

# JADE ADMINISTRATOR'S GUIDE

USAGE RESTRICTED ACCORDING TO LICENSE AGREEMENT.

Last update: 29-January-2002. JADE 2.5

Authors: Fabio Bellifemine, Giovanni Caire, Tiziana Trucco (TILAB S.p.A., formerly CSELT)  
Giovanni Rimassa (University of Parma)

Copyright (C) 2000 CSELT S.p.A.

Copyright (C) 2001 TILAB S.p.A.

Copyright (C) 2002 TILAB S.p.A.

JADE - Java Agent DEvelopment Framework is a framework to develop multi-agent systems in compliance with the FIPA specifications. JADE successfully passed the 1<sup>st</sup> FIPA interoperability test in Seoul (Jan. 99) and the 2<sup>nd</sup> FIPA interoperability test in London (Apr. 01).

Copyright (C) 2000 CSELT S.p.A., 2001 TILab S.p.A., 2002 TILab S.p.A.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, version 2.1 of the License.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

## TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>4</b>
<b>2</b>	<b>RUNNING THE AGENT PLATFORM</b>	<b>4</b>
2.1	Software requirements	4
2.2	Getting the software	4
2.3	Running JADE from the binary distribution	4
2.3.1	Command line syntax	5
2.3.2	Options available from the command line	5
2.3.3	Launching agents from the command line	7
2.3.4	Example	7
2.4	Building JADE from the source distribution	8
2.4.1	Building the JADE framework	8
2.4.2	Building JADE libraries	8
2.4.3	Building JADE HTML documentation	8
2.4.4	Building JADE examples and demo application	9
2.4.5	Cleaning up the source tree	9
2.5	Support for inter-platform messaging with plug-in Message Transport Protocols	9
2.5.1	Command line options for MTP management	10
2.5.2	Configuring MTPs from the graphical management console.	10
2.5.3	Agent address management	11
2.5.4	Writing new MTPs for JADE	11
2.5.4.1	The Basic IIOP MTP	11
2.5.4.2	The ORBacus MTP	12
2.5.4.3	The HTTP MTP	12
2.6	Support for ACL Codec	13
2.6.1	XML Codec	13
2.6.2	Bit Efficient ACL Codec	13
<b>3</b>	<b>AGENT IDENTIFIERS AND SENDING MESSAGES TO REMOTE AGENTS</b>	<b>13</b>
<b>4</b>	<b>GRAPHICAL USER INTERFACE TO MANAGE AND MONITOR THE AP ACTIVITY</b>	<b>14</b>
4.1	Remote Monitoring Agent	14
4.2	DummyAgent	18
4.3	DF GUI	19
4.4	Sniffer Agent	20
4.5	Introspector Agent	21

**5 LIST OF ACRONYMS AND ABBREVIATED TERMS 22**

---

## 1 INTRODUCTION

---

This administrator's guide describes how to install and launch JADE. It is complemented by the HTML documentation available in the directory `jade/doc` and the JADE Programmer's Guide. If and where conflict arises between what is reported in the HTML documentation and this guide, preference should be given to the HTML documentation that is updated more frequently.

---

## 2 RUNNING THE AGENT PLATFORM

---

### 2.1 Software requirements

The only software requirement to execute the system is the Java Run Time Environment version 1.2.

In order to build the system the JDK1.2 is sufficient because pre-built IDL stubs and Java parser classes are included with the JADE source distribution. Those users, who wish to regenerate IDL stubs and Java parser classes, should also have the JavaCC parser generator (version 0.8pre or version 1.1; available from <http://www.metamata.com>), and the IDL to Java compiler (available from the Sun Developer Connection). Notice that the old `idltojava` compiler available with JDK1.2 generates wrong code and it should never be used, instead the new `idlj` compiler, that is distributed with JDK1.3, should be used.

### 2.2 Getting the software

All the software is distributed under the LGPL license limitations and it can be downloaded from the JADE web site <http://jade.cselt.it/>. Five compressed files are available:

1. The source code of JADE
2. The source code of the examples
3. The documentation, including the javadoc of the JADE API and this programmer's guide
4. The binary of JADE, i.e. the jar files with all the Java classes
5. A full distribution with all the previous files

### 2.3 Running JADE from the binary distribution

Having uncompressed the archive file, a directory tree is generated whose root is `jade` and with a `lib` subdirectory. This subdirectory contains some JAR files that have to be added to the `CLASSPATH` environment variable.

Having set the classpath, the following command can be used to launch the main container of the platform. The main container is composed of the DF agent, the AMS agent, and the RMI registry (that is used by JADE for intra-platform communication).

```
java jade.Boot [options] [AgentSpecifier list]
```

Additional agent containers can be then launched on the same host, or on remote hosts, that connect themselves with the main container of the Agent Platform, resulting in a distributed system that seems a single Agent Platform from the outside.

An Agent Container can be started using the command:

```
java jade.Boot -container [options] [AgentSpecifier list]
```

An alternative way of launching JADE is to use the following command, that does not need to set the CLASSPATH:

```
java -jar lib\jade.jar -nomtp [options] [AgentSpecifier list]
```

**Remind** to use the “-nomtp” option, otherwise an exception will be thrown because the library iiop.jar is not found.

### 2.3.1 Command line syntax

The full EBNF syntax of the command line is the following, where common rules apply for the token definitions:

```
java jade.Boot Option* AgentSpecifier*
```

```
Option          = "-container"
                | "-host" HostName
                | "-port" PortNumber
                | "-name" PlatformName
                | "-gui"
                | "-mtp" ClassName "(" Argument* ")"
                | (";" ClassName "(" Argument* ")")*
                | "-nomtp"
                | "-aclcodec" ClassName (";" ClassName)*
                | "-nomobility"
                | "-version"
                | "-help"
                | "-conf" FileName?

ClassName       = PackageName? Word

PackageName     = (Word ".")+

Argument        = Word | Number | String

HostName        = Word ( "." Word )*

PortNumber      = Number

AgentSpecifier  = AgentName ":" ClassName "(" Argument* ")"?

AgentName       = Word

PlatformName    = Word
```

### 2.3.2 Options available from the command line

*-container* specifies that this instance of JADE is a container and, as such, that it must join with a main-container (by default this option is unselected)

*-host* specifies the host name where the RMI registry should be created (for the main-

container) / located (for the ordinary containers); its value is defaulted to localhost. This option can also be used when launching the main-container in order to override the value of localhost; **a typical example of this kind of usage is to include the full domain of the host (e.g. `-host kim.cselt.it` when the localhost would have returned just `'kim'`) such that the main-container can be contacted even from outside the local domain.**

- `-port` this option allows to specify the port number where the RMI registry should be created (for the main-container) / located (for the ordinary containers). By default the port number 1099 is used.
- `-name` this option specifies the symbolic name to be used as the platform name; this option will be considered only in the case of a main container; the default is to generate a unique name from the values of the main container's host name and port number. Please note that this option is strongly discouraged since uniqueness of the HAP is not enforced. This might result in non-unique agent names.
- `-gui` specifies that the RMA (Remote Monitoring Agent) GUI of JADE should be launched (by default this option is unselected)
- `-mtp` specifies a list of external Message Transport Protocols to be activated on this container (by default the JDK1.2 IIOP is activated on the main-container and no MTP is activated on the other containers)
- `-nomtp` has precedence over `-mtp` and overrides it. It should be used to override the default behaviour of the main-container (by default the `-nomtp` option unselected)
- `-aclcodec` By default all messages are encoded by the String-based ACLCodec. This option allows to specify a list of additional ACLCodec that will become available to the agents of the launched container in order to encode/decode messages. JADE will provide automatically to use these codec when agents set the right value in the field *aclRepresentation* of the Envelope of the sent/received ACLMessages. Look at the FIPA specifications for the standard names of these codecs
- `-nomobility` disable the mobility and cloning support in the launched container. In this way the container will not accept requests for agent migration or agent cloning, option that might be useful to enhance the level of security for the host where this container is running. Notice that the platform can include both containers where mobility is enabled and containers where it is disabled. In this case an agent that tries to move from/to the containers where mobility is disabled will die because of a Runtime Exception.  
 Notice that, even if this option was selected, the container would still be able to launch new agents (e.g. via the RMA GUI) if their class can be reached via the local CLASSPATH.  
 By default this option is unselected.
- `-version` print on standard output the versioning information of JADE (by default this option is unselected)

- `-help` print on standard output this help information (by default this option is unselected)
- `-conf` if no filename is specified after this option, then a graphical interface is displayed that allows to load/save all the JADE configuration parameters from a file. If a filename is specified, instead, then all the options specified in that file are used to launch JADE. By default this option is not selected

### 2.3.3 Launching agents from the command line

A list of agents can be launched directly from the command line. As described above, the `[AgentSpecifier list]` part of the command is a sequence of strings separated by a space.

Each string is broken into three parts. The first substring (delimited by the colon ':' character) is taken as the agent name; the remaining substring after the colon (ended with a space or with an open parenthesis) is the name of the Java class implementing the agent. The Agent Container will dynamically load this class. Finally, a list of string arguments can be passed delimited between parentheses.

For example, a string `Peter:myAgent` means "create a new agent named Peter whose implementation is an object of class `myAgent`". The name of the class must be fully qualified, (e.g. `Peter:myPackage.myAgent`) and will be searched for according to `CLASSPATH` definition.

Another example is the string `Peter:myAgent("today is raining" 123)` that means "create a new agent named Peter whose implementation is an object of class `myAgent` and pass an array of two arguments to its constructor: the first is the string `today is raining` and the second is the string `123`". Notice that, according to the Java convention, the quote symbols have been removed and the number is still a string.

### 2.3.4 Example

First of all set the `CLASSPATH` to include the JAR files in the `lib` subdirectory and the current directory. For instance, for Windows 9x/NT use the following command:

```
set CLASSPATH=%CLASSPATH%;.;c:\jade\lib\jade.jar;
c:\jade\lib\jadeTools.jar;c:\jade\lib\Base64.jar;c:\jade\lib
\iiop.jar
```

Execute the following command to start the main-container of the platform. Let's suppose that the hostname of this machine is "kim.cselt.it"

```
prompt> java jade.Boot -gui
```

Execute the following command to start an agent container on another machine, by telling it to join the Agent Platform running on the host "kim.cselt.it", and start one agent (you must download and compile the examples agents to do that):

```
prompt> java jade.Boot -host kim.cselt.it -container
sender1:examples.receivers.AgentSender
```

where "sender1" is the name of the agent, while `examples.receivers.AgentSender` is the code that implements the agent.

Execute the following command on a third machine to start another agent container telling it to join the Agent Platform, called "facts" running on the host "kim.cselt.it", and then start two agents.

```
prompt> java jade.Boot -host kim.cselt.it -container
           receiver2:examples.receivers.AgentReceiver
           sender2:examples.receivers.AgentSender
```

where the agent named sender2 is implemented by the class `examples.receivers.AgentSender`, while the agent named receiver2 is implemented by the class `examples.receivers.AgentReceiver`.

## 2.4 Building JADE from the source distribution

If you downloaded JADE in source form and want to compile it, you basically have two methods: either you use the provided makefiles (for GNU make), or you run the Win32 .BAT files that you find in the root directory of the package. Of course, using makefiles yields more flexibility because they just build what is needed; JADE makefiles have been tested under Sun Solaris 7 with JDK 1.2.0 and under Linux under JDK 1.2.2 RC4 and JDK 1.3. The batch files have been tested under Windows NT 4.0 and under Windows 95, both with JDK 1.2.2 or JDK1.3

### 2.4.1 Building the JADE framework

If you use the makefiles, just type:

```
make all
```

in the root directory; if you use the batch files, type

```
makejade
```

in the root directory. Beware that the batch file will not be able to check whether IDL stubs and parser classes already exist, so either you have `idltojava` and `JavaCC` installed, or you comment out them in the batch file.

You will end up with all JADE classes in a `classes` subdirectory. You can add that directory to your `CLASSPATH` and make sure that everything is OK by running JADE, as described in the previous section.

### 2.4.2 Building JADE libraries

With makefiles, type

```
make lib
```

With batch files, type

```
makelib
```

This will remove the content of the `classes` directory and will create some JAR files in the `lib` directory. These JAR files are just the same you get from the binary distribution. See section 2.3 for a description on how to run JADE when you have built the JAR files. Beware that, with both makefiles and batches, you must first build the classes and then the libraries, or you will end up with empty JAR files.

### 2.4.3 Building JADE HTML documentation

With makefiles, type

```
make doc
```

With batch files, type

```
makedoc
```

You will end up with Javadoc generated HTML pages, integrated within the overall documentation. Beware that the Programmer's Guide is a PDF file that cannot be generated at your site, but you must download it (it is, of course, in the JADE documentation distribution).

#### 2.4.4 Building JADE examples and demo application

If you downloaded the examples/demo archive and have unpacked it within the same source tree, you will have to set your CLASSPATH to contain either the `classes` directory or the JAR files in the `lib` directory, depending on your JADE distribution, and then type:

```
make examples
```

with makefiles, or

```
makeexamples
```

with batch files.

In order to compile the Jess-based example, it is necessary to have the JESS system and to set the CLASSPATH to include it. The example can be compiled by typing:

```
make jessexample
```

with makefiles, or

```
makejessexample
```

with batch files.

#### 2.4.5 Cleaning up the source tree

If you type

```
make clean
```

with makefiles, or if you type

```
clean
```

with batch files, you will remove all generated files (classes, HTML pages, JAR files, etc.) from the source tree. If you use makefiles, you will find some other make targets you can use. Feel free to try them, especially if you are modifying JADE source code, but be aware that these other make targets are for internal use only, so they have not been documented.

### 2.5 Support for inter-platform messaging with plug-in Message Transport Protocols

The FIPA 2000 specification proposes a number of different *Message Transport Protocols* (*MTPs* for short) over which ACL messages can be delivered in a compliant way.

JADE comprises a framework to write and deploy multiple *MTPs* in a flexible way. An implementation of a FIPA compliant MTP can be compiled separately and put in a JAR file of its own; the code will be dynamically loaded when an endpoint of that MTP is activated. Moreover, every JADE container can have any number of active MTPs, so that the platform administrator can choose whatever topology he or she wishes.

JADE performs message routing for both incoming and outgoing messages, using a single-hop routing table that requires direct visibility among containers.

When a new MTP is activated on a container, the JADE platform gains a new address that is added to the list in the platform profile (that can be obtained from the AMS using the action `get-description`). Moreover, the new address is added to all the `ams-agent-description` objects contained within the AMS knowledge base.

### 2.5.1 Command line options for MTP management

When a JADE container is started, it is possible to activate one or more communication endpoints on it, using suitable command line options. The `-mtp` option activates a new communication endpoint on a container, and must be given the name of the class that provides the MTP functionality. If the MTP supports activation on specific addresses, then the address URL can be given right after the class name, enclosed in brackets. If multiple MTPs are to be activated, they can be listed together using commas as separators.

For example, the following option activates an IIOP endpoint on a default address.

```
-mtp jade.mtp.iiop.MessageTransportProtocol
```

The following option activates an IIOP endpoint that uses an ORBacus-based<sup>1</sup> IIOP MTP on a fixed, given address.

```
-mtp
orbacus.MessageTransportProtocol(corbaloc:iiop:sharon.cselt.it:12
34/jade)
```

The following option activates two endpoints that correspond to two ORBacus-based IIOP MTP on two different addresses:

```
-mtp
orbacus.MessageTransportProtocol(corbaloc:iiop:sharon.cselt.it:12
34/jade);orbacus.MessageTransportProtocol(corbaloc:iiop:sharon.cs
elt.it:5678/jade)
```

When a container starts, it prints on the standard output all the active MTP addresses, separated by a carriage return. Moreover, it writes the same addresses in a file, named:

```
MTPs-<Container Name>.txt.
```

If no MTP related option is given, by default a basic IIOP MTP is activated on the Main Container and no MTP are activated on an ordinary container. To inhibit the creation of the default IIOP endpoint, use the `-nomtp` option.

### 2.5.2 Configuring MTPs from the graphical management console.

Using the `-mtp` command line option, a transport endpoint lives as long as its container is up; when a container is shut down, all its MTPs are deactivated and the AMS information is updated accordingly. The JADE RMA console enables a more flexible management of the MTPs, allowing activating and deactivating transport protocols during normal platform operations. In the leftmost panel of the RMA GUI, right-clicking on an agent container tree node brings up the popup menu with an *Install a new MTP* and *Uninstall an MTP*.

---

<sup>1</sup> ORBacus is a CORBA 2.3 ORB for C++ and Java. It is available from Object Oriented Concepts, Inc. at <http://www.ooc.com>. An alternate IIOP MTP for JADE, exploiting ORBacus features, is available in the download area of the JADE web site: <http://jade.cselt.it/>.

Choosing *Install a new MTP* a dialog is shown where the user can select the container to install the new MTP on, the fully qualified name of the class implementing the protocol, and (if it is supported by the chosen protocol) the transport address that will be used to contact the new MTP. For example, to install a new IIOP endpoint, using the default JDK 1.3 ORB, one would write `jade.mtp.iiop.MessageTransportProtocol` as the class name and nothing as the address. In order to install a new IIOP endpoint, using the ORBacus based implementation, one would write `orbacus.MessageTransportProtocol` as the class name and (if the endpoint is to be deployed at host `sharon.cselt.it`, on the TCP port 1234, with an object ID `jade`) `corbaloc:iiop:sharon.cselt.it:1234/jade` as the transport address.

Choosing *Uninstall an MTP*, a dialog is shown where the user can select from a list one of the currently installed MTPs and remove it from the platform.

### 2.5.3 Agent address management

As a consequence of the MTP management described above, during its lifetime a platform, and its agents, can have more than one address and they can be activated and deactivated during the execution of the system. JADE takes care of maintaining consistence within the platform and the addresses in the platform profile, the AMS knowledge base, and in the AID value returned by the method `getAID()` of the class `Agent`.

For application-specific purposes, an agent can still decide to choose explicitly a subset of the available addresses to be contacted by the rest of the world. In some cases, the agent could even decide to activate some application specific MTP, that would not belong to the whole platform but only to itself. So, the preferred addresses of an agent are not necessarily the same as the available addresses for its platform. In order to do that, the agent must take care of managing its own copy of agent ID and set the sender of its `ACLMessages` to its own copy of agent ID rather than the value returned by the method `getAID()`.

### 2.5.4 Writing new MTPs for JADE

To write a new MTP that can be used by JADE, all that is necessary is to implement a couple of Java interfaces, defined in the `jade.mtp` package. The MTP interface models a bi-directional channel that can both send and receive ACL messages (this interface extends the `OutChannel` and `InChannel` interfaces that represent one-way channels). The `TransportAddress` interface is just a simple representation for an URL, allowing separately reading the protocol, host, port and file part.

#### 2.5.4.1 The Basic IIOP MTP

An implementation of the FIPA 2000 IIOP-based transport protocol is included with JADE. This implementation relies on the JDK 1.2 ORB (but can also use the JDK 1.3 ORB, requiring recompilation of the `jade.mtp.iiop` package). This implementation fully supports IOR representations such as `IOR:000000000000001649444c644f4...`, and does not allow to choose the port number or the object key. These limitations are due to the underlying ORB, and can be solved with other JADE MTPs exploiting more advanced CORBA ORBs. The MTP implementation is contained within the `jade.mtp.iiop.MessageTransportProtocol` class, so this is the name to be used when starting the protocol. Due to the limitation stated above, choosing the address explicitly is not supported.

The default IIOP MTP also supports a limited form of `corbaloc:` addressing: A `corbaloc:` address, generated by some other more advanced ORB and pointing to a different platform, can be used to send ACL messages. Interoperability between a JADE platform using ORBacus and a JADE platform using the JDK 1.3 ORB has been successfully tested. In a first test, the first platform exported a `corbaloc:` address generated by ORBacus, and then the second platform used that address with the JDK 1.3 ORB to contact the first one. In a second test, the IOR generated by the second platform was converted into a `corbaloc:` URL via the `getURL()` method call in the `IIOPAddress` inner class (a non-public inner class of the `jade.mtp.iiop.MessageTransportProtocol` class); then the first platform used that address to contact the second one.

So, the `corbaloc:` support is almost complete. The only limitation is that it's not possible to export `corbaloc:` addresses with the JDK 1.3 ORB. JADE is able to convert IORs to `corbaloc:` URLs, but the CORBA object key is an arbitrary octet sequence, so that the resulting URL contains forbidden characters that are escaped using '%' and their hexadecimal value. While this conversion complies with CORBA 2.4 and RFC 2396, the resulting URL is just as unreadable as the plain old IOR. The upcoming JDK 1.4 is stated to feature an ORB that complies with the POA and INS specifications, so that it has persistent object references, and natively supports `corbaloc:` and `corbaname:` addresses. It is likely that a more complete IIOP MTP will be provided for the JDK 1.4, when it will be widely available.

#### 2.5.4.2 *The ORBacus MTP*

A Message Transport Protocol implementation that complies with FIPA and exploits the ORBacus ORB implementation can be download as an add-on from the JADE web site. A tutorial is available in the JADE documentation that describes how to download, install, compile and use this MTP. This MTP fully supports `IOR:`, `corbaloc:` and `corbaname:` addresses.

According to the OMG specifications, three syntaxes are allowed for an IIOP address (all case-insensitive):

```
IIOPAddress ::= "ior:" (HexDigit HexDigit+)
              | "corbaname://" NSHost ":" NSPort "/" NSObjectID
              | "#" objectName
              | "corbaloc:" HostName ":" portNumber "/" objectID
```

Notice that, in the third case, `BIG_ENDIAN` is assumed by default, while in the first and second case, the endianness information is contained within the IOR definition. In the second form, `HostName` and `PortNumber` refer to the host where the CORBA Naming Service is running.

#### 2.5.4.3 *The HTTP MTP*

A Message Transport Protocol implementation that complies to FIPA and uses the HTTP protocol can be download as an add-on from the JADE web site. A tutorial is available in the JADE documentation that describes how to download, install, compile and use this MTP.

## 2.6 Support for ACL Codec

By default, all ACLMessages are encoded via the String format defined by FIPA. However, at configuration time it is possible to add additional ACLCodecs that can be used by agents on that container. The command line option `-aclcodec` should be used for this purpose. Agents wishing to send messages with non-default encodings should set the right value in the *aclRepresentation* field of the Envelope.

### 2.6.1 XML Codec

An XML-based implementation of the ACLCodec can be download from the JADE site as an add-on. A tutorial is available in the JADE documentation that describes how to download, install, compile and use this codec.

### 2.6.2 Bit Efficient ACL Codec

A bit-efficient implementation of the ACLCodec can be download from the JADE site as an add-on. A tutorial is available in the JADE documentation that describes how to download, install, compile and use this codec. Take care that this codec is available under a different license, not LGPL.

---

## 3 AGENT IDENTIFIERS AND SENDING MESSAGES TO REMOTE AGENTS

---

According to the FIPA specifications, each agent is identified by an Agent Identifier (AID). An Agent Identifier (AID) labels an agent so that it may be distinguished unambiguously within the Agent Universe.

The AID is a structure composed of a number of slots, the most important of which are **name** and **addresses**.

The **name** parameter of an AID is a globally unique identifier that can be used as a unique referring expression of the agent. JADE uses a very simple mechanism to construct this globally unique name by concatenating a user-defined nickname to its home agent platform name (HAP), separated by the '@' character. Therefore, a full valid name in the agent universe, a so-called GUID (Globally Unique Identifier), is [peter@kim:1099/JADE](#) where 'peter' is the agent nickname that was specified at the agent creation time, while 'kim:1099/JADE' is the platform name. Only full valid names should be used within ACLMessages.

The **addresses** slot, instead, should contain a number of transport addresses at which the can be contacted. The syntax of these addresses is just a sequence of URI. When using the default IIOP MTP, the URI for all the local addresses is the IOR printed on stdout. The address slot is defaulted to the addresses of the local agent platform.

---

#### 4 GRAPHICAL USER INTERFACE TO MANAGE AND MONITOR THE AP ACTIVITY

---

To support the difficult task of debugging multi-agent applications, some tools have been developed. Each tool is packaged as an agent itself, obeying the same rules, the same communication capabilities, and the same life cycle of a generic application agent.

##### 4.1 Remote Monitoring Agent

The Remote Monitoring Agent (RMA) allows controlling the life cycle of the agent platform and of all the registered agents. The distributed architecture of JADE allows also remote controlling, where the GUI is used to control the execution of agents and their life cycle from a remote host.

An RMA is a Java object, instance of the class `jade.tools.rma.rma` and can be launched from the command line as an ordinary agent (i.e. with the command `java jade.Boot myConsole:jade.tools.rma.rma`), or by supplying the `'-gui'` option the command line parameters (i.e. with the command `java jade.Boot -gui`).

More than one RMA can be started on the same platform as long as every instance has a different local name, but only one RMA can be executed on the same agent container.

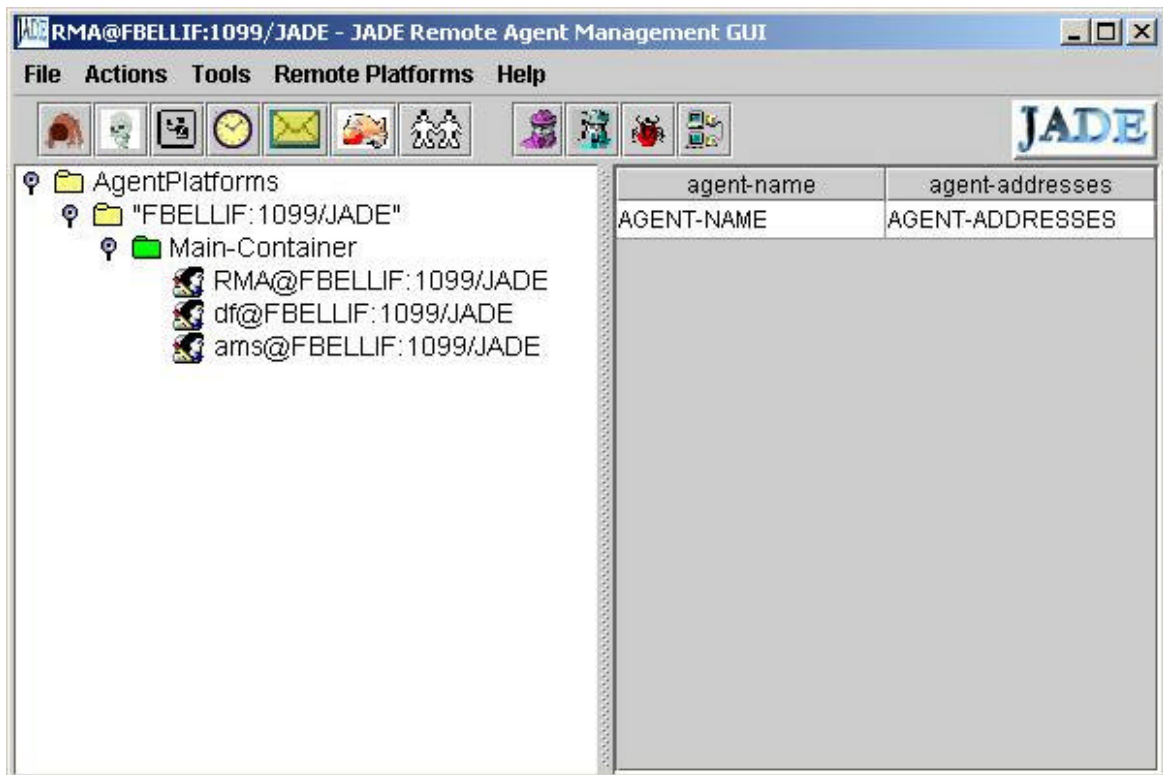


Figure 1 Snapshot of the RMA GUI

The followings are the commands that can be executed from the menu bar (or the tool bar) of the RMA GUI.

◆ File menu:

This menu contains the general commands to the RMA.

◆ Close RMA Agent

Terminates the RMA agent by invoking its `doDelete()` method. The closure of the RMA window has the same effect as invoking this command.

◆ Exit this Container

Terminates the agent container where the RMA is living in, by killing the RMA and all the other agents living on that container. If the container is the Agent Platform Main-Container, then the whole platform is shut down.

◆ Shut down Agent Platform

Shut down the whole agent platform, terminating all connected containers and all the living agents.

◆ Actions menu:

This menu contains items to invoke all the various administrative actions needed on the platform as a whole or on a set of agents or agent containers. The requested action is performed by using the current selection of the agent tree as the target; most of these actions are also associated to and can be executed from toolbar buttons.

◆ Start New Agent

This action creates a new agent. The user is prompted for the name of the new agent and the name of the Java class the new agent is an instance of. Moreover, if an agent container is currently selected, the agent is created and started on that container; otherwise, the user can write the name of the container he wants the agent to start on. If no container is specified, the agent is launched on the Agent Platform Main-Container.

◆ Kill Selected Items

This action kills all the agents and agent containers currently selected. Killing an agent is equivalent to calling its `doDelete()` method, whereas killing an agent container kills all the agents living on the container and then de-registers that container from the platform. Of course, if the Agent Platform Main-Container is currently selected, then the whole platform is shut down.

◆ Suspend Selected Agents

This action suspends the selected agents and is equivalent to calling the `doSuspend()` method. Beware that suspending a system agent, particularly the AMS, deadlocks the entire platform.

◆ Resume Selected Agents

This action puts the selected agents back into the `AP_ACTIVE` state, provided they were suspended, and works just the same as calling their `doActivate()` method.

◆ Send Custom Message to Selected Agents

This action allows to send an ACL message to an agent. When the user selects this menu item, a special dialog is displayed in which an ACL message can be

composed and sent, as shown in the figure.

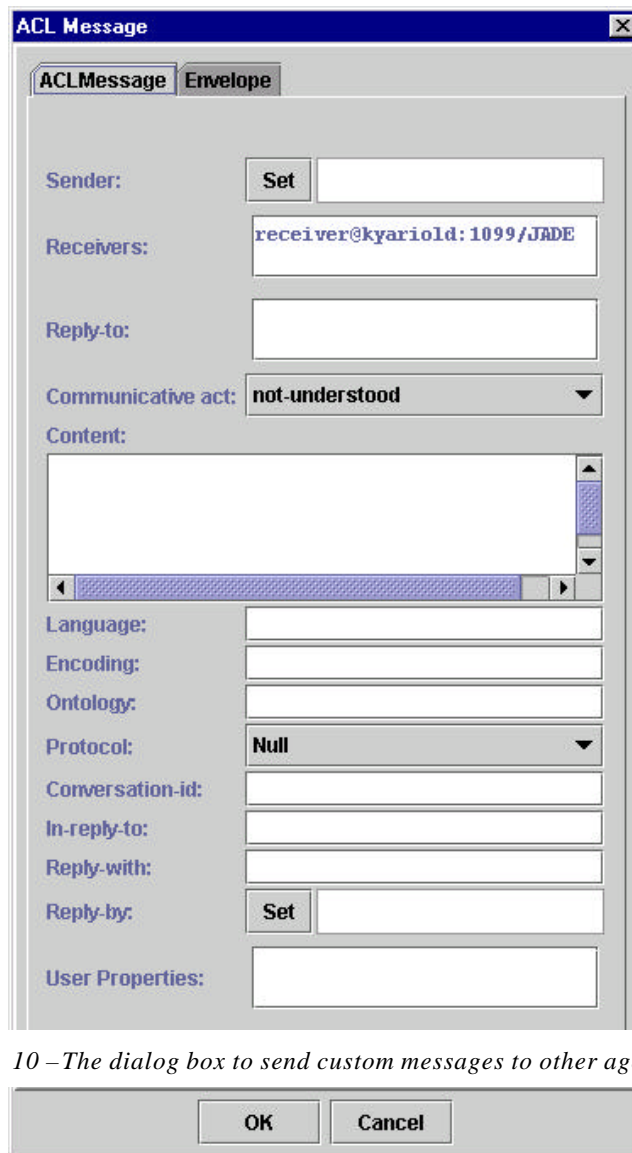


Figure 10 –The dialog box to send custom messages to other agents

◆ Migrate Agent

This action allows to migrate an agent. When the user selects this menu item, a special dialog is displayed in which the user must specify the container of the platform where the selected agent must migrate. Not all the agents can migrate because of lack of serialization support in their implementation. In this case the user can press the cancel button of this dialog.

◆ Clone Agent

This action allows to clone a selected agent. When the user selects this menu item a dialog is displayed in which the user must write the new name of the agent and the container where the new agent will start.

◆ Tools menu:

This menu contains the commands to start all the tools provided by JADE to application programmers. These tools will help developing and testing JADE based agent systems.

◆ RemotePlatforms menu:

This menu allows controlling some remote platforms that comply with the FIPA specifications. Notice that these remote platforms can even be non-JADE platforms.

◆ Add Remote Platform via AMS AID

This action allows getting the description (called APDescription in FIPA terminology) of a remote Agent Platform via the remote AMS. The user is requested to insert the AID of the remote AMS and the remote platform is then added to the tree showed in the RMA GUI.

◆ Add Remote Platform via URL

This action allows getting the description (called APDescription in FIPA terminology) of a remote Agent Platform via a URL. The content of the URL must be the stringified APDescription, as specified by FIPA. The user is requested to insert the URL that contains the remote APDescription and the remote platform is then added to the tree showed in the RMA GUI.

◆ View APDescription

To view the AP Description of a selected platform.

◆ Refresh APDescription

This action asks the remote AMS for the APDescription and refresh the old one.

◆ Remove Remote Platform

This action permits to remove from the GUI the selected remote platform.

◆ Refresh Agent List

This action performs a search with the AMS of the Remote Platform and the full list of agents belonging to the remote platform are then displayed in the tree.

## 4.2 DummyAgent

The DummyAgent tool allows users to interact with JADE agents in a custom way. The GUI allows composing and sending ACL messages and maintains a list of all ACL messages sent and received. This list can be examined by the user and each message can be viewed in detail or even edited. Furthermore, the message list can be saved to disk and retrieved later. Many instances of the DummyAgent can be started as and where required.

The DummyAgent can both be launched from the Tool menu of the RMA and from the command line, as follows:

```
Java jade.Boot theDummy:jade.tools.DummyAgent.DummyAgent
```

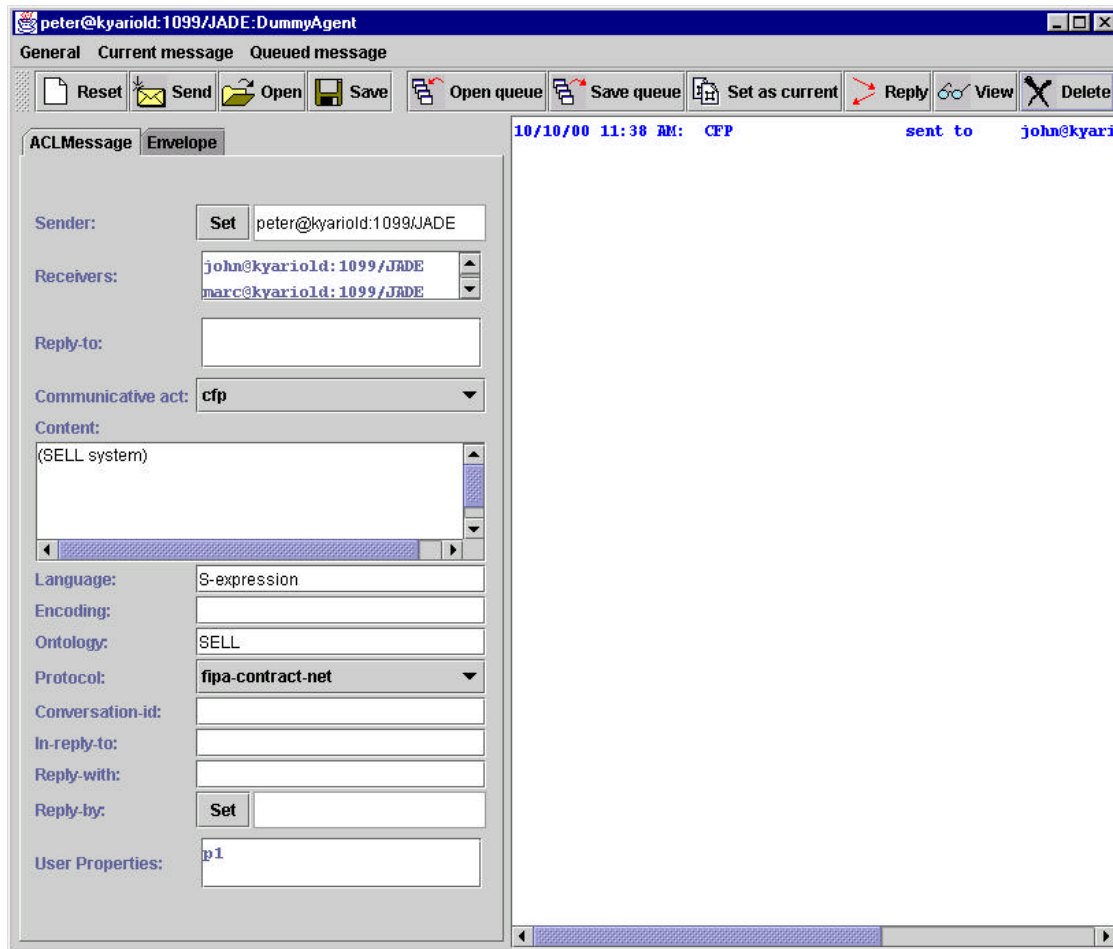


Figure 2 Snapshot of the DummyAgent GUI

### 4.3 DF GUI

A GUI of the DF can be launched from the Tools menu of the RMA. This action is actually implemented by sending an ACL message to the DF asking it to show its GUI. Therefore, the GUI can just be shown on the host where the platform (main-container) was executed

By using this GUI, the user can interact with the DF: view the descriptions of the registered agents, register and deregister agents, modify the description of registered agent, and also search for agent descriptions.

The GUI allows also to federate the DF with other DF's and create a complex network of domains and sub-domains of yellow pages. Any federated DF, even if resident on a remote non-JADE agent platform, can also be controlled by the same GUI and the same basic operations (view/register/deregister/modify/search) can be executed on the remote DF.

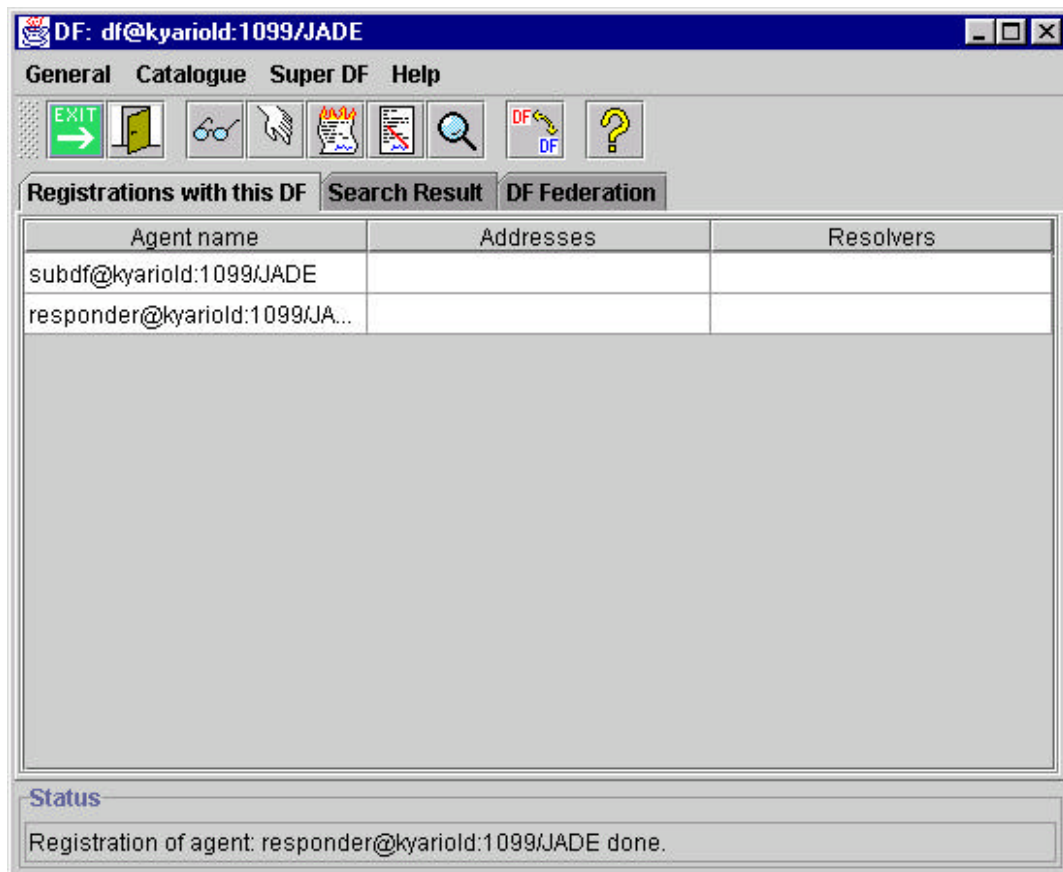


Figure 3 – Snapshot of the GUI of the DF

#### 4.4 Sniffer Agent

As the name itself points out, the Sniffer Agent is basically a Fipa-compliant Agent with sniffing features.

When the user decides to sniff an agent or a group of agents, every message directed to/from that agent / agentgroup is tracked and displayed in the sniffer Gui. The user can view every message and save it to disk. The user can also save all the tracked messages and reload it from a single file for later analysis.

This agent can be started both from the Tools menu of the RMA and also from the command line as follows:

```
java jade.Boot sniffer:jade.tools.sniffer.Sniffer
```

The figure shows a snapshot of the GUI.

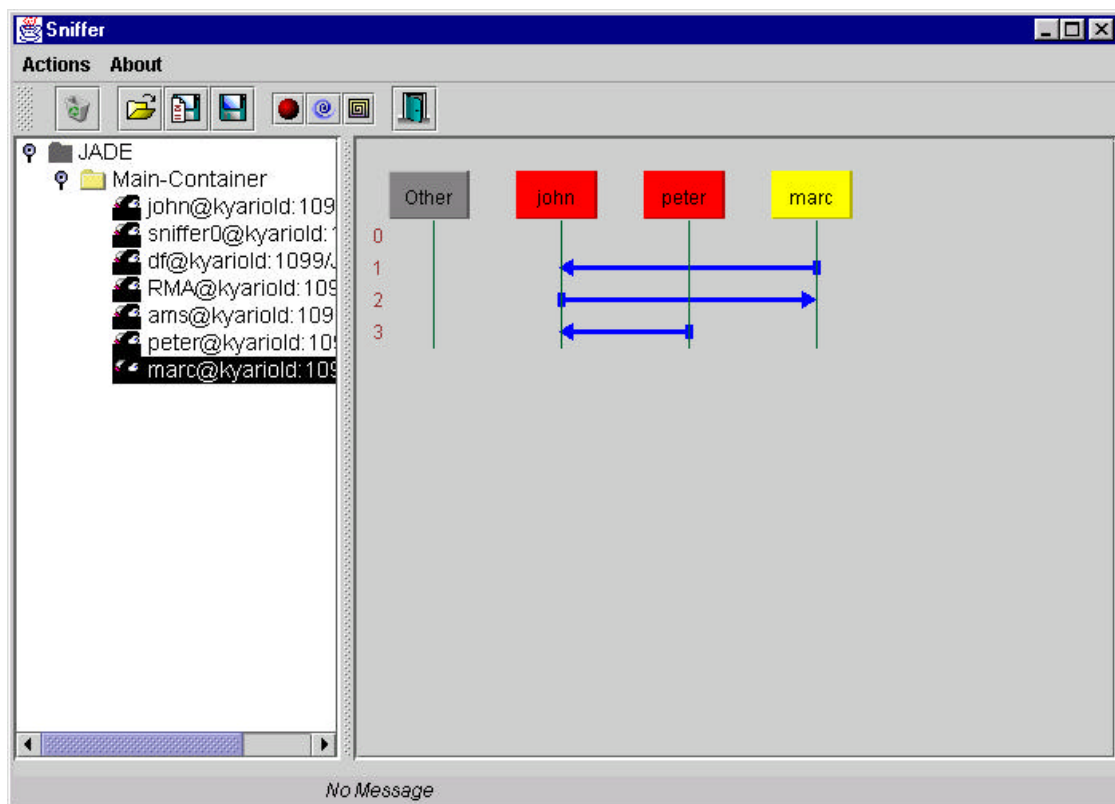


Figure 4 - Snapshot of the sniffer agent GUI

#### 4.5 Introspector Agent

This tool allows to monitor and control the life-cycle of a running agent and its exchanged messages, both the queue of sent and received messages.

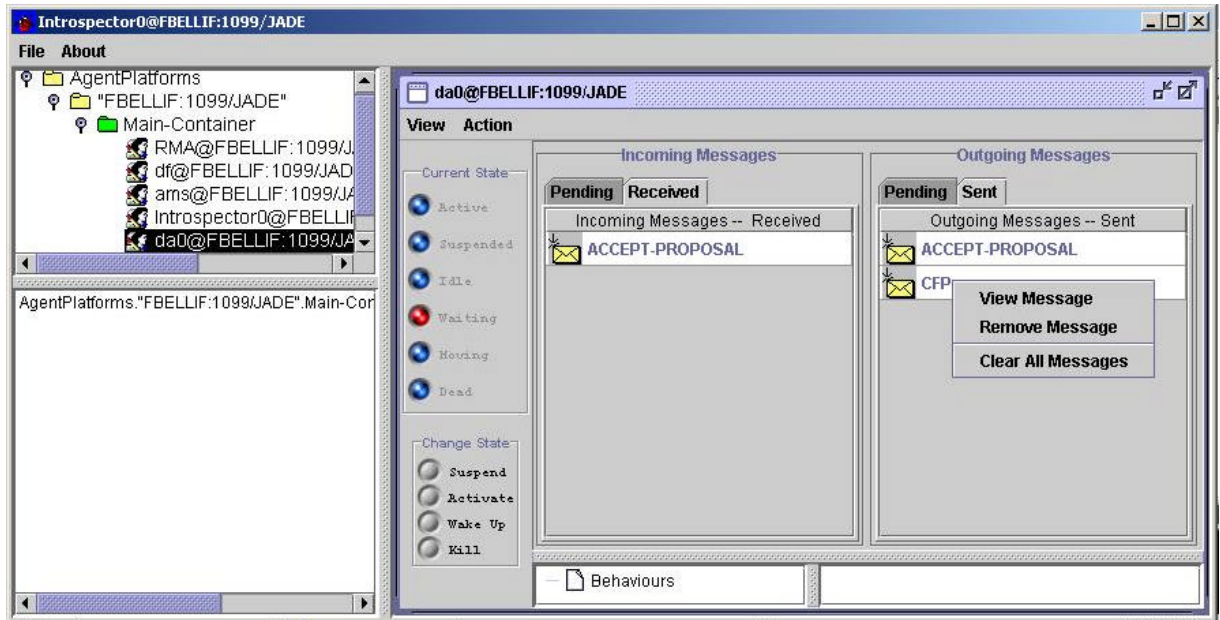


Figure 3 - Snapshot of the Introspector Agent GUI

---

**5 LIST OF ACRONYMS AND ABBREVIATED TERMS**


---

ACL	Agent Communication Language
AID	Agent Identifier
AMS	Agent Management Service. According to the FIPA architecture, this is the agent that is responsible for managing the platform and providing the white-page service.
AP	Agent Platform
API	Application Programming Interface
DF	Directory Facilitator. According to the FIPA architecture, this is the agent that provides the yellow-page service.
EBNF	Extended Backus-Naur Form
FIPA	Foundation for Intelligent Physical Agents
GUI	Graphical User Interface
GUID	Globally Unique Identifier
HAP	Home Agent Platform
HTML	Hyper Text Markup Language
HTTP	Hypertext Transmission Protocol
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
INS	
IOR	Interoperable Object Reference
JADE	Java Agent DEvelopment Framework
JDK	Java Development Kit
LGPL	Lesser GNU Public License
MTP	Message Transport Protocol. According to the FIPA architecture, this component is responsible for handling communication with external platforms and agents.
ORB	Object Request Broker
POA	Portable Object Adapter
RMA	Remote Monitoring Agent. In the JADE platform, this type of agent provides a graphical console to monitor and control the platform and, in particular, the life-cycle of its agents.
RMI	Remote Method Invocation
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language