JADE PERSISTENCE ADD-ON

PROGRAMMING AND MANAGEMENT TUTORIAL

USAGE RESTRICTED ACCORDING TO LICENSE AGREEMENT.

last update: 29 September 2006. JADE 3.4.1

Authors: Giovanni Rimassa (FRAMeTech s.r.l.)
Giovanni Caire (Telecom Italia S.p.A.)

TABLE OF CONTENTS

1 INTRODUCTION

This tutorial deals with the JADE add-on that enables JADE-based applications to exploit persistent storage in the form of a variety of relational DBMS systems. This is achieved through a schema mapping from the relevant Java classes to relational table sets. The popular and effective **Hibernate** library is used for this purpose (see http://www.hibernate.org). The Hibernate library deals with DBMS portability and performance issues, and defines an XML-based mapping format, a Java API and an object-oriented associative query language that allow to define, store and manage almost any data structure that can be defined in Java, with a limited impact on user code. Readers of this guide are supposed to be familiar with Hibernate.

Hibernate can be configured to work with most relational database systems, however this tutorial will stick with HSQLDB (see http://hsqldb.sourceforge.net) for the demo included in the add-on. HSQLDB is an open-source, Java-only implementation of an RDBMS, that can work both as a library and as a standalone server.

This tutorial will start with a brief discussion of the main concepts that are related to JADE Persistence add-on. Most of them are not really specific of JADE or multi-agent systems, but belong rather to the general domain of persisting Java objects. Only brief remarks will be made for them, and the interested reader is invited to lookup Hibernate documentation about these topics.

The subsequent section will discuss the configuration and management of the persistence service of a JADE platform, showing the resources that need set up (configuration files, DBMS servers and the like), and illustrating the graphical management console provided within the Persistence add-on.

The Java API exposed by the persistence service, which application agents can leverage for their own purposes are detailed in the Javadoc included in the add-on.

2 JADE PERSISTENCE CONCEPTS

The aim of JADE persistence add-on is twofold. On the one hand, an API and a runtime system are provided, that can save and retrieve JADE agents on a persistent storage system (a relational DBMS managed through the Hibernate library). On the other hand, it is envisaged that JADE application developers will freely adopt Hibernate to persist their own application-specific classes (e.g. business objects pertaining to their application domain). Therefore, it is possible to add application-specific database schemes to the basic ones provided by JADE; Hibernate uses XML *mapping files* to describe how Java classes are to be represented as database tables.

The JADE Persistence add-on is fully integrated with the kernel services architecture introduced with release 3.2 of JADE. Therefore persistence functionality are activated by starting the **`jade.core.persistence.PersistenceService`** service in all containers of the platform as detailed in 3.4.

The JADE persistence add-on is designed to support persisting agents and containers in different places identified as **repositories**. This makes it possible to support multiple applications on the same platform, while keeping their persistent data separate. A repository is basically a set of tables within a DB and is uniquely associated to a Hibernate session factory, and is configured using an ordinary Hibernate properties file. Therefore different repositories can be hosted at different URLs and even by different DBMS. The descriptions of available repositories is kept in an ad-hoc repository called the "**meta repository**". When

the Persistence Service is activated on a given container it gets the hibernate properties file of the meta repository as the value of the **`meta-db`** configuration property. Even if in principle it would be possible to specify different meta-repositories, typically all instances of the Persistence Service (one per container) in a platform point to the same one.

If the meta-repository tables do not exist the Persistence Service automatically creates them. Furthermore, if the meta-repository is empty (this is always the case when the meta-repository is automatically created), a default repository, called JADE-DB, suitable for persisting agents and containers is created too and is accessible through the same Hibernate session factory of the meta-repository. Figure 1 shows the minimal (and typical) configuration where the Persistence Service components in all containers point to the same meta-repository (same Hibernate property file) and only the default repository (JADE-DB) is available for persisting agents and containers.

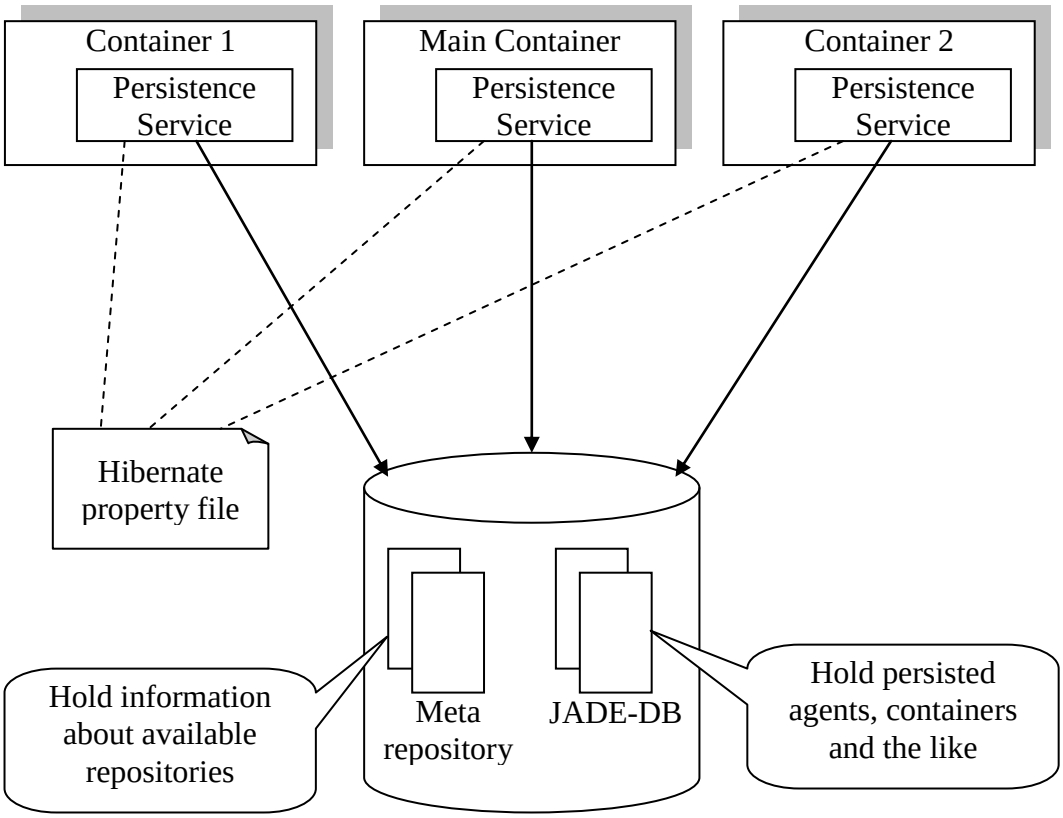


Figure 1. The typical configuration of Persistence Service repositories

2.1 Mapping files

The JADE persistence add-on defines a few mapping files to declare the basic JADE persistent entities (agents, containers, ACL messages and the like); a dynamic loading feature allows users to add other application dependent mapping files. The mapping files are:

- File **`common.hbm.xml`**: mapping for whatever types are shared by several persisted entities (agent identifiers, ACL messages and MTP descriptors).

- File **`FrozenAgent.hbm.xml`**: mapping for a *frozen* JADE agent (i.e. an agent whose state has

been swapped out and put on a database, but that still appears as alive and can receive messages).

- File **`FrozenMessageQueue.hbm.xml`**: mapping for the message queue of a frozen agent. Such an agent will have all messages persistently buffered on the database. The frozen message queue can reside on a different node with respect to the saved agent state. This allows to freeze an agent and then shutdown the container it was running on.

- File **`SavedAgent.hbm.xml`**: mapping for a persisted JADE agent. The agent state is saved using Java Serialization, so the internal agent structure (active behaviours and the like) cannot be accessed within the database.

- File **`SavedContainer.hbm.xml`**: mapping for a persisted JADE agent container. The persistent state of an agent container contains the container name, the active agents and the installed MTPs.

- File **`SavedACLMessage.hbm.xml`**: mapping for a persisted ACL Message. The whole message structure (envelope included) is represented, so that the message database can support highly structured queries on exchanged messages.

- File **`meta.hbm.xml`**: mapping for a JADE meta-repository. Persistent JADE entities (agents, containers, etc.) are stored in *Repositories*. A Repository is uniquely associated to a Hibernate session factory, and is configured using an ordinary Hibernate properties file. A JADE Repository has a *name*, a set of associated *mappings* (i.e. Hibernate XML mapping files), and a set of *properties*. As already mentioned, besides agents, containers and the like, the JADE Persistence add-on allows persisting repositories themselves. When a repository is persisted, mappings take the form of URLs pointing to the appropriate XML resources, whereas properties can either be stored within a database or use a URL pointing to a **`.properties`** resource.

3 RUNNING JADE PERSISTENCE ADD-ON

This add-on relies on a fair amount of external software to provide its services, and the configuration effort reflects this. The actual external software dependencies are the following:

- The Hibernate library itself (The **`hibernatex.y.jar`** file, contained in the root directory of the Hibernate distribution, must be added to the class path).

- Hibernate support libraries (All the Jar files in the **`lib`** subdirectory of the Hibernate distribution must be added to the class path).

- The JDBC driver for the selected DBMS (This of course depends on the actual DBMS used. In the case of HSQLDB, the **`hsqldb.jar`** file, located in the **`lib`** directory of the HSQLDB distribution, must be added to the class path).

- The DBMS server itself (This again depends on the actual DBMS, in most cases the DBMS server will not even be a Java application, so that no additional class path entries are needed. In the case of HSQLDB, the **`hsqldb.jar`** file contains both the JDBC driver and the DBMS engine).

In the following, installation, configuration and management of JADE Persistence add-on will be discussed, and some concrete examples will be given.

3.1 Installation

To successfully install the JADE persistence add-on, one has to perform all the preliminary steps required by the software dependencies listed before.

1. Set up one or more DBMS servers to hold the persisted data. These DBMSs must be supported by Hibernate (the list is quite complete and all the most popular closed and open source systems are included). In case HSQLDB is used, it must be downloaded from http://hsqldb.sourceforge.net and installed.

2. Download and install the Hibernate library from http://www.hibernate.org.

3. Have JADE libraries compiled and ready (obviously this add-on requires the base JADE runtime to work).

The actual add-on installation consist in unzipping the Persistence add-on package into the root directory of the jade distribution. At the end of this operation you should end up with a directory structure as below.

```
<jade-home>/
     |
     |--add-ons
     |      |
     |      |--persistence
     |      |      |
     |      |      |--build.xml (ant file to recompile the add-on and execute the demo)
     |      |      |--COPYING
     |      |      |--License
     |      |      |--demo/ (includes property file for the demo (see section 3.6)
     |      |      |--doc/ (includes the persistence add-on javadoc)
     |      |      |--lib/ (includes the persistence add-on jar files)
     |      |      |--src/ (includes the persistence add-on sources)
     |      |
     |      |-- … (other previously installed JADE add-ons)
     |
     |--lib/ (includes jade libraries)
     |-- … (other jade directories)
```

3.2 Compilation

The JADE Persistence add-on package already comes with the compiled sources in the form of two jar files included in the **`lib/`** subdirectory:

`JADEPersistence.jar` – Includes the add-on classes

`persistenceExamples.jar` – Includes a sample agent that shows how to activate persistence related actions.

People interested in recompiling the Persistence add-on sources have to edit the **`hibernate-home`** property in the build.xml ant file to reflect their Hibernate installation and use the lib target to re-create the above jar files.

3.3 Configuration

The configuration of JADE persistence add-on relies exclusively on Hibernate property files, without any further information. The reader can refer to Hibernate documentation for a precise explanation of the various Hibernate properties. Generally speaking, Hibernate supports a lot of properties for fine tuning, but only a handful of them are really required to set up a concrete DBMS. In the case of HSQLDB, the following lines are used (of course the JDBC URL is an example).

```
hibernate.dialect net.sf.hibernate.dialect.HSQLDialect
hibernate.connection.driver_class org.hsqldb.jdbcDriver
hibernate.connection.username sa
hibernate.connection.password
hibernate.connection.url jdbc:hsqldb:foo.telecomitalia.it
```

The lines above (or a similar set, depending on the chosen DBMS, the installation path, etc.) will be written into a Java properties file. When a JADE Container starts up and the Persistence Service boots, it looks for the Hibernate properties according to the following policy:

- If a **`-meta-db`** command-line argument or **`meta-db`** profile parameter is present, its value is the URL pointing at the Java properties file to load. If this value is not a valid URL it is interpreted as the pathname of the properties file to load in the classpath.

- Otherwise, the default Hibernate policy is used, which is to look for a file named **`hibernate.properties`** within the class path.

The configured Hibernate properties will point to a JADE meta-repository, whose tables contain persisted instances of the **`jade.core.persistence.Repository`** class. This class embeds the information related to a repository where actual agents, containers and other business objects will be persisted. Each repository has a name, a set of mappings and a set of properties (which can be stored within the database tables or in an external properties resource). This means that, even if a JADE container holds a single meta-repository, it actually can be configured with several different repositories, possibly hosted by different DBMSs. This makes it easy to support multiple applications on the same agent container, while keeping their persistent data separate.

3.4 Activation

The JADE Persistence add-on is fully integrated with the kernel services architecture introduced with release 3.2 of JADE. Therefore persistence functionality are activated by starting the **`jade.core.persistence.PersistenceService`** service in all containers of the platform. As an example the following command line can be used to start a JADE Main Container with the Persistence Service active on it.

```
java jade.Boot -gui -services jade.core.persistence.PersistenceService
-meta-db JADE_persistence.properties
```

The above command line assumes the classpath is correctly set including the jade classes, the Persistence add-on classes, the Hibernate classes, all Hibernate supporting libraries and the JDBC driver of the selected DBMS.

3.5 Operation and Management

The JADE persistence add-on supports a wide array of operations to manage the persistent repositories.

1. **Save-Agent**: save a snapshot of the specified agent state on a given repository. The agent must be serializable, and the actual snapshot is made at the next scheduling checkpoint (i.e. as soon as a behaviour returns control to the JADE scheduler).

2. **Load-Agent**: create a new agent reading its initial state from a given repository.

3. **Reload-Agent**: reset the state of the specified agent, and replace it with the state stored in a given repository. The agent will be reloaded at the next scheduling checkpoint (i.e. as soon as a behaviour returns control to the JADE scheduler).

4. **Delete-Agent**: remove the specified saved agent from a given repository.

5. **Freeze-Agent**: save a snapshot of the specified agent state on a given repository, and stop the running agent, but do not destroy its identity. Instead, set up a message buffering mechanism, hosted by a given agent container (which may or may not be the one hosting the now frozen agent). The frozen agent will appear to be running on the buffering container, and any ACL message due to it will be persisted on the given repository.

6. **Thaw-Agent**: restart a previously frozen agent, reading its state from the given repository, and delivering to it all its buffered ACL messages.

7. **Save-Container**: save a snapshot of the specified agent container on the given repository. The container name, all hosted agents, and all installed MTP endpoints are saved.

8. **Load-Container**: reload a whole agent container from a given repository. First all active agents and MTP endpoints are terminated, and then they are reloaded back from the persistent storage. This feature could rather be called *Reload-Container*, because it works more like *Reload-Agent*; to have a new container load its state at startup time, a **`-load-from`** command line argument and a **`load-from`** profile parameter are given, whose value must contain the name of the repository where the saved container state resides. Combining the **`meta-db`** and **`load-from`** settings, one can choose exactly where to load a container from.

9. **Delete-Container**: delete the specified saved container from a given repository.

The above actions can be triggered in three ways.

- By means of the JADE Persistence service API. A **`PersistenceHelper`** interface provides application agents with methods to save, reload and freeze themselves. As usal for kernel level services, the **`PersistenceHelper`** can be retrieved by means of the **`getHelper()`** method of the **`Agent`** class specifying the **`PersistenceService.NAME`** constant as parameter.

- By requesting the JADE AMS to perform actions of the **`JADE-Persistence`** ontology. This ontology includes actions corresponding to all the above operations. Both the ontology class and its actions are included in the **`jade.domain.persistence`** package.

- By means of the Management GUI. The JADE RMA has been extended to allow basic persistence operations. Moreover, the JADE persistence add-on contains a graphic management console that allows controlling a set of platforms and meta-repositories from a single GUI.

The first two methods of accessing persistence services will be dealt with in the next section, while the JADE Persistence Management GUI will be discussed in the following.

3.6 The demo

The JADE Persistence add-on comes with a minimal demo that provides some examples of configuration files and allows users to familiarize with the different persistence related concepts and actions. The demo makes use of the HSQL DB as DBMS. Therefore, before running the demo it is necessary to

- Downloaded the HSQL DB from http://hsqldb.sourceforge.net and install it.
- Edit the **`hsqldb-home`** property in the build.xml ant file to reflect the Hsqldb installation.
- Start the HSQL DB server by means of the **`runhsqldb`** ant target.

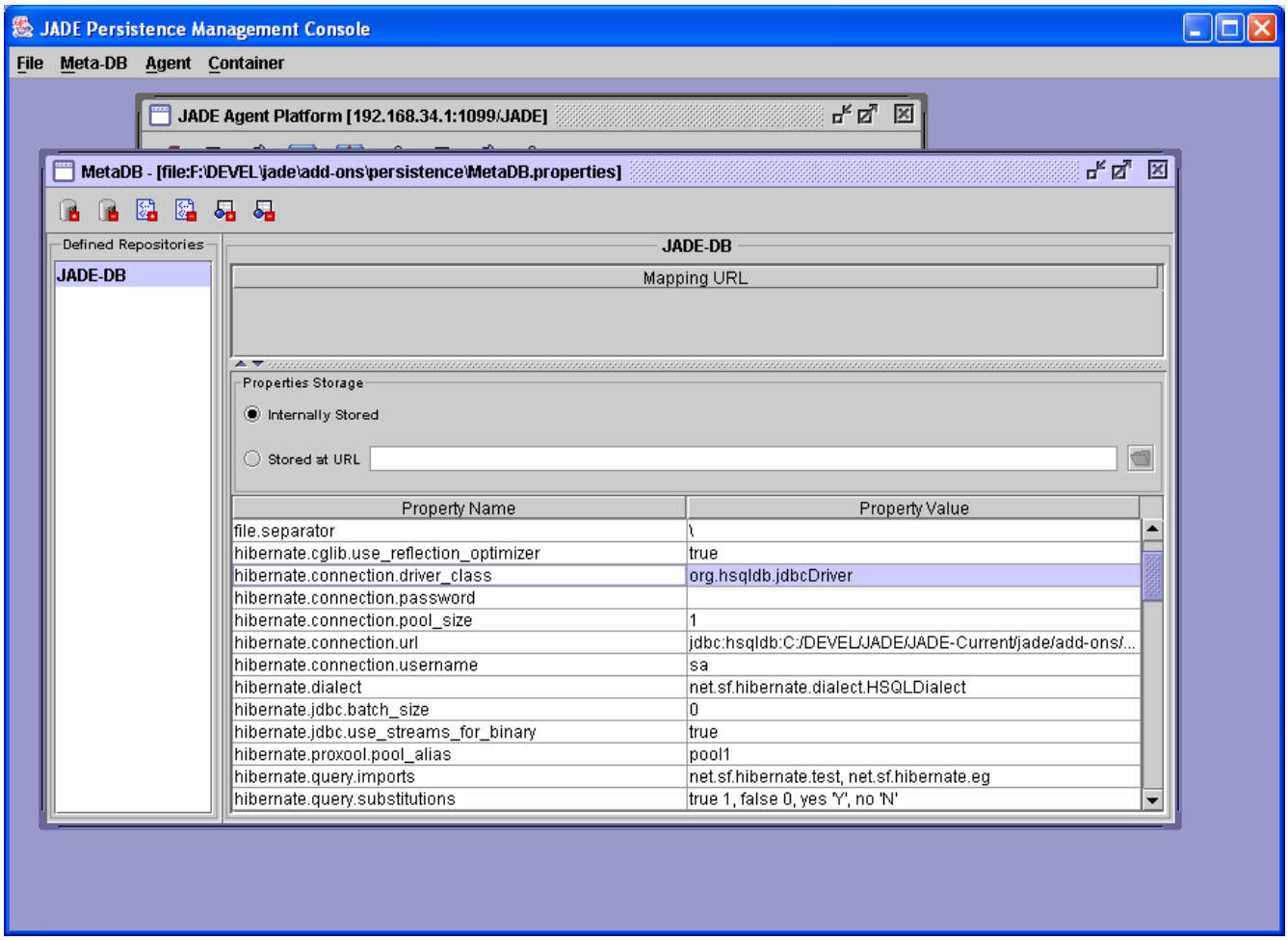
The **`demo/`** subdirectory contains a hibernate property file (hibernate_JADE_Persistence.properties) that specify where to find the meta-repository and three JADE property files to run a Main Container (main.properties), a peripheral container (container.properties) and to load the previously saved peripheral container(loadContainer.properties). In order to simplify classpath setting operations the demo can be activated by means of the **`rundemo`** ant target. This target prompts the user for inserting the name of the JADE property file to use (this feature requires ANT 1.6.5 or later). Therefore, for instance, in order to start the Main-Container it is necessary to type **`demo\main.properties`** (**`demo/main.properties`** on a Linux like machine). An agent of the **`examples.persistence.PersistentAgent`** class is also started in the peripheral container. Once both the Main-Container and the peripheral container are up and running it is possible to experiment the various persistence related actions either by means of the save, load, freeze and thaw buttons available in the JADE RMA or by activating the JADE Persistence Management Console described in the following section.

3.7 Using the JADE Persistence Management Console

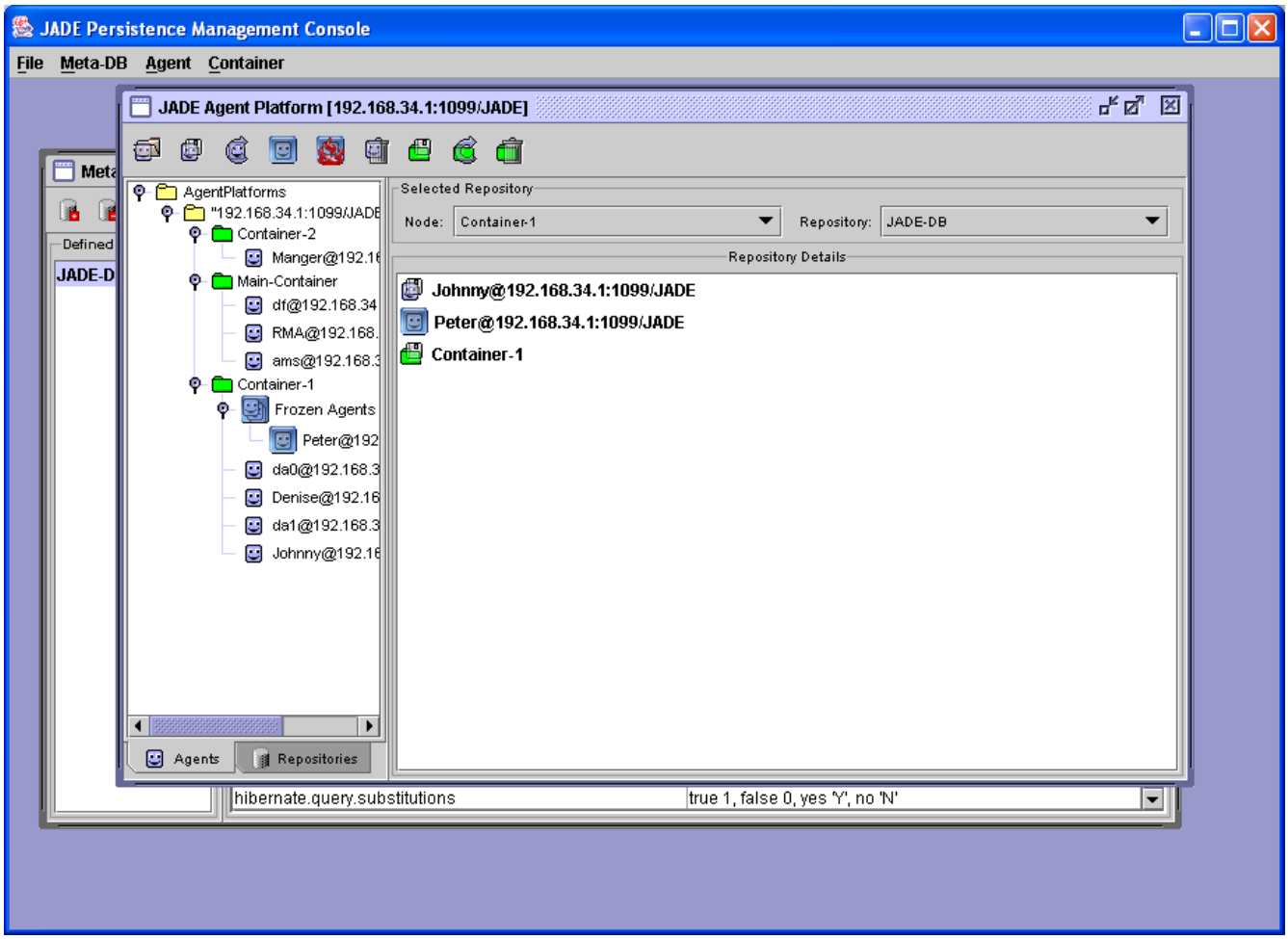
The JADE persistence add-on can be effectively managed through the provided management console. This console works as a standalone Java application, requires the Hibernate libraries and its main class is **`jade.tools.persistence.PersistenceManagerGUI`**. In order to simplify classpath setting operations the console can be activated by means of the **`runconsole`** ant target.

The console GUI is a multiple document (MDI) Swing application. The main window can contain several child windows, divided into two kinds:

1. *Meta-DB Window*. Windows of this kind are directly connected to a JADE meta-database and allow managing JADE repositories at the meta-level without the need of a running a JADE platform. Meta-DB windows are opened choosing the **`File|Meta-DB`** menu option, and then typing a symbolic alias together with an URL pointing to the Hibernate properties resource describing the meta-database. This property resource will define an Hibernate controlled DB where a set of JADE repositories are stored. The Meta-DB window will list the available repositories and will provide a toolbar to add and remove repositories, and to edit the list of the XML mappings and the set of properties (the same actions are available via the **`Meta-DB`** menu). The figure below shows the GUI with a Meta-DB window active.



2. *Platform Window*. Windows of this kind connect to a JADE platform and allow executing persistence actions on a live agent society (e.g. saving and restoring agents and containers). Platform windows are opened choosing the **`File|Platform`** menu option, and have a toolbar giving access to all the persistence operations made on agents and containers (which are also available within the **`Agent`** and **`Container`** menus). Moreover, a platform window allows selecting one platform node and one repository, so to browse its contents. The figure below shows the GUI when a Platform window is active, and the **`JADE-DB`** repository on the **`Container-1`** node holds a saved agent named **`Johnny`**, a frozen agent named **`Peter`** and a saved container named **`Container-1`**.


JADE Persistence Management Console
File
Meta-DB
Agent
Container
JADE Agent Platform [192.168.34.1:1099/JADE]
AgentPlatforms
"192.168.34.1:1099/JADE
Container-2
Manger@192.16
Main-Container
df@192.168.34
RMA@192.168.
ams@192.168.3
Container-1
Frozen Agents
Peter@192
da0@192.168.3
Denise@192.16
da1@192.168.3
Johnny@192.16
Agents
Repositories
Selected Repository
Node:
Container-1
Repository:
JADE-DB
Repository Details
Johnny@192.168.34.1:1099/JADE
Peter@192.168.34.1:1099/JADE
Container-1
Defined
JADE-D
hibernate.query.substitutions
true 1, false 0, yes 'Y', no 'N'

4 PROGRAMMING WITH THE PERSISTENCE ADD-ON

TO BE DONE.

Refer to the javadoc of the **`jade.core.persistence.PersistenceHelper`** class for persistence related actions to the local agent

Refer to the javadoc of the classes included in the **`jade.domain.persistence`** package for persistence related actions to remote agents and containers