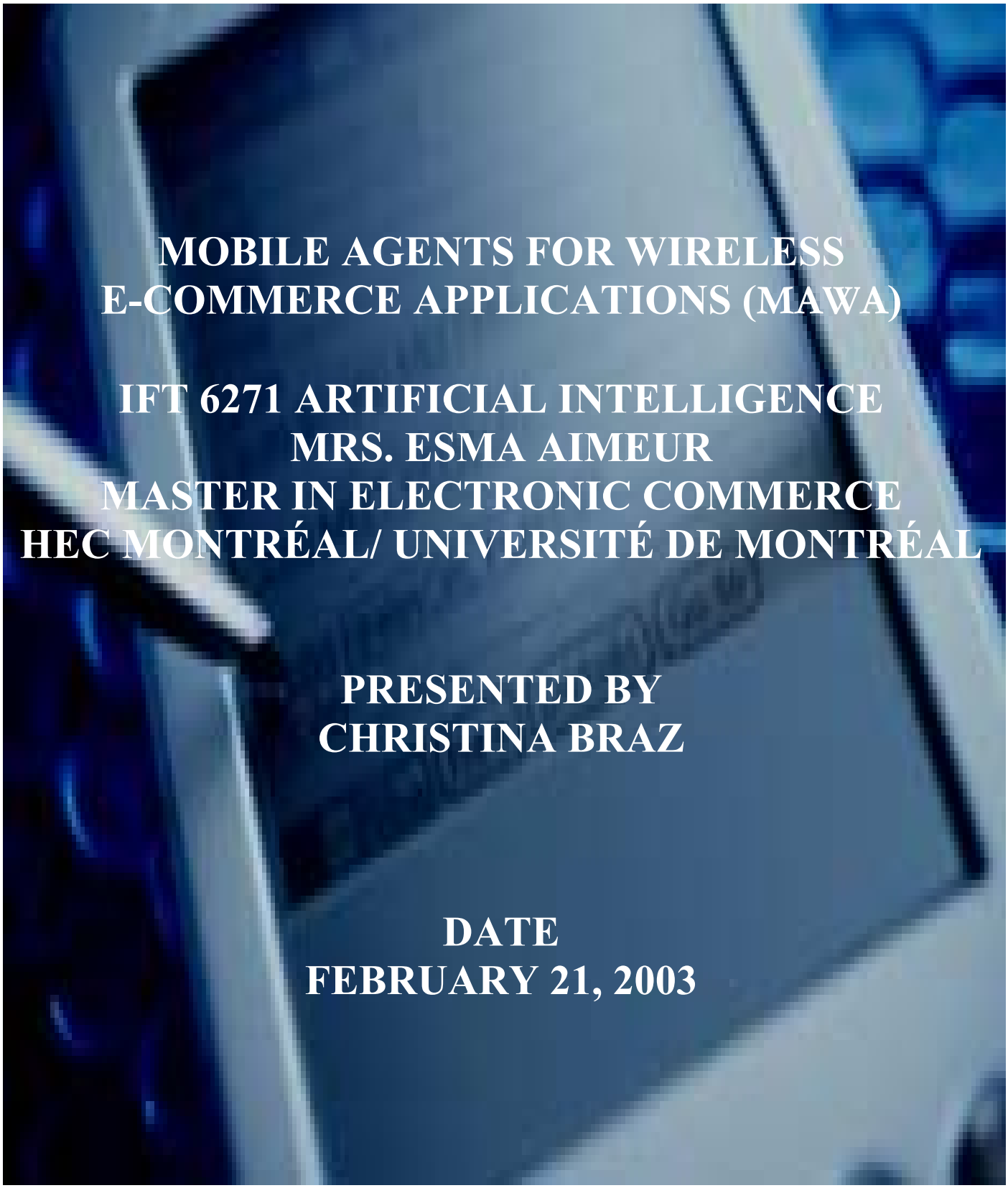


MOBILE AGENTS FOR WIRELESS E-COMMERCE APPLICATIONS (MAWA)

IFT 6271 ARTIFICIAL INTELLIGENCE
MRS. ESMA AIMEUR
MASTER IN ELECTRONIC COMMERCE
HEC MONTRÉAL/ UNIVERSITÉ DE MONTRÉAL

PRESENTED BY
CHRISTINA BRAZ

DATE
FEBRUARY 21, 2003

INDEX

MOBILE AGENTS FOR WIRELESS E-COMMERCE APPLICATIONS

Christina Braz
Master of Sciences in Electronic Commerce / Université de Montréal/HEC Montréal
Montreal, Quebec Canada - christina.braz@hec.ca
http://www27.brinkster.com/chbraz/e-profile/ecomMaster.htm

ABSTRACT

The focus of this synthesis is how to deal with the complexity of almost one billion people using their cell phones, PDAs, and other wireless computer devices at the same time, communicating with each other and interacting with other devices such as vending machines, POS terminals, messaging systems, their cars, entertainment, GPSs[1] (Global Positioning System), networked systems, central corporate data and applications, and the Internet through the Mobile Agent Technology, or specifically through the Mobile Agents for Wireless e-Commerce applications.

Mobile commerce is a system that allows people to convey transactions anywhere, anytime, typically refers to use of mobile phones and other portable devices (Personal Digital Assistant, etc.) to conduct a variety of transactions. Mobile agents (MAs) are software components that are able to move in a network. They are often considered as an attractive technology in electronic commerce applications, although security concerns remain. Mobile agents are an effective choice for many applications, for several reasons, including improvements in latency and bandwidth of client-server applications and reducing vulnerability to network disconnection. Security has been also identified as a top criterion for the acceptance of mobile agent technology.

In this synthesis, we will present three different architectures of MAs such as the “EMAA: An Extendable Mobile Agent Architecture”, the “MAW: Mobile Agents for a Wireless Environment” and the “eCapsule Technology” at which will be focusing on this synthesis. eCapsule Technology is a new breed of Mobile Data Agent technology to simplify the complex wireless mobile world.

[1] GPS: is a worldwide radio-navigation system formed from a constellation of 24 satellites and their ground stations. These stations, also known as the "Control Segment", monitor the GPS satellites, checking both their operational health and their exact position in space. The master ground station transmits corrections for the satellite's ephemeris constants and clock offsets back to the satellites themselves. The satellites can then incorporate these updates in the signals they send to GPS receivers.

1. INTRODUCTION

Mobile agents (MAs), will be a critical part of the Internet in a short terme, because mobile code makes new applications possible, it leads to considerably better performance than traditional techniques, and because it provides a general framework in which distributed, information-oriented applications can be implemented efficiently and easily, with the programming concern proliferate evenly across information, middleware, and client providers. In other words, mobile code gives providers the time and flexibility to provide their users with more useful applications, each with more valuable features, especially in the mEcommerce applications.

Lately, Web sites and other Internet services will not be able to efficiently provide the full range of customization desired by their clients, and clients will want to use the same information and organizing tools across many sites. Furthermore, fixed location, application-specific proxies will become bottlenecks, and as user needs change, may no longer be at the best network location for accessing the proxied services. As a result, customization tools will be specified as software, in the form of mobile code that runs either on the server, or on a dynamically selected proxy site near the server or client, as adequate. Mobile code is necessary, rather than client-side code, since many customization features (such as information monitoring) do not work if the client is disconnected, has a low-bandwidth connection, or requires frequent communication with the server. Mobile code is valuable, since servers and proxy sites need provide only a generic execution environment (along with an API that provides programmatic access to their service). The actual customization tools can be written by the services themselves, by third party middleware developers, and even by the end users.

Many clients will wish to send mobile code to various information sites as part of a single task. Although there will be applications for which the mobile code can be sent in parallel, many tasks require a sequence of subtasks, each at a different site. Therefore, the mobile code must be able to *jump* sequentially through multiple sites, and such *multi-jump* mobile code is a Mobile Agent (MA). Mobile code, and in particular MAs, will be a fundamental for allowing such access. MAs are programs that can migrate from host to host in a network, at times and to places of their own choosing. The state of the running program is saved, transported to the new host, and restored, allowing the program to continue where it left off. MAs systems differ from process migration systems in that the agents move when they choose, typically through a *jump & go* statement, whereas in a process-migration system the system decides when and where to move the

running process (normally to balance CPU load). Mobile agents differ from applets, which are programs downloaded as the result of a user action, then executed from beginning to end on one host.

According to Strategic Research Corporation [29], the business impact of Mobile Agents is enormous through activating applications to operate in a wireless world and thus their market impact is measured by four components, mobile agents & middleware, applications activated by agents, hardware, and services. The Mobile Agent enabled application software market is forecasted to reach $13B in 2006 at which point the total hardware, software, services market will be $66B. What is the contribution of Mobile Agents? They enable mCommerce, many forms of mobile communications services, enterprise applications, and move lots of data (traffic).

Impact of agent technology on mobile e-commerce applications

The significant growth of the demand for user mobility increases the needs to convey a transaction such as electronic shopping, payment, banking, gambling, ticketing, etc. anytime and anywhere. This leads to an effort to integrate existing electronic payment systems into the mobile environment. Mobile e-Commerce, also referred to as mCommerce, is a subset of e-commerce and deals with the ecommerce issues in the mobile environment. Mobile e-commerce delivers significant opportunities to those working in the banking, transport, retail, and communication industries.

The mobile environment constitutes many wireless technologies, such as UMTS (Universal Mobile Telecommunications System), GSM (Global Standard for Mobile Communications), wireless LAN (Local Area Network), Bluetooth, etc. Every technology has its own peculiarities that may impact the realisation of mobile e-commerce. However from the consumer's point of view, mCommerce should meet the following requirements (Hartmann & Keller)[17]:

- Simplicity (response times comparable to traditional electronic transactions)
- Security
- Convenience (Having in mind that mobile communication is still expensive, the bandwidth is limited and bit errors occur frequently, it is necessary to investigate dedicated solutions for mobile

eCommerce applications. Hence, the main requirements for these applications coming from the network and service provider perspective are:

- Data integrity
- Secure transmission
- Limited amount of data
- Scalability
- Rapid, easy and economic creation, testing and introduction of services

All prerequisites mentioned above have to be taken into account when building an agent-based mCommerce solution as well. However, MAs are well appropriated to well performing e-commerce applications and it likes that they are able to offer financial services to mobile users in a more flexible way. Particularly, they conduct to decreased costs in communications due to their ability to asynchronously and autonomously without a network connection.

2. AGENTS: THE STATE OF THE ART

2.1. Definition

According to Franklin & Graesser[16], they try to summarize the several definitions of an autonomous agents, with: "An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future".

Wooldridge & Jennings [10] have a more software-oriented approach, which has been accepted. "An agent is a hardware or software-based computer system that enjoys the following properties:

Autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
Social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;

Reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;
Pro-activeness: agents do not simply act in response to their environment; they are able to exhibit goal-directed behaviour by taking the initiative."

According to Brahim Chaib-draa 2003[4], an agent is an entity that perceives its environment, acts in an autonomous way, interacts in order to exchange objectives and constraints, anticipates and reacts in a flexible way, learns from its own experiences and finally, adapts itself in its environment.

As we might well notice by all those definitions an agent is a pretty complex portion of software that must exhibit some extraordinary characteristics in order to be called an agent. Actually, most agents have these properties, but most do not have all those characteristics at once. Rather than, they focus on one or a few.

2.2. Agent Classifications

The various definitions mentioned above involve a host of properties of an agent. These properties may help us further classify agents in meaningful ways. The table (Franklin & Graesser) [16] that follows lists several of the properties enumerated above:

Property	*Other Names*	Meaning
Reactive	Sensing and acting	*Responds in a timely fashion to changes in the environment autonomous exercises control over its own actions*
Goal-oriented	Pro-active purposeful	*Pro-active purposeful: does not simply act in response to the environment*
Temporally continuous		*Is a continuously running process*
Communicative	Socially able	*Communicates with other agents, perhaps including people*
Learning	Adaptive	*Changes its behaviour based on its previous experience*
Mobile		*Able to transport itself from one machine to another*
Flexible		*Actions are not scripted*
Character		*Believable "personality" and emotional state*

Every agent by this definition satisfies the first four properties. Complementing other properties produces valuable classes of agents, for example, mobile, learning agents, etc. Hence, a hierarchical classification occurs naturally. Mobile, learning agents are then a subclass of mobile agents.

There are other possible classifying schemes. For example, we might classify software agents according to the tasks they perform, for example, information gathering agents or email filtering agents. Or, we might classify them according to their control architecture. Agents may also be classified by the range and sensitivity of their senses, or by the range and effectiveness of their actions, or by how much internal state they possess.

Brustoloni's taxonomy of software agents [15] begins with a three-way classification into regulation agents, planning agents, or adaptive agents. A regulation agent, named with regulation of temperature by a thermostat, reacts to each sensory input as it comes in, and always knows what to do. It neither plans nor learns. Planning agents plan and learn, either in the usual AI (Artificial Intelligence) sense, or using the case-based paradigm, or using operations research based methods, or using various randomizing algorithms (randomizing agent).

Still, there are a lot of more possibilities to classify agents, for example, the environment in which the agent finds itself, for example software agents in contrary of artificial life agents and so on.

2.3. Agency Concept

Software agents, like people, may be most valuable when they work with other software agents in performing a task. A collection of software agents that communicate and cooperate with each other is called an agency. System designers using agents must take in account the capabilities of each individual agent and how multiple agents can work together. The agent-based approach allows the system designer to implement the system using multiple agents, with each agent specialized for a particular task. For example, a mCommerce application might have buyer agents, seller agents, stocking agents, database agents, email agents, etc. All of these agents need to communicate with each other and have the capability of working together to accomplish a common set of goals.

2.4. The Importance of Agents

Agents supply a new way of managing complexity because they provide a new way of describing a complex system or process. Using agents, it is easy to define a system in terms of processes arbitrated by agents.

Let consider, for example, the system design concerns involved in building a loan approval application[12] that attaches together branch banks, the main bank, loan underwriting companies, and credit reporting companies, and automates, in fact, much of the loan approval process. Building this kind of system using current technology is a complex and difficult task because the system decomposition forces the developer to deal with relatively low-level concepts (for example: loan applications, account balances, credit ratings) when defining the total system architecture. Besides, significant design time must be devoted to defining the communications protocol and interfaces that will allow the bank to exchange data with the credit reporting agencies and loan underwriters. In an agent-oriented system design, the system solution might include a customer service agent, a loan application analysis agent, an underwriter agent, etc. The focus is posted on the behaviour of each of these agents and communication between agents. The problem is done easier because the level of abstraction is higher and the programming problem becomes one of specifying agent behaviour.

An agent-based solution is valuable and attractive because the several agents used in the solution know how to do many things. For example, agents know how to communicate with other agents. The system developer no longer has to design communication protocols and message formats. The agent provides this capability as part of the basic agent mechanism. Agents have the intrinsic capability to build models of their environment, monitor the state of that environment, reason and make decisions based on that state. All the software developer needs to do is simply specify what the agents are to do in any given situation.

2.5. Agents Applications

Software agents are adequate for use in a large variety of applications. They can make it easier to build many types of complex systems. Software agents are adequate for use in implementing certain applications and in other problem sectors, other technologies will be more adequate. The developer must accurately analyze system specifications to define if agents are an appropriate implementation mechanism.

Agents are appropriated for use in applications that involve distributed computation or communication between components. Agent technology is appropriated for use in applications that reason about the messages or objects received over a network. This explains why agent-based approaches are common in applications that utilize the Internet. Multi-agent systems are also appropriated for applications that require distributed, concurrent processing capabilities.

Since agents keep a description of their own processing state and the state of the world around them, they are ideally appropriated to automation applications. Autonomous agents are capable of operating without user input or intervention. These agents may be used in applications such as plant and process automation, workflow management, robotics, etc.

Agents are not confined to use in applications where the individual agents communicate with each other over a LAN (Local Area Network) or the Internet. In some applications it makes sense to use multiple agents executing on one machine and communicating with each other using some type of intermediary process communication (RMI[Remote Method Invocation][2]). For example, a nested factory controller might consist of a user interface agent, a database interface agent, a machine tool interface agent, and a process monitoring and control agent. All of these agents could execute concurrently on the same processor or could be easily distributed across multiple processors.

Agents are well-suited to applications that require communications between components, sensing or monitoring of the environment, or autonomous operation. Since agents have the ability to reason, they may easily perform sequences of complex operations based on messages they receive, their own internal beliefs, and their total goals and objectives. They are ideally suited for a large variety of applications. They are particularly well-suited to:

- Process and Workflow Automation
- Electronic Commerce
- Distributed Problem Solving
- Internet Applications

[2] RMI: Part of the Java programming language library which enables a Java program running on one computer to access the objects and methods of another Java program running on a different computer.

3. MOBILE AGENTS

3.1. DEFINITION

Mobility is the degree to which the agents themselves travel through the network. Not all agents are mobile. An agent may just sit here and communicate with its surrounding by traditional means, such as various forms of remote procedure calling and messaging. We call agents that do not or cannot move stationary agents (Lange [20]).

Stationary Agent: A stationary agent executes only on the system where it begins execution. If it needs information that is not on that system or needs to interact with an agent on a different system, it normally uses a communication mechanism such as RPC (Remote Procedure Call) [3].

In contrast, a mobile agent is not delimited to the system where it begins execution. The mobile agent is free to travel among the hosts in the network. Created in on execution environment, it can transport its state and code with it to another execution environment in the network, where it resumes execution.

By the term *state*, we mean the attribute values of the agent that help it define what to do when it resumes execution at its destination. By the term *code*, we mean, in an object-oriented context, the class code necessary for the agent to execute.

Mobile agents: constitute a new approach to the architecture and implementation of distributed systems. Mobile Agents are programs that can be dispatched from one computer and transported to a remote computer for execution. Arriving at the remote computer, they present their credentials and obtain access to local services and data. The remote computer may also serve as a broker by bringing together agents with similar interests and compatible goals, thus providing a meeting place at which agents can interact. Mobile Agents are lightweight, secure, dynamic, and mobile executable content. Agents are small software objects that can

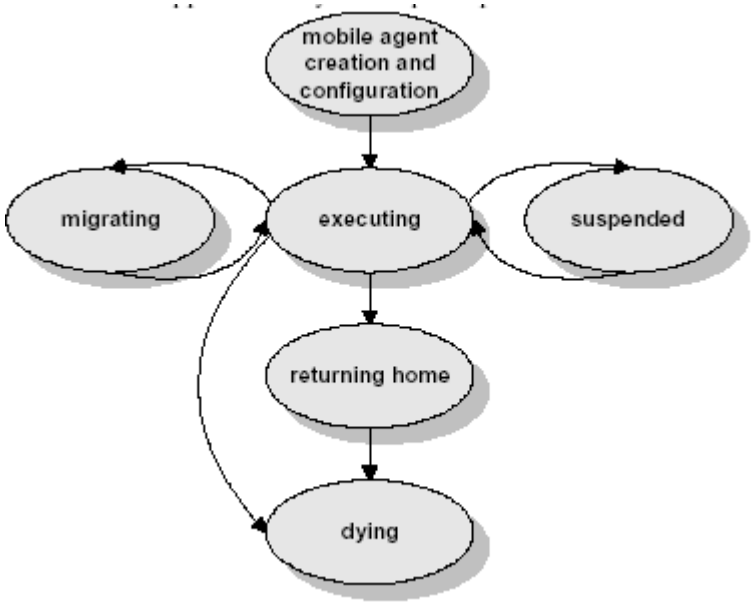
[3] Remote Procedure Call: a call to a routine that results in code being executed on a different system from the one where the request originated. An RPC system allows calling procedures and called procedures to execute on different systems without the programmer needing to explicitly code for this.

move from one host to another, along with state and security information. That is, an agent that executes on one host can suddenly halt execution, dispatch itself to a different remote host and resume execution there.

When the agent moves, it takes along its program code as well as its data. Hence, the word data is part of the description. Mobile Agents can trigger events, securely conduct and establish profiles and instructions, operate systems, function as digital money, collect and transport data, operate autonomously, and all that that suits this profile.

The Mobile Agent Lifecycle Model: As shown in Figure 1, mobile agents bind to a lifecycle which is identical to the property of temporal invariance. MAs are able to migrate, i.e. to take both their code and status along and transfer themselves to another host and continue execution there, thus fulfilling the mobility property.

Figure 1: The Mobile Agent Lifecycle Model



Source: S. Lipperts & B. Kreller. [http://dsp.jpl.nasa.gov/members/payman/swarm/lipperts99-isas.pdf]

Mobile agents can be written in a number of programming languages such as Tcl[4], Perl[5], Java and others; the agent environment supplies runtime support through dedicated libraries that add relocation and communication capabilities to the respective language.

[4] Tcl: Tool Command Language. A general-purpose scripting language. The Tcl interpreter is implemented as a C library that is easy to add to existing applications.
[5] A scripting language popular for writing CGI scripts.

3.2. THE IMPORTANCE OF THE MOBILE AGENTS

We could ask why are MAs important? Wireless networking is looking extensively for drivers and applications beyond today's communications and messaging applications (Strategic Research Corporation)[39]. The iMode[6] and WAP (Wireless Application Protocol) initiatives only address Internet access. They won't do it by themselves. One application that will stimulate change is Mobile commerce, mCommerce, which is bringing about changes of foundation in the enterprise, linking three key areas:

- Distributed and remote mobile connected employees allowing them access to the information and services needed to perform their jobs

- Distributed and remote offices, services, customers, and suppliers who provides vital information as well as use information many of whom are mobile as well as Internet connected

- Mobile users and buyers of products, services, and online corporate and information resources

MAs also enable a concept called dynamic roaming, which is what location-independence means to a distributed workforce, millions of users, active connected devices, and applications.

Dynamic Roaming embodies the principal of anytime-anywhere access to services, communication, and interaction with location independence and mobility. MAs provide the underlying services infrastructure to support dynamic roaming. Similarly, MAs provide the intelligent infrastructure for a revolution in scalability for the wireless device community and enable its convergence with wired network services.

Instead of laboriously centralizing network intelligence, as in traditional enterprise architecture, MAs may easily facilitate a functionally equivalent system at a much lower operational cost and with minimal impact to the naturally distributed network. A small, lightweight active agent is required to glue this community into a functional neural network, a virtual network, and community of connected devices, users, and applications. Another important point is, the timing of the introduction of MAs is good. We have now "left the Information Age and entered the Communication Age," (Paul Saffo[38], futurist and director of the Institute

[6] iMode is basically compact HTML (C-HTML) with a few mobile telephony additions, running over a HTTP/TCP/IP protocol stack.

for the Future. He describes the Communication Age as being characterized by the widespread, democratized use of information in the form of personal media where people can truly exploit the participative nature of the Internet at whatever location they choose. Users are not bound to physical location and their ability to interact with each other, with their environment, and with business is unparalleled. In the Communication Age, wireless devices will interact with the external environment rather than being limited to the IS department. Whereas the metaphor of communication used to be telephones—that is, people talking with people—the metaphor of the future will be a device by which machines talk to other machines on behalf of people. This development will free up people to deal with media, while machines take care of the dirty work of dealing with information.

How do they work? MAs technology runs as a very lightweight protocol and platform agnostic[7] service. It layers onto any computer, RTOS[8]-based intelligent device or mobile operating system utilizing a lightweight OS such as a Virtual Machine (Java Virtual Machine). The one challenge Agents have for adoption is that every active device has to mount the base service. Once activated, devices and agents can operate with and on each other and participate in virtual network communities. Mobile Agents (MAs) make it much easier to design, implement, and maintain distributed systems. Their genesis emerged from stationary software agents (lightweight programs) that had autonomous function. The development of software agent technology has progressed to a state where agents can negotiate with each other based on rules and policies, what seems like artificial

3.3. MOBILE AGENTS BENEFITS FOR THE CREATION OF DISTRIBUTED SYSTEMS

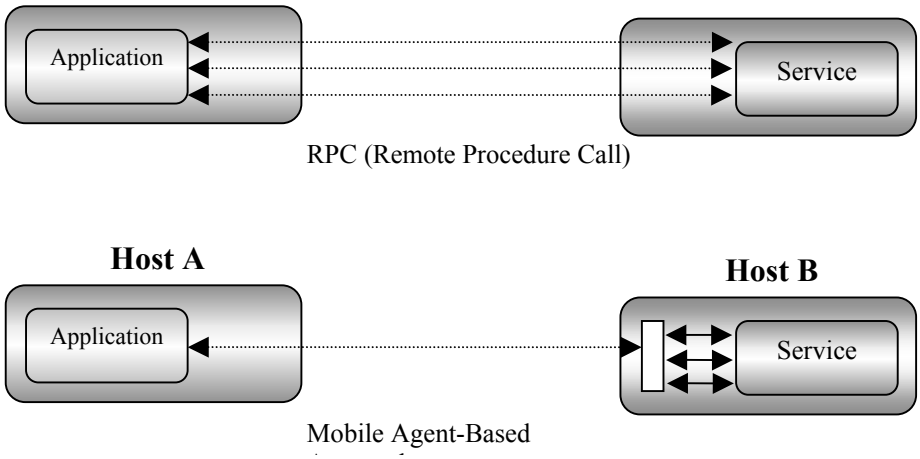
- Mobile Agents reduce the network load

Distributed systems often rely on communications protocols that involve multiple interactions to accomplish a given task (D.Lange)[20]. This is especially true when security measures are enabled. The result is a lot of

[7] A software program that will run on any computer operating system, such as Acrobat Reader.
[8] Real-Time Operating System: Operating system software designed for use in a realtime computer system, generally an embedded application.

network traffic. MAs allow you to package a conversation and dispatch it to a destination host, where the interactions can take place locally (see Figure 2).

Figure 2: Mobile Agents and Network Load Reduction



Source: D. Lange & M. Oshima [6].

MAs are also useful when it comes to reducing the flow of raw data in the network. When very large volumes of data are stored at remote hosts, these data should be processed in the locality of the data rather than transferred over the network. In one word: move the computations to the data rather than the data to the computations.

- Mobile agents overcome network latency

Critical real time systems, such as robots in manufacturing processes, need to respond in real time to changes in their environments. Controlling such systems through a factory network of a substantial size involves significant latencies. For critical real-time systems, such latencies are not acceptable. MAs offer a solution, because they can be dispatched from a central controller to act locally and directly execute the controller's directions.

- Mobile agents encapsulate protocols

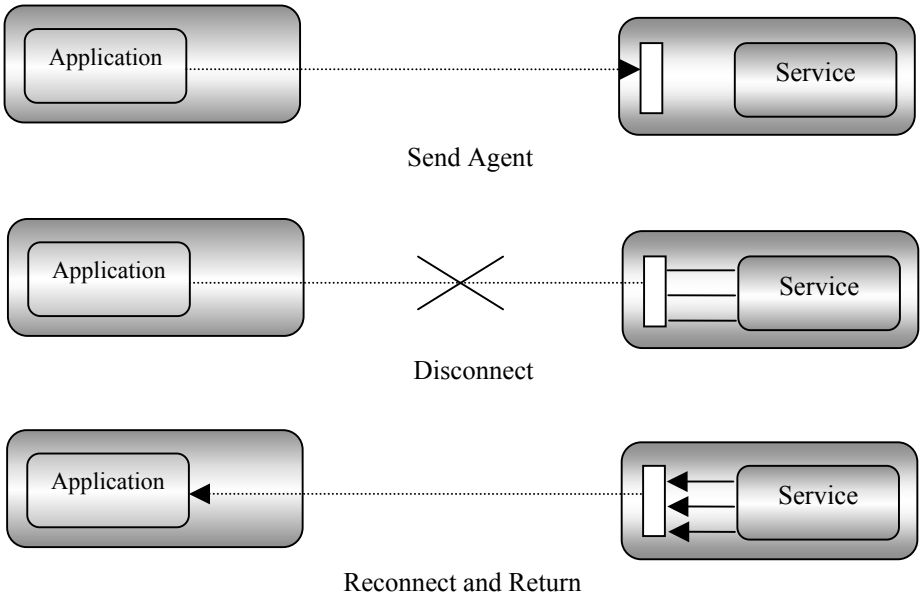
When data are exchanged in a distributed system, each host owns the code that implements the protocols needed to properly code outgoing data and interpret incoming data, respectively. However, as protocols

evolve to accommodate new requirements for efficiency or security, it is a bulky task to upgrade protocol code properly. As a result, protocols often become a legacy problem. MAs, on the other hand, can move to remote hosts to establish "channels" based on proprietary protocols.

- Mobile agents execute asynchronously and autonomously

As we might well know, mobile devices such as cellular phones, PDAs (Personal Digital Assistant, etc.) often must rely on expensive or fragile network connections. Tasks that require a continuously open connection between a mobile device and a fixed network probably will not be economically or technically feasible. To solve this problem, tasks can be embedded into MAs, which then be dispatched into the network. After being dispatched, the MAs become independent of the creating process and can operate asynchronously and autonomously (see Figure 3). The mobile device can reconnect at a later time to collect the agent.

Figure 3: Mobile Agents Allow Disconnected Operation



Source: D. Lange & M. Oshima [6].

- Mobile agents adapt dynamically

MAs have the ability to sense their execution environment and react autonomously to changes. Multiple mobile changes possess the unique ability to distribute themselves among the hosts in the network so as to maintain the optimal configuration for solving a particular problem.

- Mobile agents are naturally heterogeneous

Network computing is fundamentally heterogeneous, often from both hardware and a software perspective. Because MAs are generally computer-and-transport-layer-independent and are dependent only on their execution environment, they provide optimal conditions for seamless system integration.

- Mobile Agents are robust and fault-tolerant

The ability of mobile agents to react dynamically to unfavourable situations and events makes it easier to build robust and fault-tolerant distributed systems. If a host is being shut down, all agents executing on that machine will be warned and given time to dispatch and continue their operation on another host in the network.

3.4. ELEMENTS OF A MOBILE AGENT SYSTEM

3.4.1. Agent and Place

There are two fundamentals concepts in the MA model are the `agent` and its execution environment, the `place`.

- Agent

A MA is an entity that has five attributes:

State: needed for the agent to resume computation after travelling. When an agent travels, it transports its state with it. It must do so in order to resume execution at the destination host, a characteristic of all mobile agents.

Implementation: needed for location-independent agent execution. As any other computer program, a MA needs code in order to execute. When it travels, it has the option of either taking its implementation code or going to the destination, seeing what code is already there, and retrieving any missing code over the network (code-on-demand).

Interface: needed for agent communication. An agent provides an interface that allows other agents and systems to interact with it. This interface may be anything from a set of method signatures that allow other agents and applications to access methods on the agent to a messaging interface that allows agents to communicate such as the Knowledge Query and Manipulation Language (KQML). The infrastructure we describe is not oriented toward any specific agent language but rather describes a set of fundamental communication mechanisms that allows point-to-point communication between agents.

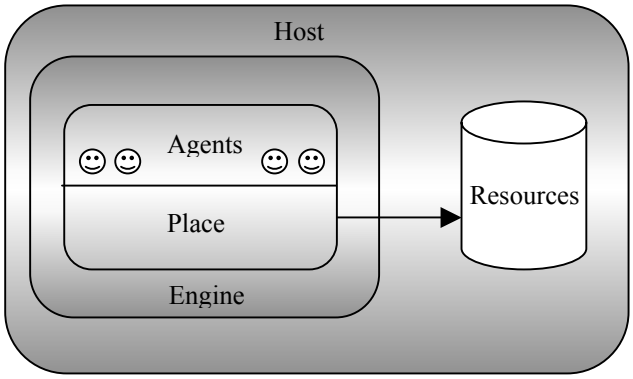
Identifier: needed to recognize and locate travelling agents. Each agent has an identifier that is unique during its lifetime (immutable). Agents demand identities so that may be identified and located via, for example, directory services.

Principals: needed to define legal and moral responsibility. A principal is an entity whose identity may be authenticated by nay system that the principal may try to access. A principal might be an individual, an organization, or a corporation. We find, for agents, at least two main principals: *Manufacturer* (the author, for example, the provider of the agent implementation) and *Owner* (the principal that has the legal and moral responsibility for the agent's behaviour, for example the creator of a mobile agent).

- Place

Now, we describe the environment in which agents operate. We might regard the place as the operating system for the agent as we might well see in the Figure 4. Four concepts play a relevant role in places:

Figure 4: Place and Engine.



Source: D. Lange & M. Oshima [6].

Engine: virtual machine for one or more places. In fact, places cannot themselves execute agents. In order to do that, agents must live inside the engine. The term engine here means that the engine serves as the virtual machine for places and their agents.

Resources: The engine and the place provide controlled access to local resources and services such as networks, databases, processors and memory, disks, and other hardware as well as software services.

Location: is a significant concept for MAs. The location of an executing agent as the combination of the name of the place in which it executes and the network address of the engine in which that place lives. This location will normally be written as an Internet Protocol (IP) address and a port of the engine with a place name attribute.

- *Principals:* As an agent, a place has two principals. A place is related to authorities that identify the person or organization for which the place acts (place master) as well as the manufacturer of the place. The manufacturer is the author (provider) of the place implementation, and the place master is the principal that has the responsibility for the operation of the place.

3.4.2. Agent Behaviour: Creation and Disposal

An agent gets created in a place. The creation may be initiated either by another agent living in the same place or by another agent or nonagent system outside the place. The class definition needed to instantiate the agent might be present on the local host or a remote host or, might be provided by the creator. Actually, creation involves three steps:

- Instantiation and identifier assignment.
- Initialization
- Autonomous execution.

An agent ends its life in getting disposed of in a place. The disposal may be initiated by the agent itself, by another agent living in the same place, or by another agent or nonagent system outside the place. An agent may be also disposed of by the system for one of the following reasons:

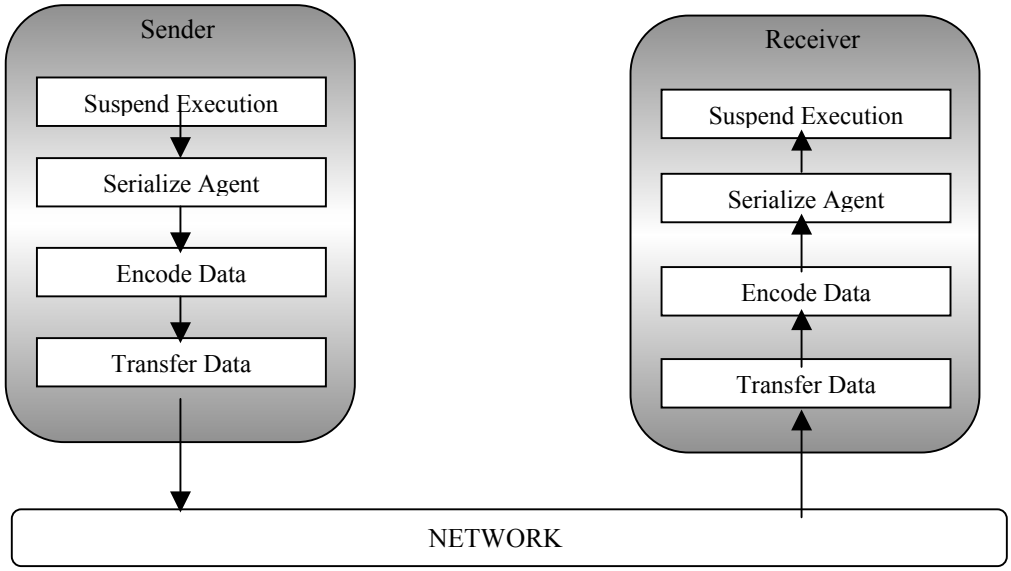
- End of lifetime: the lifetime of the gent has expired.
- No use: no one invokes to or use the agent.
- Security violation: the agent has violated the security rules.
- Shutdown: the system is shutting down.

3.4.3. Agent Behaviour: Transfer

The transfer process may be initiated by the agent itself, by another agent living in the same place, or by another agent or nonagent system outside the place. The agent is then dispatched from its current place

(origin) and received by the specified place (destination) as we might well see below in the Figure 5:

Figure 5: Agent Transfer.



Source: D. Lange & M. Oshima [6].

- Dispatching an Agent

When a MA is preparing for a trip, it must be able to identify its destination. Once the location of the destination is established, the MA informs the local agent system that it wants to transfer itself to the destination agent system. When the agent system receives the agent's trip request, it should do the following:

Suspend the agent.
Serialize the agent.
Encode the serialized agent.
Transfer the agent.

- Receiving an Agent

Only after the sender has authenticated itself to the receiving engine will the data transfer take place:

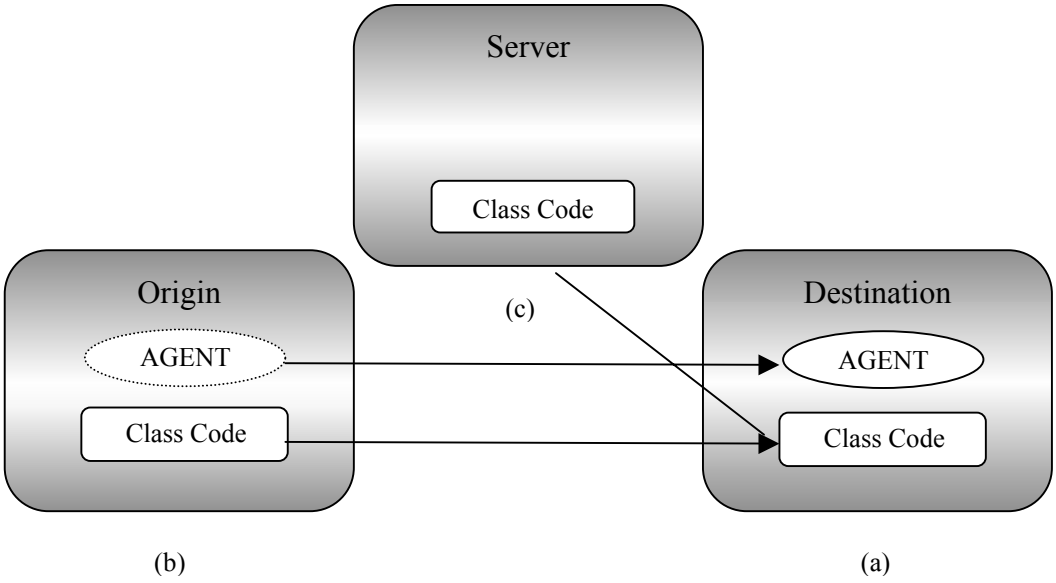
Receive the agent: when the destination engine accepts to the transfer, the encoded agents is received.
Decode the agent: the engine decodes the incoming data stream.
Deserialize the agent: the persistent representation of the agent is deserialized. The agent class is instantiated, and the transferred agent state is restored.

Resume agent execution: the re-created agent is warned of its arrival at the destination place. It may now prepare to resume its execution and its given a new thread of execution.

- Agent Class Transfer

The agent may not resume execution in the destination engine without its class being present. There are numerous ways to make the class available for the destination engine, depending on the location of the class: Class at destination (see Figure 6a); Class at origin (see Figure 6b); Code-on-demand (see Figure 6c).

Figure 6: Agent Class Transfer: (a) Class Destination, (b) Class at Origin, (c) Class at Server.



Source: D. Lange & M. Oshima [6].

3.4.4. Communications

Agents may communicate with other agents living within the same place (intraplace) or with agents living in other places (interplace and interengine). Actually, an agent may invoke a method of another agent or send it a message if it is authorized to do so. In general, agent messaging might be peer-to-peer or broadcast (one-to-many messaging scheme allowing a single agent to send a message to a group of agents).

3.5. MOBILE AGENT STRUCTURE

In SeMoA – Secure Mobile Agents (Roth & Jalali-Sohi, 1998)[37], mobile agents are transported as Java Archives (JAR files). The JAR specification of Sun Microsystems extends ZIP archives with support for digital signatures by means of adding appropriate signature files to the contents of the ZIP archive. The signature format is PKCS#7[9], a cryptographic message syntax standard. Using PKCS#7 as well extends the JAR format with support for selective encryption of JAR contents with multiple recipients. Encryption and decryption is handled transparently for agents by the filters. We implemented filters that handle agent signing and authentication as well as selective encryption of agent contents. Filters are applied transparently such that agents need not be aware of the security services provided by the server. In order to prevent encrypted parts of an agent from being copied and used in conjunction with other agents (cut & paste attacks[10]), these filters implement a non-interactive proof of knowledge of the required decryption keys.

---

[9] Cryptographic Message Syntax Standard : It describes general syntax for data that may have cryptography applied to it, such as digital signatures and digital envelopes. Available at URL: http://www.rsasecurity.com/rsalabs/pkcs/pkcs-7

[10] A cut-and-paste attack is an assault on the integrity of a security system in which the attacker substitutes a section of ciphertext (encrypted text) with a different section that looks like (but is not the same as) the one removed. The substituted section appears to decrypt normally, along with the authentic sections, but results in plaintext (unencrypted text) that serves a particular purpose for the attacker. Essentially, the attacker cuts one or more sections from the ciphertext and reassembles these sections so that the decrypted data will result in coherent but invalid information. Malicious hosts and other attackers that get a copy of the agent JAR may launch a *cut & paste attack*. The following example illustrates the attack:

1. Alice prepares a search agent. The agent collects stock quotes from Bob's server and the server of Mallet, but Alice does not want Mallet to know which quotes her agent collected from Bob. So she creates an access group G with Bob as its sole recipient and assigns the folder secret to this group. The agent is programmed to store the quotes in that folder if it is at Bob's server.
2. Alice sends her agent to Bob. Bob decrypts and installs the folder secret because he is a legal recipient. The agent collects the stock quotes and sets its next hop to the server of Mallet. Bob re-encrypts the folder and sends the agent to Mallet.
3. Mallet copies the INSTALL..MF, $name_G$.P7 and $name_V$ .EAR files from the agent to an agent of its own and sends it to Bob.
4. Bob decrypts and installs the folder secret in Mallet's agent because he is a valid recipient. The agent then copies the plain text data to another folder and sets its next hop to Mallet. Bob re-encrypts folder secret and sends the agent back to Mallet.
5. Mallet reads the plain text returned by his agent.

Each agent carries two digital signatures. The entity that signs the static part of an agent (the part that remains unchanged throughout the agent's lifetime) is taken as the rightful owner of that agent (the entity on whose behalf the agent is acting). Each sending server also signs the complete agent (static part plus mutable part); therefore it binds the new state of the agent to its static part. In other words, agent servers commit to the state changes that occurred to an agent while they hosted this agent.

Agents can use abstractions comparable to the combination of a Map interface and a file system in order to access and store data in their static and mutable part (denoted its structure).

Agent structures can be backed both by persistent and non-persistent storage, depending on the server's configuration. The agent structure also contains properties of the agent (key/value pairs), such as a human nickname of the agent and code sources to load classes from. The properties must be signed by the agent's owner, thus they are protected against tampering.

In addition, SeMoA computes implicit names from agents, by applying the SHA1 digest algorithm to its owner's signature. This renders agent names globally unique as well as anonymous. Implicit names are used in SeMoA to provide agent tracing, and will be used for scalable location-independent routing of messages among agents as well.

In summary, SeMoA supports four types of access rights for the folders of an agent:

Read-only: This data can be read on each host but cannot be modified without breaking the agent's integrity.
Read/write committed: This data can be read and modified on each host but hosts have to commit to the new state. The changes can be checked and linked to that host on the agent's next hop.
Group read: This data can be read only on a predetermined set of authorized hosts. Modification of the data breaks the agent's integrity.
Group read/write: This data can be read and modified only on a predetermined set of authorised hosts.

Groups of valid recipients can be defined flexibly. The data a mobile agent gathers on one host can be protected against eavesdropping by hosts not belonging to the access group of the folder in which the data is stored. Since the protection mechanisms are part of the server's security services, agents can remain unaware of the cryptographic operations and key management. The structure of an agent's JAR file is represented in the Figure 11:

Figure 11: The structure of an agent's JAR file.

META-INF/	MANIFEST.MF	
	OWNER.SF	
	OWNER.(DSA|RSA)	
	SENDER.SF	
	SENDER.(DSA|RSA)	
SEAL-INF/	INSTALL.MF	
	$name_i$.EAR	$i = 1, \ldots, n$
	$name_j$.P7	$j = 1, \ldots, m$
static/	agent.properties	
mutable/	instance.ser	

Source: (Roth & Jalali-Sohi, 1998)[37].

The file agent.properties consists of name/value pairs. The properties as well as the initial classes brought by an agent are covered by the owner's signature, thus any modification of the properties breaks the static part's integrity. The file instance.ser contains the serialized object instance graph of the agent. The information in SEAL-INF is used to manage selective encryption of agent contents.

3.6. MOBILITY PATTERNS FOR MOBILE AGENTS

Mobility in MAs (Mobile Agents) can be designated by the set of destinations that an MA visits, and the order in which it visits them. Therefore we identify the following parameters (Jha & Iyer)[19], to characterize the mobility of an MA:

Itinerary : The set of sites that an MA needs to visit. This could either be static (fixed at the time of MA initialization), or dynamic (determined by the MA logic).

Order: The order in which an MA visits the sites in its itinerary. This may also be either static or dynamic. Based on these parameters, we distinguish MA applications in e/m-commerce as possessing one of the following mobility patterns:

Static Itinerary (SI)

The itinerary of the MA is fixed at the time of initialization and does not change during execution of the application. We further distinguish such applications into:

- Static Itinerary Static Order (SISO): The order in which an MA visits the sites in its itinerary is static and fixed at the time of initialization. An example application is that of an auction MA, which may be required to visit a set of auction sites in a pre-specified order.
- Static Itinerary Dynamic Order (SIDO): The order in which an MA visits the sites in its itinerary is decided dynamically by the MA. An example application is that of a shopping MA which may visit a set of online shops, in arbitrary order, to find the minimum price for a product.

Dynamic Itinerary (DI)

The itinerary of the MA is determined dynamically by the MA itself. Obviously, at least the first site in the itinerary needs to be fixed at the time of initialization. An example application is that of a shopping MA that is required to find a particular product. A shop that does not have the product may recommend an alternative shop, which is included in the MA's itinerary dynamically. It may be noted that dynamic itinerary always implies dynamic order.

3.7. CONTEMPORARY MOBILE AGENT SYSTEMS

There are several mobile agent systems available, and the field is developing so dynamically and so fast that any effort to draft agent systems will be outdated rapidly. However, we will mention in this section a few interesting Java-based mobile agent systems.

3.7.1. Aglets

This system, created by Danny B. Lange & Mitsuru Oshima, imitates the applet in Java. The goal was to bring mobility to the applet. The term *aglet* is a combination of *agent* and *applet*. The Java aglet extends the model of network-mobile code made famous by Java applets. Like an applet, the class files for an aglet can migrate across a network. But unlike applets, when an aglet migrates it also carries its state. An applet is code that can move across a network from a server to a client. An aglet is a running Java program (code and state) that can move from one host to another on a network. In addition, because an aglet carries its state wherever it goes, it can travel sequentially to many destinations on a network, including eventually returning back to its original host.

3.7.2. Odyssey

General Magic Inc. invented the mobile agent and created Telescript, the first commercial mobile agent system. Based on a proprietary language and network architecture, Telescript had a short life. In response to the popularity of the Internet and later the steamroller success of Java language, General Magic decided to reimplement the mobile agent paradigm in its Java-based Odyssey. This system effectively implements the Telescript in the shape of Java classes. The result is a Java class library that enables developers to create their own mobile agent applications.

3.7.3. Concordia

Mitsubishi's Concordia is a framework for the development and management of mobile agent applications that extend to any system supporting Java. Concordia consists of multiple components, all written in Java, which are combined to provide a complete environment for distributed applications. In fact, a Concordia system is made up of a standard Java VM, a server and a set of agents.

3.7.4. Voyager

ObjectSpace's Voyager is a platform for agent-enhanced distributed computing in Java. While Voyager provides an extensive set of object message capabilities, it also allows objects to move as agents in the network. We might well say that Voyager combines the properties of a Java-based object request broker with those of a mobile agent system. In this way Voyager allows Java programmers to create network applications using both traditional and agent-enhanced distributed programming techniques.

3.7.5. Jumping Beans

Jumping Beans is a framework developed by Aramira Corporation with which a developer may create jumping applications. A jumping application is just like an ordinary application, except that while it is running it can pick itself up and physically move to another computer on the network, and then resume execution.

Note that the Java-based mobile agent systems have a lot in common. In addition to the programming language, all of them rely on standard versions of the Java virtual machine and Java's object serialization mechanism. A common server-based architecture permeates all the systems. However, agent transport mechanisms and the support for interaction (messaging) vary considerably.

Although the majority of the contemporary mobile agent systems are based on the Java language system, we shall also find other languages in use. The most significant languages are Tcl [36], Scheme[22] and Python[26].

3.8. MOBILE AGENTS ARCHITECTURES

First of all, it would be very useful to know what the MAs architectures are. A computational architecture specifies the organization of a mobile agent's behavioural modules, learning and evolution modules, as well as related competence components. It determines the working manner of the mobile agent, including the interrelationships and the communications among various components.

The domain of mobile agents for wireless applications is quite new in developments, products, services and platforms are yet in the early stages. Hence, we would like to present an overview of three formats of architectures that could well be used in the market today: EMAA - An Extendable Mobile Agent Architecture, A Mobile Agent Architecture for a Wireless Environment and Ecapsule Mobile Agent Technology Architecture. Which one, or ones, shall we choose? Clearly, it depends on our application objectives and needs. In the section Methodology and Tools we will present the Ecapsule Mobile Agent Technology Architecture in details.

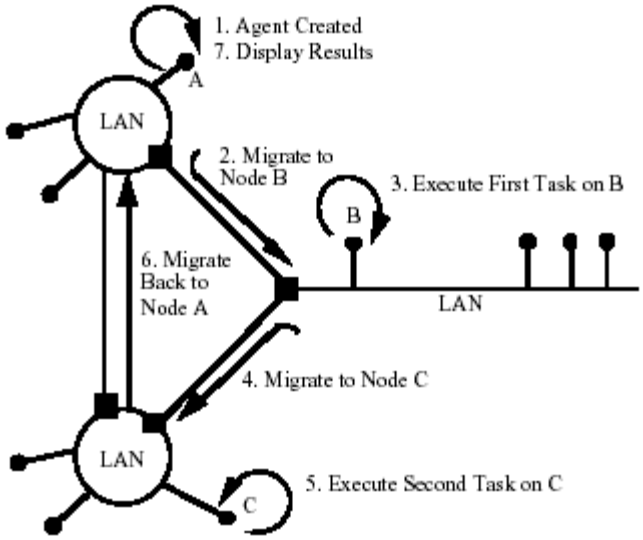
3.8.1. EMAA: An Extendable Mobile Agent Architecture

The Extendable Mobile Agent Architecture - EMAA (Lentini, Rao, Thies & Kay 1998)[25] is a mobile agent architecture specification that aids in the development of an agent system. It provides a simple way for a mobile agent to migrate from one computing node to another and to use the resources at that node. EMAA does not impose any restriction on the behaviour of the agents, thus allowing autonomous behaviour. Our purpose is to explain this mobile agent architecture, and the extendable nature of its components. One of the major benefits of a distributed computing environment is the ability to break problems up into smaller sub-problems and solve those sub-problems in parallel. EMAA exploits this characteristic by breaking an agent's high-level goal into tasks that can be executed in parallel. Another characteristic of distributed computing environments is that resources may be distributed among different nodes in the network. The concept of breaking a program up into sub-programs fits nicely with this structure because a task that needs to exploit a resource can be packaged into a separate, independent component. The resources available at a particular node are determined by the application requirements. In some situations it is convenient to package the logic needed to access these resources into a distinct component. In the EMAA methodology, these service-providing components are called servers.

EMAA is an architecture specification; its implementation is totally reusable and extendable. A single implementation of EMAA may serve as a foundation for several different agent systems. The architecture has three major components: the agents, servers and the dock. At the most basic level, the agents perform the specialized, user-defined work. Agents travel through the system via the docks. Although the dock has many

other functions, its primary purpose is to serve as a daemon[11] process used for sending and receiving agents between docks at other nodes. Furthermore, nodes that offer specialized services to agents must do so via a server. An overall picture of a distributed computing environment using the EMAA design is shown in Figure 7.

Figure 7: A Wide-Area Computing Environment;



Source: Lockheed Martin Advanced Technology Laboratories

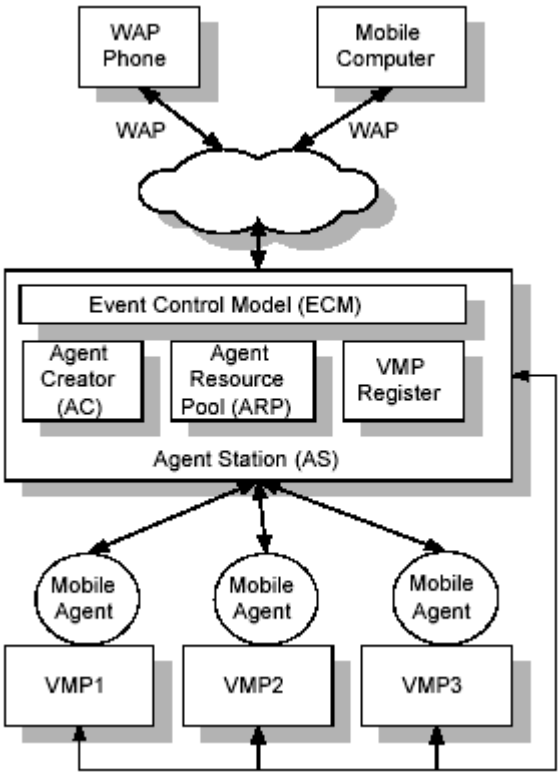
3.8.2. Mobile Agent Architecture For A Wireless Environment

The MAW - Mobile Agent Architecture for a Wireless Environment (Ieong &Guo, 1999)[20] is a client server based architecture, which consists of Agent Station module, Mobile agent module, and virtual market place modules.

[11] Daemon is a program that runs continuously and exists for the purpose of handling periodic service requests that a computer system expects to receive. The daemon program forwards the requests to other programs (or processes) as appropriate. Each server of pages on the Web has an HTTPD or Hypertext Transfer Protocol daemon that continually waits for requests to come in from Web clients and their users.

The interaction among the three modules is controlled by event control model shown in Figure 8.

Figure 8: Client-Server System Structure of MAW



Source: Ieong &Guo, 1999)[20]

- Client-Server based architecture

The Client-Server architecture is adopted for main components to communicate with each other across a range of hosts and networks. The clients such as WAP phones and mobile computer communicate with their corresponding Ass (Agent Station) via through the use of Push Access Protocol (PAP). MA is capable of finding services available in VMP (Virtual Market PLace) by build in self-discover features. MA provides interface to mobile users for agent operations while and VMP provides an agent community for various agents to interact and communicate. Multi-agents communications are established through services lookup and service discovery at VMP.

- Agent Station (AS)

Agent station is an object in a network. It has abilities to create an agent, dispatch an agent, and register VMP new service into the Agent Resource Pool (ARP). Further, AS is able to accept synchronize communication packets from WAP devices. In fact, users are browsing over ARP to select a suitable agent service. After requests from mobile users are being identified, agent creation process will produce an agent with specified attributes and dispatch it to the destined VMP.

- Mobile Agents (MAs)

As we have already mentioned, each mobile agent has five attributes. They are state, implementation, interface, identifier and principals. Agents always carry these attributes traveling from hosts to hosts. A mobile agent is not bound to the system where it begins execution. An agent will be assigned an agent ID to uniquely identify its identity and when it is created. When MA accomplishes its task, resulting message will be returned to its corresponding wireless device. If further negotiation is required, MA will bring its current execution state back to Agent Station (AS). MA has the ability to transport itself from one system in a network to another. The ability to travel allows a mobile agent to move to a system that contains an object with which the agent intends to interact. Since mobile agents are not bound to the system it was created, agents process can always transfer from host to hosts without interrupt. As a result, highly robust and fault-tolerant capabilities are found in the MAW paradigm. Making virtual connections of agent system is more reliable than wired network world.

- Virtual Market Place (VMP)

VMP is another object on the network; it consists of Voyager server and JINI[12] services protocol, when a services agent is attached into the VMP, Service Register Manager (SRM) registers its services into the services registry of the VMP through the JINI leasing services. When a user agent is attach into the VMP,

[12] Jini[tm] Network Technology: is the name for a distributed computing environment, that can offer ``network plug and play". A device or a software service can be connected to a network and announce its presence, and clients that wish to use such a service can then locate it and call it to perform tasks. Jini can be used for mobile computing tasks where a service may only be connected to a network for a short time, but it can more generally be used in any network where there is some degree of change. "Jini" isn't an acronym, so it does not stand for anything. Though, it has been stated that it does function as an anti-acronym: "Jini Is Not Initials".

SLM starts to lookup registered services available in current VMP, if it has found a desired service, user agent will communicate with the service agent through JINI registry services.

- Agents communications

Two types of agents have been classified in MAW architecture; they are service agent and user agent respectively. Both of them are created by Agent Station (AS), creation requests are either from wireless device or network terminals. When service agent dispatch to VMP. It will discover if the same service is already registered. It will register into VMP's JINI service pool only if there is no duplicated service registered in current VMP. When service agent is allowed to registry, a service ID will be assigned to this service agent and then new service will be updated to all VMPR of the associated AS. In addition, when a service leases expired, service registration will be remove due to VMP memory management process. On the other hand, user agent usually is a service consumer; it is dispatched to desired VMP and implements a Service-Finder operation, accompanying pre-assigned attributes and conditions. They will lookup and discover VMP registered service. Once a suitable service agent is found, user agent will interact with that service agent by message passing, which can be peer-to-peer[13] or broadcast[14] communication. If desired service is not found, MA will try another address in VMP address list. This will reduce network communication load when MA cannot found an expected service agent in VMPs.

3.8.3. eCAPSULE MOBILE AGENT TECHNOLOGY ARCHITECTURE

We definitely need a new concept of wireless infrastructure to support the vision of everyone and everything always connected and interoperating: eCapsule Mobile Agent Technology (Warp9 Inc. 2003) [http://ecapsule.warp9inc.com/]. This is a worldview of massive-complexity, connectivity, access, and interaction.

[13] A network in which the server is "nondedicated" (meaning it is also used as a workstation). In this kind of network, every computer acts on its own (storing files and accessing peripherals) and can "see" every other computer on the network (if it has the proper access privileges). Each workstation has equivalent capabilities and responsibilities.

[14] Commonly used to describe the ability to send the same e-mail message simultaneously to multiple recipients. The term "broadcasting" is also used to describe a live event on the Internet that is streamed in real time to millions of computers. In networking, a distinction is made between broadcasting and multicasting: Broadcasting sends a message to everyone on the network, whereas multicasting sends a message to a select list of recipients.

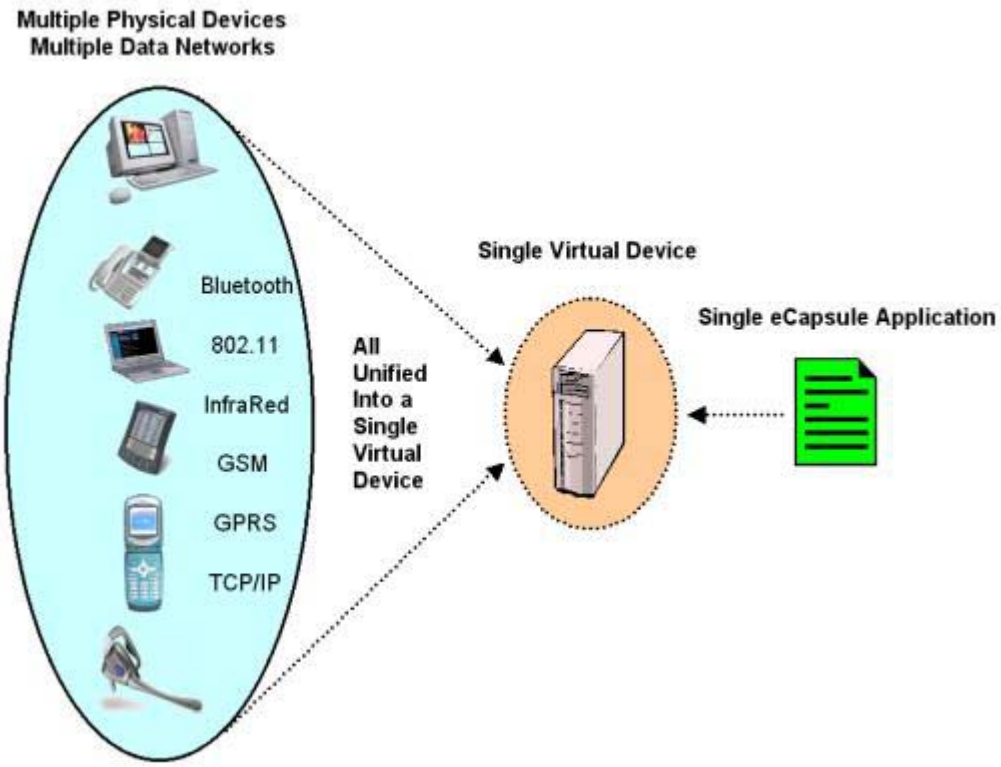
In the wired world, the only personal computing device is the personal computer (PC). However, in the wireless world, there is an evergrowing number of personal computing devices, including cell phones, personal digital assistants (PDAs), office PCs, home PCs, and car PCs. Developing applications for the mobile user is extremely more complex than for a single PC; not only are there multiple device types to consider, there are multiple data networks as well. eCapsule technology eliminates the complexities of developing applications for the mobile user by providing two powerful abstractions to the application developer. These abstractions make application development and deployment rapid and manageable. The two abstractions are:

- eCapsule Virtual Device

Unifies a group of personal computing devices, wired and wireless, into a single virtual device that appears like a larger device with the combined capabilities of the devices being unified. eCapsule technology unifies a group of computing devices, wired and wireless, into a single virtual device that appears to eCapsule programs to be like a larger device with the combined capabilities of all the devices being unified.

The value of this is clear; the complexity of developing mobile data applications for multiple devices and dealing with the chaotic wireless mobile world is simplified into writing a single application for a single virtual device as we might well see in the Figure 9 as shown below:

Figure 9: eCapsule Virtual Device.



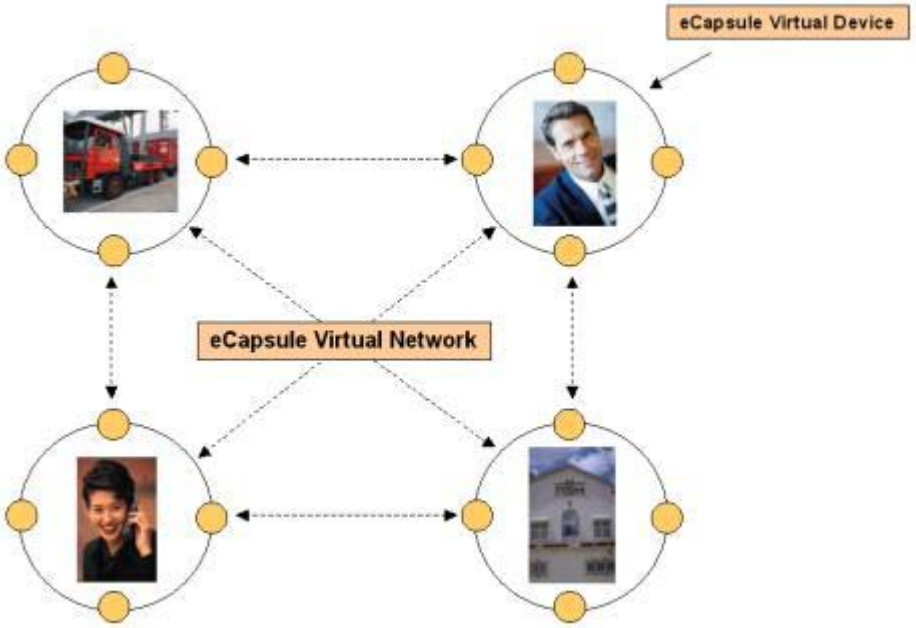
Source: Warp9 Inc. [http://ecapsule.warp9inc.com/]

- eCapsule Virtual Network

Unifies a group of eCapsule Virtual Devices into a single eCapsule Virtual Network (See Figure 10) that allows Virtual Devices to connect and interact with each other. eCapsule technology unifies a group of eCapsule Virtual Devices into a single eCapsule Virtual Network that allows Virtual Devices to connect and interact with each other. This virtual network lies on top of existing disparate networks.

This virtual network creates a manageable abstraction that enables the creation and deployment of incredibly complex applications.

Figure 10: eCapsule Virtual Network



Source: Warp9 Inc. [http://ecapsule.warp9inc.com/]

3.9. MOBILE AGENT STANDARDIZATION: MASIF

As we could well notice in the two previous sections, all these systems differ extensively in architecture and implementation, hence impeding interoperability and quick deployment of mobile agent technology in the marketplace. Thus, five companies such as FOKUS, IBM, Crystaliz, General Magic, and the Open Group have jointly developed a proposal for a Mobile Agent System Interoperability Facility (MASIF)[?], the first mobile agent standard of the Object Management Group (OMG).

MASIF speaks the interfaces between agent systems and not between agent applications and agent systems. MASIF is not about language interoperability. Language interoperability for mobile objects is very difficult, and MASIF is limited to interoperability between agent systems written in the same language but possibly by different vendors. We might say that MASIF defines the interfaces at the agent system level rather than the agent level.

MASIF standardizes the following current topics:

- *Agent Management:* One can visualize system administrator managing agent systems of different types via standard operations in a standard way: create an agent, suspend it, resume, and terminate.
- *Agent Transfer.* It is desirable that agent applications can freely move among agent systems of different types, resulting in a common infrastructure, and a larger base of available system agents can visit.
- *Agent and Agent System Names.* Standardized syntax and semantics of agent and agent system names allow agent systems and agents to identify each other, as well as clients to identify agents and agent systems.
- *Agent System Type and Location Syntax.* The agent transfer cannot happen unless the agent system type can support the agent. The location syntax is standardized so that the agent systems can locate each other.

3.10. APPLICATIONS

We will take a quick look at applications that benefit from the mobile agent paradigm.

Mobile Commerce: An example is a simple vending machine that is not connected to the Internet. How can it process digital money or be enabled for m-Commerce without a live Internet connection? The answer is within Mobile technology, which can aggregate intelligence and convergence. For example, enabling the vending machine to receive mobile communications and mounting the Mobile a Agent OS (Operating System) inside its embedded computer (See eCapsule Technology in the *Methodology & Tools Section*).

- *Electronic Commerce:* A commercial transaction may demand real-time access to remote resources such as stocks quotes and even agent-to-agent negotiation. MAs (Mobile Agent) might embody the intentions of their creators and act and negotiate on their behalf.

- *Personal Assistance:* For example in the case of a PDA device, to schedule a meeting with several other people, a user could send a MA to interact with the representatives agents of each of the people invited to the meeting. The agents could negotiate and establish a meeting time.

- *Secure Brokering:* In collaboration, the parties could let their MAs meet on a mutually agreed secure host, where collaboration can take place without the risk of the host taking the side of one of the visiting agents.
- *Distributed information retrieval:* Instead of moving large amounts of data to the search engine so that it may create search indexes, we dispatch agents to remote information sources, where they locally create search indexes that can be shipped later to the origin.
- *Telecommunications networks services:* support and management of telecommunications services are designated by dynamic network reconfiguration and customization.
- *Workflow applications and groupware:* It is all about the flow of information between coworkers. The MA is very useful here, because in addition to mobility it provides a level of autonomy to the workflow item.
- *Monitoring and notification:* An MA can monitor a given information source without being dependent on the location from which it originates. Agents can be dispatched to wait for certain types of information to become available.
- *Information dissemination:* MAs integrate the Internet push-model.
- *Parallel processing:* As MAs may create a cascade of clones in the network, one possible use of MAs technology is to administer parallel processing tasks.

3.11. METHODOLOGY & TOOLS

In this section we will describe in details the *eCapsule Mobile Agent Technology Architecture* (Warp9 Inc. 2003) [http://ecapsule.warp9inc.com/]. The Warp9's[15] mission is to be the leading provider of technology for developing and deploying multi-device and multi-network ubiquitous mobile data applications. Warp9 will enable businesses to harness the full potential and opportunities of the mobile world.

[15] Warp 9, Inc. is a software development firm that has developed a breakthrough mobile data technology that will enable efficient computing and communications anywhere, anytime and on any device. Warp 9 markets and licenses the eCapsule technology platform to the wireless mobile data industry. The company's customers include carriers, enterprises, service providers, software developers, system integrators and device manufacturers. The eCapsule technology platform is used to rapidly create, deploy and manage new and compelling mobile data applications that seamlessly and securely connect servers, personal computers, smart cell phones, personal digital assistants (PDAs) and wireless terminals in the wired and wireless worlds.

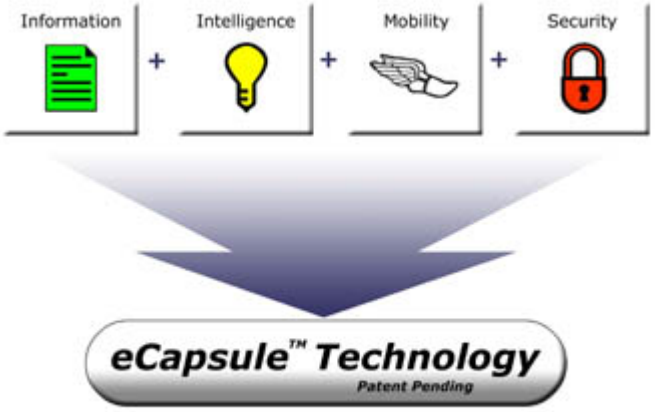
The eCapsule is a dynamic agent and fits the profile of the lightweight technology required to unlock this opportunity. Warp 9's eCapsule product family is tailored for the dynamic and massively scaleable convergence opportunity of wireless communications. eCapsules are needed to simplify this complex world and enable billions of applications and interconnected and interactive users.

3.11.1. Inside the eCapsule

eCapsule applications, aka eCapsules, combine data and logic inside an incredibly small secure software object that can move around on its own in the wired and wireless worlds, track down wired and wireless users and conduct transactions with little or no end-user interaction. eCapsules differ from traditional software programs that just live in one particular device. They are self-contained, intelligent, network aware, self-migrating software objects that live within an eCapsule Virtual Device and travel within the eCapsule Virtual Network.

An eCapsule has the following four built-in ingredients (Figure 11):

Figure 11: eCapsule Ingredients



Source: Warp9 Inc. [http://ecapsule.warp9inc.com/]

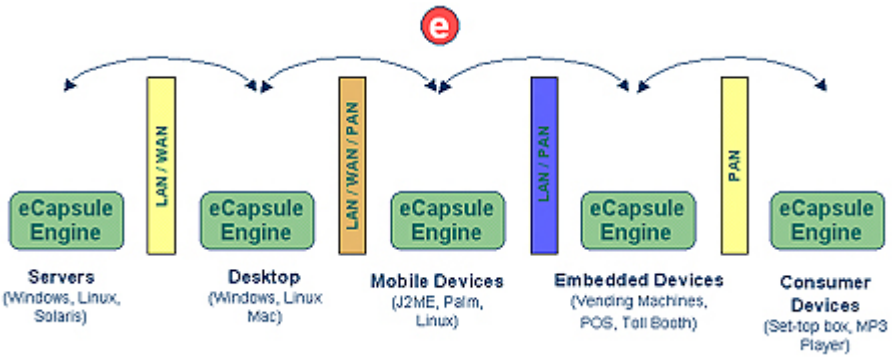
3.11.2. eCapsule Products

eCapsule technology enables the development and deployment of an unlimited number of applications that can run in both the wired and/or wireless worlds, and over all available computing devices and networks. The eCapsule technology platform is comprised of three primary components:

- eCapsule Device Engine

Installed on computing devices to enable the eCapsule Virtual Device In order to receive, process, execute and manage eCapsules, each device must have an eCapsule Device Engine. The eCapsule Engine is a software virtual machine that resides on various devices that provides an execution environment and resource interface for eCapsules. eCapsule Device Engines (Figure 12) interact with each other to seamlessly create eCapsule Virtual Devices.

Figure 12: eCapsule Device Engine



Source: Warp9 Inc. [http://ecapsule.warp9inc.com/]

As we might well notice in the Figure 12, eCapsules can easily and uniformly access device level network transport mechanisms through the Engine's built in support for various wide-area-network (WAN), local-area-network (LAN), or personal-area-network (PAN).

Supported Transport	*Supported Operating Systems*
• TCP/IP • 802.11x • Bluetooth • InfraRed	• Microsoft Pocket PC • Microsoft Windows • Palm OS • Sun Java - J2SE, J2EE, J2ME • Linux and Embedded Linux • Other UNIX • Symbian OS

The eCapsule Device Engine is available in C and Java versions. Source code and compiled object codes are available to device manufacturers, embedded systems and other OEMs. Self-installing versions of the Engine are available to solutions developers for easy deployment.

3.11.3. eCapsule Gateway Server

Management of eCapsule Virtual Devices within an eCapsule Virtual Network The eCapsule Gateway allows the IT Administrator to easily set up and manage their own eCapsule deployment infrastructure. Using any standard web browser, end-users of eCapsule applications can securely manage their own account information such as provisioning mobile devices, managing eCapsule Virtual Device configuration, viewing eCapsule transaction history, and managing new eCapsule services. IT professionals can monitor and manage the eCapsule traffic that flows through the gateway.

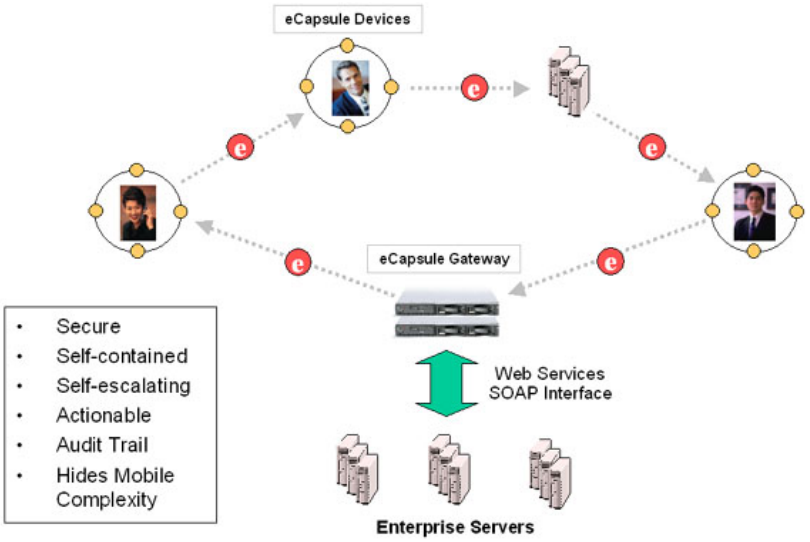
The primary components in the eCapsule Gateway are:

- *eCapsule Monitor:* Provides tracing, logging and managing of eCapsule traffic going in and out of the enterprise
- *eCapsule Authenticator:* Provides high-speed authentication and verification of eCapsules
- *eCapsule Resolver:* Provides the mapping of a set of physical devices onto an eCapsule Virtual Device
- *eCapsule Development Kit*: These eCapsule APIs and development tools allow developers to easily create standalone eCapsule applications and/or to embed them into their solutions

3.11.4. Roaming Messenger

With Roaming Messenger, a secure message in the form of e-mail, fax or voice can be encapsulated in an eCapsule along with the intelligence to roam through network personal devices, such as smart cell phones, PDAs, computers and even TV set-top boxes, delivering a message and interacting with the recipient in real-time. To date, no other technology has been able to deliver this form of messaging. See Figure 13 for details.

Figure 13: Roaming Messenger



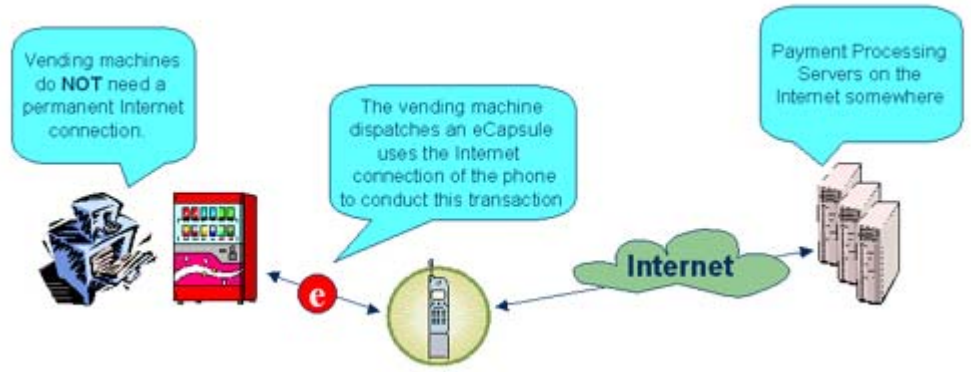
Source: Warp9 Inc. [http://ecapsule.warp9inc.com/]

3.11.5. Applications

Users might purchase from the vending machine using their Mobile Agent enabled phones, PDAs, or other wireless devices. The transaction is managed by Mobile Agents who communicate securely and directly with a remote Internet server through the Internet connection of the handheld device, completing the entire purchase transaction back to the vending machine and delivering the goods. In this example, Mobile Agents offering unlined network convergence with low operational cost. Product information (along with ordering

logic), can be encapsulated in an `eCapsule` to allow a user with a mobile device to complete a transaction anywhere, anytime.

Figure 14: Vending Machine & Mobile Commerce.



Source: Warp9 Inc. [http://ecapsule.warp9inc.com/]

According to Figure 14, wireless mobile vending solutions today demand the physical machine to have a live Internet connection that makes mass deployment troublesome and expensive. Using eCapsule technology, a purchase transaction can be completed with end-to-end security by allowing the vending machine to share the connection on the smart phone or PDA using an Infrared or Bluetooth connection securely. The eCapsule can fully encapsulate the payment transaction from the vending machine, to the handheld device, to the Internet and back.

3.12. SECURITY ARCHITECTURE

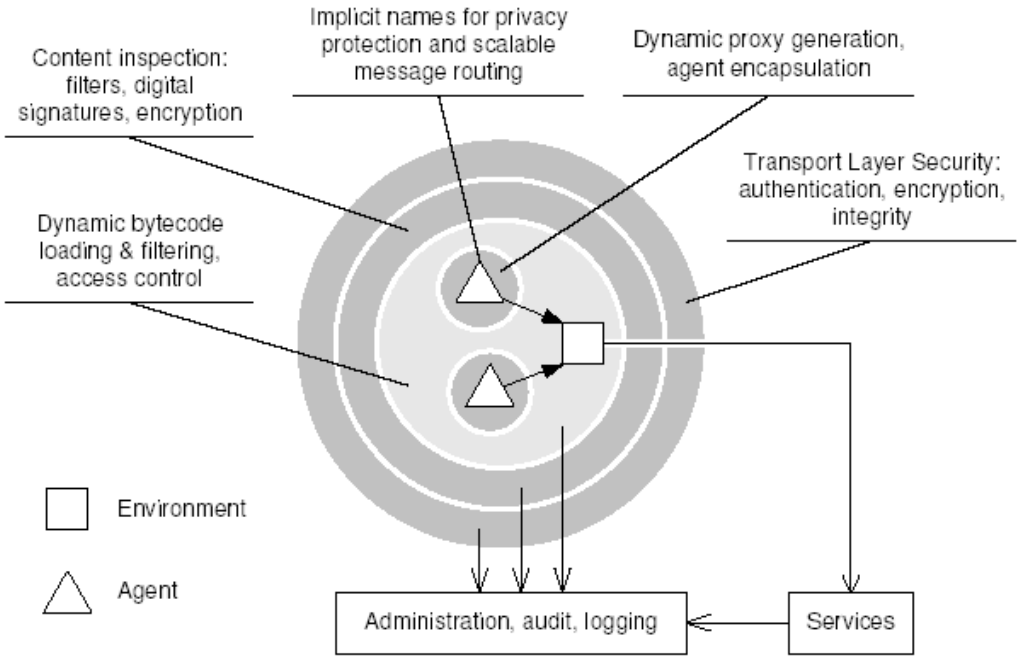
In order to exploit benefits mentioned in the *Mobile Agents Benefits for the Creation of Distributed Systems Section 5.1.*, mobile agent frameworks have to deal with a number of security threats. We will describe Security issues through an example-based approach in order to provide a detailed understanding of the security mechanisms.

- An Example in the SeMoA's Architecture

A mobile agent's itinerary in general spans a number of servers which might be run by competing operators (Roth & Jalali-Sohi, 2000)[37]. Apart from monitoring, manipulating, and stealing data from mobile agents, malicious hosts might try to abuse passing agents as Trojan Horses in attacks on competing servers while incriminating the agent's owner in the process. On the other hand, hosts have to be aware of malicious agents breaking into the server in order to harm other agents hosted by it, or to gain unauthorized system access. MAs make a perfect cover for viruses, worms, and Trojan Horses. In particular, mobile agents might try to attack remote hosts using innocent hosts as launch pads in order to cover the tracks of the attacker. Both agents and servers are threatened by attacks originating from outside the system. Eavesdroppers might snoop on agents being transferred over network connections hence compromising the privacy of agents. They might also launch active attacks on servers either directly or indirectly by manipulating agents during transport. Network-based attacks on mobile agent systems do not add new threats when compared to client/server applications, although the impact of a successful attack might be much worse. A sound security model which is able to resist these attacks is fundamental to business acceptance and market exploitation of this fascinating technology. SeMoA builds on JDK 1.31 and is a best effort to provide adequate security for mobile agent systems, servers as well as agents.

The security architecture of the SeMoA(Roth & Jalali-Sohi, 2000)[37] server compares to an onion: agents have to pass all of several layers of protection before they are admitted to the runtime system (see Figure 15) and the first class of an agent is loaded into the server's JVM.

Figure 15: SeMoA's Security Architecture



Source: (Roth & Jalali-Sohi, 2000)[37]

The SeMoAs Security Architecture of SeMoA resembles that of an onion; agents have to pass all layers before being installed and launched in the server's runtime system.

The first (outer) security layer is a transport layer security protocol such as Transport Layer Security (TLS)[16] or Security Sockets Layer (SSL)[17]. In this case, the SSL implementation is used that comes with the Java

[16] Transport Layer Security (TLS): This protocol is an industry standard designed to protect the privacy of information communicated over the Internet. TLS assumes that a connection-oriented transport, typically TCP, is in use. The TLS protocol allows client/server applications to detect: message tampering, message interception and message forgery. The full specification of the TLS Protocol is available from the IETF web site http://www.ietf.org/rfc/rfc2246.txt.

[17] Secure Sockets Layer (SSL) : Digital certificates encrypt data using Secure Sockets Layer (SSL) technology, the industry-standard method for protecting web communications developed by Netscape Communications Corporation. The SSL security protocol provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP

Secure Socket Extension (JSSE) framework provided by Sun Microsystems. This layer provides mutual authentication of agent servers, transparent encryption and integrity protection. Connection requests of authenticated peers can be accepted or rejected as specified in a configurable policy.

The second layer consists of a pipeline of security filters. Separate pipelines for incoming agents and outgoing agents are supported. Each filter inspects and processes incoming/outgoing agents, and either accepts or rejects them. We refer to this filtering procedure also as content inspection in analogy to concepts known from firewalls. At the time of writing, SeMoA features two complementary pairs of filters that handle digital signatures and selective encryption of agents (signature verification and decryption of incoming agents, encryption and signing of outgoing agents). An additional filter at the end of the incoming pipeline assigns a configurable set of permissions to incoming agents, based on information established and verified by preceding filters.

Permissions can be granted based on the authenticated identities of the agent's owner, the sponsor of its last state change, and its most recent sender. Filters can be registered and unregistered either dynamically or at boot time of the SeMoA server either programmatically or by means of configuration files. Subsequent to passing all security filters, a sandbox is set up for the accepted agent (which can be regarded as layer four). Each agent gets a separate thread group and class loader. The agent is unmarshalled[18] by a thread that is already running within the agent's thread group and becomes the first thread of that agent. Marshalling is done by the very same thread after all remaining threads in the agent's thread group have terminated. Since the Serialization Framework of Java provides callbacks that pass control back to objects to be serialized, the thread once again blocks until no more threads remain in the agent's thread group. Only then does SeMoA handle any migration requests of that agent. This prevents agents from flooding a network of agent servers by migrating and refusing to terminate at the same time.

The classes brought by an agent are annotated with tag permissions which are generated dynamically and which are unique for each agent. Whenever e.g. an agent attempts to modify a thread group the current thread

connection. Because SSL is built into all major browsers and web servers, simply installing a digital certificate turns on their SSL capabilities.

[18] The reverse of marshalling; the process of reading an XML document and constructing a tree of content objects. Each content object corresponds directly to an instance in the input document of the corresponding schema component, and the content tree represents the document's content and structure as a whole.

is traced back to the corresponding agent's thread group which then identifies a tag permission to test. This prevents agents from manipulating threads of other agents even if one agent invokes methods of another. A configurable threads filter sorts threads created by « special » classes (e.g. classes of the Abstract Window Toolkit (AWT) into separate thread groups so that these do not interfere with agent threads.

An agent's classes are loaded by its dedicated class loader. This class loader supports loading classes that came bundled with the agent, as well as loading classes from remote code sources specified in the agent. All loaded classes (save those in the class path of the server) are verified against a configurable set of trusted hash functions. The digests of verified classes must match corresponding digests signed by the agent's owner (which can be regarded as layer three). Thus, only classes authorized by the agent's owner for use with his agent are loaded into the agent's namespace.

Agents cannot share classes so one agent cannot not load a Trojan Horse class into the name space of any other agent. However, in order to allow method invocations between agents, they may share interfaces. Interfaces are deemed to be the same if their trusted digests match. In this case, an agent's class loader returns a previously loaded interface rather than loading the interface again from the served agent, so that the interfaces used by the agents are type-compatible wherever possible. Of course, this works only if the interface classes referenced by two agents are bitwise identical. Improved schemes may compare interface implementations on the API level. However, this adds overhead to the class loading process, and is not yet implemented.

Interface objects are managed in a Map that is global to all agent class loaders. Pollution attacks on this Map require breaking the trusted hash functions. Before a class is defined in the Java VM, the bytecode of that class has to pass a filter pipeline similar to the one for incoming agents. Each class filter can inspect, reject, and even modify the bytecode. SeMoA comes with an example filter that rejects classes which implement finalize( ). Malicious agents may implement this method in order to attack the garbage collector[19] thread of the hosting VM. Additional filters may implement bytecode arbitration, for example in order to add resource accounting to agent classes.

[19] Garbage collection, also known as *automatic memory management*, is the automatic recycling of dynamically allocated memory. Garbage collection is performed by a garbage collector which recycles memory that it can prove will never be used again. Systems and languages which use garbage collection can be described as *garbage-collected*.

Agents are separated from all other agents in the system; no references to agent instances are published by default. The only means to share instances between agents is to publish them in the registry. Each agent gets its own view on the registry (referred to as the agent's environment), which tracks the objects registered by that agent. All published objects are wrapped into proxys which are created dynamically. If the agent terminates or retracts a published object, then the agent's environment instructs the handler of the corresponding proxy to invalidate its link to the original object. This makes the original object unavailable even to other agents that looked up its reference in the registry. Furthermore this makes the original object available for garbage collection.

- A Java-based Mobile Agent Security Capabilities and Issues

Capabilities

Strong mobility: Capture full execution state of running agents, provides anytime mobility (simplifies writing mobile agents), and provides forced mobility – arbitrary Java code can be moved

Strong security: Dynamically control resource usage (rates and quantities), allows platform owner full control over agent execution, and protect against denial of service attacks

Issues

Protecting network communication: Protecting hosts from agents (Illegal access and Denial of service)
Protecting agents from hosts: Tampering, Extracting information, Capture / Replay.

3.13. TRENDS OF MOBILE AGENTS AND THE FUTURE OF THE INTERNET

There are several trends (Kotz & Gray, 1999)[23] affecting Internet technology:

Bandwidth: The telecommunications industry is laying down astonishing amounts of fiber. Although Internet traffic is growing extensively, the bandwidth soon to be available on the Internet backbone, as well as to many offices and neighbourhoods, is immense. Nonetheless, bandwidth to many end users will remain

limited by several technical factors. Many users will still connect via modem, or at best, ADSL[20]. Many other users will connect via low-bandwidth wireless networks. Most users can expect to see no more than 128 Kbps to 1 Mbps available at their palmtop or desktop, although some asymmetric cable modems may reach 10 Mbps (for downloads).

Mobile devices: One of the hottest areas of growth in the computer industry is portable computing devices. Everything from laptops to palmtops to electronic books, from cars to telephones to pagers, will access Internet services to accomplish user tasks, even if users have no idea that such access is taking place. Typically, these devices will have unreliable, low-bandwidth, high-latency telephone or wireless network connections.

Mobile users: Web-based email services, for example *hotmail.com*, make it clear that users value the ability to access their email from any computer. Web terminals will become commonplace in public spaces, such as cafes, airports, and hotels. Eventually, particularly with the growth in bandwidth, users will have full access to all of their files and applications from any terminal. Although this, mobile devices will proliferate, since just as with public phones.

Intranets: Organizations are increasingly using Internet protocols, particularly HTTP, to build internal intranets for their own distributed-information needs. All access to an intranet is managed by a single organization. Hence, new technologies can be deployed quickly, since little coordination is needed with outside organizations, and security (within the intranet) is of less concern.

Information overload:. Internet users are already inundated by the unconditional volume of information, and the problem will get worse as the Internet grows. Search engines, shopbots, portals, collaborative filtering, and email filtering are existing technologies that allow the user to reduce the flood to a manageable stream, but these technologies are still quite limited.

Customization: Unlike broadcast media, the Internet makes it possible to customize access for each user. Current technologies allow customization at both the client (browser) and the server. Many Web sites include

[20] ADSL : Asymmetric Digital Subscriber Line is a service that is available over an ordinary telephone line that uses spare capacity not needed for voice traffic to deliver high-speed data transfer. It is called asymmetric because the speed at which data moves from the exchange to the customer is different to the speed at which it moves from the customer to the exchange.

their own site-specific customization features, but the customization is increasingly provided by third-party proxy[21] sites.

Proxies: Such proxy sites, which today are most of ten Web sites such as the various shopbots, interpose between a user and one or more other Internet services. As a means to both reduce information overload and customize service access, proxy sites will become more and more important. In particular, as portable devices become more prevalent, highly specialized proxy sites will be provided to meet the special needs of mobile users.

3.13.1. TECHNOLOGICAL OBSTACLES

The objective of these following two sections is to present the obstacles that might be encountered when facing and analysing a mobile agent system implementation. It is important to highlight that the *mobile agent & business communities* must consider these obstacles, not in terms of *disadvantages* of a MA system but rather as fundamental issues that must be addressed urgently in order to take advantage of a MA system in the mobile world.

There are several technical obstacles (Kotz & Gray, 1999)[23] that must be cleared before mobile agents can be widely used.

Performance and scalability: Current mobile-agent systems save network latency and bandwidth at the expense of higher loads on the service machines, since agents are often written in a (relatively) slow

interpreted language for portability and security reasons, and since the agents must be injected into an appropriate execution environment upon arrival. Thus, in the absence of network disconnections, mobile agents often take longer to accomplish a task than more traditional implementations. Luckly, significant progress has been made on just-in-time compilation (most notably for Java), software fault isolation[22], and other techniques, which allow mobile code to execute nearly as fast as natively compiled code.

[21] An intermediary program that acts as both a server and a client for the purpose of making requests on behalf of other clients. Proxies are often used as client-side portals (i.e., a trusted agent that can access the Internet on the client's behalf) through the network firewall and as helper applications for handling requests via protocols not implemented by the user agent.
[22] Fault isolation is the ability for administrators to locate the source of failure and remove it from operation.

Portability and standardization: Practically, all MA (mobile agent) systems allow a program to move freely among heterogeneous machines, for example, the code is compiled into some platform-independent representation such as Java bytecodes, and then either compiled into native code upon its arrival at the target machine or executed inside an interpreter. For MAs to be largely exploited, however, the code must be portable across mobile-code systems. Making code portable across systems will require a significant standardization effort. The OMG MASIF standard[23] is an initial step, but addresses only cross-system communication and administration, leading to a situation in which an agent can not migrate to the desired machine, but instead only to a nearby machine that is running the right agent system.

Security: It is possible now to deploy a mobile-agent system that adequately protects a machine against malicious agents. Numerous challenges remain, however: (1) protecting the machines without artificially limiting agent access rights; (2) protecting an agent from malicious machines; and (3) protecting groups of machines that are not under single administrative control. An inadequate solution to any of these three problems will severely limit the use of mobile agents in a truly open environment such as the Internet.

3.13.2. NON-TECHNOLOGICAL OBSTACLES

As we will see in this section, there are several non-technological issues that must be addressed by the *mobile & business communities* for a vast adoption of mobile-agent technology.

Internet sites must have a strong motivation to overcome apathy, justify the cost of upgrading their systems, and adopt the technology (Kotz & Gray, 1999)[23]. While the technological arguments above are significant, they are not sufficient for most site administrators. In the end, the technology will be installed only if it provides substantial improvements to the end-user's experience: more useful applications, each with fast access to information, support for disconnected operation, and other important features.

Lack of a killer application: The most important obstacle is that there is no killer application for mobile agents. The MA paradigm is in many aspects a new & powerful programming paradigm, and its use drives to faster performance in many cases.

[23] OMG (Object Management Group) A consortium of software vendors, developers, and users promotes the use of object-oriented technology in software applications. The group also maintains the CORBA software interoperability standard.
MASIF (Mobile Agent System Interoperability Facility) [http://www.fokus.gmd.de/research/cc/ecco/masif/body_index.html]

Getting ahead of the evolutionary road: It is improbable that any Internet service will be willing to jump directly from existing client-server systems to full mobile-agent systems. Researchers must provide a clear evolutionary path from current systems to mobile-agent systems. In particular, although full mobile-agent systems involve all the same research issues as more restricted mobile-code systems, researchers must be careful to demonstrate that the switch to mobile agents can be made incrementally. A good example, applets, mobile code that migrates from server to client for better interaction with the user. From applets, the next step is proxy sites that accept mobile code sent from a mobile client. Probably, such proxies will be first provided by existing Internet service providers (ISPs). Since the sole function of the proxy sites will be to host mobile code, and since the ISPs will receive direct payment for the proxy service, the ISPs will be willing to accept the security risks of mobile code. Once mobile-code security is even more tested on proxy sites, the services themselves have already started to accept servlets, mobile code sent from the client directly to the server (or from the proxy to the server). Once servlets become largely used, and as researchers address the problem of protecting mobile code from malicious servers, services will start to accept mobile agents.

Another decisive evolutionary road is the migration of agent technology from intranets to the Internet. Mobile-code technologies will appear first in the normally safe intranet environment, especially intranets that are made up on high-latency networks such as a WAN (Wide Area Network) or a wireless network for mobile computers. For example, a large company, especially one with a mobile workforce, might find mobile agents the most interesting way to provide its employees with a large range of access to its internal databases.

4. CONCLUSION

There is a strong case for the use of mobile agents in many Internet applications. Further, there is a clear evolutionary road that will carry us from current technology to widespread use of mobile code and agents within the next few years. Mobile e-commerce applications will soon gain total force. However, real user demands are still ignored. Agent technology, especially MAs, might be a viable way to fulfill all requirements, such as rapid, easy and economic creation, testing and introduction of applications, and provide the user with personal as well as terminal mobility. Hence, agent-based applications might cover the needs of flexible environments, where know and trusted agents serve the user needs.

Besides, network operators and service providers should support performing and scalable agent-based concepts in order to provide their clients adaptable and efficient service environments, which allows mobile communication.

5. REFERENCES

5.1. Printed References

[1] G. Allée, Thesis: « Sécurité des agents mobiles : Protocole d'enregistrement d'itinéraire par agents coopérants », École Polytechnique de Montréal, T7 U54 2001 v.036.

[2] J.P. Bigus and J. Bigus, « Constructing Intelligent Agents Using Java: Professional Developer's Guide », 2001, 2nd Edition, John Wiley & Sons.

[3] S. Bosworth & M. Habay, « Computer Security Handbook », 2002, 4th Edition, John Wiley & Sons.

[4] B. Chaib-draa (Université Laval) : « Programmation multiagent: méthodes, outils et applications » Seminar, 2003, UQAM-Université du Québec à Montréal.

[5] S. Goutet, Thesis: « Conception d'une architecture multi-agents supportant des agents mobiles intelligents », École Polytechnique de Montréal, T7 U54 v.093.

[6] D. B. Lange & M. Oshima, « Programming and Deploying Java Mobile Agents with Aglets », 1998, Addison-Wesley Longman, Inc.

[7] J. Liu, « Autonomous Agents and Multi-Agent Systems », 2001, World Scientific Publishong Co.

[8] C. Reveilli, « Les agents intelligents », Paris : Dunod, 2000.

[9] B. Thierno, Thesis : « Une approche basée sur les agents mobiles », École Polytechnique de Montréal, T7 U54 2002 v.058.

[10] M. J. Wooldridge and N. Jennings, « *Agent Theories, Architectures and Languages: A Survey* », Springer-Verlag Lecture Notes in AI Volume 890, February 1995. 40

[11] BOOK TEACHER aimeur, « », 2002, 2nd Edition, John Wiley & Sons.

5.2. Web References

[12] Agent Builder, IntelliOne Technologies.[http://www.agentbuilder.com/Documentation/whyAgents.html]

[13] E. Aimeur, « Traitement des connaissances" course's Web site, Université de Montréal, Master of Science in Electronic Commerce, Montreal, Canada, 2003 [http://www2.iro.umontreal.ca/~aimeur/].

[14] « Agents for E-Commerce ». [http://agents.umbc.edu/Agents_for_.../.Electronic_commerce/index.shtml]

[15] K. Blixt and R. Öberg, « Software Agent Framework Technology », LiTH-IDA-Ex00/14, March 9, 2003. [http://www-und.ida.liu.se/~karbl058/saft/SAFT.pdf].

[16] J. Brustoloni, "Autonomous Agents: Characterization and Requirements," 1991 [http://www.msci.memphis.edu/~franklin/AgentProg.html]

[17] O. Etzioni & D. Weld, 1994, « A Softbot-Based Interface to the Interne », Department of Computer Sciences and Engineering, University of Washington.
[http://www.cs.washington.edu/homes/weld/papers/cacm.pdf]

[18] S. Franklin & A. Graesser, « Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents », Institute for Intelligent Systems, University of Memphis, 1996. [http://www.msci. memphis.edu/~franklin/AgentProg.html]

[19] J. Hartmann & R. Keller, "Nomadic Communication – Impacts of agent-based mobile e-commerce applications ». [http://www.jens-hartmann.de/papers/nomadic.pdf]

[20] S. Ieong and Z. Guo, « MAW - The Architecture of Mobile Agent in Wireless Environment », University of Macau, 1999. [http://www.mca.org.mo/it-cong/articles/2001/the_architecture_of_mobile_agent_in_wireless_environment.pdf]

[21] IBM Research, « Aglets », February, 2003. [http://www.trl.ibm.com/aglets/about_e.htm]

[22] R. Jha & S. Iyer, “Performance Evaluation of Mobile Agents for E-Commerce Applications”, Kanwal Rekhi School of Information Technology, Indian Institute of Technology Bombay, Powai, Mumbai - 400 076. [http://www.pspl.co.in/Persistent/publications/Performa.pdf]

[23] D. Kotz and R. Gray, « Mobile Agents and the Future of the Internet », Department of Computer Science/Thayer School of Engineering, Dartmouth College, Hanover, New Hampshire 03755. [http://www.cs.dartmouth.edu/%7Edfk/papers/kotz:future2.pdf]

[24] D. Lange, “Mobile Objects and Mobile Agents: The Future of Distributed Computing?, [http://www.moe-lange.com/danny/ecoop98.pdf]

[25] R. Lentini, G. Rao, J. Thies, and J. Kay, « EMAA: An Extendable Mobile Agent Architecture », Lockheed Martin Advanced Technology Laboratories, 1998. [http://www.atl.external.lmco.com/overview/papers/930-9823.pdf]

[26] A. Lingnau and O. Drobnik, « An Infrastructure For Mobile Agents: Requirements And Architecture », Fachbereich Informatik (Telematik), JohannWolfgang Goethe-Universit¨at Frankfurt amMain, Germany, 1995. [http://kuba.korea.ac.kr/ ~ixix/Article/agent/13dis-paper.pdf]

[27] M. Luck, P. McBurney & C. Preist, « Agent Technology: Enabling Next Generation Computing. A Roadmap for Agent-Based Computing », 2003, version 1.0. [http://www.agentlink.org/roadmap/roadmap.pdf]

[28] MASIF - The OMG Mobile Agent System Interoperability Facility (MASF), The Standard. [http://www.fokus.gmd.de/research/cc/ecco/masif/]

[29] Massachussets Institute of Technology. [http://www.swiss.ai.mit.edu/projects/scheme/]

[30] Mobile Agents Tutorial, University of West Florida, 2001. [http://www.objs.com/agent/00-12-05.ppt]

[31] J. Newmarch, “Jan Newmarch's Guide to JINI Technologies”, Version 2.08, 12 June 2001. [http://jan.netcomp.monash.edu.au/java/jini/tutorial/Jini.xml]

[32] T. Qian , « Cherubim: A Mobile Agent Based Security Architecture », Systems Software Research Group, Department of Computer Science, University of Illinois at Urbana-Champaign, 2003. [http://devius.cs.uiuc.edu/Security/cherubim/]

[33] Pyhton Language Web Site, 2003. [http://www.python.org/]

[34] Reticular Systems, « Using Intelligent Agents for Wireless Ecommerce Applications: The YellowStone Project », Revision 1.0, September 2, 1999. [http://www.reticular.com/Library/yellowstone.pdf]

[35] A. Reyes and A. Barba (Department of Engineering Telematics Technical University of Catalonia), Victor Callaghan, and Graham Clarke (Department of Computer Science University of Essex), « The Integration of Wireless, Wired Access and Embedded Agents in Intelligent Buildings ». Submitted to SCI 2001, The 5th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, Florida, July 22-25, 2001. [http://iieg.essex.ac.uk/papers/communications.pdf]

[36] V. Roth (Fraunhofer Institut fur Graphische Datenverarbeitung - Darmstadt, Germany) and V. Conan (Thomson-CSF Communications - Cedex, France), « Encrypting Java Archives and its Application to Mobile Agent Security », 1998. [http://www.igd.fhg.de/igd-a8/publications/MobileAgents/Roth01b.pdf]

[37] V. Roth and M. Jalali-Sohi, « SeMoA - Concepts and Architecture of a Security-Centric Mobile Agent Server », 2000. Fraunhofer Institut fur Graphische Datenverarbeitung - Darmstadt, Germany. [http://www.igd.fhg.de/igd-a8/publications/MobileAgents/Roth01b.pdf]

[38] P. Saffo, Director and Roy Amara Fellow, Institute for the Future. [http://www.saffo.org/biography.html]

[39] « Strategic Research Profile », Strategic Research Corporation, February, 2003. [http://www.sresearch.com/strat_profiles/src-warp9-mobile-data.pdf]

[40] J. Vaucher, « IFT6802 : Commerce électronique: systèmes et architectures » Course Web site, Université de Montréal, Master of Science in Electronic Commerce, Montreal, Canada, 2003. [http://www.iro.umontreal.ca/~vaucher/ift6802/Guide.html]

[43] Sars Inc., Secure Global Fleet Tracking, 2003. [http://www.sarsinc.com/service_intro.asp]

[44] NeoSoft.com, “What is Tcl?”. [http://www.neosoft.com/tcl/whatistcl.html]

[45] “Trends Wars - Mobile agent applications ». [http://www.computer.org/concurrency/pd1999/pdf/p3080.pdf]