Chapter 4: Mobile Agent & Distributed Environment

4.1 Introduction

As discussed in chapter 3, an agent is defined as a self-contained software element that acts autonomously on behalf of a user (e.g., person or organization) [1,2]. Each agent has its own thread of execution, which means that it can perform tasks on its own initiative. A mobile agent has, in addition, the unique ability to migrate from one host in a network to another [1,2].

Mobile agents can be also defined as computational software those are capable of roaming different WANs. While on tour, a mobile agent interacts with foreign hosts and gathers information on behalf of its owner who has initiated it. After performing its duties set by the owner, the mobile agent comes back to the originating source/owner.

In mobile computing environment, users can access information independent of their location [3-6]. But accessing this information should not restrict mobility of users. From data management point of view, mobile users can handle only fraction of data since mobile devices are having limited resources. The development of low cost and yet portable mobile devices have enabled mobile users to work from anywhere, at anytime.

Distributed Environment itself is a term which covers all types of environments where distribution is done over any of the functional or non-functional parts are distributed or over the whole system. It may be
distributed systems, client/server system, distributed database management system, distributed computing, remote execution etc. In my work the term "Distributed Environment" is used as I have emphasised on the distributed database system which is non-homogeneous (i.e. heterogeneous).

The concept of Mobile Agent is based on the hypothesis of non-stationary agents. There are different significant benefits of using mobile agent in applications those are generally works in distributed environment. It is not so that the applications which are built using the mobile agent concept cannot build without mobile agent (using static agent). But the use of static agent will make the cost on higher side.

4.2 Mobile Code Paradigms

Before discussing the Mobile Agent, there is a basic need for understanding the Mobile Code Paradigm. Basically there are four basic types of Mobile Code Paradigms. They are as follows:

✓ Client/Server

✓ Code on Demand

✓ Remote Evaluation

✓ Mobile Agents

In case of all the systems mentioned above, the basic elements are:
Data: this basically contains the result-set fetched by the system while in execution.

Code: this is the section which includes the different modules necessary for the functionality of the system.

Program Stack: as the name indicates, it contains the current status of the program.

### 4.2.1 Client/Server Paradigm

It is the most widely used paradigm. Here, in this paradigm the services are offered by a server and the service is consumed by one or more, generally remote clients.

![Client/Server Paradigm](image_url)

Figure 4.1: Client/Server Paradigm

The figure 4.1 depicts the client/server paradigm. Here in the figure, in the client side, the "CODE" form which a procedure in the server is called and the server executes the procedure and returns the results back to the client.

Few examples are RPC, Web-Services, CORBA, EJBs etc. Here, in client/server system, the data element is mobile and both the code and the program stack element are static. The systems, which follow the
client/server paradigm, are easy to implement and hence this paradigm is very popular and wide-spreading.

4.2.2 Code on Demand Paradigm

In this paradigm, the server has different procedures for service to the client and client will get the service from the server. But the service, the client wants, is transferred to the client from the server. When the code for the particular processing is received by the client, the code is executed to perform the task for the client. Figure 4.2 represents the Code on Demand paradigm.

![Figure 4.2: Code on Demand Paradigm](image)

Here, in the figure, the “Code-1” is responsible for the request, from the client, for “Code-2” in the server. The server, on receiving the request, transfers the “Code-2” to the client. Client, on receiving the “Code-2”, it executes the code to carry out the task. In this paradigm, the data element and the program stack is static but the code element is mobile. Centralized code-base and simplicity in updating the softwares are main advantages of such type of systems.
4.2.3 Remote Evaluation Paradigm

In this paradigm, the client has the code for execution but transfers the code to the server and the server executes the code and returns the results to the client.

![Figure 4.3: Remote Evaluation Paradigm](image)

The figure 4.3 represents the Remote Evaluation paradigm. Here in this paradigm, the "Code" (in the client side), that is to be executed, is transferred to the server and the server, on receiving the "Code" executes it and sends the results back to the client. In this paradigm, the data element and the program stack is static but the code element is mobile.

4.3 Mobile Agent Paradigm

The Mobile Agent paradigm actually derives from two different disciplines and they are Artificial Intelligence and Distributed Systems. The artificial intelligence helped in creation the concept of an agent and the distributed system defines the concept of code mobility across a network. [7-10]

According to standard definitions, mobile agents have everything that a non-mobile agent has; agents are autonomous, reactive, proactive and
In addition to these mobile agents are also moveable, i.e. they can migrate between platforms in order to accomplish the assigned task.

In this paradigm, the case lies in remote execution of component. The component sends itself or another on-behalf of itself, to a remote host. The component moves with its code and data. In case of state of the state it may remain intact or modified depending upon the implementation. Unlike the case with remote execution, the component (i.e. mobile agent) will be deciding for itself that whether it wishes to move to an alternate location or not.

There is a no. of advantages of mobile agents over their static counterpart. Thus mobile agents include some of the benefits like reduction in communication costs, limited local resources, easier coordination, asynchronous computing etc.

Mobile agents, sometimes called mobile objects, constitutes of Code (behaviour), Data, Execution State and Itinerary. These elements are bundled together and are able to move as single unit. In case of stationary objects, which only consist of Code (behaviour) and Data. [11]

In case of both, the behaviour is represented by interfaces but in case of stationary objects as they do not move, the Code and Data can be platform dependent [11]. Mobile agents, on the other hand, can move and, therefore, their code, data, execution state and itinerary must all be portable, or at least convertible from and into portable forms.
Figure 4.4: Mobile Agent Paradigm

Figure 4.4 represents the concept of mobile agent paradigm. In the figure, the code in the client in initialized and after initialization the code is transferred to a server. When the mobile code receives the server the Mobile Agent System executes the code. After executing the code, the server then sends the code to another server for execution. And thus after reaching the last server the code is again executed and the overall result of all the executions is sent back to the client.

A mobile agent contains the following 3 components:

- Code - the program (in a suitable language) that defines the agent's behavior.
Chapter 4: Mobile Agent & Distributed Environment

- State - the agent's internal variables etc., which enable it to resume its activities after moving to another host.

- Data/Attributes - information describing the agent i.e.
  - its origin and owner,
  - its movement history,
  - resource requirements,
  - authentication keys etc.

  Part of this may be accessible to the agent itself, but the agent must not be able to modify the attributes.

The Code and State element/component are mobile. But the Data/Attribute is semi-mobile as only the necessary Data are mobile.

Mobile agent is a moving code along with data and program stack sections around distributed systems. This movement allows the code to be transported and executed in different locations/machines on behalf of its owner. So, the following points to be addressed while creating a mobile agent:

- Code transmission,
- Code execution
- Execution Environment assumptions (by the Code)
- Construction of Code
Owner of the Code

The figure 4.5 shows the mobile agent’s life-cycle model. The events in Mobile Agent’s life-time are as follows:

- **Creation**: a brand new agent is born and its state is initialized.
- **Dispatch**: an agent travels to a new host.
- **Cloning**: a twin agent is born and the current state of the original is duplicated in the clone.
- **Deactivation**: an agent is put to sleep and its state is saved in persistent storage.
- **Activation**: a deactivated agent is brought back to life and its state is restored from persistent storage.
- **Retraction**: an agent is brought back from a remote host along with its state to the home machine.
✓ Disposal: an agent is terminated and its state is lost forever.

✓ Communication: Notifies the agent to handle messages incoming from other agents, which is the primary means of inter-agent correspondence.

It is obvious that the systems based on mobile agent paradigm have to go for sequential programming. But it is also obvious that most agent systems also need the aspects of concurrency in communication and synchronization.

There are various programming languages which supports object movement between machines. But as in case of Expert System development, the design and development of mobile agent system from the scratch is very difficult and will take a lot of time, efforts. Also it will be a very costly affair. There are various agent development frameworks/environments, those provides the framework for developing and building an agent system with ease. But few of them may not support the mobility of agents. Also, the developer of agent systems should be very cautious while selecting a particular development environment as integration with non-agent systems at any moment will be necessary. So, if the language constructs of both the agent system and non-agent systems are compatible then it will be easier for the developer(s) to integrate them both.

4.4 Multi-Agent System

In general, in case of agent paradigm there is a very important factor which is the agent-to-agent communication. In agent paradigm, as already
mentioned, there must be an Agent Communication Language (ACL) to share information and knowledge between agents. Also there are different types of agents, as discussed in chapter 3, and therefore there are possibilities of communication and sharing of information between agents of different types. This leads to the concept of Multi Agent System (MAS).

Multi Agent System (MAS) can be defined as loosely-coupled networks of communicating and cooperating agents working together to solve problems that are beyond their individual capabilities. In order to obtain coherent system behaviour, individual agents in a MAS are not only able to share knowledge about problems and solutions, but also to reason about the processes of coordination among other agents. [12]

The choice of a proper tool to implement a Multi-Agent System can arm the developer with many advantages while, being careless about it, can prove to be constricting in the long run. The inherent difficulties encountered in implementing coordinated behaviour in any MAS are essentially the following [12]:

➢ **Communication:** how to enable agent communication, what communication protocols to use;

➢ **Interaction:** what language the agents should use to interact with each other and combine their efforts;

➢ **Coherence and Coordination:** how to ensure that the agents coordinate with each other to bring about a coherent solution to the problem they are trying to solve.
As in my research work, I was planned not only to work with mobile agent but also with static agent. Therefore I have to go for a Multi-Agent System which not only supports the mobile agent but also the static agent. Developing a Multi-Agent System from the scratch is very time consuming and a lot of efforts are to be made. For minimizing these kind of difficulties, different organizations, profit and non-profit, developed Agent Development Frameworks. The frameworks basically do not support the development of kinds of agents. So, the selection of the appropriate development framework depends on the requirements.

4.5 Mobile Agent Design Patterns

In every design and implementation there is always an issue with design pattern with which one should work. According to Software Engineering Approach, Pre-implementation phase is the design phase in case of every system design and implementation. There are different methods, according to Software Engineering, can be applied for designing a system.

Though in different case studies [13-16], the mobile agent technology has been successfully applied but the restrictions in its application in practical scenarios still exists. Different design patterns on mobile agents have been proposed in [17,18].

4.5.1 Itinerary Design Pattern

In this design pattern [18], the agent receives an itinerary on the source agency which indicates the sequence of agencies/servers it should visit. In each of the agencies/servers, the agent executes its task locally and then
continues on its itinerary. After visiting the last agency/server, the agent then goes back to its source agency/server. This design pattern is depicted in figure 4.6.

![Figure 4.6: Itinerary Design Pattern](image)

4.5.2 Star-Shaped Design Pattern

In Star-Shaped design pattern [18], the agent has a list of agencies/servers to migrate. The agent then migrates to the first destination agency/server, executes the task, going back to the source agency/server. The agent repeats this until it visits the last agency/server in its list. Figure 4.7 shows this pattern.

![Figure 4.7: Star-Shaped Design Pattern](image)
4.5.3 Branching Design Pattern

In the Branching design pattern [18], the agent receives a list of agencies/servers to visit. It then clones itself according to the numbers of agencies/servers in the itinerary. Each of the clones has to execute its corresponding task in an agency/server respectively. And the clones notify the source agency/server when the respective task of each of them is completed. Here, in this pattern, parallel execution of the task/tasks can be achieved. This pattern is shown at figure 4.8.

![Figure 4.8: Branching Design Pattern](image)

4.5.4 Master-Slave Design Pattern

Here in this Master-Slave design pattern [19], there are two types of agents, the master agent and the other is the slave agent. The master agent gives a task to be done in a given agency/server to a slave agent. The slave agent then visits the indicated agency/server to accomplish its task and then returns with the results to the source agency/server. The master agent receives the results from the slave agent and then the slave destroys himself.
4.5.5 MoProxy Design Pattern

In this design pattern, it is proposed that when an agent needs a resource, it requests to the Resource Granter, indicating the desired permissions. Then, the resource granter returns a mobile proxy for the agent in order to access the resource with the desired permissions depending on the restrictions of the resource. [20]

4.5.6 Meeting Design Pattern

The Meeting pattern, suggests a way to promote local interactions between agents distributed on the net. Such interactions make possible the execution of given tasks, as well as the optimization of results. Then, the Meeting Agent, who will meet others, has a Meeting object that keeps the place and the identification of the meeting. In this way, the Meeting Agent requests the Meeting object the place of the meeting and then migrates to it. A Meeting Manager, which is an entity that manages the meeting, is responsible for notifying the agents located in the meeting place about the arrivals and exits of new ones. The Meeting object is responsible for inter­mediating the register of the agent on the manager. [21]

4.5.7 Facilitator Design Pattern

The Facilitator pattern [19], defines a service that provides a name service and localization of agents with specific abilities, thus facilitating the localization of a given agent.
4.5.8 Mutual Itinerary Recording Design Pattern

This pattern, is a general schema that guarantees the itinerary of a given agent will be registered and tracked by other cooperative agent and vice-versa, in a disposal of mutual support. When an agent is moving between platforms, it carries the information from the last platform, the current and the next ones to the cooperative agent through an authenticated channel. The agent keeps the register of the itinerary and it always compares the itinerary that it possesses with the received one. When an inconsistency is detected it should be treated. For instance, it would either disallow the agent to visit the platform that caused the inconsistency, or suspend the functioning of an agent, or send the agent back to the source agency. [18]

4.6 Distributed Environment

I have used the term “Distributed Environment” in my research work, as because the question-fact database is distributed in different servers. And also, it is considered that the DBMSs running in the servers may not be the same. So, by Distributed Environment, I want to mean the Distributed Database Environment.

A distributed database is logically a single database but actually it spreads in different computers in a network. In case of distributed database there must be multiple DBMSs running at each computer and there exist co-operations between them.

The Figure 4.9 represents the range of distributed database environments. These environments are briefly explained by the following:
4.6.1 Homogeneous Distributed Database

This environment architecture is very simple and therefore it is much easy to design and manage. In this environment all sites/node must have the same DBMS. Also all the sites should agree to cooperate in processing user requests. This kind can be further divided into two categories:

(a) Autonomous: In autonomous kind, each DBMS at each node/site works independently. They pass messages back and forth to share data updates.

(b) Nonautonomous: In nonautonomous kind, there must be a central or master DBMS, who coordinates database access and update across the nodes/sites.

But in general, each site/node has to surrender some part of its autonomy in case of right to change the schema or the software.
The figure 4.10 [22] depicts a non-autonomous homogeneous distributed database environment. There are few characteristics those define this kind of environment and they are as follows:

- Data are distributed across all the nodes.
- At each node/site same DBMS is used.
- The distributed DBMS manages all data.
- All users access the database through one global schema (database definition).
- This global schema is just the union of all the local database schemas.
4.6.2 Heterogeneous Distributed Database

Here, in this kind of distributed database environment, the sites/nodes are not bound to use the same schema and software. Since there are differences in schemas at the sites/nodes, therefore problem arises in the processing of queries and transactions. Also, the sites may not be aware of each other.

Heterogeneous systems are useful when each of the sites/nodes may have their own hardware, software and data structure which may not be compatible. In heterogeneous system, translations are required to allow communication between different sites/nodes.

From the above figure 4.9 [22-24]:

(A) **Systems:** Supports some or all of the functionality of one logical database.

- **Full DBMS Functionality:** Supports all of the functionality of a distributed database, as discussed in the remainder of this chapter.
- **Partial-Multidatabase:** Supports some features of a distributed database, as discussed in the remainder of this chapter.
  - **Federated:** Supports local databases for unique data requests.

  - **Loose Integration:** Many schemas exist, for each local database, and each local DBMS must communicate with all local schemas.
Chapter 4: Mobile Agent & Distributed Environment

- **Tight Integration:** One global schema exists that defines all the data across all local databases.

  - **Unfederated:** Requires all access to go through a central coordinating module.

(B) **Gateways:** Simple paths are created to other databases, without the benefits of one logical database.

![Figure 4.11: Heterogeneous distributed database environment (Adapted from Bell and Grimson, 1992)](image)

The figure 4.11 depicts a heterogeneous environment which can be defined by the following characteristics [22]:

- Data are distributed across all the nodes.

- Different DBMSs may be used at each node.

- Some users require only local access to databases, which can be accomplished by using only the local DBMS and schema.
✓ A global schema exists, which allows local users to access remote data.

The adoption of a particular Distributed Database Environment for the design development of a system is basically depends on the conditions/flexibilities demanded by the system to be developed.

REFERENCES:


Chapter 4: Mobile Agent & Distributed Environment


[12] A Multi-Agent Architecture for Intelligent Tutoring, Nicola Capuano, Massimo De Santo, Marco Marsella, Mario Molinara, Saverio Salerno.


