Chapter 5: Agent Development Frameworks & JADE

5.1 Introduction

Agent based technologies are becoming one of the most fruitful and promising technological paradigms. But until and unless there must be further development in standards to support agent interoperability and also in development of adequate environments for the development of agent systems, the agent based paradigm cannot realize their full potential and will not become widespread. For standardization of agent technologies, a lot of organizations are continuously working. The first standardization effort starts from the work done by Knowledge Sharing Effort [1]. FIPA represents one of most interesting answers to the need for standards [2].

A lot of research from a no. of people and commercial organizations are involved in the realization of agent applications and development environments to build agent systems. These research efforts results in some agent development environments as outcome. For example AgentBuilder, dMARS, MOLE, Voyager, SPRINGS, RESTINA, ZEUS, Aglets, Ajanta and JADE etc. are some of the agent development environments.

5.2 Agent Development Frameworks

Every day, developers have the problem of constructing ever larger and more complex software applications. Developers are now creating enterprise-wide and world-wide applications that must operate across corporations and continents and inter-operate with other heterogeneous
systems. Such applications are difficult to produce with traditional software technologies because of the limits of these technologies in coping with dynamically changing and unmanaged environments. It would appear that agent-based technologies represent a promising tool for the deployment of such applications because they offer the high-level software abstractions needed to manage complex applications and because they were invented to cope with distribution and inter-operability. [3-8]

In recent years, the agent technology is becoming an important and exciting field in computer science. In the midst of different types of agents, the mobile agent technology is getting more and more interesting and of more importance as many commercial infrastructure is using this technology en an effective manner [1-4]. There are large no. of software products with different toolkits and platforms of different qualities and maturity.

5.2.1 AgentBuilder

The AgentBuilder [9] is a tool, which is developed is Java, used to build Java agent systems. It is consists of two components:

(1) The ToolKit and

(2) The Run-Time System

The ToolKit includes tools to manage the agent software development process. It is also used for analysing the domain of agent operations, defining, implementing and testing agent software.
The Run-time System of AgentBuilder is the agent engine. Actually it is an interpreter which is used execution environment of agent.

In AgentBuilder, agents usually can communicate through KQML messages but there is a scope for defining new communication commands to with the needs.

5.2.2 dMARS

dMARS [10] is an environment for agent-oriented development and implementation. It is basically used for building distributed systems those are based on BDI agent model offering support for system configuration, design, maintenance and reengineering. In the fields of Air-Traffic Control, Telecommunication and Business Process Management, this development environment was used to realize application.

5.2.3 MOLE

MOLE [11] is another agent development system framework developed is Java. But in MOLE, the agents do not have sufficient set of features which are required for a truly agent systems. But it was very much important because it offers solution to agent mobility. MOLE agents' interaction is done by two communication types:

1. Through RMI in case of Client/Server interactions, and

2. In case of peer-to-peer interactions message exchange method is used.
5.2.4 RESTINA

RESTINA [12] is basically used to develop agents which are reusable. Reusability is a very essential feature of any development system. In case of execution of tasks and request from other agents, RESTINA agents have four reusable modules for:

1. Communicating,
2. Planning,
3. Scheduling and
4. Monitoring

In RETSINA, there are three kinds of agents. They are:

1. interface agents to manage the interaction with users,
2. task agents to help users in the execution of tasks, and
3. information agents to provide intelligent access to heterogeneous collections of information sources.

5.2.5 Zeus

One of the environments for rapid development of Java agent systems is Zeus [13]. Zeus provides a library of components for agents. It also provides visual environments for inputting users’ specifications and also provides an environment for building agents which includes an automatics agent code generator. Apart from that the environment also includes a
collection of classes that form the building blocks of individual agents. Zeus agents are primarily composed of five layers namely:

1. API Layer: This layer allows the interaction with outer world where no agent can exist.

2. Definition Layer: This layer's prime duty is to manage the tasks that an agent must perform.

3. Organizational Layer: It's main role is to manage the knowledge about other agents.

4. Coordination Layer: The name is itself tells the meaning, i.e., it manages coordination and negotiations to be made with other agents.

5. Communication Layer: It is basically manages the communications with other agents.

5.2.6 Aglets Software Development Kit

The Aglets Software Development Kit (ASDK) is an environment for programming mobile Internet agents in Java [14]. It was developed originally by Mitsuro Oshima and Danny Lange. This was basically a project, which earlier name was AWB (Aglets WorkBench). Though it was initially developed by IBM in 1997 but in later time it had been maintaining by the open source community since 2001.
Aglets is basically a platform for mobile agents. Since, Aglets is completely written in Java so it provides a high portability of both the agents and the platform and it follows the MASIF specification [15]. It is based on single-thread model for the agents and the infrastructure for communication is basically based on message passing where both the synchronous and asynchronous messages are supported. The main components of Aglets are:

1) a Mobile Agent Platform which is completely Java based with a stand-alone server called Tahiti,

2) a library which allows the agent developers to build mobile agents.

The agent developed using Aglets technology is called an aglet which is a mobile Java object. It also can such be implemented so that it autonomously and spontaneously move from one host to another. Implementation will decide how the mobile agent will behave in different remote hosts while in execution. As security concerns, the security is taken care of by using the Java based security implementations while accessing the local resources ate the remote hosts with the authorization required. The ASDK is basically a development framework and environment for designing and developing mobile agents.

5.2.7 Ajanta

Ajanta [16,17] is a Java-based framework for developing mobile agents. It is a system for programming agent based applications over the Internet. This framework is developed to serve as an infrastructure for research in secure distributed programming using mobile autonomous agents.
Ajanta is completely implemented using the Java language and its security mechanisms. The other facilities of Java, like object serialization, reflection and RMI etc., are also used by the Ajanta. Thus Ajanta has a no. of design choices as well as mechanisms which are greatly influenced by Java computation model and security architecture. The binary is free but it has poor GUI.

Ajanta supports JDK 1.1.x and Perl is required for its installation. The communication mechanisms used here are - Java Method Invocation, authenticated RMI, ATP, agent collaboration XML etc. The mobility is weak mobility. [18,19]

### 5.2.8 Tryllian’s Agent Development Kit

Tryllian’s Agent Development Kit (Tryllian’s ADK) is a complete development and implementation environment. This is developed by the homonym company in 2001. It is based on a sensing-reasoning action mechanism. It allows programmers to define a reactive and proactive behaviour of agents. The programming model of Tryllian is taskbased. [20]

Here, the communication among agents is achieved through message passing and in accordance with the FIPA standard. The other standards supported are - SOAP, XML, JXTA, JNDI.

In Tryllian environment an agent application typically consist of several Habitats spread across a number of servers, each hosting a number of Rooms. A Habitat is a collection of one ore more Rooms that share a Java Virtual Machine. It provides services such as agent lifecycle management,
communication, inter-Habitat travel, Room and agent persistence and the basic security model. It supports strong mobility, i.e., the agents can move to other Habitats during runtime, taking along their (new) Java code and state. [20]

5.2.9 JACK Intelligent Agent

JACK Intelligent Agent is an agent oriented development environment fully integrated with the Java programming language. JACK provides agent-oriented extensions to the Java programming language. [21]

The JACK provides a multi-agent platform and was written for commercial application implementing the multi-agent paradigm. JACK supports FIPA standard but does not support mobility. For communication, JACK needs DCI network, which is similar to TCP/IP. The security in JACK is basically internal and is provided by JDK.

5.2.10 Grasshopper

Grasshopper [22] is a mobile agent development and runtime platform. It is fully implemented in Java. It is basically an integration of traditional client/server paradigm and mobile agent paradigm. Its design is based on OMG-MASIF standard (discussed in chapter 4). But the latest Grasshopper version is also compliant with the specifications of the FIPA.

In Grasshopper, the agents usually can communicate through ACL. The communication may be of synchronous or asynchronous type. It also
supports multi-cast communication and different transport protocols like RMI, IIOP, Sockets etc [22].

5.2.11 Voyager

Voyager was developed initially by ObjectSpace in 1997 [23]. But it is now maintained by the Recursion Software (www.recursion.com). It is basically a distributed computing middleware. Simplification in the management of remote communications of the traditional CORBA and RMI protocols is the main focus area of Voyager.

Voyager offers various important facilities which include the dynamic generation of CORBA proxies, mobile code and mobile agents etc. Voyager provides location transparency through forwarding chains of proxies [23].

Thus Voyager can be considered to be one of the important platforms those offers a large no. of functionalities which can make the development process of distributed systems much easier as compared to the other platforms [23]. The major disadvantage of Voyager is that it is a commercial product. The Recursion Software does not provide any free optimal version of Voyager for academic purpose and which prevent the researchers from using it.

5.2.12 SPRINGS

The Distributed Information Systems Group at the University of Zaragoza of Spain developed SPRINGS [24]. The official site of SPRINGS is
http://sid.cps.unizar.es/SPRINGS. The development of SPRINGS is inspired by the features of platforms like Voyager, Grasshopper etc.

The main focus of SPRINGS is on the scalability and reliability in the situations related with a moderate and high no. of mobile agents’ existence in a system. In case of movements of mobile code or mobile agent, the programmer does not need to specify network addresses but just the name of the destination as the mapping is dynamic and does not rely on configuration files. The experimental results in [24] show that SPRINGS outperforms other platforms in scenarios with a high number of very dynamic mobile agents. Some of the major disadvantages of SPRINGS are mentioned below:

- the agent platform does not conforms with FIPA standard,
- the security mechanism provided is not very sophisticated,
- it does not provide any graphical tool for the users,
- also it is a known fact that without a proper documentation one cannot use any kind of software product. Here is the same case happening with SPRINGS as there is little documentation available about it.

Apart from the different agent development frameworks/environments discussed above there are other agent development frameworks such as:

- JADE (Java Agent Development Framework) is a software development framework aimed at developing multi-agent systems
and applications conforming to FIPA standards for intelligent agents. [32]

- April Agent Platform (AAP) is a FIPA-compliant lightweight and powerful solution for developing agent-based systems. It is implemented using the April language (majority of agent platforms is implemented using Java) and IMC. [26]

- Comtec Agent Platform (CAP) is an open-source, free implementation of FIPA standard. Its unique is the implementation of FIPA Ontology Service and Agent/Software Integration, which require SL2 as the content language. This platform provides minimal documentation that may lead to many difficulties to use. [26]

The different agent environment/tools, mentioned above, offer their individual architecture for agents. But no one of them can be used to design and develop efficient and reusable agent software. They are not of such type that every kind of agents can be built. Some of them have supports for some kind of agents others may not have that support.

As I have mentioned in earlier chapters that in my research work I have used JADE as the agent development framework. Now, in the section 5.3, which follows immediately, I will discuss about JADE in detail.
5.3 JADE: An Introduction

JADE (Java Agent Development Framework) is a software development framework. It supports the FIPA standard and also has supports for development of multi-agent systems and applications. It basically includes: a runtime environment, a library of classes agent for development and a graphical suite of tools that give an easy way for administrating and monitoring the activity of running agents.

JADE is written in Java language and is made of various Java packages, giving application programmers both ready-made pieces of functionality and abstract interfaces for custom, application dependent tasks. Java was the programming language of choice because of its many attractive features, particularly geared towards object-oriented programming in distributed heterogeneous environments; some of these features are Object Serialization, Reflection API and Remote Method Invocation (RMI). [33]

JIPMS (JADE Inter-platform Mobility Service) is a service which allows Inter-platform mobility support to JADE. It is an extra module, which does not come with JADE package and can be downloadable from the net. [33]

5.4 Components of JADE

JADE basically includes [27]:

✓ A runtime environment where JADE agents can "live" and that must be active on a given host before one or more agents can be executed on that host.
A library of classes that programmers have to/can use (directly or by specializing them) to develop their agents.

A suite of graphical tools that allows administrating and monitoring the activity of running agents.

5.4.1 JADE Run-time Environment

5.4.1.1 Containers and Platforms

Container is a running instance of the JADE Run-time Environment. A container can hold multiple agents. A platform is basically the set of active containers. There is only one special container, called Main Container, which should always be remained active in a platform. It can be said that the Main Container is the parent of all the other containers because when a container starts it must register with the Main Container. Developers using JADE may not have to know how the JADE runtime environment works. They only have to know how to start the environment and should start it before executing their agents.

5.4.1.2 AMS and DF

The Main Container also holds two special agents, namely AMS and DF, which are automatically started when the Main Container starts.

The AMS (Agent Management System) that provides the naming service (i.e. ensures that each agent in the platform has a unique name) and represents the authority in the platform (for instance it is possible to create/kill agents on remote containers by requesting that to the AMS). This
tutorial does not illustrate how to interact with the AMS as this is part of the advanced JADE programming. [27]

The DF (Directory Facilitator) that provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals. [27]

5.4.2 Packages in JADE

JADE is solely written in Java language and therefore it includes different Java packages. These packages give the developers of agent systems different ready-made pieces of functionalities. They also provide different interfaces for customized application dependent task. The main packages among these packages are as follows [28]:

- The "package jade.core" is for the implementation of kernel of the system. This package includes the Agent class which needs to be extended by the developers for their implementation of respective agents. It also contains a hierarchy of classes for the implementation of Behaviours.

- The Agent Communication Language processing is provided by the sub-package "jade.lang.acl" which conforms to the FIPA standard.

- The Java classes under the package "jade.domain" are for the representation of the entities for agent management defined by the FIPA standard. The main entities involved here are the AMS and the DF. The sub-packages, namely
Chapter 5: Agent Development Frameworks & JADE

- "jade.domain.FIPAAgentManagement" contains the classes which represent the Agent Management Ontology according to FIPA.

- "jade.domain.JADEAgentManagement" the classes for the JADE extensions for Agent Management and representation of its concepts.

- "jade.domain.introspection" contains the concepts used for the domain of discourse between the JADE tools and the JADE kernel.

- jade.domain.mobility contains all concepts used to communicate about mobility.

➤ The package "jade.gui" contains a set of classes which provides the facilities for the creation of GUIs for displaying and editing of the Agent Identifiers, Agent Description, ACL Messages etc.

➤ For the implementation of the Message Transport Protocols the package "jade.mtp" provides all the needful facilities.

➤ For standard interaction protocol modelling the package "jade.proto" provides classes. These classes are also helpful for the application developers to create their own protocols.
5.4.3 Tools in JADE

There are some tools in JADE which are helpful for the administration of the platform and development of applications. These tools are contained in a separate package, named “jade.tools” [28]. The table 5.1 represents the tools which are of importance:

Table 5.1: Important Tools in JADE

<table>
<thead>
<tr>
<th>Name of the Tool</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote Management Agent (RMA)</td>
<td>It acts as a graphical console for platform management and control. The “-gui” option is required in the command line for creation of the 1\textsuperscript{st} instance of an RMA.</td>
</tr>
<tr>
<td>Dummy Agent</td>
<td>This is a monitoring and debugging tool with graphical user interface. This is used for composing ACL messages and sending the messages to other agents. This also provides display the list of all the ACL messages sent or received.</td>
</tr>
<tr>
<td>Sniffer</td>
<td>It is an agent which intercepts ACL messages when they are in fly and displays them graphically using a notation similar to UML sequence diagrams.</td>
</tr>
<tr>
<td>Introspector</td>
<td>This agent is useful for monitoring the life cycle of an agent, its exchanged ACL messages and the behaviours in execution.</td>
</tr>
<tr>
<td>DF GUI</td>
<td>It is also a GUI which is used not only by the default Directory Facilitator of JADE but also by every other DF that the user might need.</td>
</tr>
<tr>
<td>LogManagerAgent</td>
<td>This is an agent is used for the setting of runtime logging information for both JADE and application</td>
</tr>
</tbody>
</table>
### 5.5 Features of JADE

The different features that JADE offers for agent development are as follows [29]:

- **FIPA-compliant Agent Platform**, which includes the AMS (Agent Management System), the DF (Directory Facilitator), and the ACC (Agent Communication Channel). All these three agents are automatically activated at the agent platform start-up;

- **Distributed agent platform**: The agent platform can be split on several hosts (provided that there is no firewall between them). Only one Java application, and therefore only one Java Virtual Machine, is executed on each host. Agents are implemented as one Java thread and Java events are used for effective and light-weight communication between agents on the same host. Parallel tasks can be still executed by one agent, and JADE schedules these tasks in a more efficient (and even simpler for the skilled programmer) way than the Java Virtual Machine does for threads;

| SocketProxyAgent | This simple agent acts as bi-directional gateway between JADE platform and a TCP/IP connection. This agent is useful, e.g. to handle network firewalls or to provide platform interactions with Java applets within a web browser. |
A number of FIPA-compliant DFs (Directory Facilitator) can be started at run time in order to implement multi-domain applications, where the notion of domain is a logical one as described in FIPA97 Part 1;

Programming interface to simplify registration of agent services with one, or more, domains (i.e. DF);

Transport mechanism and interface to send/receive messages to/from other agents;

FIPA97-compliant IIOP protocol to connect different agent platforms;

Light-weight transport of ACL messages inside the same agent platform, as messages are transferred encoded as Java objects, rather than strings, in order to avoid marshalling and un-marshalling procedures. When sender or receiver does not belong to the same platform, the message is automatically converted to/from the FIPA compliant string format. In this way, this conversion is hidden to the agent implementers that only need to deal with the same class of Java object;

Library of FIPA interaction protocols ready to be used;

Automatic registration of agents with the AMS;

FIPA-compliant naming service: at start-up agents obtain their GUID (Globally Unique Identifier) from the platform;
Graphical user interface to manage several agents and agent platforms from the same agent. The activity of each platform can be monitored and logged.

In JADE, the FIPA platform architecture is hidden from the programmer. There is no requirement of implementation of the Agent Platform and Agent Services. No agent programmer needs to implement the agent-management ontology and functionalities like:

- Agent registration with Agent Platform,
- Agent naming and addressing, and
- Accessing the services of the Directory Facilitator (DF) for registration, searching etc.

The above functionalities are built-in and some of them have interfaces as necessary. Also, the implementation of Message Transport, Parsing are not required as the framework does these automatically and efficiently when messages are sent and received. If the programmer wants to implement their own Interaction Protocol then they must extend the built-in Interaction Protocols via handle methods.

5.6 JADE Communication System

There is a front-end container in JADE which maintains an RMI registry internally. This registry is required by the other agent containers to register them-selves at the bootstrap time. The front-end container also maintains an Agent Global Descriptor Table which maps each agent name with its AMS
data with container's RMI object reference. Generally there are two types of communication related to agents. One is intra-platform communication and the other one is the inter-platform communication. The figure 5.1 represents the JADE Messaging Architecture.

![Figure 5.1: JADE Messaging Architecture](image)

The first case, as the name itself suggests, is related communication between agents those are living in the same platform. In this case JADE uses the internal message transport protocols (IMTPs) for implementing delivery services. Here also, there are two situations: (1) when both the sender and the receiver agents are living in the same container, and (2) when the sender and the receiver agents are in different containers but in the same platform. In case (1), JADE uses event passing to deliver messages. Here the message is not serialized, but cloned and new object reference is passed to the receiver. In the case (2), JADE uses RMI.

In case of the inter-platform communication, the Agent Communications Channel (ACC) plays a vital role. The ACC is basically physically distributed
across all the containers in a platform. In may be noted that each of the containers can be launched with one or more MTPs. The platform also is able to internally route messages and select the best MTP for each situation. There are MTPs which are currently available for JADE [30]:

- CORBA IIOP MTP based on standard Sun ORB provided with the JDK.
- CORBA IIOP MTP based on ORBACUS [31].
- HTTP-based MTP.

### 5.7 JADE Agent Model

In JADE, there is class called "Agent" which is the base class for all the user defined agent classes. So, creating a JADE agent is as simple as defining a class extending the "jade.core.Agent" class and implementing the "setup()" method on their own. Each agent is given an identifier, AID, and is composed of a unique name plus an address [32].

The form of agent names is

```
<local-name>@<platform-name>
```

Also the form of the default platform name is

```
<main-host>:<main-port>/JADE
```

But, within a single JADE platform agents are referred through their names only. A JADE agent can be in one of several states, according to Agent
Platform Life Cycle in FIPA specification (discussed in chapter 3); these are represented and detailed below [28]:

- **INITIATED**: the Agent object is built, but hasn't registered itself yet with the AMS, has neither a name nor an address and cannot communicate with other agents.

- **ACTIVE**: the Agent object is registered with the AMS, has a regular name and address and can access all the various JADE features.

- **SUSPENDED**: the Agent object is currently stopped. Its internal thread is suspended and no agent behaviour is being executed.

- **WAITING**: the Agent object is blocked, waiting for something. Its internal thread is sleeping on a Java monitor and will wake up when some condition is met (typically when a message arrives).

- **DELETED**: the Agent is definitely dead. The internal thread has terminated its execution and the Agent is no more registered with the AMS.

- **TRANSIT**: a mobile agent enters this state while it is migrating to the new location. The system continues to buffer messages that will then be sent to its new location.

"Autonomy", the distinguishing property of a software agent according to which an agent is not only limited to external stimuli, but also able to start a new communicative act autonomously. For this, there must be an internal
thread of control per agent. However, an agent can engage multiple simultaneous conversations.

An agent should have to carry out tasks on some-behalf. The tasks are may be concurrent in nature in response to various external events. The Behaviour abstraction is used by JADE to model the tasks to be performed by agents. Thus, in general, tasks of agents are represented as behaviours. The agents should instantiate their behaviours according their needs. In JADE, Behaviours are created by extending “jade.core.behaviours.Behaviour” class. JADE uses a single thread per agent instead of single thread per behaviour. This basically to reduce the no. of threads required to run the agent platform.

In JADE there are different Behaviour sub-classes, some of them are shown in the figure 5.2. Developers can implement their agent specific tasks by defining one or more Behaviour subclasses. Then instantiate them and add these behaviour objects to the agent task list. There are two methods in the Agent class those allow to manage the ready tasks queue of a specific agent:

- addBehaviour(): to add Behaviour to the Behaviour queue of the respective agent,
- removeBehaviour(): to remove Behaviour from the Behaviour queue when the task is completed.
Chapter 5: Agent Development Frameworks & JADE

Behaviour

Models a complex task i.e. a task that is made up by composing a number of other tasks.

ComposedBehaviour

Models a simple task i.e. a task that is not composed of sub-tasks.

SimpleBehaviour

Models an atomic task (its done() method returns true).

OneShotBehaviour

Models a cyclic task (its done() method returns false).

CyclicBehaviour

Models a complex task whose sub-tasks are executed sequentially.

SequentialBehaviour

Models a complex task whose sub-tasks are executed concurrently.

ParallelBehaviour

Models a complex task whose sub-tasks correspond to the activities performed in the states of a Finite State Machine.

FSMBehaviour

Figure 5.2: UML Model of the Behaviour class hierarchy (adapted from B. Fabio, C. Giovanni, T. Tiziana, R. Giovanni, 2007)

Developer must implement two basic methods in the defined Behaviour subclass, i.e., each Behaviour class must contain the following methods:

- public void action(): the task associated to the behaviour.

- public boolean done(): to check whether the behaviour is completed or not.
Developer must implement two basic methods in the defined Behaviour subclass, i.e., each Behaviour class must contain the following methods:

- **public void action()**: the task associated to the behaviour.

- **public boolean done()**: to check whether the behaviour is completed or not.
The scheduler is implemented in the base Agent class and is hidden to the developer. The scheduling policy of this scheduler is round-robin non-preemptive scheduling policy among the behaviours which are in the queue.

The following figure (Figure 5.3) represents the Agent Execution Model.

Figure 5.3: Agent Execution Model (adapted from C. Giovanni, 2009)
In short, the execution of behaviours [33]:

- Each agent has its own thread.

- Agent's core uses simple queue for sequential execution of behaviours.

- Until the "action()" of the current behaviour returns, no other behaviour can be executed.

- If behaviour is supposed to have more iterations/steps, its "done()" method has to return false. Then, the behaviour will be put in the end of the queue and its action will be called again when time comes.

- If behaviour's "done()" returns true, the behaviour is removed as finished.

5.8 JADE Agent Mobility & Cloning

As discussed in earlier chapters, by agent mobility the movement of agent program is meant basically along with data. There are two situations related to migration of agent program and they are:

- migration from one container to another those are within the same platform, called intra-platform mobility,

- migration from one host to another host (may be in the same network or in another network).
Basically, JADE supports intra-platform mobility and cloning. The base class, Agent (jade.core package), contains a set of methods which are used for managing agent mobility. The methods are –

- **doMoveQ**: This method needs to be called to move an agent by passing the new destination location as parameter. The location is referred by a Location object.

- **beforeMoveQ**: The tasks to be performed before the migration of the agent.

- **afterMoveQ**: The tasks to be performed after the migration of the agent.

Apart from migration capability of Agent (the Mobile Agent), there is another capability, known as Agent Cloning. As the name itself suggests, Cloning of agent means creation an exact copy of the agent itself. JADE supports agent cloning. Like agent mobility, JADE also contains a set of methods which are used for managing agent cloning. The methods are –

- **doCloneQ**: This is method which facilitates the cloning of an agent. Whenever there is a requirement for an agent to cloned, then the "doCloneQ" method needs to be called. It takes as parameters: the location (where the cloned agent will start) and the name to be given (to the cloned agent).

- **beforeCloneQ**: The tasks to be performed before cloning of the agent.

- **afterCloneQ**: The tasks to be performed after cloning of the agent.
In my research work the support of the agent mobility between two platforms (inter-platform) is required. So, to have this support I have used a library, known as Jade Inter-Platform Mobility Support (JIPMS). JIPMS is a service which allows Inter-platform mobility support to JADE. It is an extra module, which does not come with JADE package and can be downloadable from the net [25].

REFERENCES:


Documento de JADE 3 (2003).


