JADE OSGI GUIDE

USAGE RESTRICTED ACCORDING TO LICENSE AGREEMENT.

Version: 1.0

Last update: 16-Apr-2010.

Authors: Elena Quarantotto, Giovanni Caire

TABLE OF CONTENTS

1 INTRODUCTION

This document describes the how to run JADE agents inside an OSGi environment. OSGi (www.osgi.org) is a technology that provides a component system managing, among others, software dependencies and components interactions in a clean and formal way. Readers are assumed to be already familiar with both JADE and the OSGi technology.

Since its version 3.7, besides the usual distribution, JADE is also available in form of an OSGi bundle. This JADE-OSGi bundle makes it possible to

- Launch a JADE container (main or non-main) inside an OSGi environment (see section 2).
- Create agents on top of that container from other bundles in the same OSGi environment (see section 3).
- Package the code of one or more agents inside a separated bundle and exploit the bundle-update feature of OSGi to update agents (see section 4).
- Give agents access to all typical OSGi features such as registering and using OSGi services (see section 5).

1.1 Installation

The JADE-OSGi bundle can be directly downloaded from the download area of the JADE web site (http://jade.tilab.com) and installed in whatever OSGi framework compliant to the release 3.4.0 of OSGi. How exactly to perform the installation depends on the specific OSGi framework.

1.2 Compatibility

JADE-OSGi has been successfully tested on the Equinox (http://www.eclipse.org/equinox/) and Felix (http://felix.apache.org/site/index.html) OSGi environments, but, since it makes use of standard OSGi APIs only, it is expected to work on any OSGi framework compliant to the release 3.4.0 of OSGi.

2 STARTING JADE IN AN OSGI ENVIRONMENT

The JADE-OSGi bundle includes a BundleActivator that at startup activates a JADE container. Therefore starting JADE in an OSGi environment is as simple as installing the JADE-OSGi bundle in the OSGi runtime and start it.

In general when starting a JADE container it is necessary to specify configuration options for instance to indicate whether the starting container must be a main container or not and (in case a peripheral container is started) where to find (host and port) the Main Container. All these configurations can be specified by means system properties whose names are the names of the usual JADE configuration options prefixed with "`jade.`". For instance the `jade.host` property specifies the host where to find the Main container. How to specify system properties depends on the actual OSGi framework used. Typically this is done by means of a properties file whose location and name depends on the framework.

As an example the following snippet shows the properties to specify to start a peripheral container connecting to a Main Container running on host `avalon.tilab.com` and using port `1111`.

```
...
jade.main=false
jade.host=localhost
jade.port=1111
...
```

Besides starting JADE the JADE-OSGi bundle registers a service called `jade.osgi.service.runtime.JadeRuntimeService` (JRS) in the OSGi Service Registry. This service can be used by other bundles to create agents (as described in section 3 and 4) and control the life cycle of the local JADE container. For instance the following code snippet shows how another bundle can kill the local JADE container.

```
String jrsName = JadeRuntimeService.class.getName();
ServiceReference jrsRef = context.getServiceReference(jrsName);
JadeRuntimeService jrs = (JadeRuntimeService) context.getService(jadeRef);
jrs.kill();
```

3 CREATING AGENTS

The `JadeRuntimeService` class described in previous section provides basically the same methods of the `jade.wrapper.ContainerController` class that allows controlling a container and creating agents on it from a Java application.

Therefore a bundle can easily create agents whose code is included locally by retrieving the `JadeRuntimeService` and calling its `acceptNewAgent()` method as shown below.

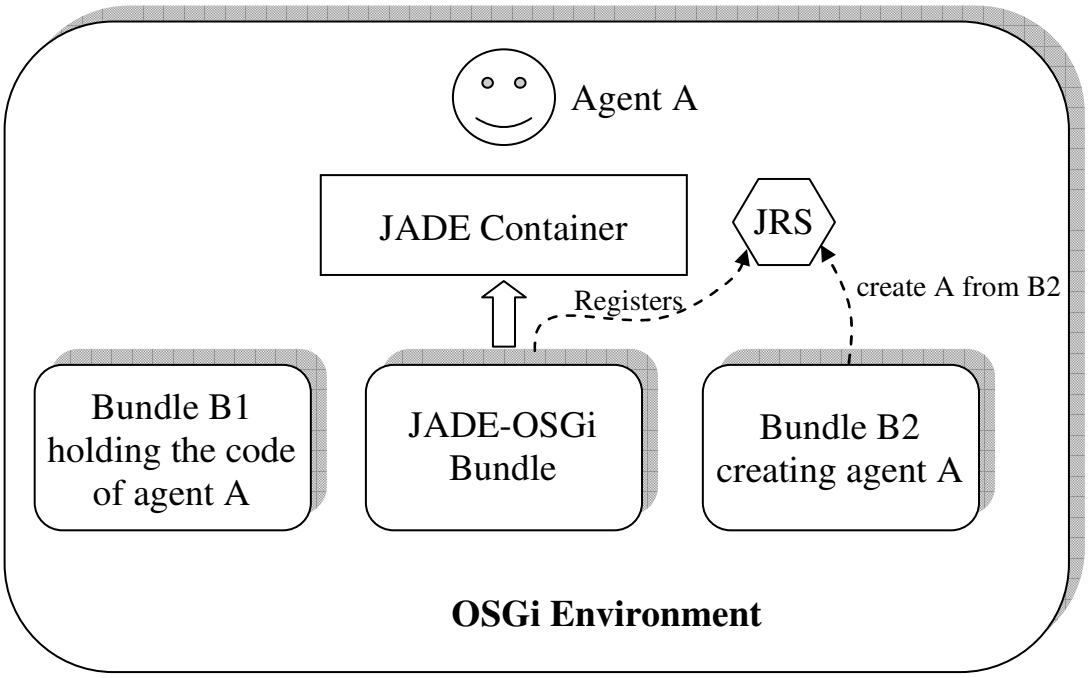
```
// Retrieve the JRS service
String jrsName = JadeRuntimeService.class.getName();
ServiceReference jrsRef = context.getServiceReference(jrsName);
JadeRuntimeService jrs = (JadeRuntimeService) context.getService(jadeRef);
// Create the agent instance
Agent myAgent = new MyBundleAgent();
// Make JADE accept the new agent
AgentController ac = jrs.acceptNewAgent("john", myAgent);
ac.start();
```

4 PACKAGING AGENTS IN SEPARATED BUNDLES

As mentioned JADE-OSGi allows packaging the code of one or more agents into separated bundles. Such agents can then be started either from other bundles by means of the `JadeRuntimeService` or remotely via the JADE AMS (e.g. from the usual JADE administration GUI). JADE-OSGi automatically takes care of killing an agent if the bundle

holding its code is stopped and restarting an agent if the bundle holding its code is updated so that any modification is immediately reflected. To clarify that, with reference to the following figure,

- Bundle B1 holds the code of agent A
- Bundle B2 creates by means of the `JadeRuntimeService` agent A
- If bundle B1 is stopped JADE-OSGi automatically kills agent A.
- If bundle B1 is updated JADE-OSGi automatically restarts (kills and starts again) agent A.



In order for a bundle to start an agent whose code is packaged into a separated bundle the createNewAgent() method of the JadeRuntimeService must be invoked as shown below.

```
// Retrieve the JRS service
String jrsName = JadeRuntimeService.class.getName();
ServiceReference jrsRef = context.getServiceReference(jrsName);
JadeRuntimeService jrs = (JadeRuntimeService) context.getService(jadeRef);
// Make JADE create the new agent
AgentController ac = jrs.createNewAgent("john", "foo.bar.MyClass", null, "B2");
ac.start();
```

Where “B2” represents the symbolic name of the bundle holding the code of the agent to start

In order to start an agent whose code is packaged in a separated bundle the AMS (e.g. from the JADE administration GUI) it is necessary to specify the agent class according to the format below.

`<class-name>[bundle-name=<bundle-symbolic-name>]`

For instance, with reference to the above example

`foo.bar.MyClass[bundle-name=B2]`

Bundles that include the code of agents that must be started either from other bundles or via the AMS must provide an `jade.osgi.service.agentFactory.AgentFactoryService` (AFS) that allows JADE-OSGi to load it. The following code snippet shows the `start()` method of the `BundleActivator` of a Bundle holding agents code.

```
AgentFactoryService agentFactory;
...
public void start(BundleContext context) throws Exception {
  agentFactory = new AgentFactoryService();
  agentFactory.init(context.getBundle());
}
```

The `init()` method of the `AgentFactoryService` class automatically manages the registration of the `AgentFactoryService` into the OSGi service registry so that the JADE-OSGi bundle can then retrieve it when requested to create agents whose code belongs to this bundle.

Similarly the `AgentFactoryService` must be deregistered when the bundle is stopped. The `clean()` method of the `AgentFactoryService` class allows doing that taking into account all necessary clean-up operations.

```
public void stop() {
  agentFactory.clean();
}
```

5 GIVING AGENTS ACCESS TO OSGI FEATURES

Agents running in an OSGi environment can access all typical OSGi features, such as registering and retrieving services in the OSGi Service Registry, by means of the helper of a JADE kernel service called OSGIBridgeService. Pay attention not to confuse OSGi services with JADE kernel services: they are both called services, but are completely different things.

The following code snippet shows how an agent can access the `BundleContext` of the Bundle containing its code. This allows the agent to access all OSGi features.

```
OSGIBridgeHelper helper;
helper = (OSGIBridgeHelper) getHelper(OSGIBridgeHelper.SERVICE_NAME);
BundleContext context = helper.getBundleContext();
```

6 APPENDIX 1. RE-COMPILING AND RE-CREATING THE JADE-OSGI BUNDLE

People interested in re-compiling and re-creating the JADE-OSGi bundle must perform the following steps.

- Download the JADE sources distribution and unzip it somewhere on the disk.

- Download the JADE-OSGi add-on from the add-ons area of the JADE web site and unzip it inside the `<jade-home>` directory.

At that point you should end up with the following directory structure

```
<jade-home>/
     |---add-ons/
     |       |---osgi/
     |             |---...
     |---...
```

- Move to the <jade-home> directory and recreate JADE for Java Standard Edition by typing

```
ant lib
```

The above command requires ANT 1.6.5 or later (http://ant.apache.org/) to be installed.

Finally move to the `<jade-home>/add-ons/osgi` directory, edit the `build.properties` file to specify the path of jar file of the OSGi framework you intend to use and re-compile/re-create the JADE-OSGi bundle by means of the `bundle` target as below

```
ant bundle
```