiver agent server's keystore. After successful verification of the digital certificates, the signatures are verified using the corresponding public keys which are retrieved from keystore. We assume that public keys have already been distributed among agent servers. The agent server keeps a record on the agent's authentication status, as well as the identities of its authorities (sender, owner and writer). The authentication mechanisms are based on our assumption that the level of trust that a receiver agent server attributes for an incoming agent depends on the level of trust that the receiver agent server had in the sender agent server, the agent owner, and the agent program writer, which can be configured by an administrator.

D. Authentication Algorithm on Cooperation Request

RREQ($destHost, destAg, sig_o, sig_s, sig_w, cert_o, cert_s, cert_w$)

```
1  ▷ Retrieve Public Key of CA
2  ▷ from keystore
3  PK_C A ←retrieve(keystore)
4  ▷ Retrieve Public Key
5  ▷ of authorities from keystore
6  PK_o ←retrieve(keystore)
7  PK_s ←retrieve(keystore)
8  PK_w ←retrieve(keystore)
9  v_o ←isValidSign(cert_o, PK_C A, PK_o, sig_o)
10 v_s ←isValidSign(cert_s, PK_C A, PK_s, sig_s)
11 v_w ←isValidSign(cert_w, PK_C A, PK_w, sig_w)
12 ▷ If all authorities must be authenticated
13 if enableHostAll and enableAgentAll
14    then if v_o falseor v_s falseor v_w false
15            then sendAuthInvalid(client)
16    else adClient(client, cert_o, v_o, cert_s, v_s, cert_w, v_w)
17         ▷ Return a capability
18         capa ← getCapa(destA, client)
19 elseif
20 ▷ Owner authentication required?
21 hostOwnerAuth or destAgOwnerAuth
22    then
23 if v_o false
24    then sendAuthInvalid(client)
25 elseif
26 hostSenderAuth or destAgSenderAuth
27    then
28 if v_s false
29    then sendAuthInvalid(client)
30 elseif
31 hostWAuth or destAgWAuth
32    then
33 if v_w false
34    then sendAuthInvalid(client)
35    else adClient(client, cert_o, v_o, cert_s, v_s, cert_w, v_w)
36 ▷ Return a capability
37 capa ← getCapa(destAg, client)
```

LOCALREQ($destAg$)

```
1 ▷ Already authenticated?
2 trusted ←getAuthClient(client)
3 ▷ If all the authority actors must be authenticated,
  ▷ then reject if any actor is not authenticated
4 if trusted isTRUE
5    then
6         ▷ Returns a capability
7         capability ←getCapa(destAg, client)
8    else then sendAuthInvalid(client)
9
```

As stated previously, mobile agents start cooperating by consulting a secure naming server. When the client agent consults the naming service for cooperating with a server agent, the naming server authenticates the client, verifies whether access rights have been exported for the client by the server agent, and establishes a secure communication

channel with the client agent. As stated previously the requirements for authentication differ according to the location of the client agent. We distinguish the following cases:

- Client agent is remote. On cooperation request (see above algorithm), the agent server on which the client is running sends to the destination naming server, the cooperation request along with the digital signature of the agent's authority which is the signatures of its (owner, sender agent server, writer). The agent's owner and writer digital signatures are part of the agent's state. As agent is mobile, the agent does not have the private keys, once left its home, and therefore the signatures must move with the agent. For the sake of the performance, the agent only keeps the signature of its home and its last dispatcher. Keeping the signatures of the itinerary nodes is beneficial for tracing purposes if needed. However, this will increase the agent's size with each migration.

  Upon authentication of the client, the secure naming server grants only initial capabilities to the client according to the server agent access control policy, over a secure communication channel. For the sake of performance, further communications can be done using these capabilities without further authentication if the communications pursue on the same secure encrypted channel. We argue that authenticating a client agent on every communication with a server agent is costly. Therefore, we opted for one-time authentication over a secure communication channel. However, if the channel is broken due the server agent mobility for instance, then an authentication is required, as capabilities may have been passed to another client agent or have been stolen, it is then necessary to verify authentication on the next establishment of a communication channel.
- Client agent is created locally. The behavior as in the local request algorithm (see algorithm above). If the client agent has been created locally by the agent server, then the agent is considered trusted. The reference to the agent is only known locally by the execution context of the agent server. This reference and other information concerning the owner and the writer of the agent are added to a table of authenticated agents by the local agent server.
- Client agent is a visitor. The behavior as in the local request algorithm (see algorithm above). If the client agent is a visitor agent, then the agent has already been authenticated upon its reception by the agent server. On cooperation request, the secure naming server responsible to authenticate agent lookups its table of authenticated agents to verify that the agent has been already authenticated.

E. Access Control Mechanisms

Our access control mechanisms are based on capabilities. Agents can control accesses to their own objects (resources) by use of capabilities. A capability is a token that identifies an object and contains access rights, i.e. the subset of the object's methods whose invocation is allowed. An agent is represented by a graph of objects, where a global unique name is generated for the root object and a relative name is generated for each internal object. In order to access an object, an agent must own a capability to that object with the required access rights (Figure 2). When an object is created, a capability is returned to the creator that usually contains all rights on the object. The capability can thus be used to access the object, but can also be copied and passed to another agent, providing it with access rights on that object. When a capability is copied, the rights associated with the copy can be restricted, in order to limit the rights given to the receiving agent. Agent move with their capabilities. Those capabilities must be protected against the man in the middle attacks. Capability protection is not our focus in this paper.

Fig. 2. Definition of a Capability

Capabilities are generated automatically by our system from the semantic definitions of access control policies by our security generator. When an agent migrates, it moves with its policy code in a transparent way agent's application. Each agent defines in a semantic way its own policy using an interface definition language (IDL) that we have defined (Figure 3). The purpose of the IDL is to separate the definition of protection from the agent's application code. The language defines the capabilities to be exchanged between cooperating agents. The semantic consists of the following terminology:

- *policy*. A *policy* is the definition of the set of methods to be accessed. It is the view that the agent is willing to export to other cooperating agents.
- *disallow*. It identifies unauthorized method call. A security violation message will be sent by the system informing the caller of security violation.
- *users*. It identifies the list of authorities that must be authenticated before capabilities are granted. As stated previously an agent runs under the authority of its owner, sender agent server, and writer principles. The authorities are *AND/ORed* according to authentication requirements of the agent. *ANDed* authorities must be all authenticated before an agent can get access.
- *grant*. It is used to grant different access rights to different agents if they have been authenticated as required.

As an example, consider a distributed agenda application using mobile agents. The agents organize a meeting between their corresponding owners (Figure 3). A mobile agent migrates from the agent server of its owner to the other owners' agent servers to decide on a meeting time. Each agent should be able to read the agenda maintained

```
Agenda_itf
{
   policy participant_policy
   {
      void disallow initAgenda ();
      readSlot_policy access (readPage_policy page);
   }
   policy administrator_policy
   {
      void initAgenda ();
      writeSlot_policy access (writePage_policy page);
   }
}
Meeting_itf
{
   policy readPage_policy
   {
      read();
      disallow write();
   }
}
Slot_itf
{
   policy readSlot_policy
   {
      read();
      disallow write();
   }
}
Users
{
  (owner=leila@uoe.ac.ae , and sender =cit2.uoeu.ac.ae ),
  (owner=richard@uoeu.ac.ae , and sender =cit3.uoeu.ac.ae, and writer =emirates),
  (writer=uoeu),
}
grant participant_policy
Users[(writer=france)]
grant participant_policy
```

Fig. 3. Agent Access Control Policy

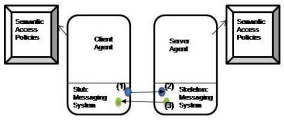


Fig. 4. Dynamic Exchange of Capabilities

by the other agent in order to determine a free common slot. An agent must not tamper with another agent agenda. An agent will then disallow other agents to call the *init()* method, which can be called only by the administrator agent. However, other agents can call the *access()* method to read the agenda.

F. Exchanging Capabilities

Agents communicate and exchange capabilities in a transparent way to the agents applications code through the messaging system (Figure 4). The messaging system is similar to the Java Remote Invocation (RMI) [32]. Each agent defines semantically its own access control policy using the IDL. This definition includes the agent's view of protection and the authorized users. Then, based on this definition, stub and skeleton classes are generated by a pre-processor (that we have implemented). Those classes include security modules which implement the defined policies. Instances of those stubs and skeletons classes are then inserted by the mobile-agents platform to protect the agent. The stubs and skeletons ensures that only authorized capabilities are exchanged among agents. The capabilities are constructed and exchanged dynamically through the security modules in a flexible way. When a client agent communicates with a server agent, the client stub sends a message along with the capability for access (1). Based on the received capability (2), the server agent skeleton grants access to the client and a reply message is sent. The server agent can send with the reply message a capability (3) which increments the client access rights for future communication.

VI. Evaluation

In order to evaluate the impact of our protection mechanisms on the performance of a mobile agent platform, we have developed the digital-based authentication mechanisms and the capability-based access control mechanisms and integrated them within a mobile agent platform that we have implemented and which we know the implementation details. This allows us to identify the Java mechanisms involved in the implementation of a secure mobile agent platform and to measure the cost of these mechanisms. Our system is composed of 2 distributed machines which run the secure mobile agent platform. We first measure the cost of the Java mechanisms involved in the implementation of security in our experimental environment. This gives us an idea about elementary costs for protection. Then, we compare the operation of the platform (agent migration) with security and without security. This gives us an information about the actual penalty on the performance of agents migrations.

In this section we describe the implementation of our secure mobile agent platform, and identify the main Java mechanisms involved in the implementation of the protection. We then present both a qualitative as well as a quantitative evaluation of the main components of the security mechanisms and the impact of those components on agent execution.

A. Experimental Environment

Our mobile agent platform is implemented in Java. The following Java characteristics are used in our experiment:

- Code mobility. Java allows classes to be dynamically loaded from remote nodes. Code portability is provided by interpretation of byte code.
- Object serialization and de-serialization. The java language provides an object serialization and de-serialization features [33], which allow instances to be exchanged between different execution environments. Serialization feature provides a means for translating Java objects into a stream of bytes which can be sent as a message over the network or written on a file disk. De-serialization is the reverse process, where a graph of objects can be reconstructed from the byte stream which includes the serialized graph.
- Polymorphism. Another important aspect of Java that we used is polymorphism. Polymorphism refers to the ability to define Interfaces (or types) and classes separately. An interface can therefore be implemented by several classes. It is possible to declare a variable whose type is an interface and which can reference objects from different classes that implement the same interface.
- Dynamic binding. Java also implements dynamic binding, which is crucial to mobile code. Dynamic binding means the ability to determine at run-time the code to be executed for a method invocation. Since Java allows classes to be dynamically loaded, a variable of an interface type can be assigned to a reference that points to an object, whose class was loaded dynamically. Java postpones the binding of the code (of this variable) until invocation time, thus allowing dynamically loaded classes to be executed.

- Digital signature. Java provides mechanisms [34] for digitally signing an object using a Digital Signature Algorithm (DSA) for digital signature [35], and using SHA-1 [**?**] for hashing. It provides mechanisms for signature verification. The java language provides the Sun Java Cryptography, an implementation for encryption and digital signature.
- Keys and Digital Certificates. The Java environment provides a feature for the management of keys and their associated X509 digital certificates [15]. In particular, it provides an interface to a keystore repository to manage private keys and their associated X.509 digital certificates.

B. Agent Authentication

The agent migration consists of the following steps.

a) Serialization of the agent state.
b) Retrieval of the sender agent server's private key and digital certificate, and the digital certificates of owner and writer from the local keystore.
c) Creation of an object signature to be used to sign the agent.
d) Initialization of the signature object with the server private key.
e) Updating the signature object using the agent's state for encoding.
f) Production of the signature of the mobile agent using the signature object from step d).
g) Construction of a message which includes the agent's state, code, signature and the (owner, sender, writer)'s digital certificates and sending of the above message to the destination agent server using a TCP/IP connection.
h) Destruction of the agent in the origin agent server.
i) Reception of the message in the destination agent server and creation of a new thread for the execution of the agent.
j) De-serialization of the agent's state.
k) Checking the digital certificates validity (valid or expired) of the (owner, sender, writer)'s digital signatures.
l) Retrieval of the (owner, sender, writer)'s public keys from a local keystore.
m) Verification of the (owner, sender, writer) 's digital certificate using the CA's public key from step k). This step verifies whether the (owner, sender, writer)'s certificates have been issued by a trusted CA.
n) Initialization of the signature object used to verify the agent's signatures.
o) Verification of the agent's signatures.
p) Run the agent if agent authorities are authenticated according to the receiving host security policy.

C. Hardware and Software Environment

Our hardware environment is described in Table I. The two servers *Agent Server1* and *Agent Server2* are connected by a local area network of 100Mbits/sec.

TABLE I
HARDWARE CONFIGURATION

	Server 1	*Server2*
Nb of Cpus	1 proc, 2 core	1 proc
Cpu Type	Intel T2500	Pentium M
Cpu Mhz	2000	1800
Memory	1GB	512MB
OS	WindowsXP	WindowsXP

We used the JDK 1.5.0_09 for all the experiments described in this paper. The security mechanisms implemented within our mobile-agents platform use the Sun Java Cryptography mechanisms [27].

D. Experiments

We measure the average cost of elementary functions used for protection. We program a minimal agent which iterates a pingpong between *Agent Server1* and *Agent Server2* 100 times.

In an experiment, we measure the average total cost of a one-way trip for an agent. We increase the agent size for each experiment. The agent sizes are 1.8Kbytes (a minimal agent), 100Kbytes, 500Kbytes, 1000Kbytes, 1500Kbytes and 2000Kbytes respectively. The experiments are repeated in two environments: with protection and without protection.

E. Protection Cost

The cost of the elementary protection mechanisms that are used in our environment is presented in the Table II. This gives us the basic elementary cost of these mechanisms in our environment under experiment during platform operation. Mainly these components are relative to authentication services. For the implementation of access control, we used the fact that Java object references are capabilities and that to invoke an object, the invoker needs that capability. These elementary components correspond to the following security operations as described in the authentication process of the agent (VI-B)

- Private Key and agent server digital certificate retrieval. This phase corresponds to step b) in the description of the agent authentication.
- Agent signature initialization. This phase corresponds to step c), d) and e) in the description of the agent authentication.
- Agent signature production. This phase corresponds to step f) in the in the description of the agent authentication.

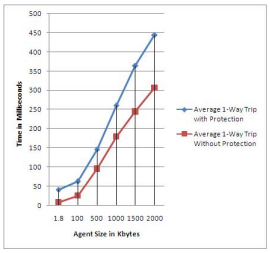


Fig. 5. Comparison between Agent Migration with Protection and without Protection

- Digital Certificate validation. This phase corresponds to step k) in the description of the agent authentication.
- Public keys retrieval. This phase corresponds to step l) in the description of the agent authentication.
- Digital Certificate verification. This phase corresponds to step m) in the description of the agent authentication.
- Signature verification initialization. This phase corresponds to step n) in the description of the agent authentication.
- Signature verification. This phase corresponds to step o) in the description of the agent authentication.

TABLE II
SECURITY ELEMENTARY COSTS OF OUR EXPERIMENTAL ENVIRONMENT

Protection Component	μs	μs
Private Key, Certificates retrieval	3.42	4.35
Agent signature init	0.11	0.11
Agent signature	7.70	7.82
Certificates validation	0.33	0.36
Public keys retrieval	1.77	2.04
Digital Certificates verification	0.03	0.03
Signatures verification init	0.69	1.13
Signatures verification	40.8	47.79

The measurements in Table II represent the execution time of the Java methods dealing with digital signature manipulation and keystore access. We observe that the signature verification is the most expensive operation. The verification algorithm, as implemented by the Java Sun security provider, uses the signature and the public key of the signer to verify that the signature has been produced by a corresponding private key.

As shown in Figure 5, the protection overhead is important in case of migration of a minimal agent. We observed an overhead of 412% on a minimal agent migration performance when protection is used. The cost of a minimal agent migration did not compensate the protection overhead which is mainly due to security function used for protection. As the agent size increases, the overhead decreases to 45%. It is obvious that secure mobile agent platform is interesting for mobile agents deployed in large-scaled environments. The protection overhead would compensate the agents migration and local cooperation can take place securely.

VII. CONCLUSION AND PERSPECTIVES

Many researchers have investigated the development of protection mechanisms in a mobile agent platform. However, the protection mechanisms provided focused on protecting host resources from malicious agents. In these works suspicious cooperating agents have to include protection mechanisms at the application level, which makes the programming of agents a difficult task. In this paper, we proposed the use of semantics for agents to define protection policies using an Interface Definition Language (IDL) that we have defined. Consequently, the agent application code and the definition of protection policies can be done separately, thus enhancing modularity, and easing programming task. The authentication of agents and the enforcement of their access control policies are done in a transparent way to the agents applications by our messaging system.

An agent has to be signed every time it is sent, and at a request for cooperation. By this way the verification of the integrity of the agent and the non-repudiation are met. Therefore, it is easier to identify a malicious agent authority that owns, sends or fabricates a malicious agent. We introduced a secure naming service which ensures authentication for initiating cooperation among agents. Our mechanisms are transparent to the agents' application codes. Available mobile agents applications can use our secure platform without any change to their code. The experiments showed the feasibility and usability of the implemented protection mechanisms.

The performance evaluation that we have conducted shows the impact of the protection on the operation of the platform. The significance of this impact depends largely on the underlying implementation and the performance of the encryption/decryption algorithms by the security service providers. We measured the cost of the elementary methods used for protection in our implementation.

The work presented in this paper continues. In particular, we are extending our platform to implement and evaluate a dynamic digital certificates and public key distribution among agent servers, and its impact on the performance and the usability of the system.

VIII. ACKNOWLEDGMENT

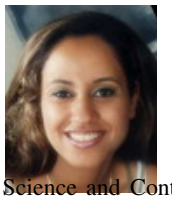
We would like to thank the anonymous reviewers for their detailed and valuable comments.

REFERENCES

[1] D. Chess, C. Harrison, and A. Kershenbaum. "Mobile Agents: Are They a Good Idea?". IBM Research Division, T. J. Waston Research Center, Yorktown Heights, New York, March 1995, URL: http://www.cs.dartmouth.edu/?agent/papers/chapter.ps.Z

[2] International Business Machines Corporation: "Mobile Agent Facility Specification". OMG TC Document cf/96-08-01, August 1996. URL: http://www.trl.ibm.co.jp/aglets/maf.ps

[3] OMG. "Mobile Agent Facility Specification". January, 2000. http://www.omg.org/docs/formal/
[4] Leila Ismail and Daniel Hagimont. "A Performance Evaluation of the Mobile Agents Paradigm". In Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications OOPSLA'99. ISBN = "1-58113-238-7, 0-201-48561-3 (Addison Wesley Longman)", pp. 306-313
[5] J. Gosling and H. McGilton. "The Java Language Environment: a White Paper". Sun Microsystems Inc., 1996. URL: http://java.sun.com/docs/white/langenv
[6] Scott Oaks. "Java Security". O'Reilly
[7] Christian Jensen and Leila Ismail. "Capability Based Protection for Hosting Mobile Code". In ERSADS'97, March 17-21, 1997.
[8] Daniel Hagimont and Leila Ismail. "A Protection Scheme for Mobile Agents on Java". In the 3rd ACM/IEEE International Conference on Mobile Computing and Networking, pp: 215-222, Budapest, Hungary, September 1997
[9] Volker Roth and Mehrdad Jalali-Sohi. "Access Control and Key Management for Mobile Agents". Elevier Preprint, November 2001
[10] Leila Ismail. 'Evaluation of Authentication Mechanisms for Mobile Agents on top of Java". icis, pp. 663-668, 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007), 2007
[11] Ozgur Koray Sahingoz and Nadia Erdogan. "A Two-Leveled Mobile Agent System for E-commerce with Constraint-Based Filtering". Lecture Notes in Computer Science. Springer Berlin/Heidelberg. Vol.3036/2004, ISBN: 978-3-540-22114-2, 2004
[12] C. Badica, M. Ganzha, and M. Paprzycki. Mobile agents in a multi-agent e-commerce system. Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2005. SYNASC, ISBN: 0-7695-2453-2, pp: 8, 2005
[13] J. E. White. "Telescript Technology: The Foundation for the Electronic Market Place". General Magic Inc., Mountain View, CA
[14] Robert S. Gray, George Cybenko, David Kotz, and Daniela Rus. "Agent Tcl. Itinerant Agents: Explanations and Examples with CDROM". William Cockayne and Michael Zypa (editors), Manning Publishing and Prentice Hall, 1997
[15] D. Lange and O. Mitsuru. "Programming and Deploying Java Mobile Agents with Aglets. Addison-Wesley Pub Co, ISBM: 0-201-32582-9, August 1998
[16] Mitsuru Oshima, Guenter Karjoth, and Kouichi Ono. Aglets Specifications 1.1 Draft 0.65. September 1998. http://www.trl.ibm.com/aglets/spec11.htm
[17] J. Baiumann, F. Hohl, K. Rothermel, and M. Straer. "Mole, Concepts of a MobileAgents System". World Wide Web, vol. 1, No. 3, pp. 123-137, 1998.
[18] D. S. Milojici, W. LaForge, and D. Chauhan. "Mobile Objects and Agents, Design, Implementation and Lessons Learned". Distributed systems Engineering IEEE, pp. 1-14, 1998.
[19] G. Glass. "Object Space Voyager - The Agent ORB for Java". Lecture Notes in Computer Science, no.1368, pp. 38-55, 1998.
[20] JADE Board. "JADE Security Guide". February, 2004. http://jade.tilab.com/
[21] G. Vigna. "Protecting Mobile Agents through Tracing". Proceedings of the Third Workshop on Mobile Object Systems. Finland, June, 1997
[22] Tomas Sander and Christian F. Tschudin. "Protecting Mobile Agents Against Malicious Hosts". Springer Lecture Notes in Computer Science 1419. Springer-Verlag, 1998
[23] Tomas Sander and Christian F. Tschudin. "Towards Mobile Cryptography". Proceedings of the IEEE Symposium on Security and Privacy. 1998
[24] G. Karjoth, N. Asokan, and C. Glc. "Protecting the computation results of free-roaming agents", Second International Workshop on Mobile Agents (MA'98), Lecture Notes in Computer Science 1477, pages 195-207. Springer-Verlag, 1998
[25] C. Cachin, J. Camenisch, J. Kilian, and J. Mller. One-Round Secure Computation and Secure Autonomous Mobile Agents. In Proc. 27th Colloquium on Automata, Languages and Programming (ICALP), Geneva, Lecture Notes in Computer Science, Springer-Verlag, 2000
[26] John Zachary. "Protecting Mobile Code in the Wild". IEEE Computer Society. March/April 2003 (Vol. 7, No. 2) pp 78 - 82
[27] A. Puliafito and O. Tomarchio. "Design and development of a practical security model for a mobile agent system". Proceedings of the Seventh International Symposium on Computers and Communications (ISCC02), vol. 02, pp. 1530-1346.
[28] Robert S. Gray, David Kotz, George Cybenko, and Daniela Rus. "D'Agents: Security in a multiple-language, mobile-agent system". In Mobile Agents and Security, Giovanni Vigna (Editor), Lecture Notes in Computer Science, Springer Verlag, pp. 154-187, 1998
[29] Matt Bishop. "Computer Security Art and Science". Pearson Education, Inc. ISBN 0-201-44099-7
[30] W. Stalling. "Cryptography and Network Security". Third Edition. Upper Saddle River, NJ: Prentice Hall. 2003
[31] Charles P. Pfleeger and Shari Lawrence Pfleeger. "Security in Computing". ISBN-10: 0-13-035548-8; ISBN-13: 978-0-13-035548-5. Prentice Hall. Published: 02 December 2002. Upper Saddle River, NJ: Prentice Hall
[32] A. Wollrath, R. Riggs, and J. Waldo. "A Distributed Object Model for the Java System". Computing Systems, vol. 9, no. 4, pp. 291-312, 1996.
[33] R. Riggs, J. Waldo, A. Wolrath, and K. Bharat. "Pickling State in the Java System". Computing Systems, 9(4), pp. 313-329, Fall 1996
[34] Pankaj Kumar. "J2EE Security for Servlets, EJBs, and Web Services. ISBN-10: 0-13-140264-1; ISBN-13: 978-0-13-140264-5; Prentice Hall. Published: 04 September, 2003
[35] Federal Information Processing Standards (FIPS). "Announcing the Standard for Digital Signature Standard (DSS)". Publication 186, 19 May 1994
[36] Sun Microsystems. "JDK 5.0 Documentation". Sun Microsystems, 2004. URL: http://java.sun.com/j2se/1.5.0/docs



Leila Ismail got her Ph.D. in Computer Science from the National Polytechnic Institute of Grenoble (INPG), France, in September 2000 with very honorable degree. During her Ph.D., she conducted her research work in distributed systems and security at the French National Institute for Research in Computer Science and Control (INRIA). Before joining the United Arab Emirates University, UAE, in 2005-2006, as an Assistant Professor in Computer Science, she was working for Sun Microsystems Research and Development Center. She was also a visiting professor in the American University of Beirut, Lebanon, during 2004-2005. She participated to the deposit of a US patent in the domain of networking while working for Sun Microsystems. Her current research interests include security in mobile-agents systems, Grid computing, high performance computing, autonomous computing, and parallel processing programming models. She is currently leading the Grid and High Performance Computing laboratory at the United Arab Emirates University. She won the IBM Shared University Award in 2007 and the IBM Faculty Award in 2008. She is actively participating in international journals and conferences programs committees.