The Annals of "Dunarea de Jos" University of Galati
Fascicle I – 2008. Economics and Applied Informatics. Years XIV - ISSN 1584-0409

# A Multiagent Platform for Developments of Accounting Intelligent Applications

**Adrian LUPAŞC**
*alupasc@ugal.ro*
*University "Dunărea de Jos" of Galaţi*

**Abstract.** AOP – Agent Oriented Programming – is a new software paradigm that brings many concepts from the artificial intelligence. This paper provides a short overview of the JADE software platform and the principal's components constituting its distributed architecture. Furthermore, it describes how to launch the platform with the command–line options and how to experiment with the main graphical tools of this platform.

**Keywords:** JADE, multiagent system, container, message, communication, accounting.

## 1. Introduction

Agent and multiagent technology has been the subject of many discussions within the scientific community, but it is maybe only recent that it has seen significant degree of exploitation in many types of application, including accounting application.

The initial software developments, that later became the JADE (*Java Agent Development Framework*) platform, were started by Telecom Italia in 1998, motivated by the need to validate the FIPA specifications. JADE went open source in 2000. JADE has a website[1] from where the software, documentation, example code, and a wealth of information about usages of JADE are available. The project welcomes the participation of the open source community with a variety of means to become involved and contribute to the project. In order to facilitate industrial involvement, in 2003 was defined a collaboration agreement and formed the JADE Governing Board, a not–for–profit organization of companies committed to contributing to the development and promotion of this platform. The Board was forms into contractual consortium with well–defined rules specifying the rights and obligations toward generated IPR. The Board is open with members able to join and leave according to their needs.

When JADE was became public, it was used almost exclusively by the FIPA community but as its feature set grew far beyond the FIPA specifications, so did its use by a globally distributed developer community. It is important to note that this platform contributed to wide diffusion of the FIPA specifications by providing a set of tools and software abstractions that hid the specifications themselves; programmers could essentially implement according to the specifications without the need to study them.

## 2. The agent's paradigm and JADE

JADE is a platform that provides basic middleware–layer functionalities which are independent of the specific application and which simplify the realization of distributed applications that

[1] http://jade.tilab.com

exploit the software agent abstraction. An important merit of JADE is that it implements this abstraction over a well–known object–oriented language, Java: providing a simple and friendly API (*Application Programming Interface*). The following simple design choices were influenced by the agent abstraction.

⇨ *Any agent must be proactive and autonomous* – an agent can not provide call–backs or its own object reference to other agents in order to decrease any chance of other entities co–opting control of its services. An agent must have its own thread of execution, using it to control its life cycle and decide autonomously when to perform and which actions.

⇨ *Any agents can always say 'Yes' or 'No', and they are loosely coupled* – message–based asynchronous communication is the main form of communication between different agents in JADE; an agent wishing to communicate must send messages to an identified destination (or more destinations). There is no temporal dependency between the receiver and sender: a receiver might not be disposable when the sender issues the message. There is also no need to have the object reference of receiver agents but just, name identities that the message transport system is able to resolve into proper transport addresses. It is even possible that a precise receiver identity be unknown to the sender, which instead may define a receiver set using an intentional grouping (all the agents that provide the "Accounting information retrieving" service) or mediated by a proxy agent (propagate this message to all agents in a domain).

This type of communication enables the receiver to select which messages to process; and which to discard, as well as to define its own processing priority (read all message, coming from a domain "accounting.com"). It enables the sender to control its thread of execution and thus not be blocked until the receiver processes the message. Also, it provides an important advantage when implementing multi–cast communication as an atomic operation rather than as a consecutive method calls.

⇨ *The system is Peer–to–Peer:* each agent is identified by a unique name (AID–*Agent Identifier*, as defined by FIPA specifications). It can join and leave a host platform anytime and can discover other agents through both yellow–page services (provided in JADE by AMS (Agent Management System) and the DF (*Directory Facilitator*) agents as defined by the FIPA). An agent can start a communication with other agent at anytime it wishes and can equally be the object of an incoming communication at any time.

- A distributed system inhabited by agents, each running as a separate thread, potentially on different remote machines, and capable of transparently communicating with one another, i.e. the platform provides a unique location–independent API that abstracts the underlying communication infrastructure.
- Compliance with the FIPA specifications – the platform successfully participated in all FIPA interoperability events and was used as the middleware for many platforms in the Agentcities network (Agentcities). A great facilitator of this was active contribution by the JADE team to the FIPA standardization process.
- Transport of asynchronous messages via a location–transparent API. The platform selects the best available means of communication and, when possible, avoids marshalling/unmarshalling Java objects. When crossing platform boundaries, messages are automatically transformed from JADE's own internal Java representation into proper FIPA–compliant syntaxes, encodings and transport protocols.
- Implementations of yellow page services. Federated systems can be implemented to represent domains and sub-domains as a graph of federated directories.
- A simple agent life–cycle management – when agents are created they are automatically assigned a globally unique identifier and a transport address which are used to register

with their platform's white page service. Simple APIs and graphical tools are also provided to both locally and remotely manage agent life cycles, i.e. create, suspend, resume, freeze, thaw, migrate, clone and kill.

- Support for agent mobility – agent code and agent state can migrate between processes and machines. Agent migration is made transparent to communicating agents that can continue to interact even during the migration process.
- A subscription mechanism for agents, and even external applications, that wish to subscribe with a platform to be notified of all platform events, including life–cycle–related events and message exchange events,
- A set of graphical tools to support programmers when debugging and monitoring. These are particularly important and complex in multi–threaded, multi–process, multi–machine systems such as a typical JADE application. Conversations can be sniffed and emulated, and agent execution can be controlled remotely and introspected, including remote step-by-step debugging of agent execution.
- Support for ontologies or content languages. Ontology checking and content encoding is per formed automatically by the platform with programmers able to select preferred content languages and ontologies (XML and RDF–based). Developers can also implement new content languages to performance specific application requirements.
- A library of interaction protocols which model typical models of communication oriented toward fulfills any goal. Application–independent skeletons are available as a set of Java classes that can he customized with application-specific code. Interaction Protocols can also be represented and implemented as a set of concurrent finite state machines integration with various web–based technologies including JSP, servlets, applets and Web service technology.
- An in–process interface for launching a platform and its distributed components from an external application.
- An extensible kernel designed to allow developers to extend platform functionality through the addition of kernel–level distributed services. This mechanism is inspired by the aspect–oriented programming approach where different aspects can be woven into application code and coordinated at kernel level.

**JADE architecture**

*Figure 1* shows the main architectural elements of a JADE platform. A JADE platform is composed of agent containers that can be distributed over the network. Agents are in containers which are the Java process that provides the JADE run–time and all the services needed for hosting and executing agents. There is a special container, called the *main container,* which represents the bootstrap point of a platform: it is the initial container to be launched and all other containers must join to this container by registering with it. The UML (*Unified Modeling Language*) diagram in *figure 2* summary shows the relations between the important architectural elements of JADE platform.

The developer identifies containers by simply using a logical name; by default the main container is named "Main Container" while the others are named "Container–1", "Container–2", etc. Command–line options are available to override this default names.

As a initial point, the main container has the following special responsibilities:

- ⇨ Managing the container table (CT), which is the registry of the object references and transport addresses of all container nodes composing the platform;
- ⇨ Managing the global agent descriptor table (GADT). which is the registry of all agents present in the platform, including their current status and location;
- ⇨ Hosting the AMS and the DF, the special agents that provide the agent management and white page service, and the default yellow page service of the platform.

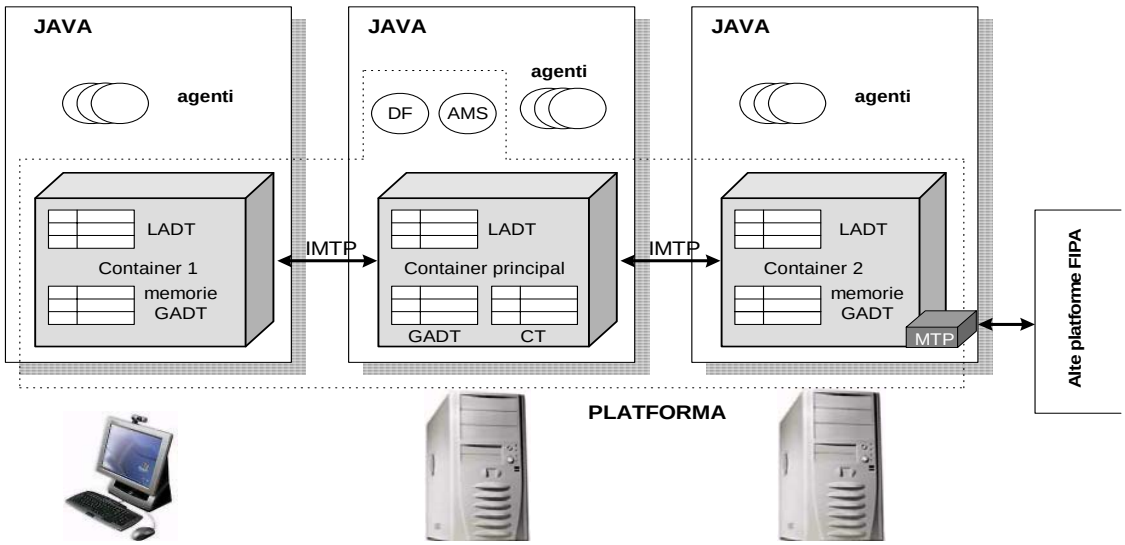


***Figure 1. The main architectural elements of JADE platform***

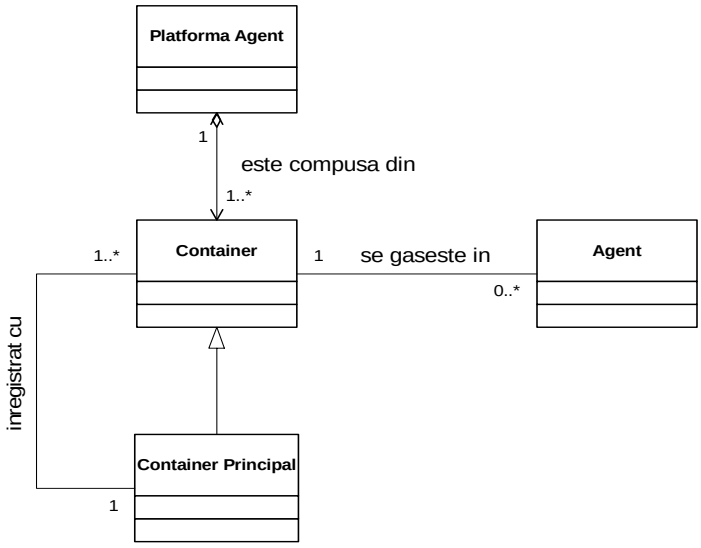


***Figure 2. Relationship between the main architectural elements***

A common query is whether the main–container is a system bottleneck. In fact this is not the case as JADE provides a cache of the GADT that each container manages locally. Platform operations do not always involve the main–container, but instead just the local cache and the two containers hosting the agents which arc the subject and the object of the operation (receiver and sender of the message). When a container must discover where the recipient of a message lives, it first searches its LADT (local agent descriptor table) and then, when the search fails, is the main–container contacted in order to obtain the proper remote reference which, consequently, is cached locally for future use. Because the system is dynamic, the system may use a stale cached value resulting in an invalid address. In this case, the container receives an exception and is forced to refresh its cache against the main–container. The cache replacement policy is LRU (least recently used), which was designed to optimize

long conversations rather than sporadic, single message exchange conversations which are actually fairly uncommon in multi–agent applications.

Agent identity is contained within an Agent Identifier (AID), composed of a set of slots that comply with the structure and semantics defined by FIPA. The most basic elements of the AID arc the agent name and its addresses. The name of an agent is a globally unique identifier that JADE constructs by concatenating a user–defined nickname (also known as a local name as it is sufficient for disambiguating intra–platform communication) to the platform name. The agent addresses are transport addresses inherited by the platform, where each platform address corresponds to an MTP (Message Transport Protocol) end point where FTPA–compliant messages can be sent and received. Agent programmers are also allowed to add their own transport addresses to the AID when, for any application-specific purpose, they wish to implement their own agent private MTP.

When the main–container is launched, two special agents are automatically instantiated and started by JADE, whose roles are defined by the F1PA Agent Management standard:

- ⇨ The Agent Management System (AMS) is the agent that supervises the entire platform. It is the contact point for all agents that need to interact in order to access the white pages of the platform as well as to manage their life cycle. Every agent is required to register with the AMS (automatically carried out by JADE at agent start–up) in order to obtain a valid AID.
- ⇨ The Directory Facilitator (DF) is the agent that implements the yellow pages service, used by any agent wishing to register its services or search for other available services. The JADE DF also accepts subscriptions from agents that wish to be notified whenever a service registration or modification is made that match some specified criteria. Multiple DFs can be started con currently in order to distribute the yellow pages service across several domains. These DFs can be federated, if required, by establishing cross–registrations with one another which allow the propagation of agent requests across the entire federation.

**Compiling the software and launching the platform**

JADE–related software is divided into two sections: the main distribution and the add–ons. The add–ons in particular include self–contained modules that implement specific extended features such as codec's for given languages. In many cases these have not been developed by the JADE team directly, but by members of the open source community who decided to return their achievements to the community itself.

The main distribution is composed of five primary archive files with the following content:

- ⇨ *jadeBin.zip* – contains only the pre–compiled JADE Java archive (.jar) files in a ready to use state.
- ⇨ *jadeDoc.zip* – contains the documentation, including the *Administrator* and *Programmer guides*. This documentation is also available online from the website.
- ⇨ *jadeExamples.zip* – contains the source code of various examples.
- ⇨ *jadeSrc.zip* – contains all the sources of JADE.
- ⇨ *jadeAll.zip* – contains all of the four files listed above.

If the above zip files are downloaded and unzipped, the directory structure should be as shown in Figure 3 (only the most relevant files and directories are actually shown). Some of the important files/folders include:

- ⇨ License, the open source license that regulates all use of the software.
- ⇨ The file *jade/doc/index-html* is a good starting point for beginners containing links to a variety of thematic tutorials, the Programmer and Administrator guide, *javadoc* documentation of all the sources, plus several other support documents.
- ⇨ The *jade/lib* folder contains all the *.jar files which must be included in the Java CLASSPATH in order to run JADE. It includes the *lib/commons-codec* subdirectory where an external Base64 codec is distributed that should also be included in the Java CLASSPATH.

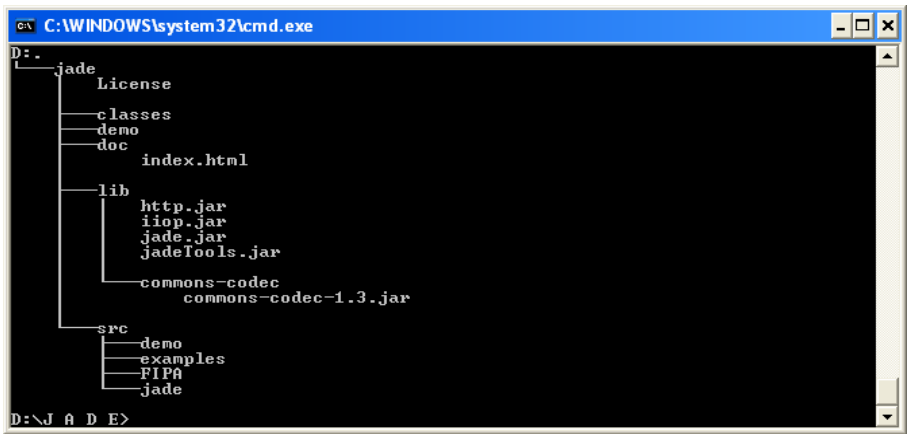


***Figure 3. JADE directory structure***

- The *jade/src* directory contains four subdirectories. Demo contains the sources of a simple demo. Examples contain several useful source code examples of various agent fragments. FIPA contains the sources of a FIPA–defined module. Jade contains all the sources of JADE itself.

The JADE sources can he compiled using the ant tool (ANT). The most important ant targets are the following:

- *jade* – to compile the sources and create the. class files under the classes subdirectory;
- *lib* – to generate the Java archive jar files under the lib subdirectory;
- *doc* – to generate the javadoc documentation files under the doc subdirectory;
- *examples* – to compile all the examples.

Experienced users might find it useful to directly access the source code repository for which read–only access is available to the JADE community. The repository is maintained and kept up to date by an administrator; instructions on how to access it arc available on the JADE website. The lib directory contains the five archive files containing the classes needed by JADE:

- *jade.jar* contains all the JADE packages except add–ons, MTPs and graphical tools;
- *jadeTools.jar* contains all the graphical tools;
- *http.jar* contains the HTTP–based MTP which is also the default MTP launched at the platform start-up;
- *iiop.jar* contains the IIOP–based MTP. This is not often used, but is the subject of a couple of examples later in the book and it implements the FIPA HOP MTP specs (FIPA75);
- *commons–codec\commons-codec-1.3.jar* contains the Base64 codec used by JADE.

The *classes* directory contains the class files of the examples. Note that to reduce the size of the distribution files, the examples arc distributed as source code and must therefore be compiled prior to use with the command ant examples. To launch the platform, the user must first set their local Java CLASSPATH, i.e. the set of directories and Java archive files where the Java Virtual Machine will took for byte code (i.e. the .class and .jar files). Issues relating to the CLASSPATH remain the topic of many questions received from JADE beginners, with the most typical concerning *classNotFound* exceptions caused by incorrect configuration of the CLASSPATH parameter.

The main-container can be launched with the JADE GUI using the command: *java jade.Boot –gui*. The result of this should be as shown in Figure 4.

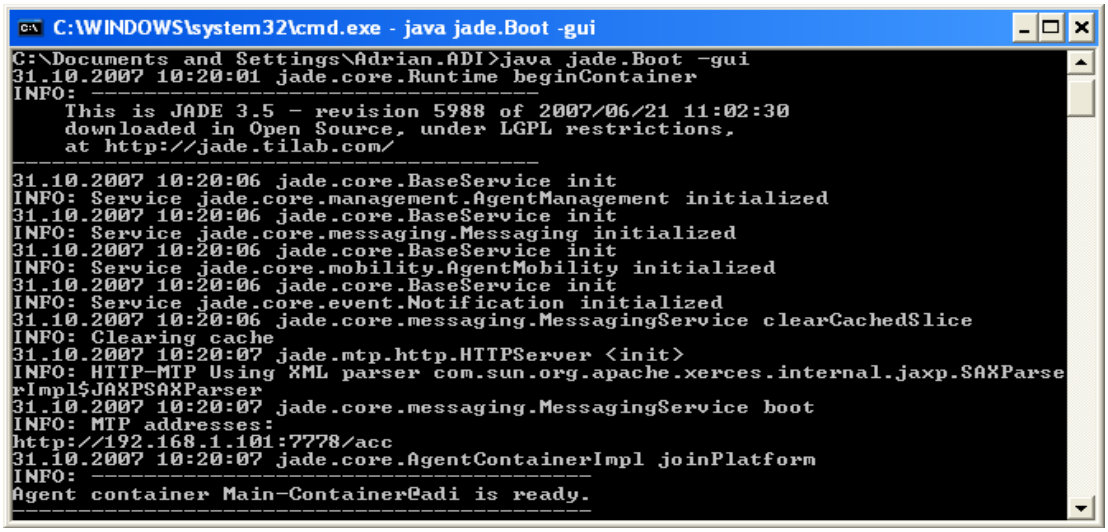


***Figure 4. Standard output at the JADE start–up***

The first part of this output is the disclaimer printed out each time a JADE run–time is started. Following that, all the standard JADE platform services are initialized, which implement the various functionalities provided by the container. Since this instance of the JADE run–time is a main container, an HTTP MTP is started by default and its local address printed. Finally, a notification indicates that a container called "main container" is ready; the JADE platform is now ready for use. As mentioned, the command–line option *-gui* has the effect of launching the primary JADE graphical interface, shown in figure 5. This GUI is actually provided by a JADE system agent called the Remote Monitoring Agent (RMA) and allows a platform administrator to manipulate and monitor the running platform. It should be noted that use of the RMA GUI, and all other graphical tools, can negatively impact system performance. This is one reason why the *-gui* option is provided. If performance is a concern, it is suggested not to use the RMA GUI at deployment time, rather to limit its use to system monitoring as required.

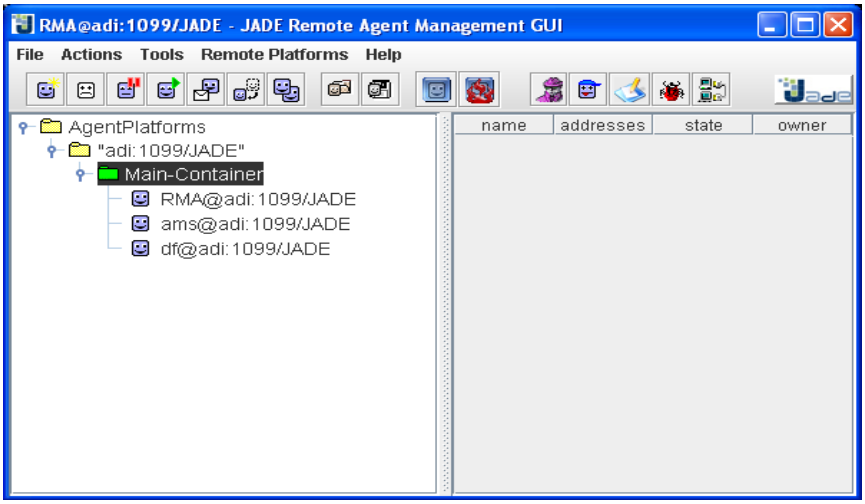


***Figure 5. GUI of the JADE RMA***

Now that the main-container has been initialized, any number of other containers can be launched on the various hosts composing the platform.

## Conclusion

Agent Oriented Programming models applications as a collection of components – agents that are characterized by autonomy, proactivity, and ability to communicate. In this paper I made a short overview of JADE, maybe the most wide agent oriented middleware in use in our days. This platform is a completely distributed middleware system with a flexible infrastructure that allows extension with add–ons modules. JADE facilitates the development of complete agent–based applications and because it is written in Java language, it benefits from the huge set of programming abstractions allowing constructing JADE multiagent systems with minimal expertise in multiagent theory.

## References

1. *Andone, I., Sisteme inteligente hibride. Teorie, studii de caz pentru aplicaţii economice şi ghidul dezvoltatorului, Editura Economică, Bucureşti, 2002.*
2. *Bellifemine, F., Caire, G., Greenwood, D., Developing multiagent systems with JADE, Wiley & Sons, Ltd., 2007.*
3. *Bordini, R.H., Hubner, J.F., Vieira., R., Jasonand the Golden Fleece of Agent-oriented Programming, In Bordini, R., Dastani, M., Dix, J., Seghrouchni, A., (eds), Multiagent Programming, Kluwer, 2005.*
4. *Bordini, R.H., Braubach, L., Dastani, M., Seghrouchni, A.E.F., Gomez-Sanz, J.J., Leite, J., O'Hare, G., Pokahr, A., Ricci, A., A Survey of Programming Languages and Platforms for Multiagent Systems, Informatica, 30(1), p. 33-44, 2006;*
5. *Negri, A., Poggi, A., Tomaiuolo, M., Turci, P., Dynamic Grid Tasks Composition and Distribution through Agents, Concurrency and Computation: Practice and Experience, 18(8), p. 875-885, 2006.*
6. *Wooldridge, M., An introduction to multiagent systems, John Wiley&Sons, LTD, August, 2002.*
7. *http://jade.tilab.com.*