“An agent is a software thing that knows to do things that you could probably do yourself if you had the time.”

- Ted Selker of the IBM Almaden Research Center
The sections of this chapter cover the design methodology of the novel approach proposed in the thesis to migrate agents. This proposal is described in the paper[45]. In addition, the protocol and ontologies used are also outlined.

4.1 JADE PLATFORM

The platform chosen for implementing migration was JADE[51], because it is a widely adopted platform within the software agent development and research communities. It is an open source and complies with FIPA[51] specifications. More details are given in chapter 3.

The platform is divided into a large number of functional modules[44], which can be placed into three categories in general terms:

- **Core.** The core of the platform is formed by all the components providing the necessary execution environment for agents' functioning.

- **Ontologies and Content Languages administration.** This consists of the agency's mechanisms for carrying out information processing in ACL messages, and the internal structures that the agency and agents will use to represent this content.

- **Message transport mechanisms.** Mechanisms and protocols used to send and receive messages at both intra-agency and inter-agency level.

At the core of the JADE platform is the concept of the container, which is the minimum execution environment necessary for an agent to operate. Each container in JADE is executed in a different Java virtual machine, but they are all interconnected by RMI (Remote Method Invocation).

In order to understand the migration proposal (discussed in the section 4.2), some understanding regarding the terms associated with agents is required, which is done in the next section.

4.2 AGENT MANAGEMENT

In addition to communication, the second fundamental aspect of agent systems addressed by the early FIPA specifications is agent management, a normative framework within which FIPA-compliant agents can exist, operate and be managed. It establishes the logical reference model for the creation, registration, location, communication, migration and
operation of agents. The agent management reference model consists of the components[32] depicted in Fig.4.1.

![Diagram of the agent management ontology]

**Fig.4.1:** Depiction of the agent management ontology

### 4.2.1 Agent Platform (AP)

This provides the physical infrastructure in which agents are deployed. The AP consists of the machines, operating systems, FIPA agent management components (described below), the agents themselves and any additional support software. The specific internal design of an AP is left to the developers of an agent system and is not a subject of FIPA standardization beyond the components discussed here. As a single AP may be spread across multiple computers, resident agents do not have to be co-located on the same host.

### 4.2.2 Agent

An agent is a computational process that inhabits an AP and typically offers one or more computational services that can be published as a service description. The particular design of these services, otherwise known as capabilities, is not the concern of FIPA, which only mandates the structure and encoding of messages used to exchange information between agents (and other third-party technologies if FIPA compliant). An agent must have at least one owner and must support at least one notion of identity which can be described using the FIPA Agent Identifier (AID) that labels an agent so that it may be distinguished unambiguously. An agent
may be registered at a number of transport addresses at which it can be contacted.

4.2.3 Directory Facilitator (DF)

The DF is an optional component of an AP providing yellow pages services to other agents. It maintains an accurate, complete and timely list of agents and must provide the most current information about agents in its directory on a non-discriminatory basis to all authorized agents. An AP may support any number of DFs which may register with one another to form federations.

Every agent that wishes to publicize its services to other agents should find an appropriate DF and request the registration of its agent description. There is no intended future commitment or obligation on the part of the registering agent implied in the act of registering. Agents can subsequently request the deregistration of a description at which time there is no longer a commitment on behalf of the DF to broker information relating to that agent. At any time, and for any reason, an agent may request the DF to modify its agent description. In addition, an agent may issue a search request to a DF to discover descriptions matching supplied search criteria. The DF does not guarantee the validity of the information provided in response to a search request. However, the DF may restrict access to information in its directory and will verify all access permissions for agents which attempt to inform it of agent state changes.

4.2.4 Agent Management System (AMS)

The AMS is a mandatory component of an AP and is responsible for managing the operation of an AP, such as the creation and deletion of agents, and overseeing the migration of agents to and from the AP. Each agent must register with an AMS in order to obtain an AID which is then retained by the AMS as a directory of all agents present within the AP and their current state (e.g. active, suspended or waiting). Agent descriptions can be later modified, under restriction of authorization by the AMS. The life of an agent with an AP terminates with its deregistration from the AMS. After deregistration, the AID of that agent can be removed by the directory
and can be made available to other agents who should request it. Agent descriptions can also be searched for within the AMS, and the AMS is the custodian of the AP description that can be retrieved by requesting the action get-description.

The AMS can request that an agent performs a specific management function, such as to terminate its execution, and has the authority to enforce the operation if the request is ignored. Only a single AMS can exist in each AP and if the AP spans multiple machines, the AMS is the authority across all those machines.

4.2.5 Message Transport Service (MTS)

The MTS is a service provided by an AP to transport FIPA-ACL messages between agents on any given AP and between agents on different APs. Messages are providing a transport envelope that comprises the set of parameters detailing, for example, to whom the message is to be sent. The general structure of a FIPA-compliant message is depicted in Fig.4.2.

![Fig. 4.2 : FIPA message structure](image)

4.3 PROPOSAL FOR MIGRATION USING ACL

The idea of creating a migration using ACL messages came from FIPA’s specification regarding mobility, where this type of migration is proposed[45]. This specification only gives a general outline of the ontologies, the protocol, and the life cycle of a mobile agent. It has not been updated due to the lack of implementations and has become obsolete within the FIPA specifications. For these reasons, it is found that there is a need to propose extensions to the specification to cover situations that it does not deal with.

The design for a migration using ACL means that transmission of mobile agents between two agencies will be carried out using the message
system between agents[9][49][58]. In other words, the agent (both the code forming it as well as the state that it is in) will travel as the content of a message between two agents. Specifically, the agent will travel in the message between the AMS agents of each of the agencies involved in a migration.

Because the agencies have mechanisms for sending and receiving messages, using a parallel transmission protocol is not necessary. This is an advantage in interoperability terms and enables agents to be transmitted using the various message transport protocols (HTTP, SMTP, etc). Furthermore, this is achieved in a totally transparent way.

The first logical step in this process is to design the ontology and the protocol that will be used in the exchange of messages between the two agencies. This protocol has the movement of the agent as its final purpose. Defining an ontology basically consists of defining the elements that will form the content of an ACL message to give a common interface between the two parties when extracting the information of the message.

The two possible migration models that are proposed in the initial FIPA specification deal with one migration directed by the agent and another directed by the agency. In our implementation, it was decided to adopt the migration directed by the agency, which is more robust, as it enables the agency to decide which migration requests are accepted and which are not.

The ontology initially specified by FIPA is made up of seven elements: five concepts ("mobile-agent-description", "mobile-agent-profile", "mobile-agent-system", "mobile-agent-language", and "mobile-agent-os") and two actions ("Move" and "Transfer").

Of these concepts, we only use the first one, "mobile-agent-description". This is because it is very difficult to develop systems that enable agents with different architectures to migrate with total interoperability, at the current level of agent technological maturity. These agents could have been written in different languages or executed in different operating systems. For this reason, these concepts are never used, assuming that mobile agents which migrate move between the same agencies. Obviously, if the agency has been developed in a language like Java, a migration between agencies lodged in different operating systems is possible.
However, this is a characteristic of this language which is transparent to the agency, and therefore does not involve the need to use the concept "Mobile-Agent-OS", for example. In any case, although we do not use it, we maintain these concepts to ensure compatibility in case it is possible to make agents migrate between agencies implemented in a different way or with different languages.

The concept "mobile-agent-description", on the other hand, is highly useful to us. Within it are several fields that define the characteristics of the mobile agent in question. Among others, these include the characterization of the code and its data.

Of the two actions specified, we have only implemented "Move", for migrations directed by the agency. The "Transfer" action is not considered, which can be used for migrations directed by the agent, although it is supervised by the agency.

Once the content of the ACL messages was described, we moved on to enumerating the protocol by which the migration process is carried out. A diagram of the messages exchanged during the migration process can be seen in Fig. 4.3.
Firstly, the agent wishing to migrate starts a conversation with the AMS agent of the local platform to make a request for migration. This request is the first step in the standard FIPA-Request protocol and consists of a Request-type message with a Move action and a MobileAgentDescription (MobileAgentDescription is the name of the class that implements the concept of "mobile-agent-description"), in which the code and data fields are empty.

When the AMS agent receives the request for migration from a mobile agent, the first thing it does is to decide whether to accept it or not according to a given criterion. If the migration is accepted, the AMS agent sends an Agree message to the agent, or if not, a Refuse message.

If the migration is accepted, the first thing that the AMS agent does is to obtain the code and data (serialized instance) of the agent that made the request and fills in the code and data fields of the MobileAgentDescription.

The next step is to make contact with the AMS belonging to the agency to which the mobile agent wishes to migrate. To this end, the local AMS must start a parallel conversation to that between the agent and the remote AMS.

When the remote AMS receives the request, the agent’s code and data travel within the MobileAgentDescription that the local AMS has prepared.

Following its own criteria, the remote platform decides whether to accept the incoming agent. If so, it responds with a Agree message,
and if the agent does not meet the requirements specified by the agency to execute it, it responds with a *Refuse* message.

- The remote AMS loads the agent's class, deserializes its instance and restores its execution. Once this entire process has been successfully completed, the standard FIPA-Request protocol is completed by sending an *Inform* message to the local AMS.
- The final step in the protocol consists of informing the agent that started the process. If the process has been successfully completed, the original agent is destroyed.

### 4.4 DESIGN COMPONENTS

After describing the ontology and the protocol used, the components necessary to carry out the entire process are listed below.

**Migration Manager:** The Migration Manager component is responsible for carrying out all operations enabling the execution of an agent to be suspended in the local agency. Also, it enables its state and code to be processed. These operations allow to insert the agent in an ACL message. In the remote agency, it is responsible for undoing the messages that contain agents, decoding them and acting together with the agency to restore the agent's execution.

**Class Loader:** The class loader is the component that constructs the representation of the class in the memory so that it can be used in the code. The bytecode of the class is extracted from the ACL message and loaded during the deserialization time of an incoming agent. At the same time, each classloader provides a space of different names from the others, so that two agents created with one class with the same name do not conflict within a single platform if they are loaded by different classloaders.

**Mobile Agent:** The *MobileAgent* class was created, inheriting the properties of a basic agent and adding the functionality of being able to migrate to it. This functionality provides it with the *doMigrate(dest)* method, which starts the migration protocol when invoked.

**Conversation Modules:** These modules were implemented using the behaviours of the JADE model. Behaviors represent agent tasks and are also a useful tool for designing the protocols that govern conversations using ACL. There are basically two components of this type developed to provide
mobility. The first is the agent's behavior. This component is launched when the function \textit{doMigrate(dest)} is invoked and is responsible for supervising the migration from the agent's point of view. The second was implemented within the AMS agent to help it administer its part of the migration protocol. This behavior has a double functionality as it was designed for playing the roles of the local AMS and the remote AMS. The most complex part of the implementation of this component is its functioning as a local AMS, parallel conversations (AMS-Agent and AMS-AMS) which depend on each other must be taken into consideration.

4.5 DYNAMIC DISTRIBUTED SERVICE (DDS)

The DDS is specifically designed to be as transparent as possible to the agent programmer by ensuring that inter-platform migration is as straightforward as intra-platform migration. As much compliance as possible has been maintained with the deprecated (due to lack of initial implementation validation) FIPA Agent Management Support for Mobility Specification (this deprecated specification is available at http://www.fipa.org/specs/fipa00087/index.html). The current version is simple, but designed to be extendable with additional features such as security and support for fault-tolerant migration.

4.5.1 The Migration Process

The core mechanism of the DDS is the movement of agents between platforms using FIPA-ACL messages as the transportation medium. These messages are sent between the AMSs of the endpoint platforms. As mentioned, a few alterations have been made to the original FIPA specification; the ontology now defines two actions, move and power-up. The first of these represents the movement of agent code and instance; the second represents the agent’s activation once migration is complete. Additionally, several concepts are defined such as mobile-agent-description which contains all agent information, including its code, data and its mobile-agent-profile. This latter concept defines the basic agent characteristics to help ensure compatibility with receiving platforms, such as the name of its native agent system, the language in which it has been written, and the mobility protocol used. It is assumed that the messages containing this
information, consistent with the mobility ontology, will be sufficient for a remote platform to decide whether it can execute an incoming agent.

Both of the actions, move and power-up, are effected using the standard FIPA Request interaction protocol. In each case a Request message is sent to the target platform with an Inform or a Failure message expected as a response. As illustrated in Fig.4.5, FIPA Request mobility protocol, the initiating agent (on the left) sends a Request to move the message to its platform which will in turn send a Request to move the message containing the agent code and data to the specified destination platform (on the right). If the destination platform successfully extracts the agent from the Request message, an Inform message is returned to the originating platform. When received, this Inform message triggers the originating platform to terminate the requesting agent and send a power up Request to the destination platform, which starts the agent. If something fails, the agent state in the originating platform is restored and any residual presence of the agent in the target platform is removed.

![Fig. 4.5: FIPA Request mobility protocol](image)

The advantage of using ACL messages to transport the agent from one platform to another is that no additional inter-platform communication channel is needed. The disadvantage is that performance is not particularly high due to the encoding and decoding processes intrinsic to ACL messaging. Standardization of MTPs designed to improve ACL message sending/receiving performance, perhaps using lightweight content encodings
such as that expressed in[37](FIPA23), could significantly improve efficiency while retaining interoperability.

### 4.5.2 Code Grouping

In order to move an agent from one location to another, all of its code must be grouped for placement into the ACL message that will transfer it. Locating this code is not necessarily a trivial task, as in JADE the Classpath tends to mix together references to platform code and libraries, and agent code. This makes it difficult to decide whether a library or a class in the Classpath may be used by the agent in the future.

Two different solutions are provided to this problem. The programmatically simpler method (although not so simple for the user) is for the programmer to manually pack the agent inside a JAR(Java Archive) file with all the code and libraries it will require. This JAR file can then be transferred alongside the agent.

The DDS provides an initial mechanism in the migration system which automatically generates a JAR file given an agent. This JAR file contains all the agent's code and the programmer will never notice its existence. This mechanism is built by recursively examining the agent's main class looking for all dependent classes and building a JAR from the final set obtained. In the present version, it is possible that in some special cases not all agent classes are properly detected. This could happen in agents where, for example, reflection is used to load classes (e.g. Class.forName("ClassName").newInstance()). If problems occur with agent migration it is highly likely that some classes have not been included in the agent's JAR.

Currently, to avoid this problem, the JAR file must be manually created. The JAR should be named such that it corresponds to the agent's main class name. For example, the agent with a main class org.example.MyAgent must be packed in a JAR file named org_example_MyAgent.jar. The JAR must then be placed in a folder that is specified on the command line when launching JADE.

The second approach, programmatically more sophisticated but transparent to the user, consists of autonomously identifying all needed classes and constructing a JAR file containing them. Class selection is
controlled by the service which may imply that non-optimal selections may be made.

### 4.5.3 The Mobility Ontology

The mobility-ontology ontology contains all the concepts and actions needed to support agent mobility. JADE provides the class jade.domain.mobility. MobilityOntology which functions as a singleton giving access to a single, shared instance of the JADE mobility ontology through the getInstance() method. The ontology, which extends the JADEMManagementOntology, contains five concepts and two actions, each associated with a class of the package jade.domain.mobility. The concept schemas are described in Tables 4.1-4.5. The actions are:

- **move-agent.** This is the action of moving an agent from one location to another. It is represented by the MoveAction class. The action has a single, unnamed slot of type mobile-agent-description. The argument is mandatory.

- **clone-agent.** This is the action of producing a copy of an agent, possibly running on another location. It is represented by the CloneAction class. The action has two unnamed slots: the first of type mobile-agent-description and the second of type String. Both arguments are mandatory.

It may be noted that this ontology currently has no counterpart in the FIPA specifications.

<table>
<thead>
<tr>
<th>Slot name</th>
<th>Slot type</th>
<th>Mandatory/optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>AID</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Destination</td>
<td>Location</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Agent-profile</td>
<td>mobile-agent-profile</td>
<td>Optional</td>
</tr>
<tr>
<td>Agent-version</td>
<td>String</td>
<td>Optional</td>
</tr>
<tr>
<td>Signature</td>
<td>String</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 4.1: Mobile-agent-description**: This describes a mobile agent. It is represented by the class MobileAgentDescription

<table>
<thead>
<tr>
<th>Slot name</th>
<th>Slot type</th>
<th>Mandatory/optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>mobile-agent-system</td>
<td>Optional</td>
</tr>
<tr>
<td>Language</td>
<td>mobile-agent-language</td>
<td>Optional</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Os</td>
<td>Mobile-agent-os</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

**Table 4.2: Mobile-agent-profile:** This describes the computing environment needed by the mobile agent. It is represented by the class MobileAgentProfile

<table>
<thead>
<tr>
<th>Slot name</th>
<th>Slot type</th>
<th>Mandatory/optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Mandatory</td>
</tr>
<tr>
<td>major-version</td>
<td>Integer</td>
<td>Mandatory</td>
</tr>
<tr>
<td>minor-version</td>
<td>Integer</td>
<td>Optional</td>
</tr>
<tr>
<td>dependencies</td>
<td>String</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 4.3: Mobile-agent-system:** This describes the run-time system used by the mobile agent. It is represented by the class MobileAgentSystem

<table>
<thead>
<tr>
<th>Slot name</th>
<th>Slot type</th>
<th>Mandatory/optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Mandatory</td>
</tr>
<tr>
<td>major-version</td>
<td>Integer</td>
<td>Mandatory</td>
</tr>
<tr>
<td>minor-version</td>
<td>Integer</td>
<td>Optional</td>
</tr>
<tr>
<td>dependencies</td>
<td>String</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 4.4: Mobile-agent-language:** This describes the programming language used by the mobile agent. It is represented by the class MobileAgentLanguage

<table>
<thead>
<tr>
<th>Slot name</th>
<th>Slot type</th>
<th>Mandatory/optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Mandatory</td>
</tr>
<tr>
<td>major-version</td>
<td>Integer</td>
<td>Mandatory</td>
</tr>
<tr>
<td>minor-version</td>
<td>Integer</td>
<td>Optional</td>
</tr>
<tr>
<td>dependencies</td>
<td>String</td>
<td>Optional</td>
</tr>
</tbody>
</table>
Table 4.5: Mobile-agent-os: This describes the operating system needed by the mobile agent. It is represented by the class MobileAgentOS

4.6 SUMMARY

In this chapter, a proposal is presented for the mobility of agents between various agencies, based on the agent communication language (ACL) proposed by FIPA. This has been the key to fulfilling the double objective of maintaining consistency with the generally accepted agency model and also permitting interoperability between platforms of a different type.