JADE TUTORIAL
Security Administrator guide

USAGE RESTRICTED ACCORDING TO LICENSE AGREEMENT.

last update: 19-September-2002. JADE 2.61

Author: Giosuè Vitaglione (TILAB, formerly CSELT)

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 1st FIPA interoperability test in Seoul (Jan. 99) and the 2nd FIPA interoperability test in London (Apr. 01).
Copyright (C) 2000 CSELT S.p.A. (C) 2001 TILab S.p.A. (C) 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

1 INTRODUCTION

1.1 Scope and target audience

This document contains information about the configuration and usage of *JADE-S*, the *JADE Secure Agent Platform.*

JADE-S is formed by the combination of the standard version of JADE with the new JADE security plug-in. It includes features such as user/agent authentication, authorization and secure communication between agents into the same platform. JADE-S enables a set of security features and provides the base technology for programmers focused on real-world, agent-based application development.

A brief description of the security model adopted and its features is provided. Also covered are practical details about permission configuration and a guide for starting a secure JADE platform.

This guide should be used by JADE platform administrators, or developers wishing to understand and experiment with JADE-S. Some basic knowledge of JADE is required [1].

Programmers who are looking for API-level description of JADE security will want to read the generated javadoc API documentation and the “JADE Security Programmers Guide”, not yet released.

1.2 How to use this document

This document is structured so as to provide a smooth learning curve, making possible its usage as both a tutorial and a reference manual. More details about the technologies used are available in the references section, at the end of this guide.

Section 2 provides a brief description of the entities involved in JADE-S and it is not intended to be a computing security manual. Section 3 guides you through the steps of starting your first secure JADE platform. At the end, a complete JADE permission reference is provided.

1.3 System requirements

In order to use the JADE security plug-in you need JADE 2.61 or later, and Sun Java SDK 1.4. The file jadeS.jar containing the JADE security plug-in needs to be in your CLASSPATH variable. There are no requirements other than the standard JADE installation.

1.4 How to Install

Unpack the file `jadesecurity.zip` in the same location of your current JADE installation. The directory `jade/add-ons/security` is created containing all the needed files. You also need to add `jade/lib/jadeS.jar` to your Java CLASSPATH.

2 JADE SECURITY MODEL

2.1 Overview

Distributed systems require a high level of security at both the infrastructure and application level. Distributed Multi-Agent systems, leveraging agent’s autonomy and mobility, require even greater attention to security issues.

JADE-S enables a set of security features and provides the base technology for programmers focused on real-world, agent-based application development.

Multi-agent systems without security support should not provide e-commerce services or other applications using the Internet. A malicious agent 'M', for example, could *kill* a reseller agent 'A', and take its place selling items, placing/accepting orders and getting payments, acting with a fake identity. As another example, anyone could connect a remote container to your platform, move a malicious agent onto your main container, and read/delete all the files on your machine.

JADE-S makes your JADE platform a controlled multi-user environment, where all the components are owned by authenticated users, whom in turn are authorized by the platform administrator to perform only certain privileged actions.

JADE-S is based on the Java security model [2] and extends it for multi-agent systems. It also takes advantage of JAAS (Java™ Authentication and Authorization Service), JCE (The Java™ Cryptography Extension) and JSSE (The Java™ Secure Socket Extension) technologies in order to provide a rich set of security features to agent-based application developers.

2.2 Basic Concepts

A JADE agent platform can be spread across multiple containers over several hosts. A JADE-S platform, which uses the JADE security support, is a multi-user environment where each component (e.g. an agent or a container) is owned by a user who is responsible for its actions. Users must be known to the platform and must be authenticated by providing username/password in order to take ownership of any components. A simple scenario with two users that own agents on three containers is showed in Figure 1.

Not all users can perform all actions available on the platform. The platform policy file contains information about which privileged actions can be performed by authorized users.

An agent proves its identity by showing its *Identity Certificate*, signed by the Certification Authority. Using digitally-signed certificates, the platform can be sure of the identity and the ownership of the agents, and can grant or deny permission of performing certain actions.

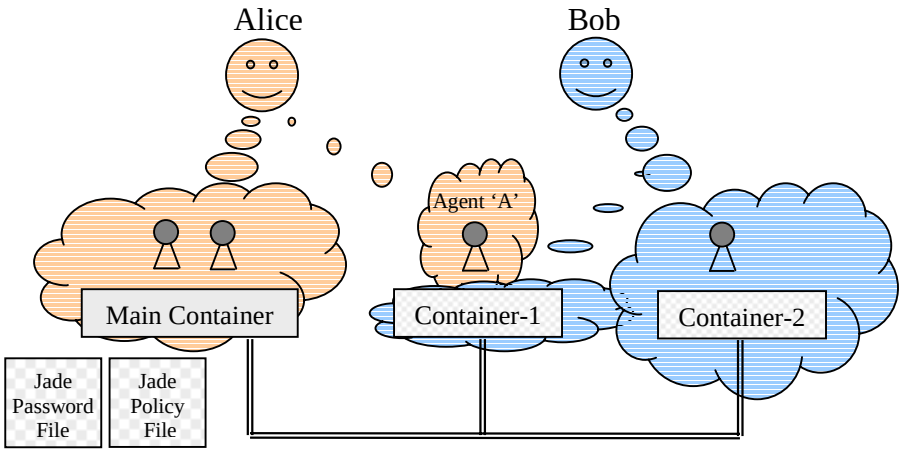


***Figure 1**: A multi-user scenario. User 'Alice' owns: the Main Container, two agents on it and the Agent 'A'. User 'Bob' owns: Container-1, Container-2 and the agent on it. Users have to login into the platform providing username and password. Authorization for performing actions is obtained only if allowed according to the JADE policy, or through the delegation mechanism. JADE agents can perform all the operations that are allowed to their owners.*

Additionally, an agent can perform other actions, using the delegation mechanism, borrowing more credentials from other agents and showing these credentials to get permission to perform those actions. These additional credentials are passed through *Delegation Certificates* that are signed by the platform's certification authority.

The sections numbered from 2.3 to 2.8 describe the mechanisms and concepts used in JADE-S, such as authentication, authorization, JADE permissions, JADE policies, certificates, the Certification Authority, the delegation mechanisms and the secure communication.

2.3 Authentication

Each component in the platform is "owned" by an authenticated user. The user who starts-up the platform initially owns the AMS, the DF and the main container. Other authenticated users can own containers or other agents.

An user must be authenticated, providing a username and password, to be able to own or perform actions on a component of the platform. This works similar to file ownership in multi-user operating systems. Usernames and passwords are checked against the JADE password file, which is unique to the whole platform and is loaded with the Main Container.

Each agent owns an I*dentity Certificate* containing the name of the agent and that of its owner. The certificates owned by an agent can be shown to prove its identity, who its owner is, and which resources it has access to.

2.4 Authorization

In a JADE-S platform, permission for accessing resources can be granted to *Principals* in a manner similar to Java Authentication and Authorization Service (JAAS)[5].

Using JAAS terminology, a *Subject* is typically a user or a role in an organization that can perform actions on resources. A certain *Subject* can have multiple *Principal*s, like each one of us can have several accounts to access to different systems.

JADE-S uses the concept of *Principal* as an abstraction for a user account, an agent or a container.

A *Principal* must be authorized by the Java security manager in order to perform privileged actions (e.g. send a message, move to a container, or play a note through the sound card). The security manager allows or denies the action according to the JADE platform's policy.

Permission to perform an action can be assigned by the platform administrator to a certain *Principal* using the policy file loaded at start-up, or can be assigned at run-time by another agent, using the delegation mechanisms.

2.5 Permissions and Policies

JADE-S uses the new Java support for principal-based authentication. Permissions are assigned not only to pieces of code, as in the old Java sandbox model, but also to who executes that code.

A Permission is an object that describes the possibility of performing an action on a certain resource, referred to as the *Target*. There are several types of permissions in Java: FilePermission, SocketPermission, AWTPermission, and many others. JADE-S introduces others: AgentPermission, ContainerPermission, etc. See section 4 for a complete reference of Permissions and actions. For each permission object there is a list with the allowed actions. Figure 2 shows a set of three permission objects that allow a Principal to perform certain actions on a Resource.

A policy specifies which permissions are available for various principals. The Java security manager permits only the actions allowed by the current policy. You can provide your policy, when you start a JADE platform, through a *Policy File* that contains a list of the allowed actions. The policy file syntax for JADE is the same as Java, and you can combine Java permissions and JADE permissions in the same policy file.

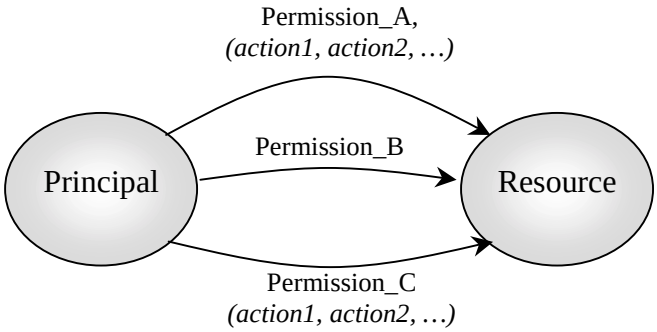


Figure 2 - A principal-based policy with more permissions on a resource

A policy file contains entries that usually look like:

```
grant principal jade.security.impl.PrincipalImpl "alice"  {
  permission jade.security.impl.AgentPermission "bob", "send-to,receive-from";
}
```

which means: allow principal called “alice” to receive and send messages from/to agents with principal “bob” (that is the resource object of this action).

The permission is defined by the class jade.security.impl.AgentPermission which allows (amongst others) the actions “send-to” and “receive-from”. This permission is granted to any agent owned by “alice”.

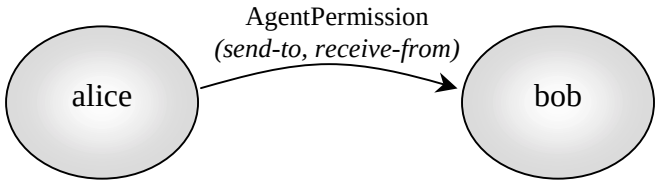


Figure 3 - Principal ‘alice’ can receive and send messages to agents owned by ‘bob’

A permission is a “positive” statement that grants a right to a principal. The current Java model does not allow “denying” statements. All permissions not explicitly granted in the policy are denied.

2.6 Certificates & Certification Authority

The Certification Authority (CA) is the entity that signs all the certificates for the whole platform. It is the only one that owns a public/private key pair. These two keys are created such that when a document is encrypted using the private key, it can be read only if the corresponding public key is owned, and vice-versa. The private key is kept secret and never given to anybody else. When the CA signs a document, it first makes a *digest* of it, which is a shorter non-reversible version of the document, a kind of a checksum. Then the digest is encrypted with the private key. This encrypted digest is maintained along with the

document. Any other entity can verify the authenticity of the document by decrypting the digest by using the CA public key, making a digest of the document again, and comparing these two digests. More information about public key encryption and certificates can be found at [6] [7].

A secure JADE platform provides a single CA into the Main Container, accessible from any container. This solution avoids a complex authorities hierarchy or problems of distributing keys to containers. It also results in a more efficient implementation since certificate verification (performed with the CA public key) occurs more frequently than certificate signing, which requires the CA private key and hence communication to the main container.

Since all JADE certificates are signed by the platform Certification Authority, they are valid only internally to the platform where they were signed.

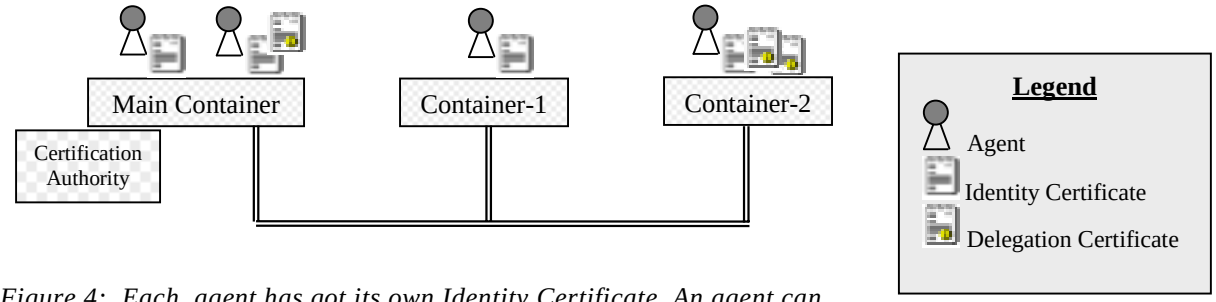


Figure 4: Each agent has got its own Identity Certificate. An agent can create Delegation Certificates, which contains the list of permissions to delegate. Such certificate can be sent to an other agent that adds it in its certificate folder and can take advantage of the delegated permissions.

2.7 Delegation

This mechanism allows the "lending" of permissions to an agent. Besides the identity certificate, an agent can also own other certificates given to it by other agents, as shown in Figure 4. Specifically an agent 'A' can delegate some permissions to another agent 'B' by sending it a *Delegation Certificate*. Once the agent 'B' has added this new certificate into its certificate folder, it is allowed to perform the privileged actions that agent 'A' delegated to it.

For example, agent 'A' has permissions to read a text file, and agent 'B' does not. Agent 'A' delegates that permission to 'B' by creating and sending to it a *Delegation Certificate*. Agent 'B' can now read that file on behalf of 'A' using the received delegation certificate.

Delegation Certificates can be limited in time so that the delegation is valid only until it expires.

2.8 Secure Communication

In order to secure the communication between agents on different containers/hosts, JADE-S enables the usage of the Secure Socket Layer (SSL) protocol, providing privacy and integrity for all intra-platform connections.

This enables a solid protection against malicious attempts of packet sniffing. More information about SSL can be found at [3]. Documentation about SSL and Java can be found at [4]. The next sections provide examples of using SSL in JADE-S.

3.1 My First JADE-S Platform

In this section we will run a secure JADE platform with a simple policy file

Open a command prompt console and go to the directory:

jade/add-ons/security/runExamples/myfirst/.

All files needed to run this example are in this directory.

3.1.1 Policy file

As explained in section 2, you need to provide a *Policy File* that says 'who' is allowed to do 'what' in the platform.

In this directory, you will find a simple policy file we use for our example: `myfirst.policy`. An explanation of the entries in this file will be provided in the following sections. At the end of this tutorial, you will be able to create and modify your own policy files.

3.1.2 Authentication and Passwords

Only authenticated users with sufficient permissions may start up a JADE-S platform. The Main container, AMS and DF are owned by the user launching the platform. To launch the platform, a user must provide a username and password matching one of those contained in the JADE password file installed in your system.

For this example you can use the password file: myfirst/myfirst.passwd. It contains two entries, a user: "alice" with password: "wannapass", and a user: "bob" with password: "letmepass".

You can also create a new password file as explained at section 3.3. If you use Unix-like machines that use DES-based (Data Encryption Standard) shadow passwords (e.g. Linux) you can use the system password (/etc/shadow) to launch a JADE-S platform.

3.1.3 Starting JADE-S

We are going to start a JADE-S platform composed of a Main Container, AMS, DF, RMA and a "Dummy Agent", as well as another container hosting a second "Dummy Agent". The scenario is shown in Figure 5 where the Main Container and its agents are owned by user 'alice'; Container-1 and its agents are owned by user 'bob'.

Please note that more complex scenarios are possible, with agents owned by same/different users on same/different containers and JVMs.

At a command prompt execute the file main.bat. After a few seconds the Main-container with all the agents, as shown in Figure 5, start up and the window of `da0` appears.

Using a second command prompt, execute the file `cont.bat` to start Container-1 with the agent `da1`. The window of `da1` will appear.

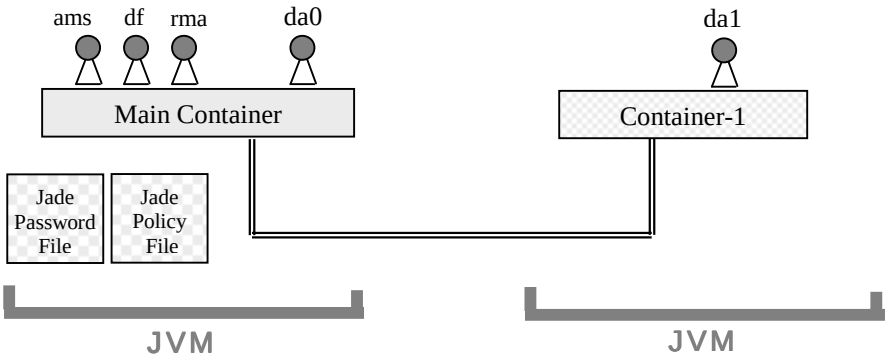


Figure 5: The JADE-S platform of the "myfirst" example with two Java Virtual Machines running the Main-Container and the Container-1.

3.1.4 Platform Start-up Details

Here we provide a description of what is contained in the files used in the previous section to start up a JADE-S platform.

In order to create the Main-Container we executed the file `main.bat`:

main.bat

```
java -Djava.security.manager
     -Djava.security.policy=myfirst.policy
     jade.Boot -gui
     jade.security.passwd=myfirst.passwd
     -owner alice:wannapass
     da0:jade.tools.DummyAgent.DummyAgent
```

- Enables Java security manager
- Policy File
- Jade Password File
- User:Password of the Platform's Owner
- Start a DummyAgent with name: "da0"

The above solution doesn't require a JADE configuration file.

As alternative, you could also use this command:

```
java jade.Boot –gui
     -conf myfirst.conf
     -owner alice:wannapass
     da0:jade.tools.DummyAgent.DummyAgent
```

providing the policy and password file name into a JADE configuration file like that:

myfirst.conf

JVM Policy File → `java.security.policy=myfirst.policy`

Jade Password File → `jade.security.passwd=myfirst.passwd`

If no password is provided with the –owner option, JADE-S will ask you for it.

The JADE password and the policy files on the main container are used by all the agents and containers for the entire platform.

In order to create the Container-1 owned by the user ‘bob’, we executed the file `cont.bat`:

cont.bat

```
java   -Djava.security.manager
       -Djava.security.policy=basic.policy
       jade.Boot -container
       -owner bob:letmepass
       da1:jade.tools.DummyAgent.DummyAgent
```

In this case no JADE configuration file is used. As alternative you could use a configuration file for setting the basic policy file name.

For all the containers in the platform (except the MainContainer) it is necessary to enable the Java security manager with the file `basic.policy` that contains the minimal Java permission to start JADE. This policy file does not contain JADE permissions, as the agents running in this container obey the JADE policy provided to the Main Container. The username**:**password is automatically checked against the main container password file, which is unique for the whole platform. As with standard JADE, this second container can be either on the same host or on a different one on the network.

3.1.5 Exchanging Messages

We will next exchange messages between `da0` and `da1`, to verify if they have the permission to do so.

Let’s send a message from the `da0` window, ACLMessage tab, add the receiver (right mouse click, add) name “`da1`”, and click on the checkbox since you provided a non-GUID name. You may fill the other fields (Content, Language, Encoding) or leave them blank. Send the message by clicking the yellow envelope icon on the top-left side of the Dummy Agent window. The message will not arrive to `da1` and an exception will be thrown.

Now[1] shut down both Container-1 and the Main-Container, and uncomment the following lines in the file myfirst.policy, as shown below:

```
//
// --- UNCOMMENT for allowing message exchange between 'alice' and 'bob' ---
//
grant principal jade.security.impl.PrincipalImpl "alice" {
     permission jade.security.impl.AgentPermission "bob", "send-to,receive-from"; };

grant principal jade.security.impl.PrincipalImpl "bob" {
     permission jade.security.impl.AgentPermission "alice", "send-to,receive-from"; };
```

Now restart `main` and `cont`, and try again to send a message from `da0` to `da1`. No security exception is thrown and the message arrives to `da1`, select it and click on the glasses icon to see its content.

The modified policy file does allow sending messages from agents owned by ‘alice’ to agents owned by ‘bob’. Keep the platform running for the next section, where we use permissions to kill agents.

[1] Current version of JADE-S does not yet support policy refresh at run-time.

3.1.6 License to Kill

The file `myfirst.policy` contains the permission for ‘alice’ to kill her own agents. Using the RMA, kill the agent `da0`; this action is allowed and the agent disappears. Now, try to kill `da1`. An exception is thrown because ‘alice’ does not have the permission to kill agents owned by ‘bob’.

Shut down the platform (both Main and Container-1) and add this to your policy file:

```
grant principal jade.security.impl.PrincipalImpl "alice" {
    permission jade.security.impl.AgentPermission "/da1", "kill";
    permission jade.security.impl.ContainerPermission "/Container-1", "kill-in";
    permission jade.security.impl.AMSPermission "/da1", "modify";  };
};
```

The first permission allows ‘alice’ to kill agents called ‘da1’, the second allows ‘alice’ to **kill** agents **in** Container-1, and the last allows the modification to state of ‘da1’ maintained by the AMS.

Note the leading slash (‘/’) used for agent’s and containers’ name. No leading slash is used for usernames.

Start `main.bat` and `cont.bat` again. Kill agent `da1` from the RMA. The agent is killed, and no exception is thrown because this action is now legal.

3.1.7 Changing Owner

During your tests, there is a useful action that can be performed while using the RMA. You can change ownership to an agent without restarting the platform. To do this, select the agent from the RMA window, choose “Change Owner” from the menu “Actions”, and provide username/password of the new owner.

The action of changing ownership has to be allowed by the policy used. In the file `myfirst.policy` this permission is defined with the following lines:

```
grant principal jade.security.impl.PrincipalImpl "bob" {
       permission jade.security.impl.AgentPermission "alice",
                                                     "take";
       permission jade.security.impl.AMSPermission "bob",
                                                     "register,deregister,modify";
       permission jade.security.impl.AuthorityPermission "bob",
                                                     "sign-ic,sign-dc";
};
```

These permissions allow ‘bob’ to “take” ownership of agents belonging to ‘alice’.

You can go further, experimenting with more scenarios, allowing, for example, the creation of a platform (jade.security.impl.PlatformPermission), or a container (ContainerPermission), agent life cycle (AgentPermission), and so on. A complete list of JADE permissions is listed in section 4.

3.2 Secure Communication

By default, JADE communication is in clear text, and can be sniffed on the network by malicious software.

Usage of SSL/TLS (Transport Layer Security) enables secure intra-platform communication. To activate such support, it is enough to include into your configuration file:

```
imtp=jade.security.impl.RMISSLIMTPManager
```

This automatically enables SSL for all intra-platform communication. Each remote container needs to be started with the same option in order to use secure communication.

Having installed the Security Plug-in, this feature can be used also by JADE platform started with the Dummy Security (i.e.: no security manager, no policy, no passwords).

3.3 Creating a JADE Password File

In the "myfirst" example we used: `myfirst.passwd` as JADE password file.

As already stated, on some Unix-like systems you can also use your Operating System password file (usually it is called: /etc/shadow) as the JADE password file. In this case you can use the tools provided by your O.S. for creating an account or modifying your password.

Otherwise, you need to create a JADE password file. In order to do that, on any system, you can use the JADE facility contained in the class: jade.security.impl.User.

To add a new JADE user entry in a JADE password file, type:

```
java jade.security.impl.User add –name alice –pass wannapass
```

if you don't provide username or password JADE will ask you for them.

To remove an entry from the password file:

```
java jade.security.impl.User remove –name alice
```

To change password:

```
java jade.security.impl.User passwd –name alice –pass wannapass
```

All these commands work on the file "`jade.passwd`" in the current directory, if it does not exist, it is automatically created. You can specify an alternative file name with the option `–file`.

If you use a Unix-like system as user 'root', do not use jade.security.impl.User to modify your system password file (i.e. /etc/shadow), because some important system information might be lost. Use the standard tools of your operating systems instead, or use this JADE facility on a different password file.

3.4 Java and JADE policy files together

As described in section 2.5, you can set the Policy File for JADE, containing all the principal-based permission entries, which uses the same Java Policy File syntax [8]. The policy you set is used in addition to the default Java policy files: the "System Policy File" and then the "User Policy File".

If you enable a security manager, Java loads first the "System Policy File" located at:

```
java.home/lib/security/java.policy  (Unix)
java.home\lib\security\java.policy  (Win32)
```

then the "User Policy File", which is at:

```
user.home/.java.policy  (Unix)
user.home\.java.policy  (Win32)
```

You can define JADE-specific permission entries in any of the policy files mentioned above.

Note that it will read only the user policy file of the user starting the platform, if one exists, and not those of other users on your system. Note also that the policy is valid for the whole JVM, and not just for JADE, when you run more applications in the same JVM (e.g. servlet calling JADE, etc…).

3.5 More complex policy files

In a complex scenario, with many users and many agents, the policy file can become more difficult to manage. It is suggested that you frequently include comments ("//") to describe what each entry allows and, when appropriate, giving details of the reason why you set a permission.

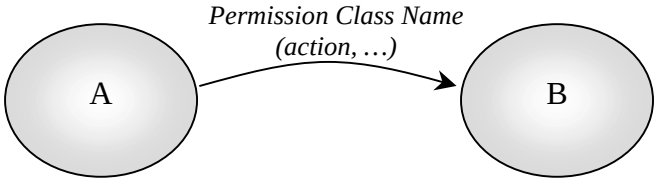
3.6 Java PolicyTool

To help with policy management, the Java SDK provides *PolicyTool* that is a graphical editor for policy files. This tool can be used for editing JADE policy file. It can be invoked with the command:

```
policytool –file myfirst.policy
```

4 JADE PERMISSIONS REFERENCE

Each of the following tables represent a JADE permission and contain the list of privileged actions permitted for each JADE permission class. The description of the actions is referred to this scenario:



That corresponds to an entry in the policy file that looks like that:

```
grant principal jade.security.impl.PrincipalImpl "A" {
      permission PermissionClassName "B", "action, ...";
};
```

It should be noted that a permission can be granted to a principal that can be a username, agent, or container. The name of an agent can be provided using a leading slash ("/").

Example:

```
grant principal jade.security.impl.PrincipalImpl "/agentname-A" {
      permission jade.security.impl.AgentPermission "username-B", "send-to";
};
```

This means that the agent called "agentname-A", can send messages to agents whose owner is "username-B".

The policy entry:

```
grant principal jade.security.impl.PrincipalImpl "username-A" {
      permission jade.security.impl.AuthorityPermission "/ams", "sign-dc";
};
```

means that agents belonging to “username-A“ are allowed to sign Delegation Certificates where the AMS agent is the delegated entity. Since the certificate is physically signed by the Certification Authority, this permission means: “be able to ask the Certification Authority to sign a Delegation Certificate”.

Another example:

```
grant principal jade.security.impl.PrincipalImpl "username-A" {
      permission  jade.security.impl.ContainerPermission  "/Container-1",  "create-
in";
};
```

means that “username-A” can create agents in the Container-1.

jade.security.impl.AgentPermission

Action	Description
`create`	‘A’ can create agents owned by ‘B’. Usually ‘A’ = ’B’.
`kill`	‘A’ can kill agents with principal ‘B’.
`suspend`	‘A’ can suspend the activity of agents with principal ‘B’.
`resume`	‘A’ can resume the activity of suspended agents with principal ‘B’.
`take`	‘A’ can take the ownership of agents with principal ‘B’.
`send-to`	‘A’ can send messages to agents with principal ‘B’.
`send-as`	‘A’ can send messages as it was ‘B’.
`receive-from`	‘A’ can receive messages from agents with principal ‘B’.
`move`	‘A’ can command to move agents with principal ‘B’.
`clone`	‘A’ can command to clone agents with principal ‘B’.

jade.security.impl.ContainerPermission

Action	Description
`create`	‘A’ can create a new container owned by ‘B’. Usually ‘A’=’B’.
`Kill`	‘A’ can kill a container owned by ‘B’.

`create-in`	‘A’ can create an agent into a container owned by ‘B’.
`kill-in`	‘A’ can kill an agent into a container owned by ‘B’.
`move-from`	‘A’ can command to move agents from a container owned by ‘B’.
`move-to`	‘A’ can command to move agents to a container owned by ‘B’.
`clone-from`	‘A’ can command to clone agents from a container owned by ‘B’.
`clone-to`	‘A’ can command to clone agents to a container owned by ‘B’.

jade.security.impl.PlatformPermission

Action	Description
`create`	‘A’ can start up a platform. Usually ‘A’=’B’.
`Kill`	‘A’ can shut down a platform owned by ‘B’

jade.security.impl.AuthorityPermission

Action	Description
`sign-ic`	‘A’ can (ask the Certification Authority to) sign identity certificates. Usually ‘A’=’B’.
`sign-dc`	‘A’ can (ask the Certification Authority to) sign delegation certificates to delegate ‘B’.

jade.security.impl.AMSPermission

Action	Description
`register`	‘A’ can register agents owned by ‘B’ to the AMS.
`deregister`	‘A’ can de-register agents owned by ‘B’ from the AMS.
`modify`	‘A’ can modify the registration of agents owned by ‘B’ in the AMS.

5 REFERENCES

[1] JADE Programmers Manual, http://jade.cselt.it/

[2] Java Security, http://java.sun.com/security/

[3] "Introduction to SSL", Netscape Communication Corp., http://developer.netscape.com/docs/manuals/security/sslin/contents.htm

[4] Java™ Secure Socket Extension (JSSE) Reference Guide, http://java.sun.com/j2se/1.4/docs/guide/security/jsse/JSSERefGuide.html

[5] Java Authentication and Authorization Service (JAAS), http://java.sun.com/products/jaas/index-14.html

[6] Introduction to Public-Key Cryptography, http://developer.netscape.com/docs/manuals/security/pkin/contents.htm

[7] Understanding Public Key Infrastructure (PKI), http://verisign.netscape.com/security/pki/understanding.html

[8] Java - Default Policy Implementation and Policy File Syntax, http://java.sun.com/j2se/1.4/docs/guide/security/PolicyFiles.html

[9] Multi-User and Security Support for Multi-Agent Systems, A.Poggi, G.Rimassa, M.Tomaiuolo (DII – University of Parma), Proceedings of WOA 2001 Workshop, Modena, Sep 2001.