# Mobile agents for e-commerce

**By**

**Rahul Jha**
**Roll No.  : 99329011**

**Guided By**
**Prof. Sridhar Iyer**

**KR School of Information Technology**
**Indian Institute of Technology, Bombay**

# Abstract

In the past few years, mobile agent (MA) paradigm has received a great deal of attention. While MAs have generated considerable excitement in the research community, they have not translated into a significant number of real-world applications. One of the main reasons for this is the lack of work that quantitatively evaluates the effectiveness of mobile agents versus traditional approaches. This project contributes towards such an evaluation and implements an e-commerce application using mobile agents.

The existing mobile agent applications in the domain of e-commerce are classified and the underlying patterns of mobility are identified. These in turn determine several implementation strategies using the traditional client-server and the mobile agent paradigms. The project quantitatively evaluates various implementation strategies and identify various application parameters that influence application performance. The project also provides qualitative and quantitative comparison across three Java based mobile agent framework viz. Voyager, Aglets, Concordia, for e-commerce applications. Finally, we present the implementation and deployment issues of a complete B2C e-commerce application using mobile agent and messaging and discuss the software engineering aspects of mobile agent technology.

# Table of contents

*1 Introduction ........ 4*

**1.1 Project goals and work done ........ 5**

**1.2 Thesis organization ........ 5**

*2 Mobile agent technology ........ 6*

**2.1 Agents ........ 6**

**2.2 Mobile agents ........ 6**

**2.3 Mobile agents and the traditional model ........ 7**

**2.4 Advantages of using mobile agents ........ 7**

**2.5 Existing mobile agent systems ........ 8**

**2.6 New trends in Internet applications ........ 9**

*3 Agent frameworks ........ 10*

**3.1 Java as technology base for mobile agents ........ 10**

**3.2 Aglets ........ 11**

**3.3 Concordia ........ 15**

**3.4 Voyager ........ 17**

*4 Related work ........ 21*

**4.1 Software agents in e-commerce ........ 21**

**4.2 Mobile agents in e-commerce ........ 22**

**4.3 Summary ........ 23**

*5 Quantitative evaluation of Voyager for e-commerce applications ........ 24*

**5.1 Mobile agents in e-commerce ........ 24**

**5.2 Mobility patterns ........ 25**

**5.3 Implementation issues ........ 27**

**5.4 Experimentation and results ........ 28**

**5.5 Conclusions ........ 31**

*6 Evaluation of Aglets, Concordia and Voyager ........ 32*

**6.1 A qualitative evaluation ........ 32**

**6.2 A quantitative evaluation ........ 35**

**6.3 Conclusion ........ 39**

*7 Our Prototype of e-commerce application using mobile agents ........ 41*

**7.1 Implementation aspects ........ 41**

**7.2 Agent framework .......... 42**
**7.3 Prototype architecture .......... 42**
**7.4 Entities involved .......... 43**
**7.5 Features and suitability of our design .......... 47**
**7.6 Test bed .......... 49**
**7.7 The e- market place .......... 49**
**7.8 Conclusion .......... 49**
***8 Conclusion and future work .......... 51***
**8.1 Design paradigm evaluation .......... 51**
**8.2 Mobile agent framework evaluation .......... 52**
**8.3 Software engineering issues .......... 52**
**8.4 Ideas and future work .......... 53**
***9 References .......... 54***

# Chapter 1

## Introduction

The emergence of e-commerce applications has resulted in new net-centric business models. This has created a need for new ways of structuring applications to provide cost-effective and scalable models. The number of people buying, selling, and performing transactions on the Internet is expected to increase at a phenomenal rate. However, the potential of the Internet for truly transforming commerce is largely unrealized to date. Electronic purchases are still largely non-automated. While information about different products and vendors is more easily accessible and orders and payments can be dealt with electronically, a human is still in the loop in all stages of the buying process, which adds to the transaction costs. A human buyer is still responsible for collecting and interpreting information on merchants and products, making decisions on merchants and products and finally entering purchase and payment information.

Mobile Agent (MA) systems have for some time been seen as a promising paradigm for the design and implementation of distributed applications. A mobile agent is a program that can autonomously migrate between various nodes of a network and perform computations on behalf of a user. Some of the benefits provided by MAs for creating distributed applications include reduction in network load, overcoming network latency, faster interaction and disconnected operations[9]. In the past, a range of roles for agents have been explored such as information retrieval, automating repetitive task and workflow. MAs are useful in applications requiring distributed information retrieval since they move the location of execution closer to the data to be processed. Software mobile agents help people with tedious repetitive job and time consuming activities.

A suitable application for mobile agents is the electronic commerce. Mobile agent technologies can be used to automate several of the most time consuming stages of the buying process. Unlike "traditional" software, mobile agents are personalized, and autonomous. MA move around the network searching for a user specified product across different shops. With the MA moving to the shops, the number of information exchange is local and is not over the network, thus saving network latencies and load. Using the mobile agent technology client specific queries could be executed at the shops site. Qualities inherent to MAs are conducive for optimizing the whole buying experience and revolutionizing e-commerce over then net.

We believe that the effective use of mobile agents can dramatically reduce transaction cost involved in e-commerce, in general, and in business-to-consumer transaction, in particular.

## *1.1 Project goals and work done*

While MAs have generated considerable excitement among the research community, they have not translated into a significant number of real-world applications. One of the main reasons for this is the lack of work that tries to quantitatively compare the performance of MA implementations of specific applications with other implementations. Our project contributes in such a direction by presenting quantitative evaluation mechanism and results, for choice of a design paradigm.

Further there exist a long list of mobile agent frameworks and choice of an agent framework for an e-commerce based application itself need assessment of the existing popular frameworks. This project provides a qualitative and quantitative evaluation across three popular Java based mobile agent framework viz. Voyager, Aglets and Concordia, for an e-commerce application.

A complete e-commerce application is developed using both the traditional and mobile agent design paradigm as part of the project and the design, deployment and software engineering issues are discussed.

## *1.2 Thesis organization*

Chapter 2 introduces the mobile agent technology  and highlights on its strength and issues related to mobile agent technology. The chapter also compares mobile agent technology with the traditional models. Chapter 3 presents a study of three mobile agent framework viz., Voyager, Aglets and Concordia. The chapter details about the architecture and mobility features of these agent frameworks. Chapter 4 briefs about the related work and issues addressed in the domain of mobile agents for e-commerce. Chapter 5 introduces our quantitative evaluation of design paradigms. The chapter identifies underlying mobility patterns and applicable implementation strategies for e-commerce application. The chapter describes our experiments and presents our results. Chapter 6 provides a qualitative and quantitative comparison across Voyager, Aglets and Concordia. In Chapter 7 we present our prototype design and discuss implementation issues of business-to-customer e-commerce based application. We conclude our thesis with Chapter 8 which presents our view, experience and ideas on mobile agent technology.

# Chapter 2

## Mobile agent technology

### *2.1 Agents*

The precepts of agent technology exist in many of the applications we use today and take for granted. For example, the e-mail client is a type of agent. At your request, it goes about its business of collecting your unread e-mail from your mail server. Contemporary e-mail clients even presort your incoming messages into specified folders based on criteria you define. In this manner an agent is a software that becomes an extension of the user, performing tasks on the user's behalf.

The most promising among agents are mobile agents, which can themselves be intelligent or unintelligent. Unlike their static brethren, which are content to execute in the cozy confines of a single machine or address space, mobile agents have wheels. They migrate about the network, executing tasks at each way station, potentially interacting with other agents that cross their paths. An example of a mail delivery system can be considered to contrast a mobile and static agent. A static mail agent is that in which a POP client communicates with an SMTP server and collects mail. Both players in the transaction stay on their respective machines, using the network only for the transfer of message content. A mobile agent design of the same transaction might define a mail carrier agent that travels about the network autonomously, handing messages to mail handler agents at each stop.

### *2.2 Mobile agents*

A mobile agent is a program, which represents a user in a computer network, and is capable of migrating autonomously from node to node, to perform some computation on behalf of the user. Mobile agents are defined as objects that have behavior, state, and location [5]. Its tasks are determined by the agent application, and can range from online shopping to real-time device control to distributed scientific computing. Applications can inject mobile agents into a network, allowing them to roam the network either on a predetermined path, or one that the agents themselves determine based on dynamically gathered information. Having accomplished their goals, the agents may return to their ``home site" in order to report their results to the user.

A subset of behaviors of every agent is inherited from the model, notably those behaviors that define the means by which agents move from place to place. Finally, a mobile agent model is not complete without defining a set of events that are of interest to the agent during its lifetime. The set of events varies a bit from model to model, but the following is a list of the most common ones:

- *Creation* - Analogous to the constructor of an object. A handler for this event should initialize state and prepare the agent for further instructions.
- *Disposal* - Analogous to the destructor of an object. A handler for this event should free whatever resources the agent is using.
- *Dispatch* - Signals the agent to prepare for departure to a new location. This event can be generated explicitly by the agent itself upon requesting to migrate, or it can be triggered by another agent that has asked this agent to move.
- *Arrival* - Signals the agent that it has successfully arrived at its new location and that it should commence performing its duties.
- *Communication* - Notifies the agent to handle messages incoming from other agents and is the primary means of inter-agent correspondence.

## 2.3 Mobile agents and the traditional model

Mobile agent provides a new design model for applications as compared to the traditional client server model. First and foremost, the mobile agent shatters the very notion of client and server. With mobile agents, the flow of control actually moves across the network, instead of using the request/response architecture of client-server. In effect, every node is a server in the agent network, and the agent moves to the location where it may find the services it needs to run at each point in its execution [9]. The scaling of servers and connections then becomes a straightforward capacity issue, without the complicated exponential scaling required between multiple servers. The problem of robust networks is greatly diminished, for several reasons. The hold time for connections is reduced to only the time required to move the agent in or out of the machine. Because the agent carries its own credentials, the connection is simply a conduit, not tied to user authentication or spoofing. Last and most important, no application-level protocol is created by the use of agents. Therefore, compatibility is provided for any agent-based application. Complete upward compatibility becomes the norm rather than a problem to be tackled, and upgrading or reconfiguring an application may be done without regard to client deployment. Servers can be upgraded, services moved, load balancing interposed, security policy enforced, without interruptions or revisions to the network and clients.

## 2.4 Advantages of using mobile agents

Some of the benefits of mobile agents are:

- *Reduction in network traffic:* MA's code is very often smaller than data that it processes, so the transfer of mobile agents to the sources of data creates less traffic than transferring the data.
- *Asynchronous autonomous interaction:* Mobile agents can be delegated to perform certain tasks even if the delegating entity does not remain active. This makes it an attractive for mobile application and disconnected operations.

- *Interaction with real-time systems:* Installing a mobile agent close to a real-time system may prevent delays caused by network congestion.
- *Efficiency savings:* CPU consumption is limited, because a mobile agent execute only on one node at a time. Other nodes do not run an agent until needed.
- *Space savings:* Resource consumption is limited, because a mobile agent resides only on one node at a time. In contrast, static multiple servers require duplication of functionality at every location. Mobile agents carry the functionality with them, so it does not have to be duplicated.
- *Support for heterogeneous environments:* Mobile agents are separated from the hosts by the mobility framework. If the framework is in place, agents can target any system. The costs of running a Java Virtual Machine (JVM) on a device are decreasing. Java chips will probably dominate in the future, but the underlying technology is also evolving in the direction of ever-smaller footprints (e.g. Jini).
- *Online extensibility of services:* Mobile agents can be used to extend capabilities of applications, for example, providing services. This allows for building systems that are extremely flexible
- *Convenient development paradigm:* Creating distributed systems based on mobile agents is relatively easy. The difficult part is the mobility framework, but when it is in place, then creating applications is facilitated.
- *Easy software upgrades:* A mobile agent can be exchanged virtually at will. In contrast, swapping functionality of servers is complicated; especially, if we want to maintain the appropriate level of quality of service (QoS).

## *2.5 Existing mobile agent systems*

With the introduction of Java to the Internet world, many mobile agent projects have made use of this operating system independent language. Another benefit to using Java is that each of these systems can make use of the standards that are inherent in Java such as the Java virtual machine and object serialization mechanism . Some of these systems are listed below :

- *Aglets*, IBM's mobile agent system. The word Aglet is formed through the combination the words agent and applet, as the intention of this system is to bring mobility to Java applets [31].

- *Odyssey*, from General Magic Inc. was the first mobile agent system. It was reworked using Java and now provides a set of Java classes that developers can make use of to create their own mobile agent applications[5].

- *Concordia*, Mitsubishi's agent system which provides developers with a framework for the development and the management of mobile agent applications[35]. These applications can be extended to any system supporting Java.

- *Voyager*, an agent based system that supports both traditional and agent-based distributed computing techniques created by ObjectSpace. Voyager supports object request brokering so developers can create distributed application using both traditional messaging, such as CORBA or RMI, as well as agent-enhanced techniques [7].

## 2.6 New trends in Internet applications

There are many trends in Internet technology and activity that encourage the use of mobile agents on the Internet. These trends are outlined [15] and are briefly described below:

- *Bandwidth* : Internet access is broadening to the point where people will have a reasonable-speed access to the Internet. The Internet backbone has an enormous amount of bandwidth available, however the average user will not have this at his disposal.

- *Mobile devices* : Internet users are mobile and therefore they need their Internet access to come with them by using portable computing devices. Everything from laptops or palmtops to car telephones to pagers can access the Internet. These devices usually connect using a telephone or wireless network.

- *Mobile users* : Internet users have shown that they like to have access to everything from anywhere through the popularity of things like web-mail. Web terminals are becoming more and more popular, Internet cafes are the latest in public place Internet access.

- *Intranets* : Internal or private and smaller versions of the Internet are being used for information sharing within companies and corporations. Intranets are usually managed by a single organization and can make use of new technologies quickly since security within the intranet is of less concern.

- *Information overload* : The massive amount of information available on the Internet today is immeasurable. Users are easily overwhelmed by the sheer quantity of data that is at their disposal. Filtering technology, while still quite limited, can help reduce the stream of information to a given user to a tolerable level.

- *Customization* : Site customization for individual users is possible through the Internet and can be provided on either the client or server side.

- *Proxies* : Third party proxies can provide site wide customization for one or more Internet services. They can be used to reduce information overload and customize service access.

# Chapter 3

## Agent frameworks

This chapter describes the architectural design of three Java based mobile agent frameworks viz. Aglets, Concordia and Voyager. The architectural design and mobility features of these frameworks are outlined in this chapter. We believe that today Java is the language for mobile agent frameworks and so we discuss the details of these three popular Java based mobile agent frameworks.

### *3.1 Java as technology base for mobile agents*

In the current trend towards heterogeneous networks, which are composed by several different platforms, the mobility of binary code is problem hardly to overcome. An answer can be found in Java programming language, which combines the object-oriented programming style with the use of intermediate format called bytecode, which can be executed on each platform that hosts a Java virtual machine (JVM).

Java is strongly network oriented and provides some support for mobility of code from the dynamic class loading to the definition of applets. Java implements a form of weak mobility, by serializing objects and sending them to another JVM. The serialization mechanism permits to maintain the values of the instance variables, but it cannot keep track of the execution flow. Some of the properties of Java that make it a good language for mobile agent programming are :

- **Platform-independence :** Java is designed to operate in heterogeneous networks. To enable a Java application to execute anywhere on the network, the compiler generates architecture-neutral byte code, as opposed to non-portable native code.

- **Object serialization :** A key feature of mobile agents is that they can be serialized and de-serialized. Java conveniently provides a built-in serialization mechanism that can represent the state of an object in a serialized form sufficiently detailed for the object to be reconstructed later. The serialized form of the object must be able to identify the Java class from which the object's state was saved, and to restore the state in a new instance.

- **Reflection:** Java code can discover information about the fields, methods, and constructors of loaded classes, and can use reflected fields, methods, and constructors operate on their underlying counterparts in objects, all within the security restrictions.

- **Multithread programming :** Agents are by definition autonomous and could be implemented as individual threads.
- **Security manager :** It defines which resources a Java program is allowed to access The Java virtual machine also contains a bytecode verifier that does static checks to prevent forbidden code sequences from being loaded, thereby ensuring the unreachability of the sandbox surrounding the incoming code.

The following section describes in detail the architectural design and mobility features of Aglets, Concordia and Voyager.

## *3.2 Aglets*

Aglets was developed by IBM Tokyo Research Laboratory and is now open source. An Aglet is a composite Java object that includes mobility and persistence and its own thread of execution. Aglets uses a call-back model based on the Java event delegation model. Various action and mobility interfaces are supported by Aglets framework which determine what to do when a specific event happens.

An Aglet interacts with its environment through an AgletContext object. Aglets are always executed in AgletContexts. To interact with each other, Aglets go through AgletProxy objects. An AgletProxy object acts as an interface of an Aglet and provides a common way of accessing the Aglet behind it. In a way, an AgletProxy object becomes the shield that protects an agent from malicious agents. Figure 3.1 show the Aglet interaction model.

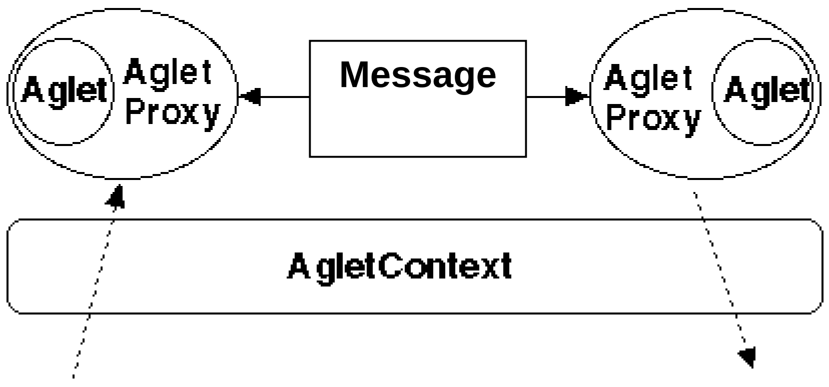

**Fig 3.1 : Aglet interaction model**

### 3.2.1 Architecture overview

The Aglets architecture consists of two layers, and two APIs that define interfaces for accessing their functions[32][33] viz., runtime layer and the communication layer. Figure 3.2 show the Aglet architecture with the two layers and sub-components.

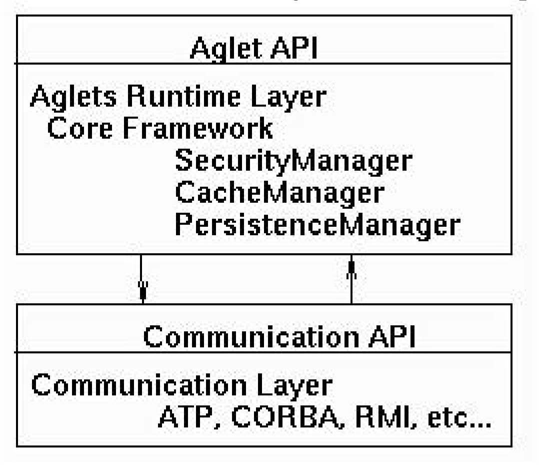

**Fig 3.2: Aglet architecture**

### 3.2.2 The Aglets runtime layer

The Aglets runtime layer implements Aglets interfaces such as AgletContext and AgletProxy it also consists of a core framework and subcomponents. The core framework provides mechanisms fundamental to Aglet execution i.e., 1) Serialization and de-serialization of Aglets 2) Class loading and transfer 3) Reference management and garbage collection.

The subcomponents are designed to be extensible and customizable because these services may vary depending on requirements or environments.

- **PersistenceManager**
  The PersistenceManager is responsible for storing the serialized agent, consisting of the Aglet's code and state into a persistent medium such as a hard disk. Persistence manager do not keep a copy of agent before dispatching and hence the system is susceptible to loss of agent over broken network.

- **CacheManager**
  The CacheManager is responsible for maintaining the bytecode used by the Aglet and its transfer when an Aglet moves, the CacheManager caches all bytecode even after the corresponding class has been defined.

- **SecurityManager**
  The SecurityManager is responsible for protecting hosts and Aglets from

malicious entities. A very preliminary form of security is supported by Aglets framework.

### 3.2.3 The communication layer

The Aglets runtime itself has no communication mechanism for transferring the serialized data of an Aglet to destinations. Instead, the Aglets runtime uses the communication API that abstracts the communication between agent systems [33]. This API defines methods for creating and transferring agents, tracking agents, and managing agents in an agent-system and protocol-independent way.

The current Aglets uses the Agent Transfer Protocol (ATP) as the default implementation of the communication layer. ATP is modeled on the HTTP protocol, and is an application-level protocol for transmission of mobile agents. To enable remote communication between agents, ATP also supports message-passing. Aglets uses ATP for agent transfer and RMI for message exchange.

### 3.2.4 The communication layer architecture

The following figure shows the architecture of the communication layer.

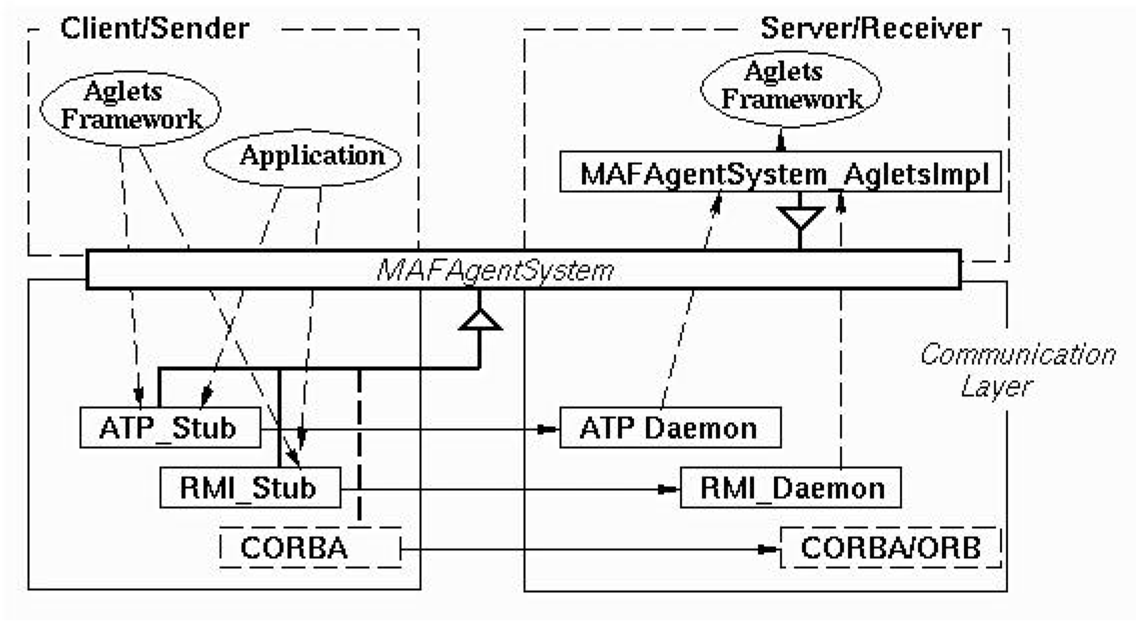

**Fig 3.3 : Architecture of communication layer**

An application or client uses a stub object to send a request to the destination. An agent system must have a stub class for each protocol it supports. Applications or clients can then get and use a stub object for a given protocol. An agent system's responsibility to instantiate and manage stub objects. Aglets supports two protocols, ATP and RMI.

On the other hand, an aglet server has an implementation of MAFAgentSystem that actually handles the requests. It is the agent-system provider's responsibility to provide the implementation of MAFAgentSystem.. Furthermore, a server has one or more daemons to accept requests from a sender. A server may support multiple protocols by having multiple daemons to handle each protocol. When a daemon accepts requests, it then forward these requests to the MAFAgentSytem_AgletsImpl.

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

### 3.2.5 Agent Transfer Protocol

ATP is a simple application-level protocol designed to transmit an agent in an agent-system-independent manner. An ATP request consists of a request line, header fields, and a content. The request line specifies the method of the request, while the header fields contain the parameters of the request. ATP defines the following four standard request methods:

- **Dispatch**
  The dispatch method requests a destination agent system to reconstruct an agent from the content of a request and to start executing the agent. If the request is successful, the sender must terminate the agent and release any resources consumed by it.

- **Retract**
  The retract method requests a destination agent system to send a specified agent back to the sender. The receiver is responsible for reconstructing and resuming the agent. If the agent is successfully transferred, the receiver must terminate the agent and release any resources consumed by it.

- **Fetch**
  The fetch method is similar to the GET method in HTTP; it requests a receiver to retrieve and send any identified information (normally class files).

- **Message**
  The message method is used to pass a message to an agent identified by a agent-id and to return a reply value in the response. Although the protocol adopts a

request/reply form, it does not lay down any rules for a scheme of communication between agents.

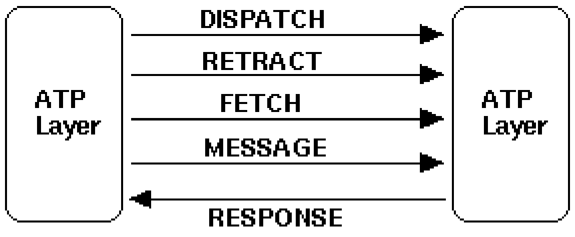

**Fig 3.4 :Agent transfer protocol**

Unlike normal Java objects, which are automatically released by garbage collector, an Aglet object, since it is active, can decide whether or not to die. Aglet programmers are responsible for releasing allocated resources such as file descriptors or DB connections, because these may not be released automatically.

## *3.3 Concordia*

Concordia is a framework for mobile agent system developed and supported by Mitsubishi Electric Information Technology Center, USA. Concordia is a complete Java based framework for network-efficient mobile agent applications which extend to any device supporting Java.

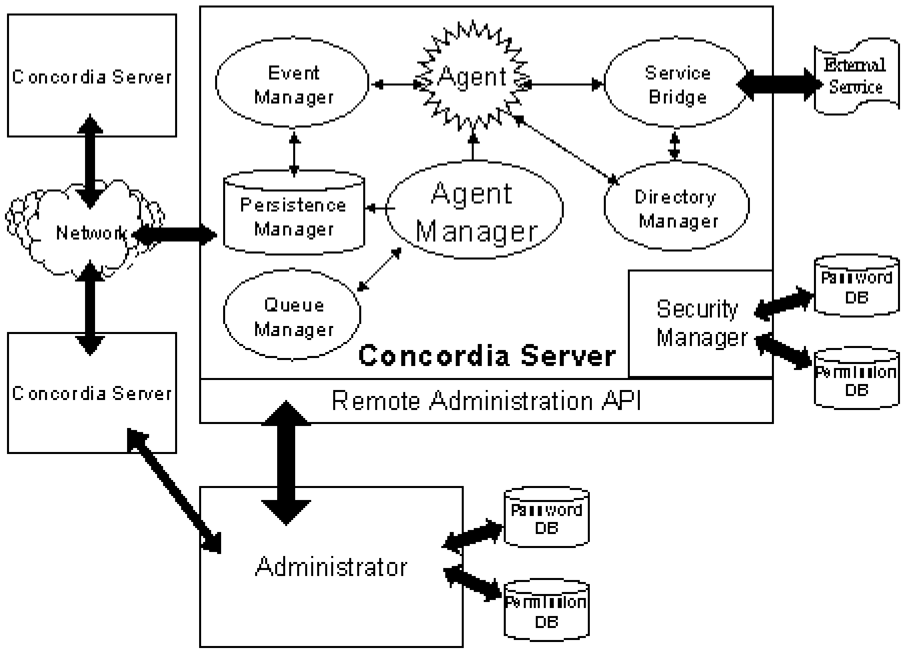

**Fig 3.5 Concordia's architecture**

### 3.3.1 Architectural overview

A Concordia system, at its simplest, is made up of a Java virtual machine sitting on any machine, a Concordia Server, and at least one agent. The Concordia server and agent are Java program, the Concordia Server manages the agent, including its code, data, and movement.

For an agent to move from one server to another the Concordia server inspects the Itinerary object, created and owned by each agent. That destination is contacted and the agent's image is transferred, where it is again stored persistently before being acknowledged. In this way the agent is given a reliable guarantee of transfer.

After being transferred, the agent is queued for execution on the receiving node. Its security credentials are transferred with it automatically and its access to services is under local administrative control at all times. In all cases, the Concordia agent is autonomous and self-determining in its operation.

### 3.3.2 Architectural components

The Concordia system is made up of numerous components, each of which integrates together to create the full mobile agent framework [35]. The Concordia Server is the major building block, inside which the various Concordia Managers reside. Each Concordia component is responsible for a portion of the overall Concordia design, in a modular and extensible fashion.

- **Agent Manager :** The Agent Manager provides the communications infrastructure using the TCP/IP stack for agent transmission. It abstracts the network interface in order that agent programmers need not know any network specifics nor need to program any network interfaces. The Agent Manager Server also manages the life cycle of the agent, providing it with agent creation, destruction, and provides an environment in which the agents execute.

- **Administrator :** The Administration Manager manages services (Agent Managers, Security Managers, Event Managers, etc) and supports remote administration from a central location, so only one Administration Manager is required in the Concordia network.

- **Security Manager :** The Security Manager is responsible for identifying users, authenticating their agents, protecting server resources and ensuring the security and integrity of agents, authorizing the use of dynamically loaded Java classes and their accumulated data objects as the agent moves among systems.

- **Persistence Manager :** The Persistence Manager is completely transparent and maintains the state of agents in transit around the network. As a side benefit, it allows for the checkpoint and restart of agents in the event of system failure.

- **Event Manager :** The Event Manager handles the registration, posting and notification of events to and from agents. The Event Manager can pass event notification to agents on any node in the Concordia network thus supporting agent collaboration.

- **Queue Manager :** The Queue Manager is responsible for the scheduling and possibly retrying the movement of agents between Concordia systems which include maintenance of agent and persistence of agent state.

- **Directory Manager :** The Directory Manager provides naming service in the Concordia network. The Directory Manager may consult a local name service or may be set up to pass requests to other, existing name servers.

- **Service Bridge :** The Service Bridge provides the interface from Concordia agents to the services available at the various machines in the Concordia network. It comprises a set of programming extensions to provide access the native API's as well as interfacing these to the Directory Manager and Security Manager.

- **Agent Tools Library :** The ATL is a library which provides all the classes needed to develop Concordia mobile agents.

The Concordia mobile agent framework pays much attention on security and reliability. Role-based access control is realized to protect resources and mobile agents. So unauthorized mobile agents can not access resources and unauthorized users can not inspect agent's contents. Concordia also utilize symmetric and public key cryptography to protect agents during their transmission as well as when being stored on disk. Concordia server can authenticate each other by exchanging digital certificates. It uses two-phase commit protocol to transmit agents from one node to another. In case of server or network failures, agents can be recovered through using persistence manager. Concordia also has an agent debugger which helps tracking the progress of an agent throughout the network.

## *3.4 Voyager*

Object Space's Voyager is a full-featured Java ORB architecture designed to support the development of powerful distributed computing systems. The product uses the Java virtual machine to load classes at runtime to create mobile objects and autonomous agents. Voyager provides a complete and seamless distributed computing framework, that supports remote invocation, remote pass by value, distributed events, naming services, object mobility, autonomous agents, runtime object extensions, object activation, distributed security, distributed garbage collection, distributed timers, advanced messaging, multicasting, and replaceable networking protocols. Voyager ORB is a high-performance object request broker that supports CORBA, RMI, DCOM. Its innovative dynamic proxy generation removes the need for stub generators. Voyager ORB includes a universal naming service, DCOM support, activation framework, publish/subscribe and mobile agent technology[7] [29].

### 3.4.1 Architectural overview

The Voyager framework provides all of its features with only a few abstractions, a very small API, and no need for interface definition languages (IDLs) or management of stubs or proxies. Voyager works from interfaces defined in pure Java and automatically generates and distributes whatever stubs or proxies it needs at runtime.

Voyager uses introspection to discover the features of whatever class users want to make distributable (at runtime). It then generates any required wrapper classes on-the-fly by writing the bytecode to memory and registering the just-created classes with the JVM. Voyager lets users deploy code to a single location by leveraging Java's class-loading mechanism; it supports flexible and custom security by embracing standard Java security; it supports pass-by-value and object mobility by leveraging Java serialization.

### 3.4.2 Universal architecture

Voyager offers a universal architecture that isolates user code from the intricacies of communications and messaging protocols. Figure 3.6 depicts Voyager's universal architecture.

In a Voyager system message calls made to a proxy are forwarded to its object. If the object is in a remote program, the arguments are serialized using the standard Java serialization mechanism and de-serialized at the destination. The morphology of the arguments is maintained. By default, parameters are passed by value. However, if an object's class implements *com.objectspace.voyager.IRemote* or *java.rmi.Remote*, the object is passed by reference instead. An appropriate proxy class will be generated dynamically if needed. Voyager also provides oneway, sync, and future messages.

Multicast and Publish-subscribe features are provided by Voyager ORB, a Java message can be multicast to a distributed group of objects without requiring the sender or receiver to be modified in any way. The publish-subscribe facility supports server-side filtering and wildcard matching of topics.

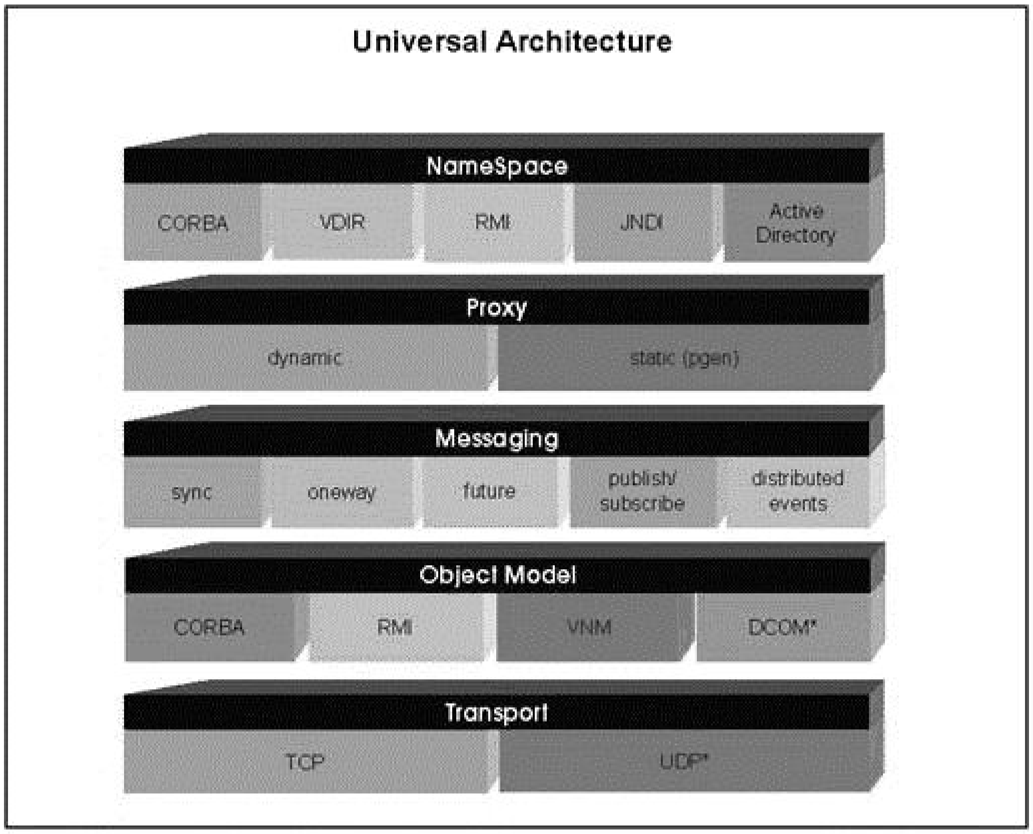

**Fig 3.6 : Universal architecture of Voyager**

### 3.4.3 Features supported

The Voyager ORB simplifies and unifies access to the most common industry standards. There are several aspects of Voyager that are universal:

- **Communications :** The universal communications architecture allows Voyager programs to be both a universal client and a universal server by supporting simultaneous bi-directional communication with other CORBA, RMI, and DCOM programs.
- **Messaging :** The universal messaging layer allows different types of messages such as synchronous, oneway, and futures to be sent to an object regardless of its location or object model.
- **Naming :** The universal naming service allows access to the many commercially available naming services through a single API.

- **Directory :** The universal directory is a single directory that can be accessed and shared by all clients, for example, an RMI server can bind an object into a universal directory using the native RMI registry API and a CORBA client can lookup up the same object using the CORBA naming service API.

- **Gateway :** The universal gateway allows Voyager to automatically bridge protocols between clients and servers that are not written using Voyager.

- **Dynamic Aggregation :** Facility to attach secondary objects, or *facets*, to a primary object at runtime. Classes don't have to be modified in any way for instances to act as facets or to be extended by them.
- **Remote Invocation :** In object-oriented programming, remote invocation refers to the ability to invoke methods on remote objects (objects that exist in a different machine) as if they were local. Remote invocation is  implemented through proxies, which provide a facade that hides the mess of the underlying stub generations.
- **Activation :** The activation framework allows objects to be persisted to any kind of database and automatically re-activated in the case that the program is restarted.
- **Security :** An enhanced security manager is included, as well as hooks for installing custom sockets such as SSL.

# Chapter 4

## Related work

The growth of WWW and the emergence of e-commerce applications has resulted in new net-centric business models. Support for on-line commerce has created a need for new ways of structuring applications to provide cost-effective and scalable models. The most promising among all being, the use of agents for e-commerce. A lot of work use the notion of agent in e-commerce applications. But most of the work involve agents in the sense as a piece of  code that acts on user's behalf. Not much work has been done in the direction of incorporating mobile agents for the design of e-commerce applications. A clear distinction need to be maintained between agent and mobile agents in e-commerce.

### *4.1 Software agents in e-commerce*

Some of the work which deals with agents in e-commerce are :

- MIT Media Lab's Kasbah [18] is an online World Wide Web marketplace for buying and selling goods. A user creates an agent, provides it with a set of criteria, and dispatches it into the marketplace. The criteria include price, time constraints and quantity of merchandise desired. Users can select one of several price decay functions for goods they are attempting to sell. Seller agents post their offers on a common blackboard and wait for interested buyer agents to establish contact. The buyer agent filters the available deals according to the user's specifications, and then proceeds to negotiate a deal on its owner's behalf. The system also provides some measure of post-purchase evaluation that can be used to develop trust based on reputation and repeat business. Kasbah supports continued operation despite user disconnection and requires minimal user intervention during negotiations.

- The Minnesota AGent Marketplace Architecture (MAGMA) [25] is a prototype for a virtual marketplace system targeted towards items that can be transferred over the Internet (such as information). It consists of Java-based trader agents (buyer and seller agents), an advertising server that provides a classified advertisement service, and a bank that provides financing support and payment options. Independent agents can register with a relay server that maintains unique identifiers for the agents and routes all inter-agent messages.

- AuctionBot [16, 24] is a generic auction server that allows suppliers to auction products by specifying the acceptable parameters for the sale. Buyer and seller agents can conduct negotiations with AuctionBot, enforcing the user-specified parameters to control the bidding. AuctionBot provides an API that allows a user to create agents that are primed with customized bidding strategies, but this requires coding knowledge on the user's behalf. AuctionBot is basically an information service; it does not enforce the exchange of goods or provide post-transaction services.

- Others similar work include BargainFinder [17] which is uses a database search agent that queries several online music stores for the best deal on CDs and cassettes. Jango [19, 23] is similar to BargainFinder, but allows comparison-shopping based on price. Firefly [21], on the other hand, is a mail-order system that provides automated collaborative filtering.

It is important to note that none of the above use mobile agents as the design paradigm. These implementation however highlight the strengths of convergence of the two fields i.e. agents and e-commerce. The above applications provide the motivation for use of mobile agents in e-commerce to gain performance edge over other implementations.

## *4.2 Mobile agents in e-commerce*

A few of research effort which uses MA for e-commerce applications are :

- Mobile Agents for Networked Electronic Trading (MAgNET) is a system for networked electronic trading, that is based Aglets. Architecture of MAgNET is designed for supply chain management where mobile agents deal with procurement of the many components needed to manufacture a complex product. In the MAgNET system, the buyer site maintains a list of potential suppliers along with their lists of products. A buyer who is interested in acquiring a product creates a MA, specifies criteria for the acquisition of the product, and dispatches the MA to the potential suppliers. The MA visits each supplier site, searches the product catalogs according to the buyer's criteria, and returns to the buyer with the best deal it finds. The buyer either confirms the deal and proceeds with the monetary transaction, or aborts the query and disposes the agent.

- A system implementing another model of e-commerce using MA is a supplier driven marketplace [3]. This approach is particularly attractive for products with a short shelf-life. A supplier creates and dispatches an MA to potential buyers by giving it a list of sites to visit. The MA carries with it information about available stock and price of the product. Since the MA moves to the destination, the network and processing latencies that contribute to delays in servicing orders may be reduced.

### *4.3 Summary*

Our project provides a framework for e-commerce application development using mobile agents and is of its kind providing a methodical design. Our project quantitatively evaluates design paradigm for e-commerce applications. We identify mobility patterns and implementation strategies for e-commerce application. Based on the mobility patterns of e-commerce applications we evaluate three Java based MA frameworks viz. Aglets, Concordia and Voyager. A complete B2C e-commerce application is then implemented and software engineering issues discussed.

While there exist some qualitative studies of MAs in e-commerce [2], [6], [13] and some quantitative studies of MAs in other application domains [8], [12], to the best of our knowledge, there is no literature on the quantitative study of MA applications in the domain of e-commerce. Also, no effort has geared towards identifying mobility patterns for e-commerce applications and evaluating mobile agent frameworks based on mobility patterns.

# Chapter 5

## Quantitative evaluation of Voyager for e-commerce applications

The emergence of e-commerce applications has resulted in new net-centric business models. This has created a need for new ways of structuring applications to provide cost-effective and scalable models. One of the main reasons of MA not being used in large number of applications is the lack of work that tries to quantitatively compare the performance of MA implementations of specific applications with other implementations [9].

In this chapter, we first classify existing MA applications in the domain of e-commerce and identify patterns of mobility that underly these applications. We discuss several strategies for implementing each of these patterns using the traditional Client-Server (CS) and the MA paradigms. We also identify various application parameters that influence performance, such as, size of CS messages, size of MA, number of remote information sources, etc., and study their effect on application performance. We have chosen an application representative of the domain of e-commerce and have implemented it using both CS and MA paradigms. These implementations consist of CS applications using Java and MA applications using the Voyager framework [7]. We present our observations and conclusions regarding the choice of appropriate implementation strategies for different application characteristics.

## *5.1 Mobile agents in e-commerce*

The number of people buying, selling, and performing transactions on the Internet is expected to increase at a phenomenal rate. The application of MAs to e-commerce provides a new way to conduct business-to-business, business-to-consumer, and consumer-to-consumer transactions [10]. We classify existing MA applications in e-commerce into three categories, viz., shopping agents, salesman agents, and auction agents.

### 5.1.1 Shopping agents

These MAs make purchases in e-marketplaces on behalf of their owner according to user-defined specifications. This model of e-commerce uses a customer-driven market place. A typical shopping agent may compare features of different products by visiting several online stores and report the best choice to its owner. The MA carries the set of features to be considered and their ideal values as specified by its owner. It is given one or more sites to visit and may dynamically visit other sites based on subsequent information. Since the MA moves to the source of information, the overhead of repeatedly transferring potentially large amounts of information over a network is eliminated. One example of a system that implements shopping agents is MAgNET, where agents deal with procurement of the many components needed to manufacture a complex product [11].

#### 5.1.2 Salesman agents

These MAs behave like a traveling salesman who visits customers to sell his wares. This model of e-commerce uses a supplier driven marketplace and is particularly attractive for products with a short shelf-life. A supplier creates and dispatches an MA to potential buyers by giving it a list of sites to visit. The MA carries with it information about available stock and price of the product. Since the MA moves to the destination, the network and processing latencies that contribute to delays in servicing orders may be reduced. A system implementing salesman agents is discussed in [10].

#### 5.1.3 Auction agents

These MAs can bid for and sell items in an online auction on behalf of their owners. Each MA carries along with it information about its owners bidding range, time within which the item is to be procured, bidding pattern, and other relevant attributes. In the presence of multiple auction houses, MAs can be used for collecting information across them. An agent can make a decision to migrate to one of them dynamically, depending on the amount of information transmitted, latency, etc. Some advantages of using MAs include allowing disconnected operation of auction agents, reducing network traffic, and facilitating quicker response during auction. One example of a system that implements mobile auction agents is Nomad [14].

From the above classification, it may be observed that mobility in MAs can be characterized by the set of destinations that an MA visits, and the order in which it visits them.

### *5.2 Mobility patterns*

We have identified the following parameters to characterize the mobility of an MA:

- **Itinerary :** the set of sites that an MA has to visit. This could either be statically fixed at the time of agent initialization, or dynamically determined by the MA.

- **Order :** the order in which an MA visits sites in its itinerary. This may also be determined statically or dynamically.

Based on these parameters, we distinguish MA applications in e-commerce as possessing one of the following mobility patterns:

### 5.2.1 Static Itinerary (SI)

The itinerary of the MA used in the application is known a priori and does not change. We further distinguish such applications based on order as:

#### 5.2.1.1 Static Itinerary Static Order (SISO)

The order in which an MA completes its itinerary is static and known a priori. An example application is an auction MA which is required to visit a set of auction houses in a specified order.

#### 5.2.1.2 Static Itinerary Dynamic Order (SIDO)

The order in which an MA completes its itinerary is decided dynamically by the MA. An example application is a shopping MA which finds the minimum price for a product from a set of on-line shops. The order in which the shops are visited may be irrelevant and could be dynamically determined by the MA.

### 5.2.2 Dynamic Itinerary (DI)

The itinerary of the MA used in the application is determined dynamically by the agent itself. However, at least the first site in the itinerary should be known a priori. An example application is a shopping MA that is required to find a particular product. A shop that does not contain the product may recommend an alternative shop, and this recommended shop is included in the MAs itinerary dynamically.

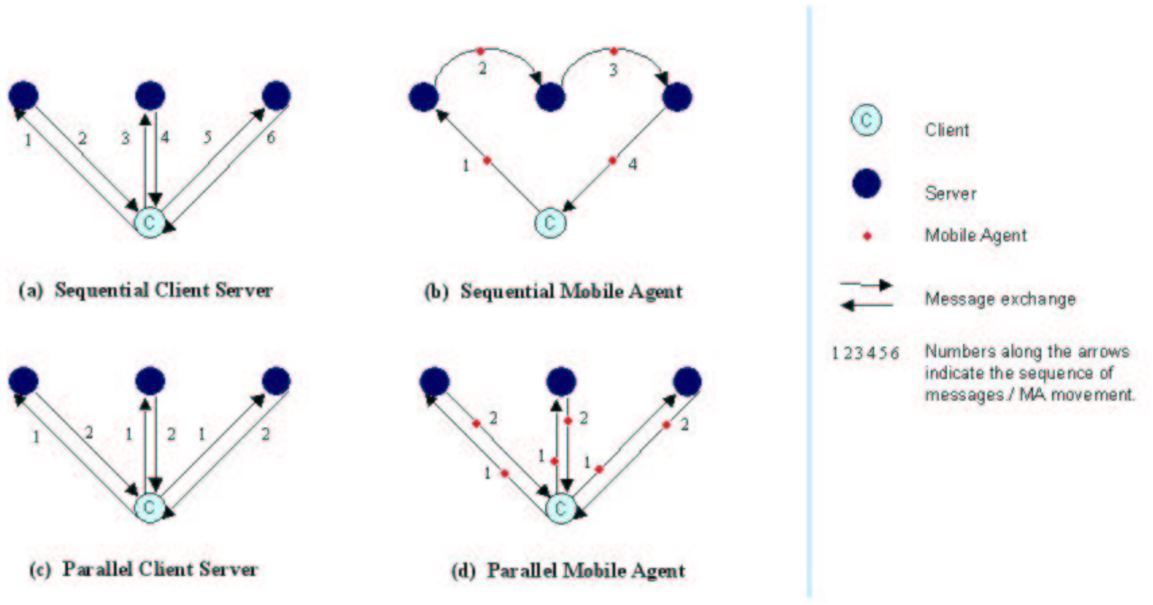
1
2
3
4
5
6
C
(a) Sequential Client Server
1
2
3
4
C
(b) Sequential Mobile Agent
1
2
1
2
1
2
C
(c) Parallel Client Server
1
2
1
2
1
2
C
(d) Parallel Mobile Agent
C
Client
Server
Mobile Agent
Message exchange
1 2 3 4 5 6
Numbers along the arrows indicate the sequence of messages / MA movement.

**Fig 5.1 : Implementation strategies**

It may be noted that dynamic itinerary implies dynamic order and the distinction between static order and dynamic order is not meaningful in this case.

## *5.3 Implementation issues*

### 5.3.1 Implementation strategies

We have identified four implementation strategies that may be adopted by e-commerce applications:

- **Sequential CS**
  This is based on the traditional client-server paradigm. The client makes a request to the first server and after processing the reply, makes a request to the second server and so on, till the list of servers to be visited is exhausted. This strategy is illustrated in figure 5.1(a).

- **Sequential MA**
  In this case a single MA moves from its source of origin (client) to the first site (server) in its itinerary. It then moves to the next site and so on, till it has visited all the sites in its itinerary. This strategy is illustrated in figure 5.1(b).

- **Parallel CS**
  This also based on the client-server paradigm. However, instead of sequential requests, the client initiates  parallel threads of execution where each thread concurrently makes a request to one of the servers and processes the reply. This strategy is illustrated in figure 5.1(c).

- **Parallel MA**
  In this case the client initiates multiple MAs, each of which visits a subset of the servers in the itinerary. The MAs then return to the client and collate their results to complete the task. This strategy is illustrated in figure 5.1(d).

It is also possible to use combinations of the above strategies. In our experiments, we restrict ourselves to these four strategies only.

### 5.3.2 Implementation for different mobility patterns

The feasible implementation strategies for different mobility patterns identified in section 5.2 are as follows:

- **SISO :** Since the order of visit is fixed statically, the possible implementation strategies in this case are:
  - Sequential CS
  - Sequential MA

Parallel CS and parallel MA strategies cannot be used for SISO applications since the order in which the MA visits servers may be important to the application being implemented.

- **SIDO :** Since the order of visit is determined dynamically, all the strategies outlined in section 5.3.1 are possible, namely :
    - Sequential CS
    - Sequential MA
    - Parallel CS
    - Parallel MA

- **DI :** Since the itinerary is determined dynamically, the possible implementation strategies in this case are :
    - Sequential CS
    - Sequential MA

  Parallel CS and parallel MA strategies cannot be used for DI applications since information about the servers to be visited is not known a priori.

The selection of the "ideal" implementation strategy from those feasible for a given application could be based on several criteria such as ease of implementation, performance, availability of technology, etc. The following section describes the details of implementing an application using different strategies.

## *5.4 Experimentation and results*

### 5.4.1 Experimentation

We have chosen a typical e-commerce application, viz., that of a single client searching for information about a particular product from the catalogs of several on-line stores. We assume that the client requires a highly customized search which the on-line store does not support. This would require the client to fetch a relevant subset of the catalog and implement a search at its end. We have implemented such an application using all four strategies mentioned in section 5. 3.1.

We have used the Voyager Framework for MA implementations. The CS implementation consists of a server that sends a catalog on request and a multi-threaded client that requests a catalog from one or more servers. The client and the server have been implemented in Java.

The experiments were carried out on P-III, 450 MHz workstations connected through a 10 Mbps LAN with typical student load. We have considered the following parameters for comparing the performance of these implementations:

- number of stores (varies from 1 to 26);
- size of catalog (varies from 20 KB to 1 MB);

- size of client-server messages (varies proportionately with catalog size);
- size of an MA (fixed at 4.6 KB);
- processing time for servicing each request (varies from 10 ms to 1000 ms);
- network latencies on different links (assumed constant since all workstations were on the same LAN);

Our performance metric is the user turnaround time, which is the time elapsed between a user initiating a request and receiving the results. This includes the time taken for agent creation, time taken to visit/collect catalogs and the processing time to extract the required information.

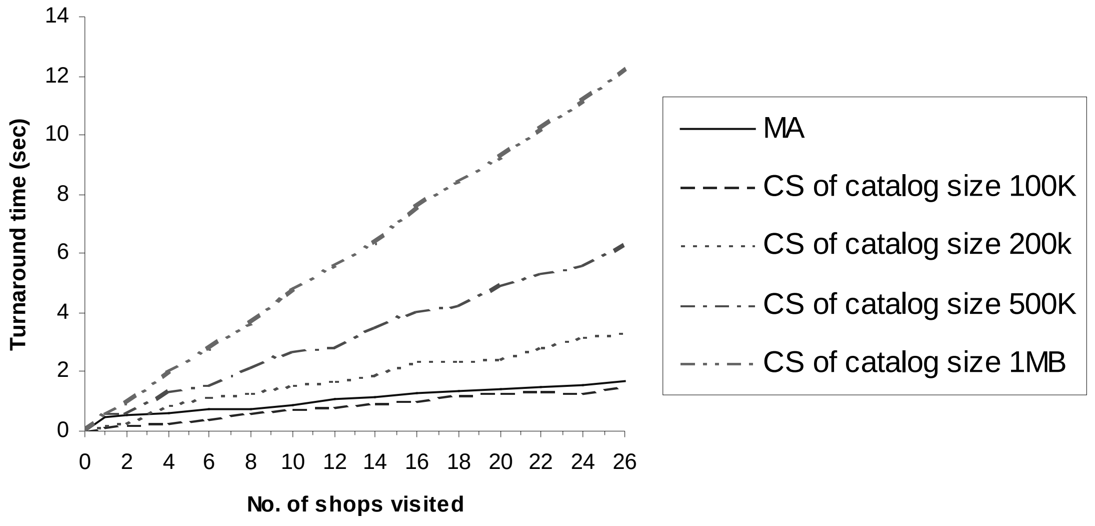

**Fig 5.2 : Effect of catalog size on turnaround time for sequential MA & sequential CS**

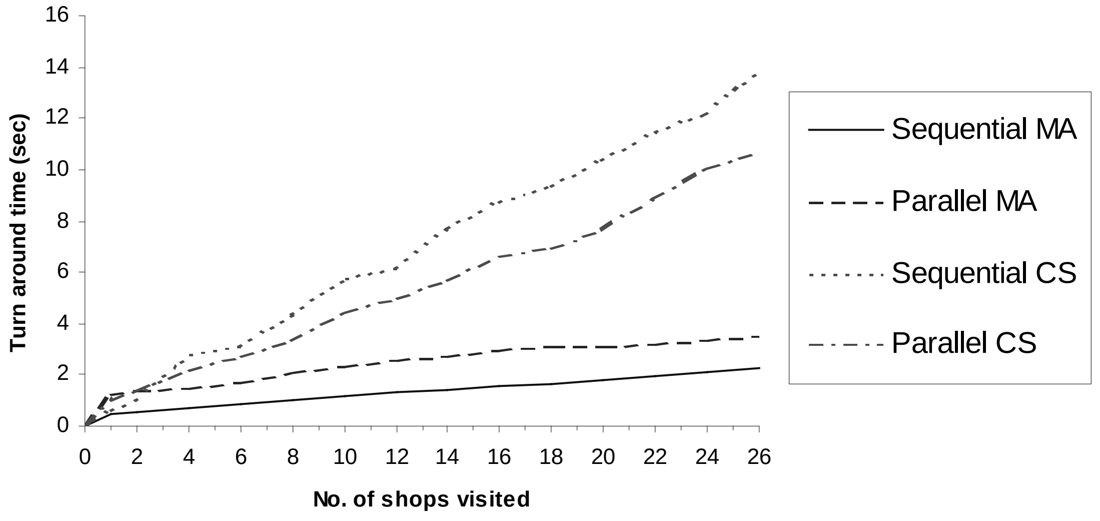

**Fig 5.3 : Turnaround time for different implementation strategies for a processing time of 20 ms (catalog size of 1 MB).**

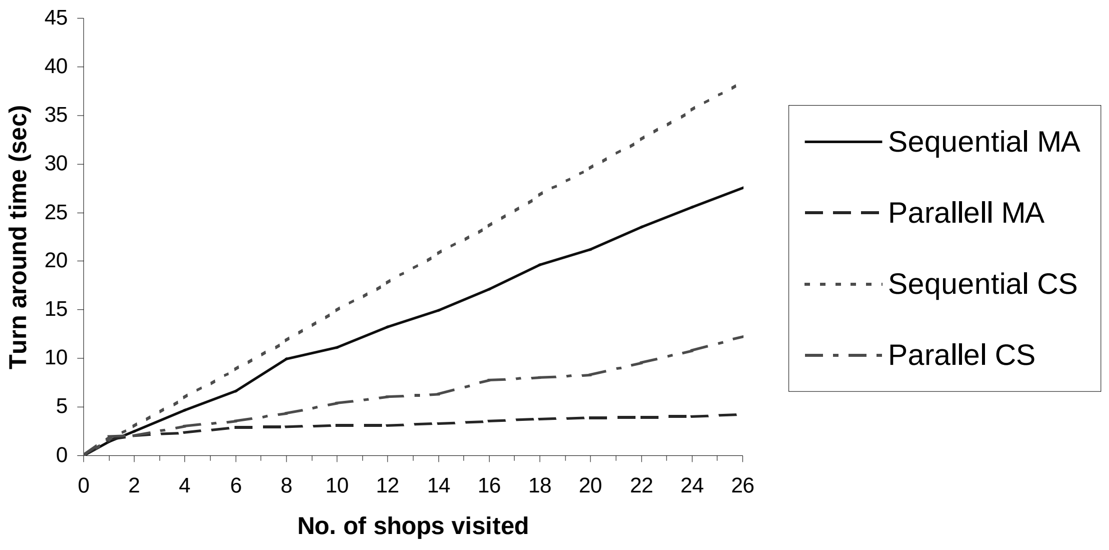

**Fig 5.4 : Turnaround time for different implementation strategies for a processing time of 1000 ms (catalog size of 1 MB).**

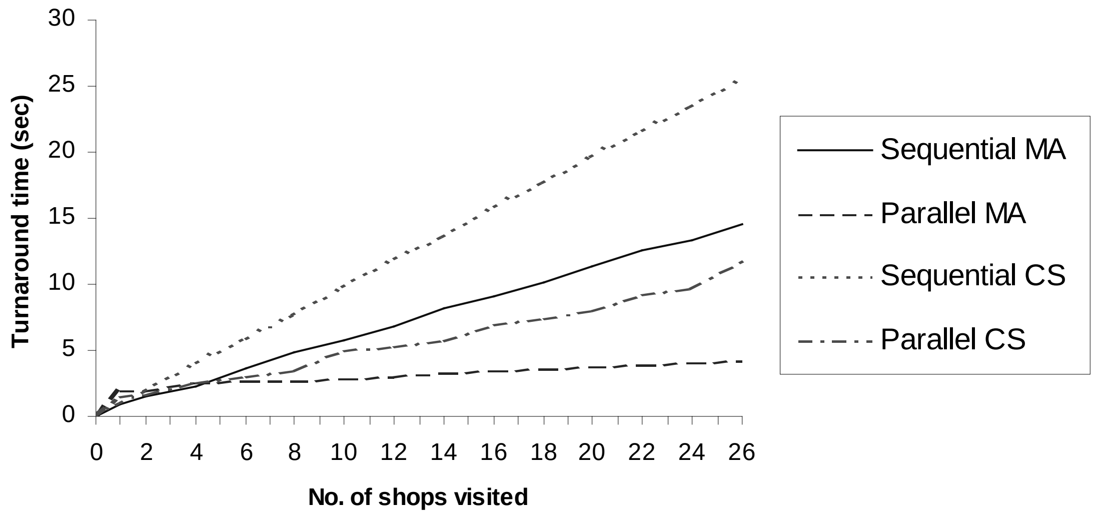

**Fig 5.5 : Turnaround time for different implementation strategies for a processing time of 500 ms (catalog size of 1 MB).**

### 5.4.2 Results

The results of our experiments are shown in the graphs of figures 5.2, 5.3, 5.4, and 5.5. Some of our observations are:

- The performance of sequential MA remains the same for different catalog sizes while performance of sequential CS degrades with increase in catalog size (Fig. 5.2).

- Sequential CS implementation in our case performs better than sequential MA for a catalog size less than 100 KB (Fig. 5.2).
- With a catalog size of 200 KB, sequential MA starts to perform better than sequential CS when the number of shops to visit is 4 (Fig. 5.2).
- For small processing delays (20 ms), sequential MA performs better than all other strategies (Fig. 5.3).
- For higher processing delays (1000 ms), parallel implementations show better performance than sequential implementations. Also, parallel MA performs better than parallel CS in this case (Fig. 5.4).
- With a processing time of 500 ms, parallel MA and parallel CS implementations begin to perform better than sequential MA when the number of shops to visit is 6 (Fig. 5.5).

## *5.5 Conclusions*

We have classified existing MA applications in e-commerce, identified underlying mobility patterns for these applications and discussed possible implementation strategies for these patterns using the client-server and mobile agent paradigms. We have performed experiments to quantitatively evaluate these different strategies using the Voyager framework for MA implementations and Java for CS implementations.

Sequential CS implementations are most suitable for applications where a small amount of information has to be retrieved from few remote information sources (servers), and the degree of processing required is low (our experiments provide a good quantitative indication of these parameters). However, these conditions do not hold for most real-world e-commerce applications. MAs scale effectively across the above parameters, and with scalability being one of the needs of net-centric computing, we find that MAs are an appropriate technology for implementing e-commerce applications. Parallel implementations are effective when processing information contributes significantly to the turnaround time.

# Chapter 6

## Evaluation of Aglets, Concordia and Voyager

With a long list of mobile agent frameworks existing, some from the industry and most of them are contributions from the research community; selection of a mobile agent framework among the existing ones (around 60 of them are listed in the mobile agent list) is a difficult decision to make and require methodical assessment.

Different applications have different mobility patterns and requirements. For some applications, the overall performance is of concern, for others the economy of usage of resource could be important. We argue that application's inherent pattern plays an important role in choosing an agent framework for its implementation. Use of a very generic framework, or a framework which is meant for applications in some other domain will result in a deteriorated performance and just portraying the use of a new paradigm rather than highlighting on its strength. The need is to identify specific mobile agent frameworks, which suits best for e-commerce application. Towards this end we perform qualitative and quantitative evaluation of three popular Java based mobile agent framework viz. Aglets, Concordia and Voyager.

## *6.1 A qualitative evaluation*

### 6.1.1 Comparison parameters

Study of the agent framework in Chapter 3 provides us with architectural insights and the feature supported by these mobile agent frameworks. For a detailed comparison of these three mobile agent framework, we have performed a qualitative comparison across these framework. We have identified the following parameter of interest for qualitative evaluation.

- **Category**

The category of framework classifies whether the framework is specifically design for mobile agents or provides an additional support for mobility.

- **Form of mobility**

Does the MA framework support weak or strong mobility.

  - Strong mobility : the code, the data state (values of internal variables) and the execution state (stack and the program counter) of the moving entity are transferred to the new location
  - Weak mobility : Only code and data state of the moving entity are transferred to the new location

- **Java messaging**

Framework's support for sending regular Java messages to other agents, even if they are moving and regardless of where they are in the network.

- **Multicast**

Support for multicast messaging i.e. addressing a message to large number of host.

- **Publish / Subscribe**

Framework's support to send a message or event to all objects in a group which are registered subscribers, interested in message of the selected subject.

- **Scalability**

Feature to support large number of agents and host interactions. Scalable architecture provides support for multicast messaging, distributed events, publish/subscribe features.

- **Authentication and Security**

Support to prevent unauthorized code from executing a preset variety of operations. To support access restriction on objects operations to provide a secure computing environment.

- **Agent persistence**

Framework's feature to have a backup copy of the agent in a database. A persistent agent is automatically recovered if the agent is terminated or if it is flushed from memory to the database.

- **Naming service**

Support for naming service that enables connecting to an existing object, based on a name. This is particularly useful for launching a mobile agent from one application to another and then locating it after it moves.

- **Remote agent creation**

The ability to create agents at remote sites i.e. the place where a request is made and where and agent is instantiated are two different node on the network.

- **Messaging modes between agents**

Framework's support for different messaging modes for agent communication

  - Synchronous : the caller blocks until the message completes and the returns a value.
  - one-way : A one-way message does not return a result and the caller does not blocks.
  - future message : A future message immediately returns an object, which is a placeholder to the return value.

- **Database integration**

Support for performing transaction and firing queries to popular databases.

- **Execution environment**

Terminologies and divisions used for agent execution environment

- **Group / Collective**

Grouping of agents into physical or logical units as a means to manage, locate, and communicate with agents. Are these groupings defined by an agent's physical location, or are the groupings logical.

- **Proxy auto-update on agent migration**

Automatic proxy update when an agent moves and not an update using forwarding.

- **Garbage collection**

Facility for distributed garbage collection of agents when there is no local or external reference to the agent.

### 6.1.2 Qualitative comparison table

| **Features** | **Voyager** | **Aglet** | **Concordia** |
|---|---|---|---|
| Category | ORB with mobility support | MA based framework | MA based framework |
| Form of Mobility | Weak | Weak | Weak |
| Execution environment | Voyager server, space, subspace | Context | Server |
| Java messaging | Transparent | No | No |
| Multicast | Yes | No | No |
| Publish/Subscribe | Yes | No | No |
| Scalability | Space | No | No |
| Authentication and security | Strong implementation | Weak implementation | Strong implementation |
| Agent persistence | Yes | No | Yes |
| Naming service | Federated | No | No |
| Remote agent creation | Yes | No | No |
| Messaging modes between agents | One way, synchronous, future | One way, synchronous, future | No |
| Database integration | Most popular one's | No | proprietary |
| Grouping / Collective | Logical | Physical | Physical |
| Proxy auto update on agent migration | Yes | No | Yes |
| Garbage collection | Yes | No | No |

## *6.2 A quantitative evaluation*

To analyze the performance of these three mobile agent frameworks, Aglets, Concordia and Voyager; and to understand the importance of application's mobility pattern, we have performed comparative experiments for applications having e-commerce characteristics across these framework. The experiments are designed in such a way as to isolate the performance properties and parameters of interest at different level of detail and come up with a performance analysis result.

With the goal to identify a mobile agent framework which suits most for an e-commerce application we have performed comparative qualitative performance analysis across these mobile agent frameworks.

Results from quantitative performance analysis will help improving the performance and scalability of a system. Such quantitative analysis can identify application bottlenecks and determine the best implementation strategy that enhance application performance and would reduce the network load as a whole. The issue of performance is also very important in emerging Internet-systems: numerous studies show that performance of systems and application determines to a large extent the popularity of Internet services and user-perceived Quality of Service [38].

### 6.2.1 Issues in quantitative comparison

Large number of factor influence the performance of a mobile agent system, because of numerous component involved. Hence the quantitative evaluation of such a system becomes more complex [38]. Issues such as the following need to be considered while carrying out a performance study.

1) The absence of global time, control and state information: this makes it hard to define and determine unequivocally the condition of a particular MA-system at a particular moment.
2) The complex architecture of MA platforms, which makes it difficult to describe performance properties with the familiar, simple metrics used in parallel and distributed systems.
3) The variety of distributed computing models that are applicable to mobile agent applications dictates the design of a variety of experiments to explore different performance problems that may arise under different circumstances.
4) The dynamic nature of MA systems running on Internet, which makes it hard to establish a concise and stable representation of the set of system resources a MA performance.
5) The additional complexity introduced by issues that affect the performance of Java, such as interpretation versus compilation, garbage collection, etc.

### 6.2.2 The quantitative evaluation setup

We have chosen a typical application from the domain of e-commerce to quantitatively evaluate the three framework. The characteristic of the experimental applications are chosen such that they are common and recurring patterns of e-commerce applications. The experimental application represent the basic application pattern of any e-commerce applications, isolating other features which are not of interest in determining the choice of an application framework.

To analyze the performance of mobile-agent frameworks, we have developed an approach for capturing basic performance properties of these platforms. These properties are defined independently of the various ways each particular mobile-agent API can be used to program and deploy applications and systems on Internet. To this end, our approach focuses on identification of basic elements of mobile agent system involved in an application. The interaction of these basic element result in a mobility pattern, which in turn decides an implementation strategies. Our experiment also seeks to expose the performance behavior of these functionalities by identifying the parameters involved and measuring them against our performance metric viz. the user turn-around time.

### 6.2.3 The application

The application for our experiment is of product discovery in a typical e-commerce scenario viz., that of a single client searching for information about a particular product from the catalogs of several on-line stores.

We assume that the client requires a highly customized search which the on-line store does not support and hence the filtering logic is carried along with the mobile agent to each host it visits. At each shop the mobile agent only uses the basic operation provided by the shop's database engine. The lager part of filtering is done by the logic that the mobile agent carries along with it, which represents a user's specific taste and requirement for a give product in request.

### 6.2.4 Basic elements

The basic entities involved in all these mobile agent framework in consideration and their functionalities for performance behavior are identified as

- i) Buyer's shopping agent
- ii) Shop
- iii) Buyer's site
- iv) Agent behavior

### 6.2.5 Application's mobility pattern

The arrangement and the fashion in which the basic elements interacts decide the application's mobility pattern. The performance traits of an application depends on the characteristics of its basic elements, how these elements are combined and influence each other. For our performance analysis experiment the pattern is that of an agent searching for a product in a given order over various shops across a local area network.

### 6.2.6 Experimental test bed

The experiments were carried out on P-III, 450 MHz workstations connected through a 10 Mbps LAN with typical student load. We have experimented with three Java mobile agent frameworks namely Voyager, Aglets and Concordia. These agents run with within hosts, which were free of additional processing load from other applications.

### 6.2.7 Experimental parameters

In order to capture the intrinsic performance properties of basic elements, we have considered agents with limited functionalities and interface, which carry the minimum amount of code and data needed for their basic behavior. An agent execution environment for different agents sit at each host. We have considered the following parameters for performance analysis.

- Number of stores (varies from 1 to 26)
- Size of catalog (1 MB)
- Processing time for servicing each request ( 20 ms )
- Network latencies on different links (assumed constant since all workstations were on the same LAN)
- Message size (kept constant for all three frameworks)

Our performance metric is the user turnaround time, which is the time elapsed between a user initiating a request and receiving the results. This includes the time taken for agent creation, time taken to visit shops and carry out the filtering operation from the shop's catalogs, carrying forward the desired result to the next host to make its final decision of choosing the best deal and returning back at the clients site.

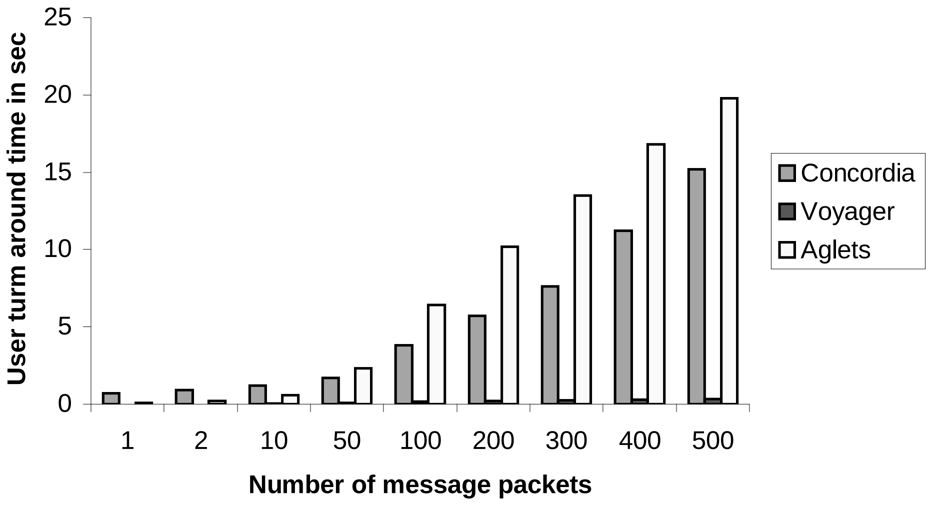

**Fig 6.1 : Synchronous message exchange cost for Voyager, Aglets and Concordia.**

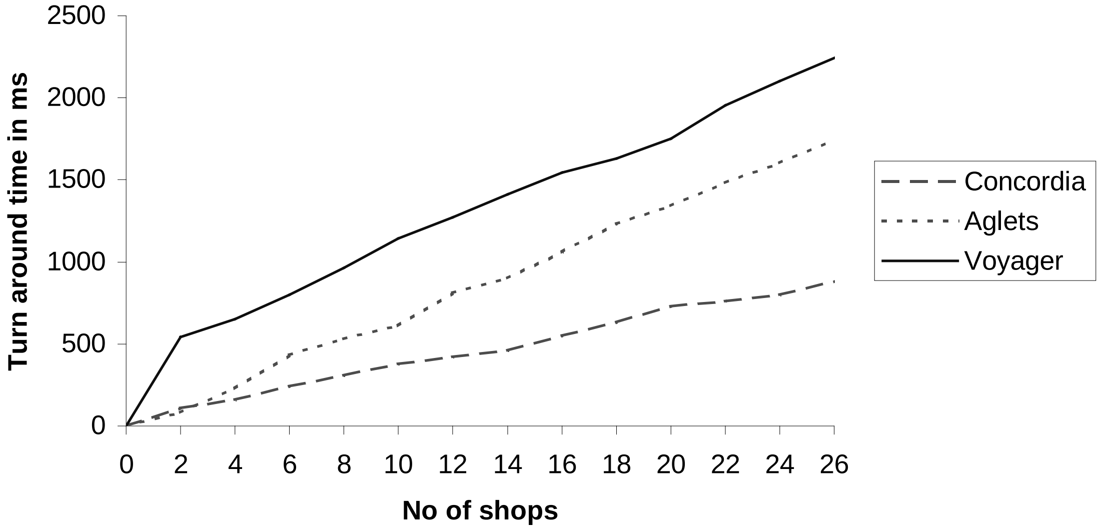

**Fig 6.2 : Code shipment cost for Voyager, Aglets and Concordia**

### 6.2.8 Experiments

The most basic feature that need to be provided by an agent framework are

- Agent mobility and
- Messaging for agent communication

These features are responsible for output performance of a mobile agent application. Our experiments examine :

**1) Message transfer cost for different mobile agent framework**
A small message of fixed size was send in synchronous mode across two nodes in the same LAN and the user turn-around time for this was observed. The number of message was varied from 1 to 500 and the turn-around time observed.

**2) Code shipment cost for different mobile agent framework**
A mobile agents of each of the system was made to visit varying (1 to 26) number of shops and the user turn-around time was measured.

### 6.2.9 Results

**1) Message transfer cost for different mobile agent framework**
Fig 6.1 shows our result for cost of synchronous message exchange for Voyager, Aglets and Concordia. We observe that Voyager has a much better and more responsive message transfer support as compared to the other two frameworks. Voyager performs comparatively very well as compared with other two. Aglets show a better performance than Concordia.

**2) Code shipment cost for different mobile agent framework**
Fig 6.2 shows our result for cost of mobile agents shipment for Voyager, Aglets and Concordia. Concordia gives the best performance for code shipment compared to the other two. Concordia and Voyager both uses RMI and serialization for agent transfer, but still we see a large difference between the two.

## *6.3 Conclusion*

Experimental results help us isolate the performance characteristics of mobile agent platforms examined, and lead us to the discovery and explanation of basic performance properties of mobile agent systems. They provide a solid base for the assessment of the relative merits and drawbacks of the platforms mined from a performance perspective.

The detailed qualitative study highlights the strengths of different mobile agent frameworks. We observe that Voyager supports almost the super set of functionalities and features as compared with the other two. Voyager's multicast and publish subscribe features facilitate development of scalable system. Also Voyager support advanced messaging and direct Java messaging to remote objects. The integration of security an naming service help in developing real life applications.

Voyager being an ORB has advanced messaging support and hence performs much better than Aglets and Concordia. Aglets performs better than Concordia because of weak messaging architectural support of Concordia.

Concordia and Voyager both uses RMI and serialization for agent transfer, but still we see a large difference between the code shipment performance of the two this is because

- Voyager is not specifically designed as a mobile agent framework. Voyager is an ORB with support for mobility and autonomous agents.
- difference in performance between Concordia and Voyager is also because of the large set of functionality supported by Voyager.

Aglets uses it own agent transfer protocol for shipping agents and performs worse than Concordia and better than Voyager.

# Chapter 7

## Our Prototype of e-commerce application using mobile agents

Mobile agent paradigm provides a cleaner design for several real life application as compared to the traditional client server model. The mapping of implementation components to real life objects is direct while using mobile agent technology. In an e-commerce scenario, a mobile agent is launched from a buyer's site with a list of market to visit, the products to look for, the desired product attributes and some other user preferences. A real life problem of getting a product or a set of product can be directly mapped onto the mobile agent design paradigm, where in, real life you send an assistant with all these parameters to get the job done, here an agent act as an assistant. Compared to the client server design model where a real life scenario has to be forced to fit in a request-response model, a mobile agent is a better and more clear design technology.

The activity of buying and selling among end-consumers is a particularly time-consuming and inefficient form of shopping and often includes additional steps such as negotiating on price and other features. We believe that the effective use of mobile agents can dramatically reduce transaction cost involved in electronic commerce, in general, and in business-to-consumer transaction, in particular.

### *7.1 Implementation aspects*

We have implemented a complete business-to-customer e-commerce based application using mobile agent technology. The goal of our prototype was to gain practical experience with mobile agent in e-commerce as well as the software engineering aspect of mobile agent technology.

In our prototype model we have created an e-market place for buying and selling of goods using both the traditional client server paradigm and the mobile agent paradigm. Ten e-shops were hosted which handle large database of products and understand both of the design paradigms. A buyer could use any of the implementation mechanisms to get a deal . The e-commerce application that we have choose is that of a user searching for a book with desired attributes across shops of interest. The product are represented in XML in a separate file and the product information are dealt using XML tags. By changing the XML file and tags the application could easily be used for any other product discovery.

## *7.2 Agent framework*

We have used Voyager ORB as the mobile agent framework for our applications implementation. In Voyager an agent is a special kind of object that can move independently, can continue to execute as it moves, and otherwise behaves exactly like any other object. Voyager enables objects and other agents to send standard Java messages to an agent even as the agent is moving. In addition, Voyager allows to remote-enable any Java class, even a third-party library class, without modifying the class source in anyway. Voyager also includes a rich set of services for transparent distributed persistence, scalable group communication, and basic directory services.

Voyager's API for agent mobility provides us with the common basic feature of agent mobility and hence allows a more customizable application to be developed using pure Java code.

## *7.3 Prototype architecture*

We have implemented a customer driven market place where, the system is based on pull model of marketing i.e., the buyer moves around several shops searching for a product. The entities involved in our prototype model are buyer, mobile agent, shops and Voyager's naming server. The buyer is an entity searching for a product of its interest at various shops. The shops maintain huge products catalog and support some basic services to access and operate on product catalogs which are stored as XML files. Figure 7.1 show architecture of our prototype model.

A buyer interested in a product launches a mobile agent and provides it with a list of shops to visit, the product specification and product evaluation logic. The buyer's mobile agent visits each of the shops in it itinerary in the specified order. On arrival at a shop, the buyer's mobile agent contacts a stationary local agent to get the required product. The shop's local agent hands over the buyer's mobile agent to a local salesman agent, which deals with a particular category of products. The salesman agent uses local services to search the product catalog according to buyer's criteria and returns the result to the buyer's agent. The buyer's agent then uses its evaluation logic to evaluate the product from the filtered list which match best to his taste. The buyer agent rates each of its entries then carries this information along with it and hops on to the next shop in its itinerary. The buyers mobile agent also has a list of sites to visit in-order to support mobile devices which may be disconnected for small interval of time. On completion of its itinerary the buyer's MA returns back to the buyer's site and contacts the stationary agent and handover the information. The stationary agent then displays the result and the best deal to the user. It is then to the user to make a choice and go ahead with the purchase.

## *7.4 Entities involved*

The entities involved in our prototype are :

- Voyager's naming server
- Buyer
- Mobile agent and
- Shops

These entities have several components which constitute the whole system. Figure 7.1 show the components of our prototype model and their interaction.

### 7.4.1 Voyagers naming server

Voyager contains an integrated directory service that allows to associate an alias with any object and connect to the object via its alias at a later time, even if it has moved. Voyager's directory service allows to create and connect network directories together to form a large federated directory service.

The shops and host in our prototype designed are registered with the Voyager's naming service and are identified by unique names.

### 7.4.2 Buyer

1) **Buyer**
Buyer is the user who is willing to get a product from several online stores and is not interested in searching for information all over the net. The buyer interacts with a graphical user interface and provides it with his preferences for the product. The submission in turn lead to the creation of a buyer's mobile agent to get the job done. A buyer site is registered with the Voyager's naming service and is identified by a unique name.

2) **Buyer's Graphical User Interface (BGUI)**
The buyer's graphical user interface is the interface between the buyer (user) and the stationary agent sitting at that host. A buyer interface allows the user to input parameters for a specified product. A buyer's-id is required to uniquely identify the buyer in the marketplace, this could be the buyer's email-id or some agreed upon protocol of identity assignment. For our prototype we use the buyer's email-id as buyer's-id.

3) **Buyer's Stationary Agent (BSA)**
Buyer's stationary agent is the local agent the sit at the buyer's site and is responsible for managing resources and service at the buyer's end. The buyer's stationary agent executes in the agent execution environment and is responsible for management of mobile agent i.e. agent creation, receiving and dispatching agent. The buyer's stationary agent is the interface for the mobile agent to access

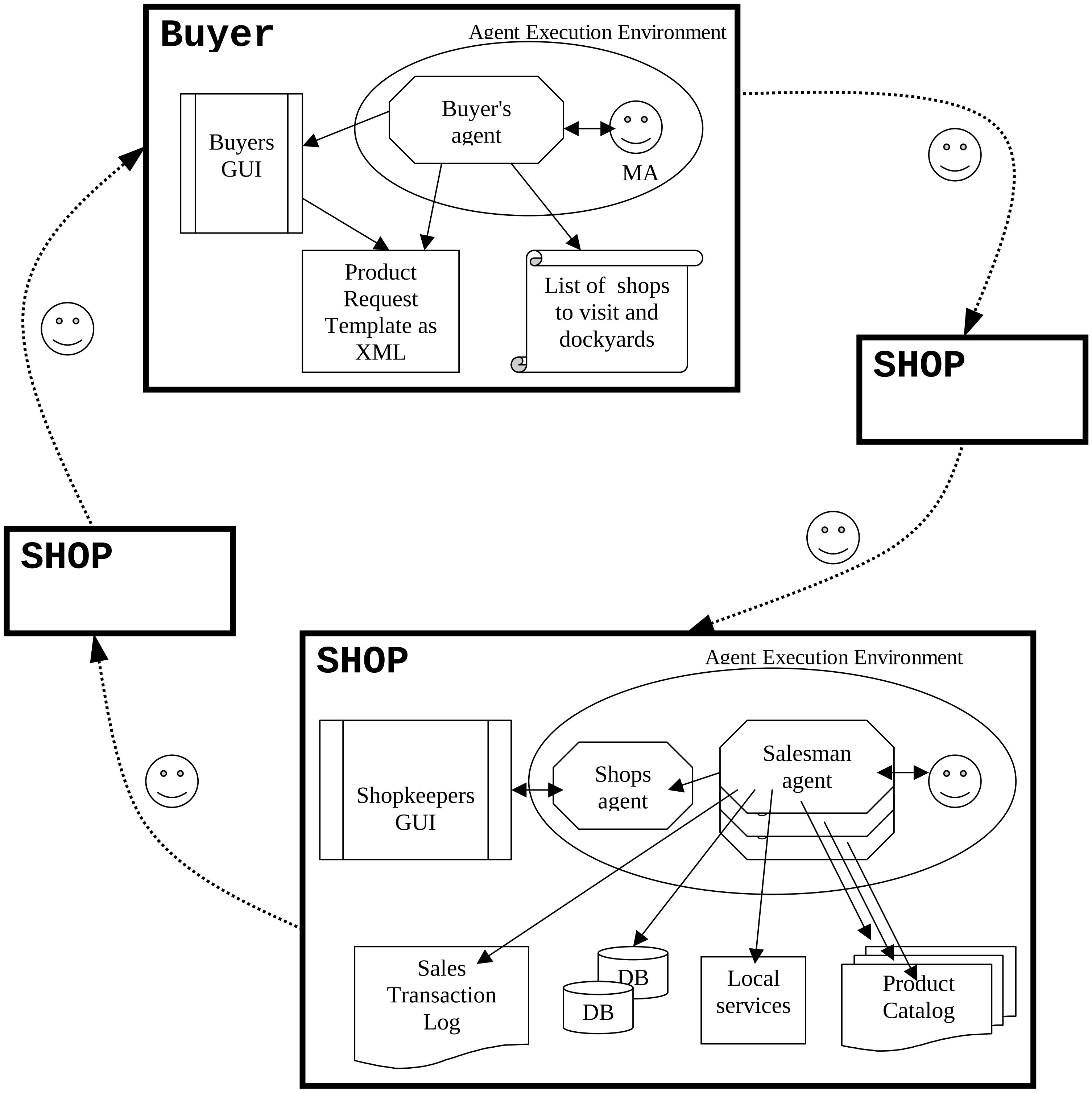

**Fig 7.1 :The Architecture of our prototype model**

resources and services. A mobile agent visiting the buyer's site passes on his request to access some of the local resource or services to the stationary agent which in turn carries out the operation for the mobile agent. Buyer's stationary agent can thus also be used for security and authentication purposes. On user's submission of product parameters using the buyer's graphical user interface the information is collected by the buyer's stationary agent and stored in a data structure. The buyer's stationary agent then creates a buyer's mobile agent to get the product. The buyer's mobile agent is provided with a list of shops to visit, the product specification, product evaluation logic and sent off to get the product.

**4) Agent Execution Environment (AEE)**

Voyager's ORB is our agent execution environment which provides support for agent creation, arrival, dispatch and agent management. Voyager is an ORB (Object Request Broker) implemented in Java and provides support for remote messaging, mobile objects and autonomous mobile agents.

**5) Buyer's Mobile Agent (BMA)**

The Buyer's mobile agent is a Voyager's mobile agent which move around the network visiting shops in its itinerary to get its job done. Buyer's mobile agent is created and launched from the buyer's site to search a product. The buyer's mobile agent is provided with a list of shops to visit, the product specification and product evaluation logic. The BMA visits all the shops in its itinerary and returns back to the buyer with the result. At each shop the BMA interacts with the shop's stationary agent and further with the salesman agent. The list of products that matches the user's parameters are stored in vector of product data structure and carried along with the mobile agent. On completion of its itinerary the mobile agent returns back to the buyer's site and handover the information to the buyer's local agent which in turns displays the result to the user.

**6) Mobile Agent's Evaluation Logic (MAEL)**

MAEL is the buyer's product evaluation criteria which is carried along with the mobile agent and executed at each shop before adding a product offer to the mobile agent's product list. MAEL evaluates and rates products that match user's specification and carries the top three offers from the list. The MAEL provides user specific operation on the product catalogs, so a buyer is not restricted to just the operation provided at the shops. The MAEL is created at the buyer's end and attached to the mobile agent hence it helps in selecting products according to user choice.

**7) Product Request Template (PRT)**

The PRT is the product parameters specified by the user which are stored in XML data format structure and is attached to the mobile agent by buyer's local agent. At a shop the mobile agent passes on its product request XML tree to the salesman agent which in turn does the filtering of products form the product catalog which is stored as DOM tree.

**8) List of shops to visit**

On agent creation the agent is provided with a set of shops to visit in its itinerary. Shop's name are registered at the Voyager's naming service and the mobile agent gets the network addresses of these shops using the Voyager's naming service.

**9) List of host supporting disconnected operations**

A buyers mobile agent also has a list of site to sit in case the host is a mobile device and faces small interval of disconnection. This provides support for disconnected operations for mobile devices. The mobile agents pings for the mobile host at regular intervals to see if it is connected.

### 7.4.3 Mobile agent

The mobile agent in our prototype is the buyer's mobile agent (BMA). The mobile agent is responsible for automating the whole task of shopping by acting on behalf of the user. The mobile agent move around different nodes in the network and accesses local resources at the site by communicating with the local agents. The mobile agent uses Voyager's naming service to move across shops in our system. The mobile agent carries along with it, list of shops to visit, the product specification and product evaluation logic. In case of a mobile host the mobile agent has a list of hosts to sit when disconnected.

### 7.4.4 Shops

**1) Shopkeeper**

Shopkeeper manages his shop at a given site. The shopkeeper is responsible for adding new products, catalogs. The Shopkeeper interacts with the system using the shopkeeper's graphical user interface. The Shopkeeper uses shopkeeper's graphical user interface to see the list of products at his shop and the transactions made at his shop. The shopkeeper keeps track of sales transaction with the help of sales transaction log, which is also displayed at the shopkeeper's graphical user interface.

**2) Shopkeeper's graphical user interface (SGUI)**

The shopkeeper's graphical user interface provides an interface to the shopkeeper to surf through its product catalogs. SGUI also provides information of transactions made at the shop. SGUI could also be used to add and modify product catalogs; in our prototype model we have not implemented functionalities of shopkeeper being able to add and modify its product catalogs.

**3) Shop's stationary agent (SSA)**

Shop's stationary agent is the local agent the sits at the shopkeeper's site and in responsible for

- Mobile agent management i.e., receiving and dispatching mobile agents to / from its site. A buyer's mobile agent looking for a shop, first contacts the SSA. The SSA handles all request from the mobile agent and depending on requirement spawns salesman agent. Further interaction is between the salesman agent and the mobile agent.

- Interacting with the shopkeeper for product and catalog management

The SSA also maintains a list of mobile agent visiting and currently active at its shop. All access to local resource and services by the mobile agent are passed on to either SSA or the salesman agent; which then execute it, for the mobile agent thus imposing security and authority restrictions.

**4) Agent execution environment (AEE)**

Voyager's ORB is our agent execution environment which provides support for agent creation, arrival, dispatch and agent management. Voyager is an ORB (Object Request Broker) implemented in Java and provides support for remote messaging, mobile objects and autonomous mobile agents.

**5) Shop's Salesman agent (SSmA)**

A SSmA is spawned on a mobile agents request to the SSA for accessing a specific product catalog. Each product catalog is maintained by a specific type of SSmA which has all the information about the product. The SSmA uses local service such as filtering, searching, etc. to serve mobile agents requests. The buyer's mobile agent operate on product catalog through SSmA which returns the buyer's mobile agent the resultant product sub-tree.

**6) Product catalogs in XML**

With regard to interoperability and platform-independence required for the Internet application for information interchange. We have used XML as our data representation format for product catalogs. Catalogs of different products are maintained by different stationary salesman agents.

**7) Local services**

Each shops host some local service to provide access and operation to local resources. Local services could be availed by mobile agent by request to local agents. For our product catalog we have provided two local service of filtering and searching. Local services provides support and functionalities for salesman agents.

**8) Sales transaction log (STL)**

Each shop maintains a STL which records user request and transaction made by the user. This could help the shopkeeper study user's behavior.

## *7.5 Features and suitability of our design*

In the domain of e-commerce where, in the current scenario a lot of time is being spent on product discover, finding best deal and negotiating at different stages. These operations not only involve larger user interaction time but leads to extra network load and larger number of message interchange over the network. e-commerce applications require :

- e-commerce application demands faster response and real time decision making, for example in an auction house or a stock market the user would want to have the price changes as soon as possible, make his decision on the changed price and convey to the other party.

In traditional computing models network delay and disconnected operation provide a major hindrance to real time interaction.

- Also marketing is always customer driven i.e. a user buys goods to his requirements and specifications. Hence a need of customer specific queries and operation are must for e-commerce.

In current scenario the queries are limited to as that provided by the shops. Also with mobile devices being pervasive support for disconnected operation is must.

Our prototype implementations addresses all these problems and provide solution for them.

### 7.5.1 Data representation

The data representation format for our application is XML. XML represents data in a familiar (HTML-like) tagged textual form, and explicitly separates the treatment of data from its representation. This achieves the platform-independence required for the Internet and the well-appreciated feature of human-readability. In addition, XML can be made capable of representing whatever kind of data and entity one is likely to find in the Internet: complex documents, service interfaces and objects, communication protocols, and agents. These characteristics let us think that interoperability in the Internet will be information-oriented and based on XML, rather than service oriented and based on CORBA. With regard for inter-operability and as a communication mode for mobile agent the product catalogs, product requests and filtered product results are all represented in XML.

### 7.5.2 Static and mobile agents

We have distinctly separated and identified use of static and mobile agents in our design. We have static agents at the buyer's and shoppers' end which are responsible for handling mobile agents and mobile agent's request. Our static agent presently only manage mobile agents and resource at that site but they could be extended to being AI agent which study the buyers behavior and then accordingly creates a mobile agent on behalf of the buyer. Similarly the idea could be extended to the shopkeeper's stationary agent which could mine the users transaction to identify user's buying behavior.

### 7.5.3 Evaluation Logic

The Mobile Agent's Evaluation Logic (MAEL) is another strong and extendable feature of our prototype. MAEL is the users evaluation and rating criteria, for a product in request. This module is dynamically aggregated with the mobile agent and hence any time a change in preference is to be implemented a new module need to be attached.

#### 7.5.4 Salesman agent

Use of salesman agent is a cleaner and better way of implementing a shops. Different salesman agents are responsible for managing different product catalogs. A user (mobile agent) interested in a product need to contact that specific salesman agent. This help in providing different administrative right to different products.

#### 7.5.5 Support for disconnected operations

Our mobile agent also maintains a list of host to sit in case it could not migrate it self to a mobile host from where it was launched because of temporary disconnection. This feature provides support for disconnected operations. The agent moves to any of these host (dockyards) and ping the destination at regular intervals to determine when the host is connected.

## *7.6 Test bed*

The experiments were carried out on P-III, 450 MHz workstations connected through a 10 Mbps LAN with typical student load. Shops were hosted on ten machines, which all had the Voyager framework installed. One machine was used as the buyer's site. Each shop and the buyer's site was registered with the Voyager's naming service with a unique name.

## *7.7 The e- market place*

We created an e-market place for buying and selling of books using both the traditional client server paradigm and the mobile agent paradigm. Ten e-shops were hosted which handle large databases of books and understand both of the design paradigms. A user interested in some book enters parameters such as, title of the book, publisher's name, author's name etc. A mobile agent is created then, which visit all the ten shops, searching for the book. At each shop the mobile agent contacts the shop's local agents. At each shop it evaluate the result and finally returns back to the buyer with the matching results.

## *7.8 Conclusion*

Our experience with mobile agent implementation show that mobile agent are better design paradigm for real life application development. With mobile agents the mapping of application logic with real life object is direct. The API's provided by Voyager are good and easy enough to develop a customized applications with pure Java code. Mobile agent provide large flexibility in design and dynamic aggregation support of Voyager helps add module on the run for specific cases. Applications can easily be extended and modeled for different jobs across the network by just changing the Mobile Agent's Evaluation Logic (MAEL).

e-commerce using mobile agent provide a solution for quicker and smarter marketing. Mobile agents in e-commerce applications help users with

- Tedious repetitive job and time consuming activities of product discovery and comparisons.
- Faster and real time interacting at shops : the agent sits at the shops so message exchange are local
- Reducing network load : message exchanges are local thus reducing network load.
- Support for disconnected operation.
- Introduce concurrency of operations : multiple agents doing different sub part of a big job.
- Client specific functionalities at the shops: with agent carrying the clients product filtering logic we have support for client specific functionalities at the shops.

# Chapter 8

## Conclusion and future work

In this project, we identified mobile agent as a promising design paradigm for the development of e-commerce applications. Our work provides a methodical way for implementation of a complete e-commerce based application over the Internet. The project involved work in three logical areas:

- Quantitative evaluation of mobile agent design paradigms.
- Qualitative and quantitative evaluation of Aglets, Concordia and Voyager.
- Design, deployment and software engineering issues of an e-commerce based mobile agent application.

### *8.1 Design paradigm evaluation*

We classified existing MA applications in e-commerce, identified underlying mobility patterns for these applications and discussed possible implementation strategies for these patterns using the client-server and mobile agent paradigms. Our experience and the results suggest that mobility pattern play an important role in deciding the implementation strategy to be used for high performance applications. We have performed experiments to quantitatively evaluate these different strategies using the Voyager framework for MA implementations and Java sockets for CS implementations.

Sequential client server implementations are most suitable for applications where a small amount of information has to be retrieved from few remote information sources, and the degree of processing required is low (our experiments provide a good quantitative indication of these parameters). However, these conditions do not hold for most real-world e-commerce applications. MAs scale effectively across the above parameters, and with scalability being one of the needs of net-centric computing, we find that MAs are an appropriate technology for implementing e-commerce applications. Parallel implementations are effective when processing information contributes significantly to the turnaround time. We show that only when the cost of shipping a mobile agent is less than the message exchanges cost, mobile implementations are useful. We also noted that large size of message (catalog) exchanges and large number of interaction are typical characteristics of e-commerce application which are conducive for mobile agents implementation.

From our experiments we obtained the performance *cross-over points* for different implementation strategies and design paradigm. We believe that a *hybrid-model implementation* of client server and mobile agent paradigm would result a in better performance results over the long run.

## *8.2 Mobile agent framework evaluation*

This project identified the basic element involved and the application's mobility pattern for an e-commerce application. The arrangement and the fashion in which the basic elements interacts decide the application's mobility pattern. The performance traits of an application depends on the characteristics of its basic elements, how these elements are combined and influence each other. In our performance analysis experiment for a product discovery pattern, we found that Concordia performs better than Aglets and Voyager, and has the least code shipment cost. Voyager's advanced messaging support provides a much faster messaging than Aglets and Concordia. Voyager's low performance in code shipment is because of the large set of functionality supported by Voyager and it is not tailored specifically for mobile agent system.

Still the overall performance and features supported by Voyager make it a good framework choice for developing real life mobile agent applications.

Our experimental results provide a solid base for the assessment of the relative merits and drawbacks of the platforms mined from a performance perspective for an e-commerce application.

## *8.3 Software engineering issues*

Our experience with mobile agent implementation show that mobile agent are better design paradigm for many real life e-commerce application development. The mapping of application design with real life object is direct. Also mobile agent provide large flexibility in application design, extendibility and easy integration of new functionalities. We note that e-commerce using mobile agent provide a solution for quicker and smarter marketing. Mobile agents help users with tedious repetitive job and time consuming activities of product discovery and comparisons. With the mobile agent moving to different shops and interacting at these shops, the message exchange are local and not over the network, hence reducing network load. As the mobile agent sits with all the logic at the shop we have faster response to changes, which also provide support for disconnected operation. Mobile agent introduce concurrency and with agent carrying the clients product filtering logic we have support for client specific functionalities at the shops.

Our performance analysis experiments using the application mobility pattern provided measurements and observations that help us establish causality relationship between conclusions from our qualitative evaluation experiments to the observations made with the application implementation of our prototype. Our prototype provides support for both client server and mobile agent implementations and can switch between the two implementations. The better performance of mobile agent technology for our bookstore application buys its justifications from our performance analysis experiment which were based on product discovery pattern.

## *8.4 Ideas and future work*

We believe that a hybrid-model implementation of client server and mobile agent paradigm would result a in better application model, mined from a performance perspective. A metric of *performance cross-over points* over different parameters would be used by the hybrid model to decide the most economical implementation for a given job in hand. A real time learning algorithm would help further improve and update the knowledge of our performance metric and thus helping in better choice of a design paradigm for a job with given set of parameters.

We believe that mobile agent frameworks should be geared toward application's mobility pattern, rather than providing a generic framework which support mobility. To enhance performance we require application specific framework which is designed for a set of application 's mobility behavior.

Our project provide a framework for e-commerce based application design using mobile agents. The project does not look at the security aspects of mobile agent system. This work is to be further extended with detailed study of security issues in mobile agent and their use in e-commerce. Also study of suitability of  existing e-payment protocols for mobile agents is essential for incorporating payment system with mobile agents. Design of a new payment protocol which suits most to the mobile agent paradigm could be carried out as further extension.

## 9 References

[1] Jonathan Bredin, David Kotz and Daniela Rus, "Market-based Resource Control for Mobile Agents", Proceedings of Autonomous Agents, May 1998.

[2] Antonio Carzaniga, Gian Pietro Picco and Giaovanni Vigna, "Designing Distributed Applications with Mobile Code Paradigms ", Proceedings of the 19th International Conference on Software Engineering(ICSE '97) , pp. 22 - 32, ACM Press, 1997.

[3] P. Dasgupta, N. Narasimhan, L.E. Moser and P.M. Melliar-Smith, "A Supplier Driven Electronic Marketplace Using Mobile Agents", Proceedings of the First International Conference on Telecommunications and E-Commerce, Nashville, TN, Nov. 1998.

[4] P. Dasgupta, N. Narasimhan, L.E. Moser and P.M. Melliar-Smith, "MAgNET: Mobile Agents for Networked Electronic Trading ", IEEE Transactions on Knowledge and Data Engineering, Special Issue on Web Applications , July-August 1999.

[5] Alfonso Fuggetta, Gian Pietro Picco and Giovanni Vigna , "Understanding Code Mobility ", IEEE Transactions on Software Engineering , vol. 24(5), 1998.

[6] Carlo Ghezzi and Giovanni Vigna, "Mobile code paradigms and technologies: A case study ", Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany , vol. 1219 of Lecture Notes on Computer Science, Springer, April 1997.

[7] G. Glass, "ObjectSpace Voyager Core Package Technical Overview ", Mobility: process, computers and agents , Addison-Wesley, Feb. 1999.

[8] Dag Johansen, "Mobile Agent Applicability ", Proceedings of the Mobile Agents 1998, Berlin , Springer-Verlag, Lecture notes in computer science; vol. 1477, ISBN 3-540-64959-X, (1998), pp. 9-11 September, 1998.

[9] Danny B. Lange and Mitsuru Oshima, "Seven Good Reasons for Mobile Agents ", Communications of ACM , vol. 42, no. 3, March 1999.

[10] Pattie Maes, Robert H. Guttman and Alexandros G. Moukas, "Agents That Buy and Sell ", Communications of ACM , vol. 42, no. 3, pp. 81 - 91, March 1999.

[11] Todd Papaioannou, "Mobile Information Agents for CyberSpace - State of the Art and Visions ", To appear in Proc. of Cooperative Information Agents (CIA- 2000) , 2000.

[12] Stavros Papastavrou, George Samaras and Evaggelia Pitoura, "Mobile Agents for WWW Distributed Database Access ", Proceedings of IEEE International Conference on Data Engineering (ICDE99) , 1999.

[13] Gian Pietro Picco and Mario Baldi, "Evaluating Tradeoffs of Mobile Code Design Paradigms in Network Management Applications ", Proceedings of 20th International Conference on Software Engineering, ICSE98, Kyoto, Japan, IEEE CS Press , 1998.

[14] Thomas Sandholm and Qianbo Huai, "Nomad: Mobile Agent System for an Internet-Based Auction House ", IEEE Internet Computing , vol. 4, no.2, March-April 2000.

[15] David Kotz and Robert S. Gray. "Mobile code: The future of the Internet.", In Proceedings of the Workshop on Mobile Agents in the Context of Competition and Cooperation (MAC3)" at Autonomous Agents '99, pages 6-12, Seattle, Washington, USA, May 1999.

[16] AuctionBot URL: http://auction.eecs.umich.edu.

[17] BargainFinder URL: http://bf.cstar.ac.com/bf.

[18] A. Chavez and P. Maes, "Kasbah: An agent marketplace for buying and selling goods," Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK (April 1996), pp. 75-90.

[19] R. Doorenbos, O. Etzioni and D. Weld, "A scalable comparison/shopping agent for the World Wide Web", Proceedings of the First International Conference on Autonomous Agents, Marina del Rey, CA (February 1997).

[20] R. Guttman and P. Maes, "Agent-mediated integrative negotiation for retail electronic commerce", Mediated Electronic Trading Workshop, Minneapolis, MN (May 1998).

[21] Firefly URL: http://www.firefly.com.

[22] R. Guttman, A. Moukas and P. Maes, "Agent-mediated electronic commerce: A survey", Knowledge Engineering Review (June 1998).

[23] Jango URL: http://www.jango.com.

[24] J. Rodriguez, P. Noriega, C. Sierra and J. Padget, "FM96.5: A Java-based electronic auction house", Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK (April 1997), pp. 207-224.

[25] M. Tsvetovatyy, B. Mobasher, M. Gini and Z. Wieckowski, "MAGMA: An agent-based virtual market for electronic commerce", Applied Artificial Intelligence, vol. 11, no. 6 (September 1997), pp. 501-523.

[26] David Chess, Colin Harrison, and Aaron Kershenbaum, "Mobile Agents: Are They A Good Idea?", IBM Research Report

[27] Saurab Nog, Sumit Chawla, David Kotz , "An RPC Mechanism for Transportable Agents", Dartmouth PCS-TR96-280, Department of Computer Science, Dartmouth College, 1996

[28] Neeran M. Karnik, Anand R. Tripathi, "Design issues in mobile agent programming systems", IEEE Concurrency, Vol. 6, No. 3, pp. 52-61.

[29] Voyager 3.2 ORB Documentation, www.objectspace.com

[30] Aglets Home URL: http://www.trl.ibm.co.jp/aglets, http://www.aglets.org

[31] Danny B. Lange, "Mobile Objects and Mobile Agents: The Future of Distributed Computing", In Proceedings of The European Conference on Object-Oriented Programming '98, 1998.

[32] Yariv Aridor and Danny B. Lange, "Agent Design Patterns: Elements of Agent Application Design", In Proceedings of Agents'98, 1998.

[33] Danny B. Lange and Mitsuru Oshima , "Mobile Agents with Java: The Aglet API", World Wide Web Journal, 1998.

[34] Gunter Karjoth, Danny B. Lange and Mitsuru Oshima, A Security Model for Aglets, IEEE Internet, July/August, 1997.

[35] Concordia Home URL : http://www.merl.com/HSL/Projects/Concordia/

[36] Alberto Castillo, Masataka Kawaguchi, Noemi Paciorek, David Wong, "Concordia as enabling technology for cooperative information gathering", "Japanese Society for Artificial Intelligence Conference" in Tokyo, Japan , June 1998.

[37] G. Samaras, M. Dikaiakos, C. Spyrou, and A. Liverdos., "Mobile agent platforms for webdatabases: A qualitative and quantitative assessment." In Proc. of the 1st Int'l Symp. on Agent Systems and Applications and Third Int'l Symp. on Mobile Agents (ASA/MA'99), pages 50--64. IEEE Computer Society Press, October 1999.

[38] M. Dikaiakos and G. Samaras. "Quantitative Performance Analysis of Mobile-Agent Systems: A Hierarchical Approach.", Technical Report TR-00-2, Department of Computer Science, University of Cyprus, June 2000.