

ECE 674 – Agent Technology
Third Course Project
Agent Implementation

**Agent Based Distributed Power Grid
Simulated with JADE FRAMEWORK**

Saad Sadiq – C11954772
May 2, 2016

Table of Contents

I. Introduction.....	3
II. Distributed Power Grids.....	3
III. Agent Simulation using JADE.....	4
Installing Jade.....	5
Starting JADE	6
Testing the Hello World program.....	9
Shutting Down the Platform	10
Java Swing.....	10
Log4j.....	11
IV. Simulation & Results.....	13
Agent communication.....	13
Code Hierarchy.....	15
The program has been coded in the following hierarchy. The program starts with the main class and starts the Jade GUI and the platform manager. The load and generator agents can be initialized from the platform manager gui created in java swing. The generator agents can be configured with several power types that it will advertise. The load agents can be configured with a search criteria that it will use to filter power providers from each generator agent. The power request info and power sell info are two classes used by the generator and load agents respectively when they are negotiating the purchase. Below is the code hierarchy.....	15
V. Conclusion.....	27

I. Introduction

The terminology of Multi-agent Systems is generally referred to entities that are autonomous, social and proactive while practicing their decision powers and existing in their own environments. In these multi-agent systems, individual agents are independent and they make decisions based on their own belief, desire and intention sets. In the various multi-agent systems that have been proposed or developed recently, a wide variety of agents exist ranging from fully autonomous intelligent agents down to relatively simple entities, such as rules or cluster of rules. Multi-agent System have come forth as game changing frameworks in implementing distributed and cooperative services to difficult research problems. Such frameworks require omnipresent agents in the environment for observing, communicating, data collecting and analysis. MAS can be utilized to solve challenging problems that are impossible for humans to resolve. They can also be used in many research areas ranging from economics to computer games, and social sciences. Moreover, several complex phenomena can be represented by multi-agent systems such as trading stock markets , urban planning simulations and health care clinical trials.

To expand our understanding and research portfolio, we start by simulating a distributed power grid framework in this project. Distributed power grids are currently highly investigated by researchers around the globe. The following section further discusses the motivation behind selecting distributed power grid and its crucial importance. The rest of the report is organized as follows. Section II discusses the importance of distributed multiagent systems in Power grids and why it is the future of electrical power transmission and distribution. In Section III we provide details of the Jade platform, Java swing and log4j. We also describe the installation and setup details and various environment configurations to make them work. In section IV we describe and our simulation and also provide results of our implementation. Section V concludes this project while indicating the key aspects for future enhancements.

II. Distributed Power Grids

A distributed MAS comprises of many agents that collaborate with each other to achieve a relative global objective. Each agent is responsible for their own specific duties that may or may not be part of the global solution. The consequent result of this cooperation between different agents is a collective property of the multi-agent system. Distributed multi-agent systems is a very prominent research field and it has made its way into numerous industries to solve complex real world problems. The distributive nature of these multi-agent systems reveal that, in most cases, tasks are not achievable by individual agents acting alone, but rather by continuous fundamental interaction between the distributed agents. These distributed MAS frameworks have attracted enormous attention of the scientific community to provide solutions in a variety of industrial domains.

Autonomous control of power systems using distributed intelligent agents has overcome many limitations of the conventional power grids. distributed multi-agent solutions are composed of multiple intelligent agents that collaborate to solve challenges that may be impossible for each individual agent. In recent years, MAS have been employed in a wide range of power system applications including

modeling of electricity markets, grid protection, fault restoration and grid control. The Consortium for Electric Reliability Technology Solutions (CERTS) indicates a few critical issues in the management of micro-grids:

1. Addition of new micro sources without upgrading current systems.
2. Join or disconnect from the grid in a smooth and swift manner
3. Control of active power and voltage imbalances
4. Meeting the dynamic load requirements of major grids

The limitations of traditional modeling methods in dealing with the added complexity of real time demand and supply optimization, transmission limits and unit commitment constraints has been shown to be overcome by agent based simulation models. Distributed Agents facilitate distinct players in real-time markets to optimize their performance by maximizing their individual utility through an auction-based mechanism.

In this project a Market-driven management of distributed, micro storage devices is simulated where there are three types of agents. The generator agents (GA), load agents (LA) and directory facilitator agents (DF). The generator agents (GA) advertise their distributed energy resources (DER) to the DF agent while the load agents (LA) find out the lowest price energy unit with the highest sustainability rating, that can be paid for the required Kilo-Watt-Hour (KWH) demand. The whole simulation was performed using JADE agent simulation tool. Further details of Jade are provided in the following section.

III. Agent Simulation using JADE

This section provides an overview of the JADE Architecture introducing the notions of

- Agent
- Container
- Platform
- Main Container
- AMS and DF

Figure 1 represents the main JADE architectural elements. An application based on JADE is made of a set of components called Agents each one having a unique name. Agents execute tasks and interact by exchanging messages. Agents live on top of a Platform that provides them with basic services such as message delivery. A platform is composed of one or more Containers. Containers can be executed on different hosts thus achieving a distributed platform. Each container can contain zero or more agents. For instance, with reference to the figure 1 below, container "Container 1" in host Host 3 contains agents A2 and A3. Even if in some particular scenarios this is not always the case, you can think of a Container as a JVM (so, 1 JVM ==> 1 container ==> 0 or many agents). A special container called Main Container exists in the platform. The main container is itself a container and can therefore contain agents, but differs from other containers as it must be the first container to start in the platform and all other containers register to it at bootstrap time and secondly, it includes two special agents: the AMS

that represents the authority in the platform and is the only agent able to perform platform management actions such as starting and killing agents or shutting down the whole platform (normal agents can request such actions to the AMS). The DF that provides the Yellow Pages service where agents can publish the services they provide and find other agents providing the services they need. It should be noticed that if another main container is started, as in host *Host 4* in Figure 1, this constitutes a new platform.

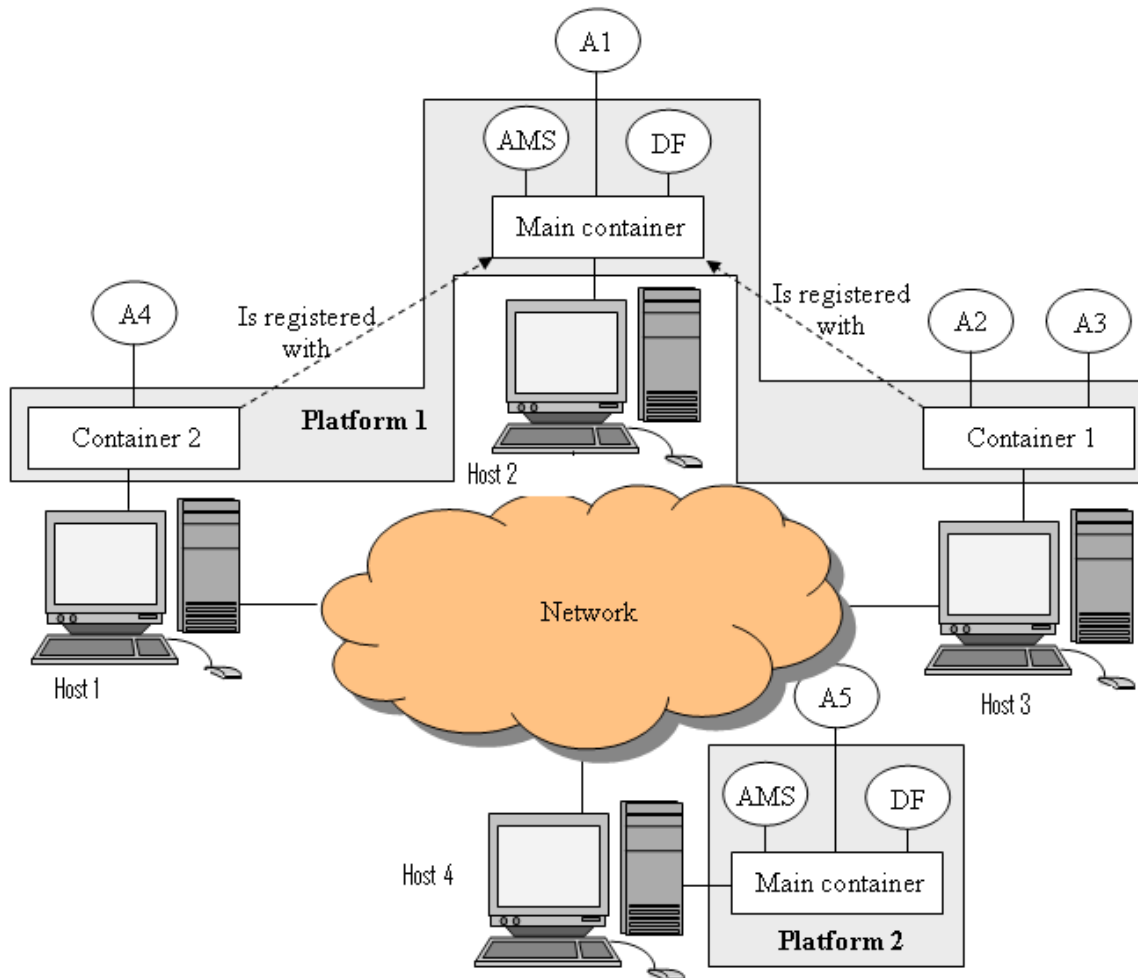


Figure 1. The JADE Architecture

Installing Jade

In this section, the necessary per-requisites and installation guidelines are presented so that any future expansions can be followed without difficulties. First it is necessary to have the latest supported version of Java installed on the system.

Step 1. Download and install the Java SE SDK (version 7u5).

Step 2. Download Eclipse (Juno release, Eclipse IDE for Java Developers package).

Step 3. Download and unzip JADE (version 4.2.0), resulting in the following directory structure:

```
c:\jade-4.2.0\classes (from JADE-src-4.2.0.zip)
c:\jade-4.2.0\demo (merge from JADE-bin-4.2.0.zip and JADE-examples-4.2.0)
c:\jade-4.2.0\doc (from JADE-doc-4.2.0.zip)
c:\jade-4.2.0\lib (merge from JADE-bin-4.2.0.zip and JADE-src-4.2.0.zip)
c:\jade-4.2.0\src (merge from JADE-src-4.2.0.zip and JADE-examples-4.2.0)
```

Starting JADE

We will now launch the JADE runtime and provides a brief description of the main features of the JADE administration GUI. Launching the JADE runtime means creating a Container, such container can then contain agents that can be started directly at container startup time or later on e.g. through the JADE Management GUI. Moreover, being the first container in the platform, such container MUST be the platform Main Container.

In the previous section, when mentioning Download Jade ver 4.2, we assumed that the following packages were downloaded

- The JADE binary distribution (JADE-bin-4.0.zip)
- The JADE source distribution (JADE-src-4.0.zip)
- The JADE examples distribution (JADE-examples-4.0.zip)

After unzipping the packages, the Jade directory structure should look something like figure 2:

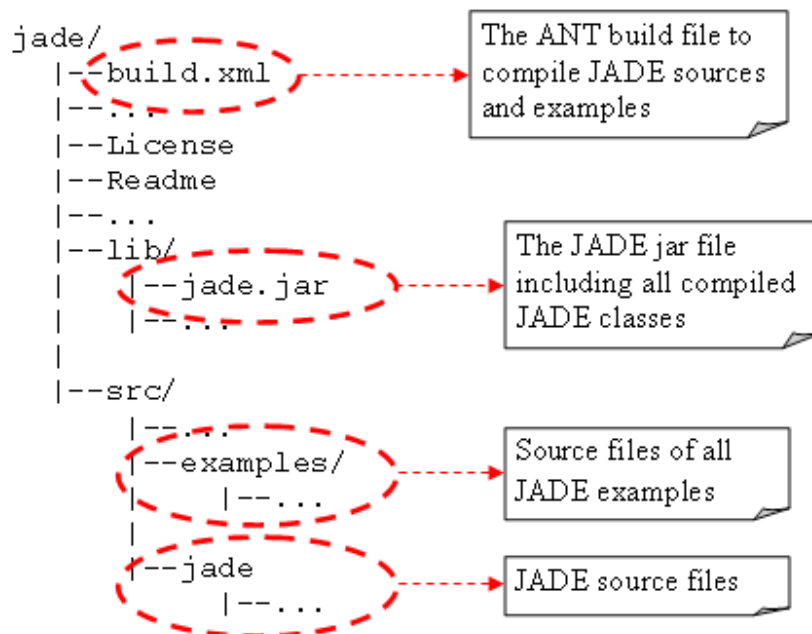


Figure 2: The Jade directory structure

When starting Jade, it should be noted that the Jade runtime can be launched in several ways. The

easiest way is to open a shell, move to the jade directory and type

```
java -cp lib\jade.jar jade.Boot -gui
```

You should see an output similar to that below.

```
May 02, 2016 12:06:12 PM jade.core.Runtime beginContainer
```

```
INFO: -----
```

```
  This is JADE 4.4.0 - revision 6778 of 21-12-2015 12:24:43  
  downloaded in Open Source, under LGPL restrictions,  
  at http://jade.tilab.com/
```

```
-----
```

```
May 02, 2016 12:06:12 PM jade.imtp.leap.LEAPIMTPManager initialize
```

```
INFO: Listening for intra-platform commands on address:
```

```
- jicp://10.33.13.153:1099
```

```
May 02, 2016 12:06:13 PM jade.core.BaseService init
```

```
INFO: Service jade.core.management.AgentManagement initialized
```

```
May 02, 2016 12:06:13 PM jade.core.BaseService init
```

```
INFO: Service jade.core.messaging.Messaging initialized
```

```
May 02, 2016 12:06:13 PM jade.core.BaseService init
```

```
INFO: Service jade.core.resource.ResourceManagement initialized
```

```
May 02, 2016 12:06:13 PM jade.core.BaseService init
```

```
INFO: Service jade.core.mobility.AgentMobility initialized
```

```
May 02, 2016 12:06:13 PM jade.core.BaseService init
```

```
INFO: Service jade.core.event.Notification initialized
```

```
May 02, 2016 12:06:13 PM jade.mtp.http.HTTPServer <init>
```

```
INFO: HTTP-MTP Using XML parser
```

```
com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser
```

```
May 02, 2016 12:06:13 PM jade.core.messaging.MessagingService boot
```

```
INFO: MTP addresses:
```

```
http://ddmnn1:7778/acc
```

```
May 02, 2016 12:06:13 PM jade.core.AgentContainerImpl joinPlatform
```

```
INFO: -----
```

```
Agent container Main-Container@10.33.13.153 is ready.
```

```
-----
```

They key information to take from this console log is the information highlighted with yellow color. The first one indicates the host (typically the local host) and port (1099 by default) where the Main Container accepts incoming connections from other containers. It is possible to make Jade use a different port by means of the `-local-port <a-port>` option. For instance to make the Main Container accept incoming connections from other containers on port 1111 it would be sufficient to type the command line below.

```
java -cp lib\jade.jar jade.Boot -gui -local-port 1111
```

Similarly it is possible to make JADE use a different host name/address with respect to that read from the underlying network stack by means of the `-local-host <host-name-or-address>` option. This is typically useful when dealing with network environments where hosts can be reached only by specifying hostnames including the network domain (e.g. `rvc.eng.miami.edu`) as exemplified below

```
java -cp lib\jade.jar jade.Boot -gui -local-host rvc.eng.miami.edu
```

The second highlighted information is the name of the newly started Container (remember that the Main Container is itself a Container). By default the name assigned to a Main Container is "Main Container". You can explicitly assign a name to a Container (regardless of it is a Main or peripheral Container) by means of the `-container-name <a-name>` option. Since we specified the `-gui` option, the JADE Management Console depicted in figure 3 should also appear.

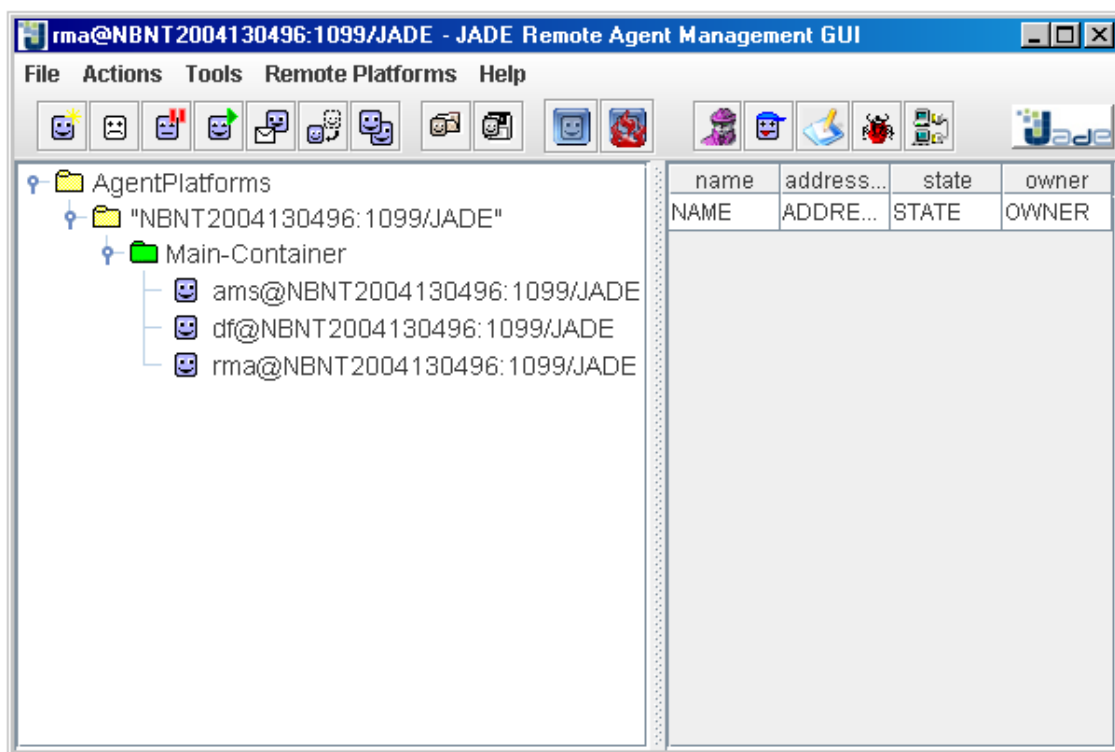


Figure 3. The JADE Management Console

In the left part of the Management Console you can see a tree showing that there is a Platform called `<Main-Container-host>:<Main-Container-port>/JADE` (note that, by the fact the we launched a Main Container, we actually created a Platform) that is composed of a single Container called Main Container, that, on its turn, contains three agents:

1. The ams i.e. agent management system
2. The df i.e. directory facilitator or the Yellow pages service
3. The rma i.e. Remote Management Agent, implementing the JADE Management Console

Looking at agent names we see that they have the form


```
<local-name>@<platform-name>
```

It is possible to set a different platform name by means of the -platform-id <a-platform-name> configuration option as in the command line below.

```
java -cp lib\jade.jar jade.Boot -gui -platform-id MyPlatform
```

If we did that, agents would have been called ams@MyPlatform, df@MyPlatform and rma@MyPlatform respectively.

Testing the Hello World program

Now we will try to connect our Eclipse IDE to the Jade platform and try running the standard Hello World program through the IDE. To achieve this we follow the below steps

- Step 1. In Eclipse: created a new Java project named HelloWorldAgent.
- Step 2. In Eclipse: navigated to menu Project > Properties; section Java Build Path; tab Libraries.
- Step 3. Add (via “Add external JARs...” button) the two JAR files jade.jar and common-codec-1.3.jar (both from the lib directory).
- Step 4. Add a new class named HelloWorldAgent to the project, specified jade.core.Agent as the parent class.
- Step 5. Override the setup method in the new class to say hello to the console. The finished agent class should look as follows:

```
import jade.core.Agent;
```

```
public class HelloWorldAgent extends Agent {  
    protected void setup() {  
        System.out.println("Hellod. I'm an agent.");  
    }  
}
```

- Step 6. Compile the project. Eclipse automatically compiles on the fly, so we didnt have to compile explicitly with the javac command.
- Step 7. Set/edit the CLASSPATH environment variable to contain the paths of the JADE JAR files, the JADE class directory and the directory of the HelloWorldAgent:

Set a “user variable”-type environment variable with name CLASSPATH and the following value: %JADE_HOME%\lib\jade.jar;%JADE_HOME%\lib\common-codec\common-codec-1.3.jar;c:\Users\Saad\workspace\HelloWorldAgent\bin;%JADE_HOME%\classes

- Step 8. Start a up a console and boot the agent:

```
java jade.Boot -agents Saad:HelloWorldAgent
```

Step 9. (optional) If Java gave an error, you either need to specify the -classpath parameter for the java command or set the CLASSPATH environment variable.

Shutting Down the Platform

In order to shut the platform down, in the Management Gui, choose File --> Shut down Agent Platform and then select Yes when prompted for confirmation. Next we will discuss about java swing and how it has been used to develop the GUI. A short code snippet has been provided to inherit the swing libraries and make a sample box. Anyone reproducing our simulation can run this code to test java swing and its capabilities.

Java Swing

Swing API is set of extensible GUI Components to create JAVA based Front End/ GUI Applications. Java has built Swing on top of AWT API and acts as replacement of AWT API as it has almost every control corresponding to AWT controls. But unlike AWT, Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Following is a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method to show how Java swing has been implemented in this project.

```
import javax.swing.*;
public class FirstSwingExample {
    public static void main(String[] args) {
        JFrame f=new JFrame();//creating instance of JFrame
        JButton b=new JButton("click");//creating instance of JButton
        b.setBounds(130,100,100, 40);//x axis, y axis, width, height

        f.add(b);//adding button in JFrame
        f.setSize(400,500);//400 width and 500 height
        f.setLayout(null);//using no layout managers
        f.setVisible(true);//making the frame visible
    }
}
```



Log4j

We have utilized Log4j logger in our project to record the communication between agents and display it on the IDE console. Log4j is basically a reliable, fast and flexible logging framework (APIs) written in Java, which is distributed under the Apache Software License. Log4j is highly configurable through external configuration files at runtime. Following are the setting up procedures of our log4j deployment.

Step 1. Declaring dependencies :

```
pom.xml
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

For non-Maven user, visit log4j official page, download the jar and put it in the project library path manually.

Step 2. Configuring log4j.properties

Create a log4j.properties file and put it into the resources folder. Refer to the step #1 above. For standalone Java app, make sure the log4j.properties file is under the project/classes directory . The log4j properties configuration file should look like this

log4j.properties

```
Bash
# Root logger option
log4j.rootLogger=DEBUG, stdout, file

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

# Redirect log messages to a log file, support file rolling.
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=C:\\log4j-application.log
log4j.appender.file.MaxFileSize=5MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

Step 3. Logging a message

To log a message, first, we create a final static logger and define a name for the logger, normally, we use the full package class name.

Java

```
final static Logger logger = Logger.getLogger(classname.class);
```

Then, logs messages with different priorities, for example, debug, info, warn, error and fatal. Normally, we just need to use debug or error.

Java

```
//logs a debug message
if(logger.isDebugEnabled()){
    logger.debug("This is debug");
}

//logs an error message with parameter
logger.error("This is error : " + parameter);

//logs an exception thrown from somewhere
logger.error("This is error", exception);
```

We can then use the logger to test in our hello world program

```
package powergrid;
import org.apache.log4j.Logger;
public class HelloExample{
    final static Logger logger = Logger.getLogger(HelloExample.class);
    public static void main(String[] args) {
        HelloExample obj = new HelloExample();
        obj.runMe("AgentTest");
    }
    private void runMe(String parameter){
        if(logger.isDebugEnabled()){
            logger.debug("This is debug : " + parameter);
        }
        if(logger.isInfoEnabled()){
            logger.info("This is info : " + parameter);
        }
        logger.warn("This is warn : " + parameter);
        logger.error("This is error : " + parameter);
        logger.fatal("This is fatal : " + parameter);
    }
}
```

Output

Bash

```
2014-07-02 20:52:39 DEBUG HelloExample:19 - This is debug : AgentTest
2014-07-02 20:52:39 INFO HelloExample:23 - This is info : AgentTest
2014-07-02 20:52:39 WARN HelloExample:26 - This is warn : AgentTest
2014-07-02 20:52:39 ERROR HelloExample:27 - This is error : AgentTest
2014-07-02 20:52:39 FATAL HelloExample:28 - This is fatal : AgentTest
```

This verifies that our logger is running fine and we can integrate it in our code.

IV. Simulation & Results

In the perspective of multi-agent organization and development, our task is to develop a multi-agent based distributed power grid. In this distributed Grid there are two parties, people who look for energy known as 'Load Agents' and the energy providers that sell power to the distributed grid known as the 'Generator Agents'. There can be any number of load agents (LA) and generator agents (GA) at any given time in the distributed power grid. There will be 5 different types of power, each one bearing different cost and sustainability ratings, that the Generator Agents will sell to the market. These will be

1. Solar
2. Hydel
3. Wind
4. Fuel cells
5. Nuclear

The Generator agents will be able to add new power sources if they are available to their dispense or remove a source if it goes out of order. More Load agents and Generator Agents can be added to the JADE platform at any time to emulate the real life systematic working of the distributed Grid. Generator Agents (GA) register with Directory Facilitator (DF) agent right after creation and Load Agents (LA) always update their generator agent list by querying the Directory Facilitator (DF) agent. As the source or consumer human we specify some criteria to the Load agent such as

1. Type of Power
2. Max price per KWH
3. Max number of generators to query

Load Agents send query to Generator Agents, collect the results and analyses them to decide the best choice of power to buy. Then initiates byt buying process and finally buys the power with minimum cost and highest sustainability power rating.

Agent communication

We start by describing the agents communication in our project. Within Jade, agents can communicate

transparently regardless of whether they live in the same container, in different containers (in the same or in different hosts) belonging to the same platform or in different platforms. Communication is based on an asynchronous message passing paradigm. Message format is defined by the ACL language defined by FIPA, an international organization that issued a set of specifications for agent interoperability. An ACL Message contains a number of fields including

1. The sender
2. The receiver(s)
3. The communicative act
 - (also called performative) that represents the intention of the sender of the message. For instance when an agent sends an IN sends a REQUEST message it wishes the receiver(s) to perform an action. FIPA defined 22 communicative acts, each one with a well defined semantics, that ACL gurus assert can cover more than 95% of all possible situations. Fortunately in 99% of the cases we don't need to care about the formal semantics behind Communicative acts and we just use them for their intuitive meaning.
4. The content i.e. the actual information conveyed by the message (the fact the receiver should become aware of in case of an INFORM message, the action that the receiver is expected to perform in case of a REQUEST message)

following is a sequence diagram as shown in figure 4, of the agent communication sequence between the Load Agent, Generator Agent and Directory Facilitator agent. The sequence diagram is described in the following steps

1. The generator agents first registers itself with the directory facilitator as soon as it is alive.
2. The DF agent serves as a yellow papers directory for the load agent and provides them with information about all the generator agents.
3. The load agents doesn't hold knowledge about the generator agents and thus looks for the DF agent for power providers.
4. The load agents, after having list of all the GAs, will start communicating with generator agents based on the search criteria
5. After receiving details from all GAs the load agents will decide which vendor to buy from
6. After deciding, the load agents will contact specific generator agent to make the purchase.

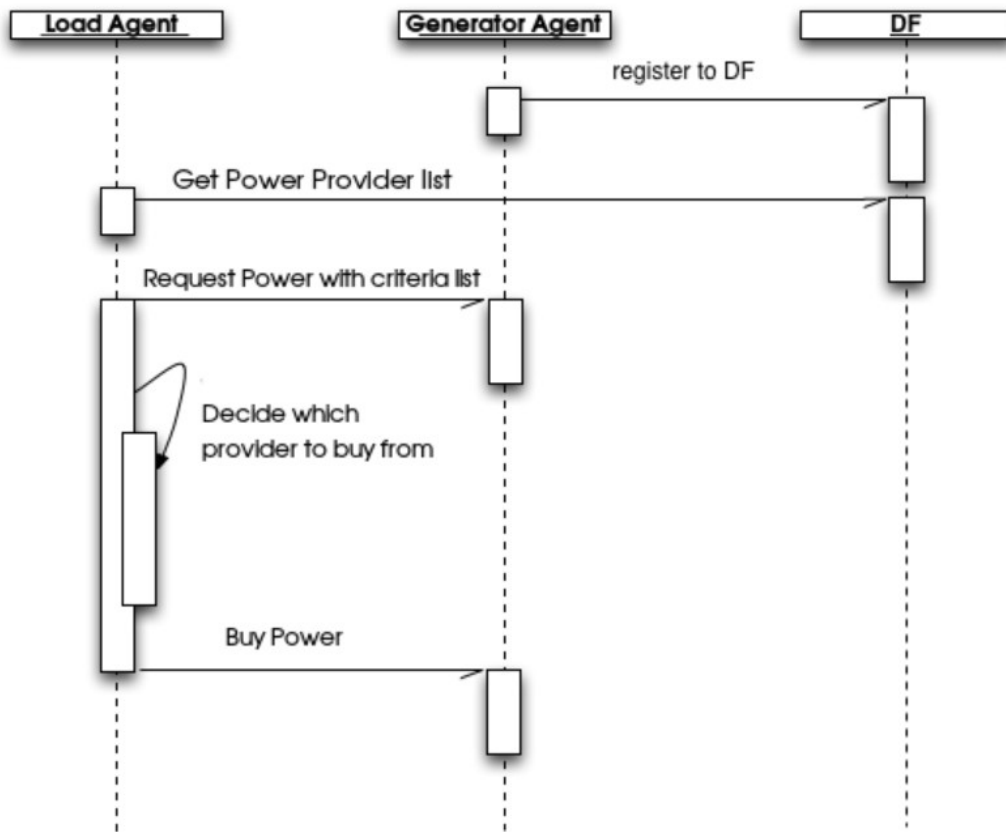


Figure 4. Sequence diagram for the agent communication

Code Hierarchy

The program has been coded in the following hierarchy. The program starts with the main class and starts the Jade GUI and the platform manager. The load and generator agents can be initialized from the platform manager gui created in java swing. The generator agents can be configured with several power types that it will advertise. The load agents can be configured with a search criteria that it will use to filter power providers from each generator agent. The power request info and power sell info are two classes used by the generator and load agents respectively when they are negotiating the purchase. Below is the code hierarchy

Main

- Jade Gui
- Platform Manager
 - Provider View
 - Generator Agent
 - PowerRequestInfo
 - Search View
 - Load Agent
 - PowerType
 - PowerSellInfo

Misc

- Logger
- Utils

Let us start the simulation and run the code. Below is the console log that is recorded using log4g. We can observe that first the jade.core.Runtime beginContainer command is used to initialize Jade 4.2. This Jade platform will be the base on which all of our agents will be created and used.

```
16:32:07,377 INFO ~ Starting platform...
Hello World
May 01, 2016 4:32:07 PM jade.core.Runtime beginContainer
INFO: -----
      This is JADE 4.2.0 - revision 6574 of 2012/06/20 15:38:00
      downloaded in Open Source, under LGPL restrictions,
      at http://jade.tilab.com/
-----
Retrieving CommandDispatcher for platform null
May 01, 2016 4:32:07 PM jade.imtp.leap.LEAPIMTPManager initialize
INFO: Listening for intra-platform commands on address:
- jicp://127.0.1.1:3250
May 01, 2016 4:32:09 PM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
May 01, 2016 4:32:09 PM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
May 01, 2016 4:32:09 PM jade.core.BaseService init
INFO: Service jade.core.resource.ResourceManagement initialized
May 01, 2016 4:32:09 PM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
May 01, 2016 4:32:09 PM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
May 01, 2016 4:32:09 PM jade.mtp.http.HTTPServer <init>
INFO: HTTP-MTP Using XML parser
com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser
May 01, 2016 4:32:09 PM jade.core.messaging.MessagingService boot
INFO: MTP addresses:
http://saadNotebook:7778/acc
```


As part of our code we have used the -gui switch to start the Jade GUI as our code runs, so we can visually monitor the progress and status of each agent. The Jade management console should start in its initial state and should look like figure 5.

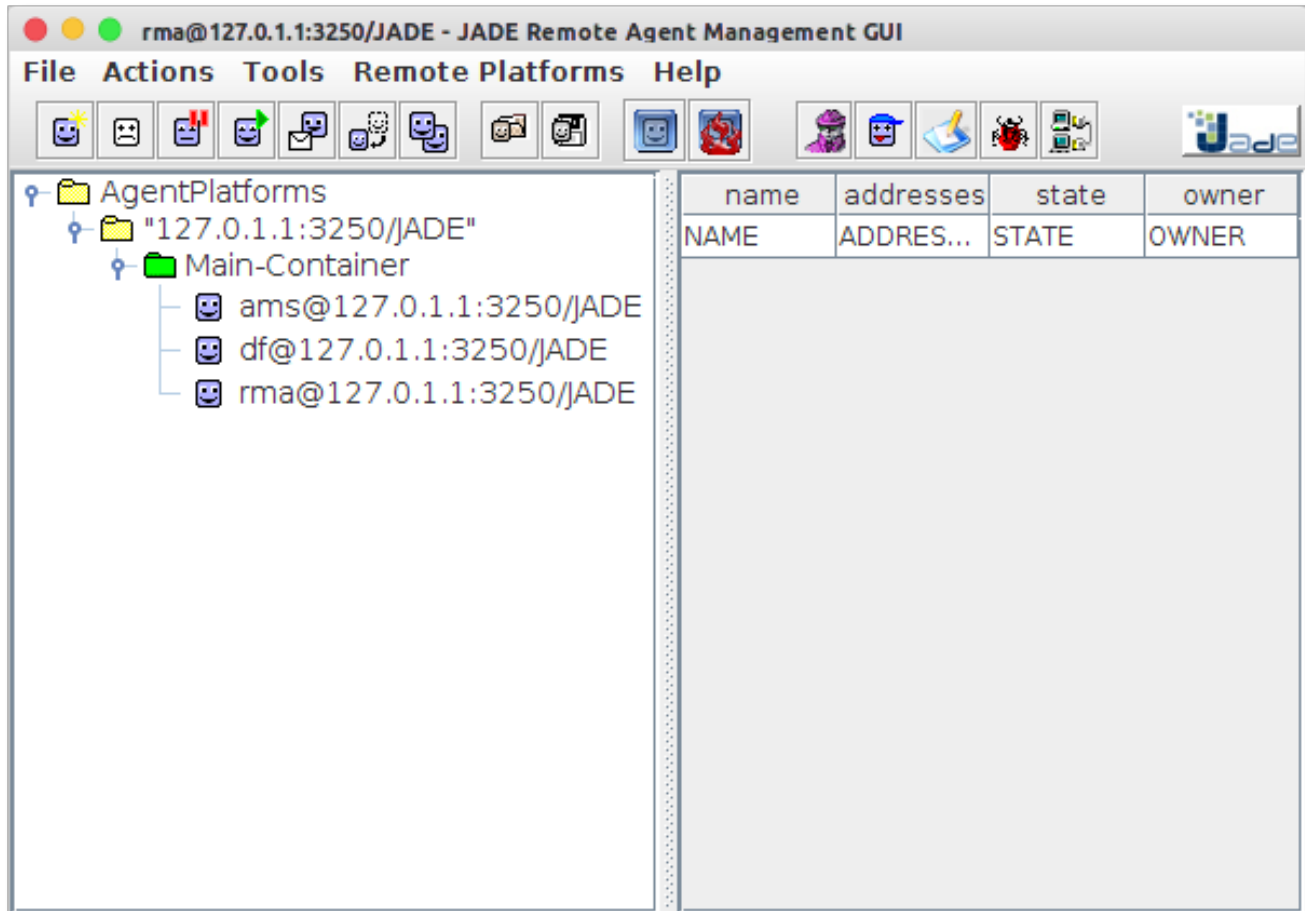


Figure 5. The Jade GUI booted from Eclipse

Following the management console our custom created platform manager will also be started as shown in figure 6. This gui management window will be used to create and start load and generator agents. Below is the console log pertinent to the platform manager.

```
16:32:09,478 INFO ~ Creating Platfrom Manager UI Starter thread...
May 01, 2016 4:32:09 PM jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Main-Container@127.0.1.1 is ready.
-----
16:32:09,530 INFO ~ Starting Platform Manager UI...
```

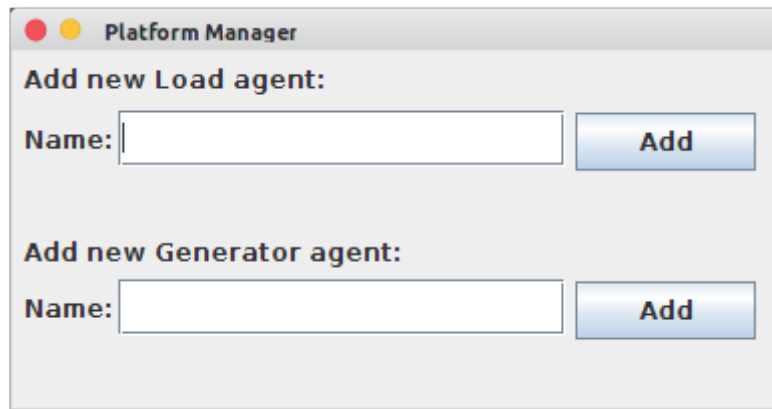


Figure 6. The Platform Manager created using Java swing

As seen in the UML class diagram below, the platform manager is used to generator the Searchview and Providerview that are the GUI consoles for the load agent and the generator agent respectively.

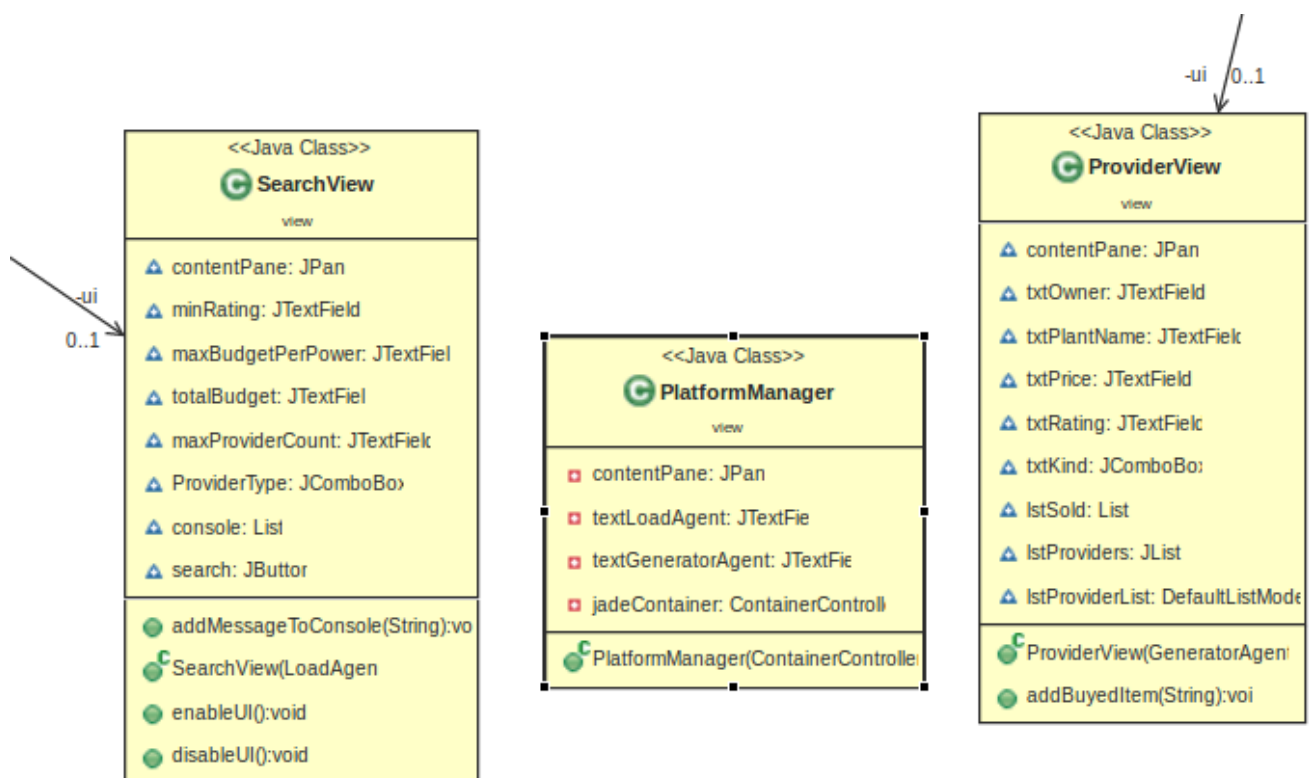


Figure 7. The UML class diagram of platform manager, Load agent and generator agent GUIs

As soon as we provide a new for a new load agent to be created and click 'Add', the following messages are passed on the console and we create a load agent management console as shown in figure 8.

```
16:36:20,449 INFO ~ Agent LA1 is being created...  
16:36:20,456 INFO ~ Agent: LA1 - Creating Load Agent UI...
```

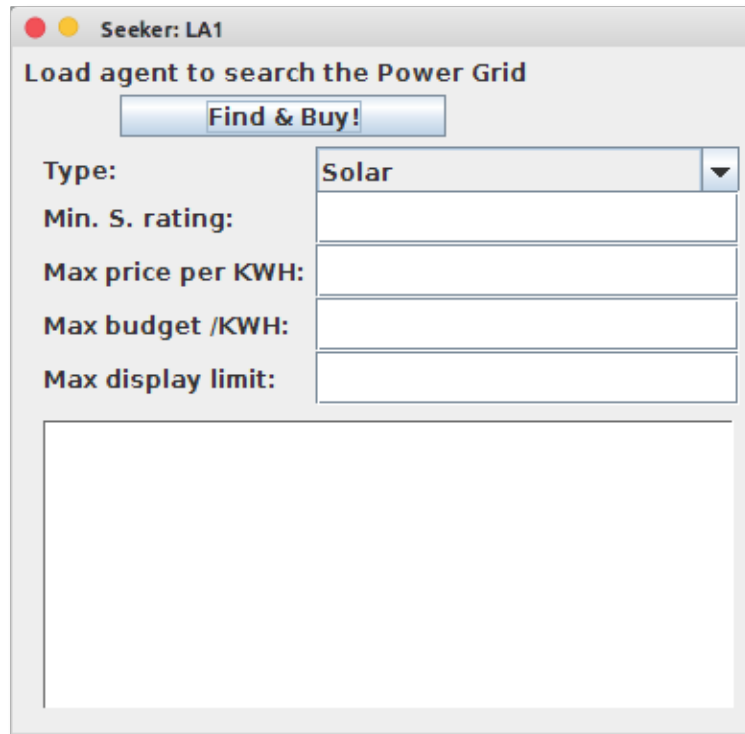


Figure 8. The Load agent management console

The GUI is created so that the load agents can specify their search criteria. Following is the description of each field in the load agent GUI.

1. Type
 - There are 5 types of power advertised around the power grid. It should be noted that a power grid is usually composed of more than one power type so that each power source can provide its unique capabilities
2. Min. S. rating
 - This is the minimum sustainability rating. In the power grid we are simulating, each power type has a rating based on the sustainability of the power type. Solar and wind power types have the highest sustainability ratings because they can provide cheaper and cleaner power without artificial sustenance. Whereas Nuclear based power types have the lowest ratings
3. Max price per KWH
 - This is a search criteria the user wants to impose. Some power types are cheaper per

KWH while solar and wind can be priced higher.

4. Max budget

- Based on the budget, the system can buy power from more than one power providers. This option was added if the user wants to buy power from more than one vendor

5. Max display limit

- This variable will set the limit on the number of power providers displayed in the results window

6. Find & Search: This button will start searching for agents under the mentioned criteria and buy the cheapest power source with the highest rating.

Below we see the UML class diagram for the load agents and the functions as shown in figure 9.

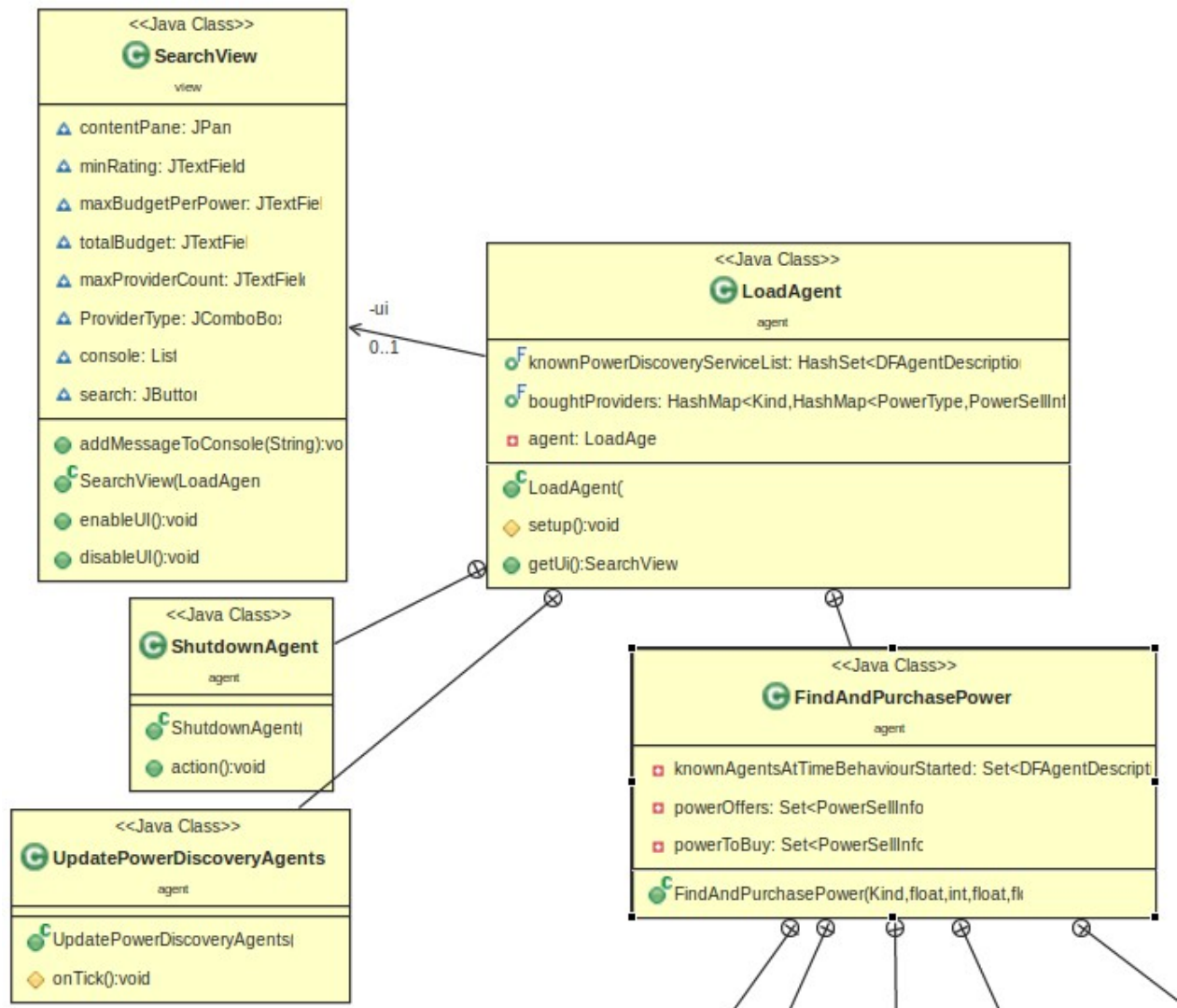


Figure 9. The UML Class diagram for Load Agent

We then use the Platform Manager to create two generator agents and name the GA1 and GA2 respectively. In our simulation these generator agents will be used to advertise several power sources to the load agent LA1. Following is the console log for created the two generator agents.

```
16:37:04,729 INFO ~ Agent GA1 is being created...
16:37:04,746 INFO ~ Agent: GA1 - Registering to DF agent...
16:37:04,752 INFO ~ Agent: GA1 - Creating Generator Agent UI...
16:37:04,755 INFO ~ Agent: GA1 - Waiting for 'Buy Power' and 'Query Power' requests...
16:39:53,447 INFO ~ Agent GA2 is being created...
16:39:53,454 INFO ~ Agent: GA2 - Registering to DF agent...
16:39:53,454 INFO ~ Agent: GA2 - Waiting for 'Buy Power' and 'Query Power' requests...
16:39:53,455 INFO ~ Agent: GA2 - Creating Generator Agent UI...
```

Figure 10 shows the GUI created for the generator agents. We can create multiple power sources under the control of each generator i.e. each generator agent can advertise multiple power sources for selling. The description of each field is given as following.

1. Type: This should be one of the 5 types of power that we are advertising
2. Owner: Name of the company for the power plant
3. Plant #: Location or code name of the power source
4. Price KWH: this is the per KWH price of the power source
5. S. rating: The sustainability rating as discussed previously in the load agent section
6. Add button: will add the filled details into a new power source that will be available for selling
7. Delete button: can be used to remove power sources from the console

The screenshot shows a GUI window titled "Provider: GA1". It is divided into two main sections. The left section, titled "Local Power Grid:", contains a large empty rectangular area and a "Delete Selected" button at the bottom. The right section, titled "Add new:", contains several input fields: "Type" (a dropdown menu currently showing "Solar"), "Owner", "Plant #", "Price KWH", and "S. Rating". Below these fields is an "Add" button. At the bottom of the right section, there is a "Sold:" label followed by an empty rectangular area for displaying sold power sources.

Figure 10. GUI console developed for the generator agents.

Figure 10 displays the UML class diagram of the generator agent and its functions.

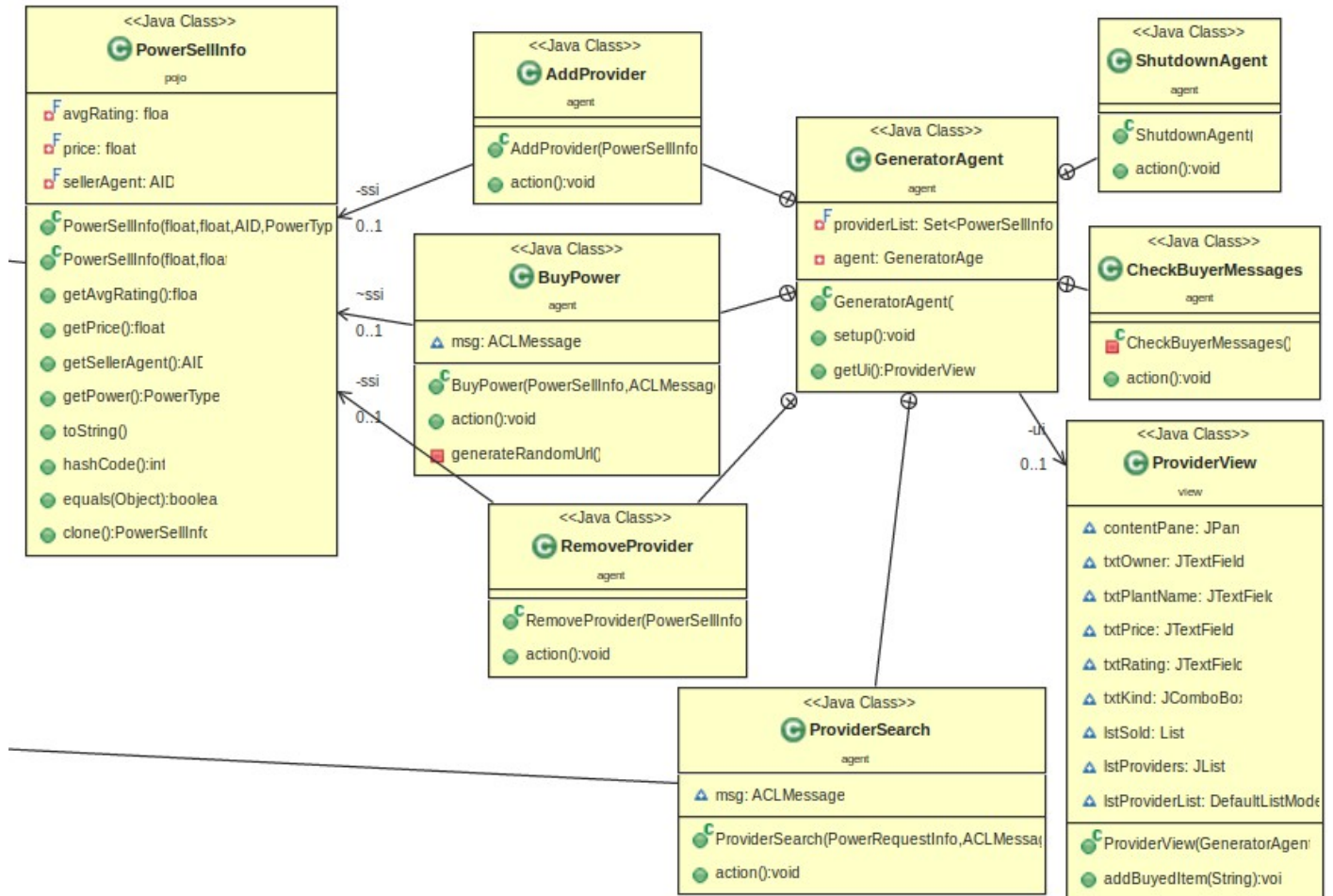


Figure 11. UML class diagram for the generator agent and its functions

With all the agents being created, we can see that reflected in the Jade management GUI console in figure 12. We can see in the JADE gui that all of our agents are initialized. We will use the default DF agent provided by the JADE engine. After we verify that the agents are in healthy status, we fill several power sources in the two generator agents and prepare them for selling the different power types. The two generator agents should look something like in figure 13.

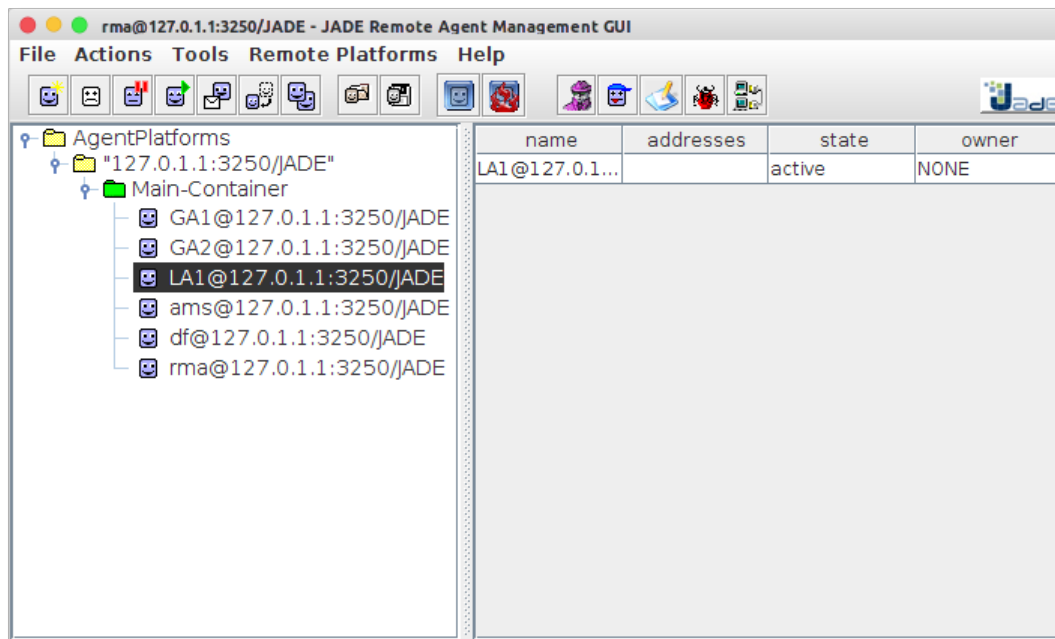


Figure 12. The updated Jade GUI management console



Figure 13. The two generator agents after adding power sources to them

These power sources will be advertised to the directory facilitator agent (DF) as sources from GA1 and GA2. The load agent LA1 can then communicate with the DF agent to get the source information. Following is the console log generated after we mentioned the purchase criteria in LA1 and then click 'Find and buy'.

```
16:45:53,156 INFO ~ Agent: LA1 - Sending Power Provider search request to Generator Agents...
16:45:53,164 INFO ~ Agent: GA1 - Waiting for 'Buy Power' and 'Query Power' requests...
16:45:53,166 INFO ~ Agent: GA2 - Waiting for 'Buy Power' and 'Query Power' requests...
16:45:53,167 INFO ~ Agent: LA1 - Recieving Power search results... (0 / 2)
16:45:53,179 INFO ~ Agent: GA1 - Waiting for 'Buy Power' and 'Query Power' requests...
16:45:53,179 INFO ~ Agent: GA2 - Waiting for 'Buy Power' and 'Query Power' requests...
```

The 'Find & buy' button corresponds to the FindAndPurchasePower function in the code. The FindAndPurchasePower function then starts the selection and purchasing process. The UML class diagram of 'find & buy' is shown below in figure 14.

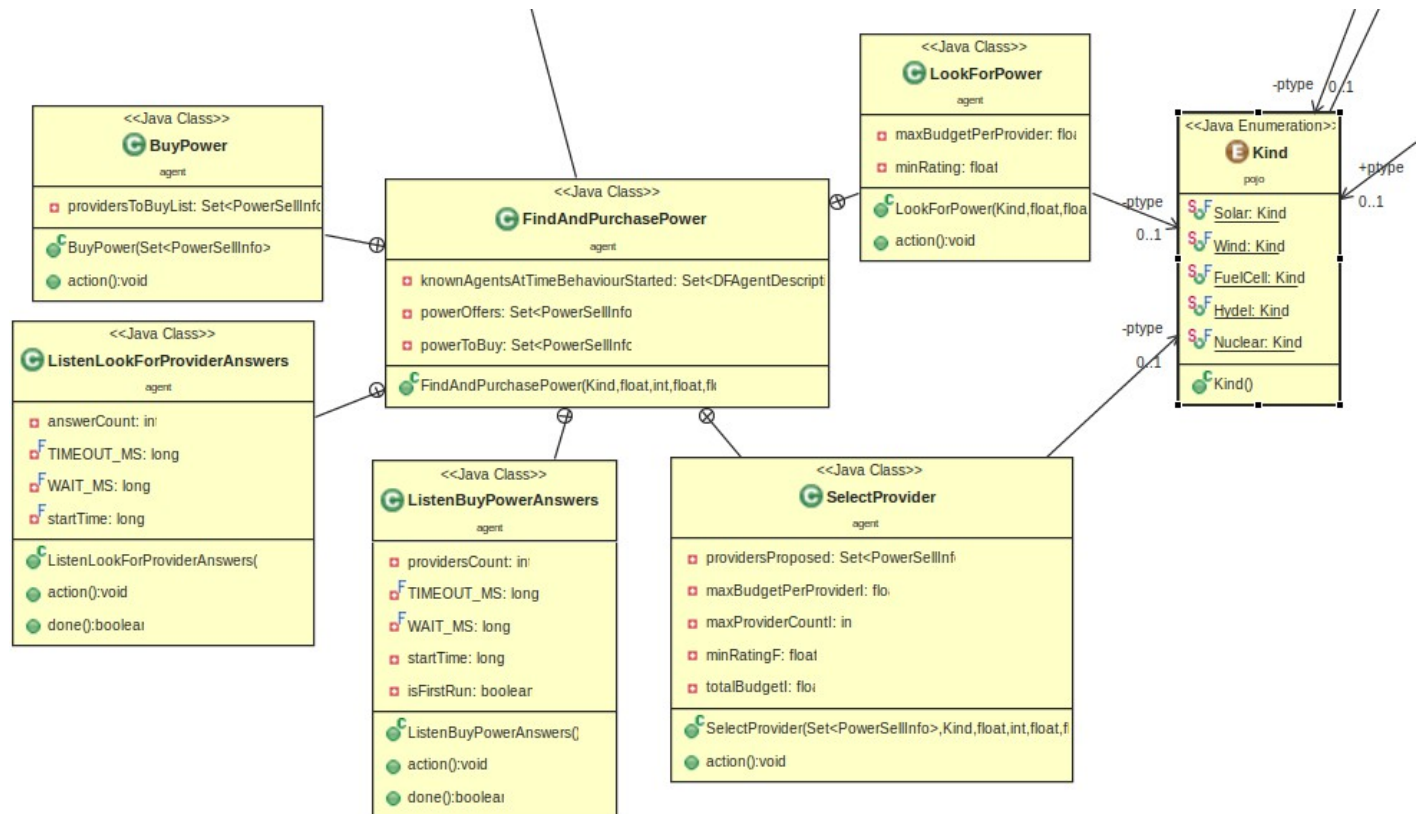


Figure 14. UML Class diagram of the Find & Buy method of Load agent

As the load agent LA1 request each GA if they have power sources that comply with its criteria, each generator agent checks if it has the power type meeting the criteria. Each GA then responds back to the LA with the compatible power sources and waits to hear back from the LA. The LA then decides which power type has the lowest price with the highest sustainability rating and then contacts the specific GA only it wants to buy power from. Please see below the agent communication of the criteria evaluation and selection process.

```

16:45:53,180 INFO ~ Agent: GA1 - Searching Power Providers with given criteria...
16:45:53,180 INFO ~ Agent: GA2 - Searching Power Providers with given criteria...
16:45:53,209 INFO ~ Agent: LA1 - Receiving Power search results... (0 / 2)
16:45:53,211 INFO ~ Agent: LA1 - Receiving Power search results... (1 / 2)
16:45:53,213 INFO ~ Agent: LA1 - Ordering Power purchase...
16:45:53,214 INFO ~ Agent: LA1 - Waiting for Power purchase results... (0 / 1)
16:45:53,214 INFO ~ Agent: GA1 - Waiting for 'Buy Power' and 'Query Power' requests...
16:45:53,215 INFO ~ Agent: GA1 - Waiting for 'Buy Power' and 'Query Power' requests...
16:45:53,217 INFO ~ Agent: GA1 - Power sold. Seller agent: LA1@127.0.1.1:3250/JADE Sale
Info: FPL - Miami
16:45:53,217 INFO ~ Agent: LA1 - Waiting for Power purchase results... (0 / 1)
16:45:53,220 INFO ~ Agent: LA1 - Saad Sadiq: FPL - Miami [Solar] [R:5.0] [P:10.0] @GA1
[URL: -----12]

```

Following is the UML diagram that relates to the evaluation and decision process.

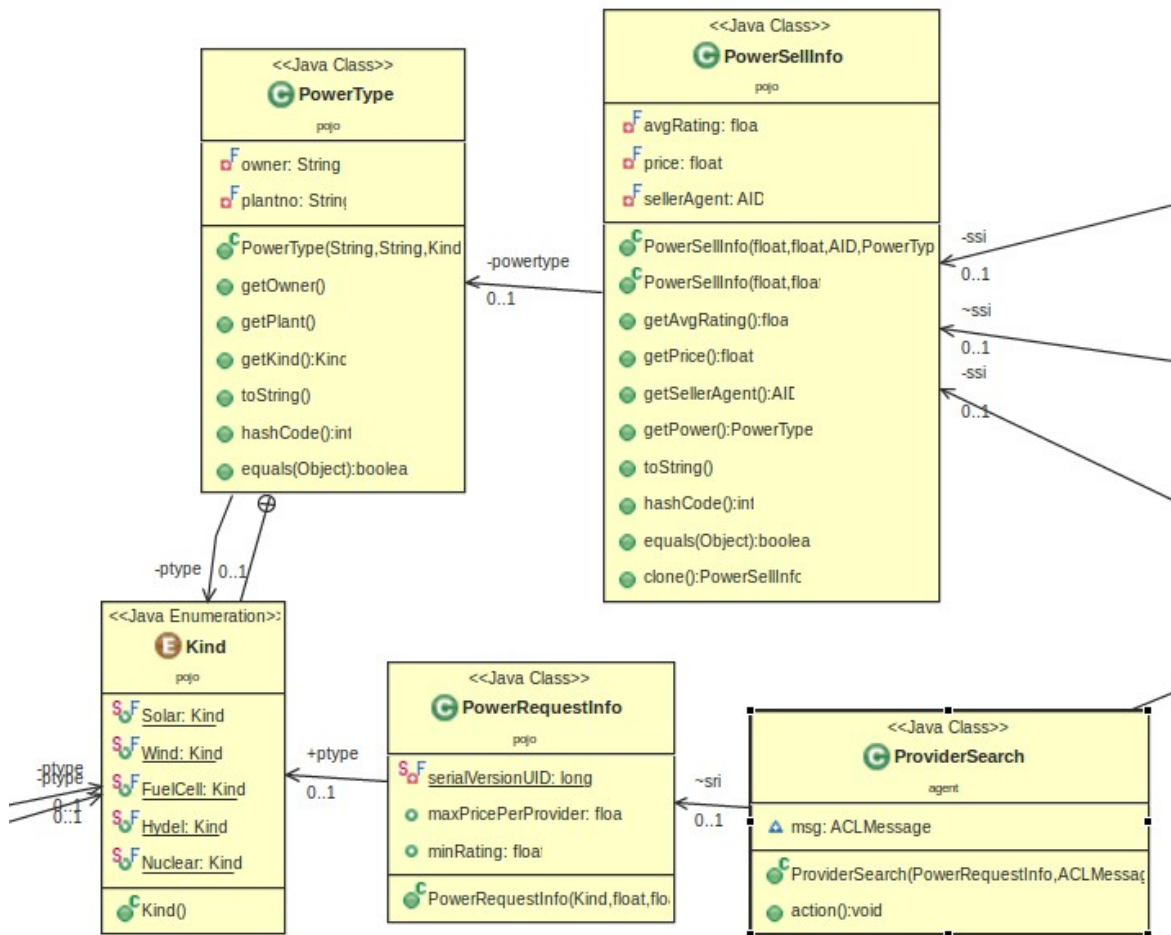


Figure 15. UML class diagram of the methods related to evaluating criteria and decision making

The load and generator agents both keep a log of the sold provider name and details in the console gui for reference. The results are displayed in the result boxes of both LA and GA as shown in figure 16. We can observe from the results that out of several power sources, our load agent autonomously purchased the lowest priced power source having the highest sustainability ratings.

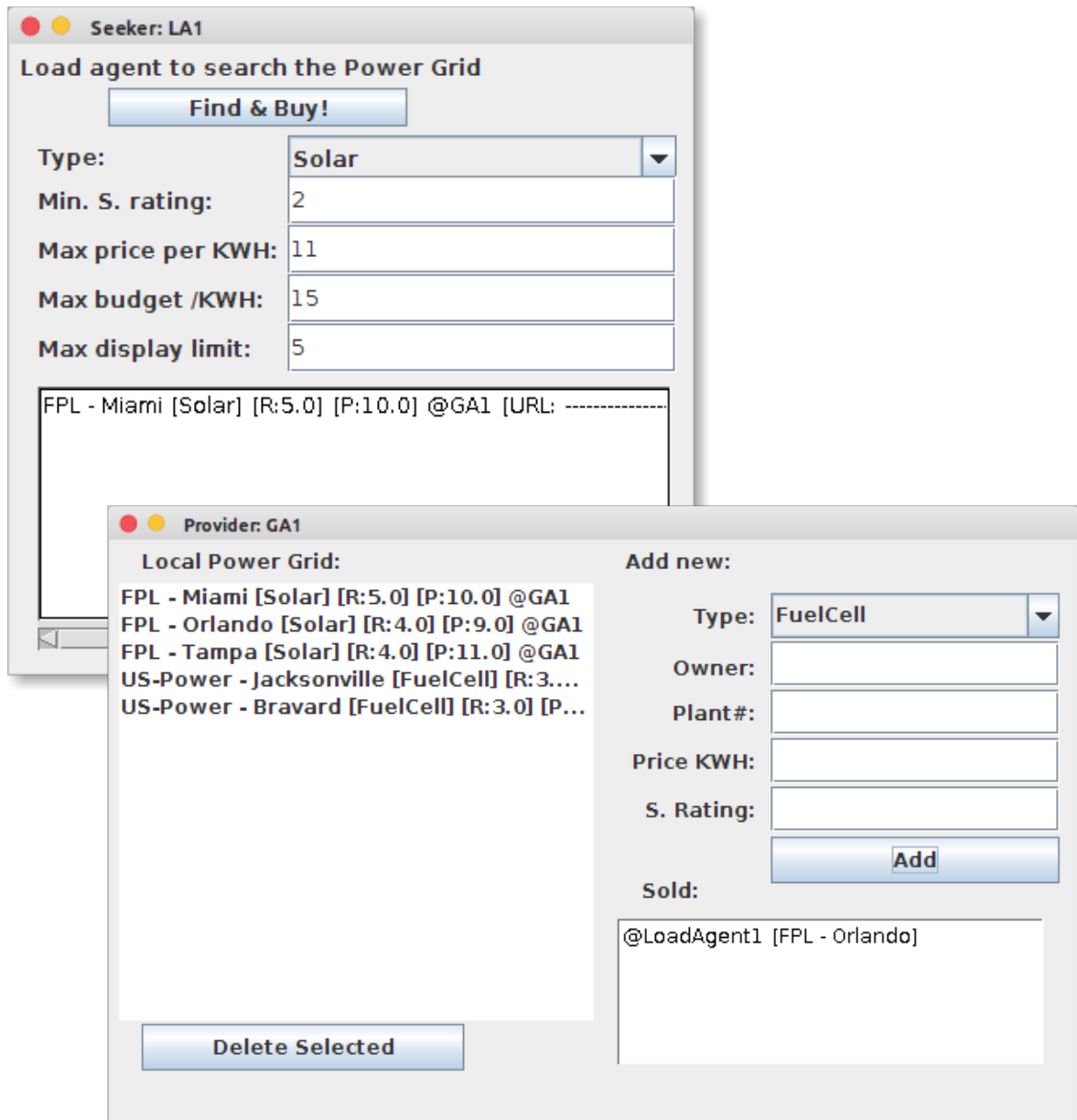


Figure 16. Details of the sold and purchased power displayed in the GUI of GA and LA respectively

V. Conclusion

Controlling distributed power systems using autonomous multiagents has been the focus of investigation of many researchers. This cutting edge and highly impactful field has overcome many limitations of the conventional power grids. In recent years, MAS have been employed in a wide range of power system applications including modeling of electricity markets, grid protection, fault restoration and grid control. We have aimed to work in the area of distributed multiagents and gain insights that will be helpful for our future research direction. We have selected modern distributed power grids as the subject of our implementation because of the above mentioned benefits and advantages of this field. This project aims to investigate the abilities of JADE framework and to that will help explore future research venues. Key areas of the project was agent communication, agent contract nets, decision making, and planning.

In this distributed grid two parties were created, agents who look for energy known as 'Load Agents' and agents who sell the energy to the distributed grid known as the 'Generator Agents'. There were 2 GA and one LA in our simulated project. We used 5 different types of power, each one bearing different cost and sustainability ratings, that the Generator Agents will sell to the market. These were

1. Solar
2. Hydel
3. Wind
4. Fuel cells
5. Nuclear

We were able to add new power sources to the Generator agents that were advertised to the directory facilitator (DF) agent. Load Agents (LA) always update their generator agent list by querying the Directory Facilitator (DF) agent. Load Agents send query to Generator Agents, collect the results and analyze them to decide the best choice of power to buy. Then the buying process is initiated and finally power with minimum cost and highest sustainability power rating is bought from an agent. From the successful implementation and simulation, we observed the results that out of several power sources, our load agent autonomously purchased the lowest priced power source having the highest sustainability ratings.

Future aspects of this project include simulating distributed multiagents for the Hadoop big data environment. Big data is the perfect candidate to streamline its processes using distributed multiagents. Very few researchers have touched this critical area of research. We intend to investigate possible solutions that multiagents can bring to the world of big data.