# JADE TUTORIAL

# Simple Example for using the JadeGateway class

**last update: 25 October 2006.**
**author: Viktor Kelemen ( MTA - SZTAKI )**

## Table of Contents

## Introduction

In some cases we'd like to create a web interface based on JSP and servlets for our JADE multi-agent system.
JADE offers some utility classes these could help us.

I assume you are familiar with Java/JSP and the Jade API.

This document describes a short example which I've made during my work to show how agents and servlets can communicate.
*Ive used the Jade-Leap libraries and its optimized for mobile devices.*
*But if you used the simple Jade libraries it'd be ok and changes in the source are not needed.*

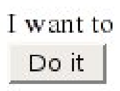
The silverbullet may be the **`jade.wrapper.gateway`** package in the communication.
It contains these classes:

- **`JadeGateway`**
- **`GatewayAgent`**

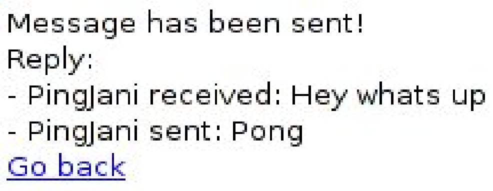
**JadeGateway** class is a singleton so it has static methods only.
**GatewayAgent** class will be extended by one of our agent.

## What will it do?

This little web application shows a button (below) and if you click on it a message will be sent to an agent created previously and this agents will send a reply.



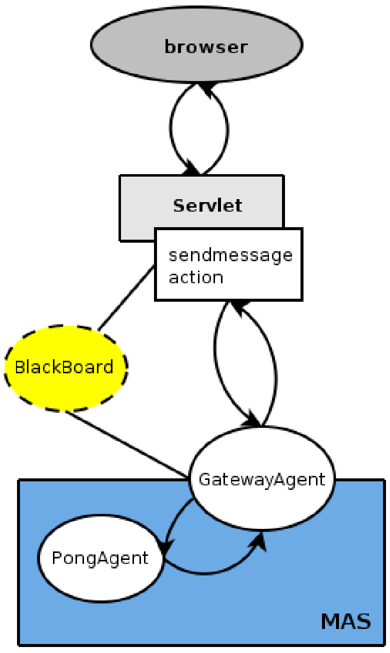
This repy will appear on the webpage.


Message has been sent!
Reply:
- PingJani received: Hey whats up
- PingJani sent: Pong
Go back

## System structure

**Browser** runs on the client side.
**Servlet** runs in our application server.

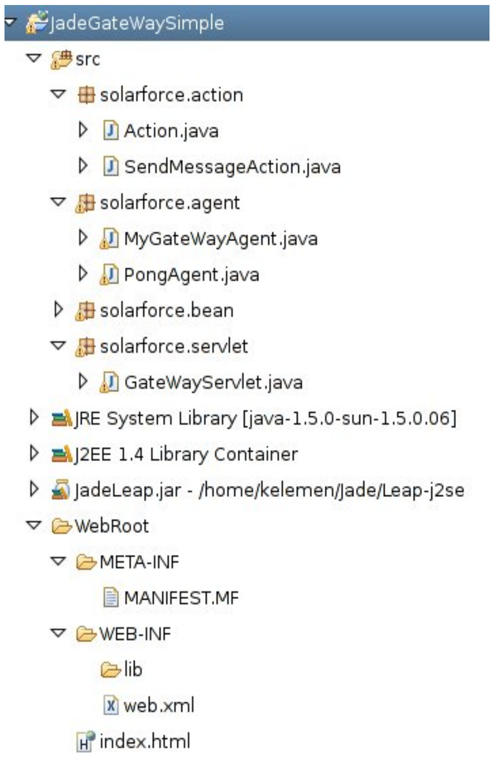
**PongAgent** already exists in a MAS.
**BlackBoard** is an object created by the servlet and used as a communication channel.
**GatewayAgent** is created by the servlet too and it behaves such as a dispatcher



**Timeline:**

1. In the browser the user causes an event and it generates a POST message.
2. The servlet handles it and the sendmessage action is invoked.
3. The action creates a new **BlackBoard** object which will be the message channel between the **GatewayAgent** and the **servlet**.
4. The **GatewayAgent** gets the dashboard object created previously and extracts who is the recipient and what's the message.
   After that it sends the message.
5. **PongAgen**t who is now the recipient and responses to the **GatewayAgent**.
6. The **GatewayAgent** packes the reply and sends it via **BlackBoard** to the **servlet**.
7. The **servlet** forwards it to the browser.

My project structure is look like this:



```
JadeGateWaySimple
  src
    solarforce.action
      Action.java
      SendMessageAction.java
    solarforce.agent
      MyGateWayAgent.java
      PongAgent.java
    solarforce.bean
    solarforce.servlet
      GateWayServlet.java
  JRE System Library [java-1.5.0-sun-1.5.0.06]
  J2EE 1.4 Library Container
  JadeLeap.jar - /home/kelemen/Jade/Leap-j2se
  WebRoot
    META-INF
      MANIFEST.MF
    WEB-INF
      lib
      web.xml
    index.html
```

***solarforce.action***
servlet uses these actions

**solarforce.agent**
our well-known PongAgent and MyGateWayAgent

**solarforce.bean**
BlackBoard object

**solarforce.servlet**
Our servlet who receives post messages and sends the proper response

## Details, Step by Step

### 1. starting

At first let's launch the Main-Container for our local MAS. In this case the main-container and the tomcat server run on the same host this makes our task easier.

So, start the main-container and a gui agent.

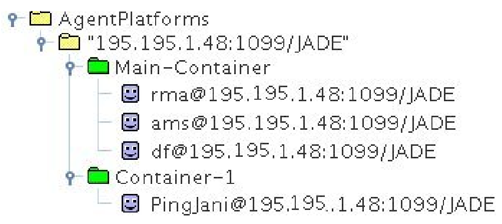
```
:> java -classpath .:/home/kelemen/Jade/Leap-j2se/JadeLeap.jar  jade.Boot -gui
```

After that let's start an **agent** called PingJani **who will receive the message** from the browser.

```
:> java -classpath .:/home/kelemen/Jade/Leap-j2se/JadeLeap.jar jade.Boot -container PingJani:solarforce.agent.PongAgent
```

*Note: the class of this agent is located in WebRoot/WEB-INF/classes*

If everything is fine the next will appears in the Jade GUI:



```
AgentPlatforms
  "195.195.1.48:1099/JADE"
    Main-Container
      rma@195.195.1.48:1099/JADE
      ams@195.195.1.48:1099/JADE
      df@195.195.1.48:1099/JADE
    Container-1
      PingJani@195.195..1.48:1099/JADE
```

### 2. html page

After that we **should write a html file** contained the form where the “Do it” button appears.
This is simple we just need some html tags.

**index.html**

```
<body id="page">
  I want to
  <form id="operationBox" method="get" action="GateWayServlet">
      <input type="hidden" name="action" value="sendmessage"/>
             <input type="submit" value="Do it"></input>
  </form>
</body>
```

This contains a hidden field which will indicate the type of the action sent to the servlet.

### 3. servlet

So, the html page sends proper post message to the servlet.
Let's have a look.

**GateWayServlet.java**

In the `init` function we set the proper class which will be derived as a GatewayAgent.
In this case the second parameter is `null` because the main container of the MAS is running on the same host and on the default 1099 port.

```
              // This sets that which class will be the GateWayAgent
              JadeGateway.init("solarforce.agent.MyGateWayAgent",null);
```

In the *doPost* function we catch the post message sent by our html page. It's quite generic.

```
        String actionName = request.getParameter("action");

        if (actionName == null)      {
                response.sendError(HttpServletResponse.SC_NOT_ACCEPTABLE);
                return;
        }

        Action action = (Action) actions.get(actionName);
        if (action == null)    {
                response.sendError(HttpServletResponse.SC_NOT_IMPLEMENTED);
                return;
        }
        action.perform(this, request, response);
```

in this case the **“action”** is the **“sendmessage”**

so the **SendMessageAction** will be performed

In the **SendMessageAction** (below) we create a **BlackBoard** object and set up its content (receiver, message).
Then we call the `JadeGateway.execute` with a param which is the BlackBoard object created previously.

`JadeGateway.execute` sends this object to the gatewayagent tuned earlier who will forward it or fill it with the proper content.

**So in the SendMessageAction:**

1. creating a BlackBoard object
2. filling the BlackBoard object with the proper content
3. sending the object to the GatewayAgent with the `JadeGateway.execute`

After that the servlet will be waiting until the gatewayagent says that: Ok, ive finished, here is your object.
Yes, this is a synch thing.

**SendMessageAction.java**

```
        // create a BlackBoard for the session if it not exist
        BlackBoardBean board = new BlackBoardBean();

        board.setReceiver("PingJani");
        board.setMessage("Hey whats up");

        try     {
                JadeGateway.execute(board);
        } catch(Exception e) { e.printStackTrace(); }
```

## 4. entering to the MAS

Now our gatewayagent gets this blackboard object.
The `processCommand` will be invoked when the servlet does an execute.

```
    BlackBoardBean board = null;

    protected void processCommand(java.lang.Object obj) {

        if (obj instanceof BlackBoardBean)   {

            board = (BlackBoardBean)obj;

            ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
            msg.addReceiver(new AID( board.getReceiver(), AID.ISLOCALNAME) );
            msg.setContent(board.getMessage());
            send(msg);
        }
        }
```

In this case we make an ACL message and send it.

In the background we have an `CyclicBehaviour` who is waiting for the response. If we get it a `releaseCommand` will be invoked.

```
                addBehaviour(new CyclicBehaviour(this)
                {
                        public void action() {

                              ACLMessage msg = receive();

                              if ((msg!=null)&&(board!=null))      {
                                     board.setMessage(msg.getContent());
                                     releaseCommand(board);
                              } else block();
                        }
                });
```

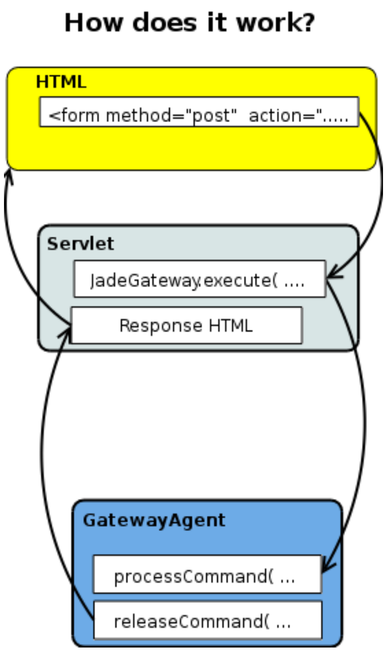
## 5. back to the servlet

After that we will return to the servlet and it'll be continued.
It makes a simple html file contained the proper response.

**SendMessageAction.java**

```
                try     {
                        JadeGateway.execute(board);
                } catch(Exception e) { e.printStackTrace(); }

                response.setContentType("text/html");

                PrintWriter out = response.getWriter();

                out.print("Message has been sent!<br/>");
                out.print("Reply:"+board.getMessage());
                out.print("<br/><a href='index.jsp'> Go back </a>");

                out.flush();
                out.close();
```

## 6. a sort overall



## Conclusion

First of all the best thing in this conception (servlet – gatewayagent – MAS) is that it works correctly and absolutely scalable because the agent world is totally separated.

In some cases it is possible that the gateway agent will be the bottle neck but if we use a pool for the gateway agents this would be not a serious problem.

***Note:***
*1. The complete source code is included and i hope it makes things more clearly.*
*I've used the Eclipse IDE so the source is in its format.*

Viktor Kelemen
kelemen.viktor@sztaki.hu