Java Agent Development Framework (JADE)

Laboratory of Multiagent Systems LM
Laboratorio di Sistemi Multiagente LM

Elena Nardini
elena.nardini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2010/2011
1. JADE Platform

2. Programming with JADE – Basic Features

3. JADE Download

4. Exercises
   - Exercise 1
   - Exercise 2
JADE and the Agent Abstraction

JADE Main Features

- Software platform providing basic middleware-layer functionalities for the realisation of distributed application by exploiting the agent abstraction
- Full compliant with FIPA specifications
- Provides graphical tools to support programmers when debugging and monitoring

JADE Agents

- Identified by a globally unique name: AgentIdentifier or AID
- Can join and leave a host platform at any time and can discover other agents through both white-page and yellow-page services
- Can initiate communication with any other agent at any time and can equally be the object of an incoming communication at any time
- Can be mobile
JADE Architecture

[Diagram showing the architecture of the JADE Platform]

- **JADE Platform**
- **JADE Architecture**
- **Elena Nardini (Università di Bologna)**
- **A.Y. 2010/2011 4 / 30**
JADE Platform

JADE Architecture

Agents and Containers

- A JADE platform is composed of *agent containers* that can be distributed over the network
- Agents live in containers
- A container is a Java process providing the JADE run-time and all the service needed for hosting and executing agents

Main Container

- Is a special container representing the bootstrap point of the platform
- All the containers must join to a main container by registering with it
- By default the *Main Container* contains two agents:
  - *Agent Management System* (AMS) that supervises the entire platform
  - *Directory Facilitator* (DF) that implements the yellow pages service
Admin & Debugging Tools

Complexity in Multi-agent Applications

- Often distributed across several hosts
- Composed of perhaps hundreds of multi-threaded processes
- They are dynamic in that agents can appear, disappear and migrate

→ Difficulties in management and especially debugging

→ JADE has an event notification service which forms the basis of
  - The *JADE RMA* management console
  - A set of graphical tools

→ They are provided to help in the management and debugging phase
Admin & Debugging Tools

JADE RMA (Remote Monitoring Agent)

- Implements a graphical platform management console
- Provides a visual interface to monitor and administer a distributed JADE platform composed of one or several hosts and container nodes
- It includes a “Tools” menu through which other tools can be launched
Admin & Debugging Tools

**Dummy Agent**

- A simple tool that is useful for sending stimuli, in the form of custom ACL messages, to test the behaviour of another agent
Sniffer Agent

- A tool used for debugging or, simply documenting conversations between agents
While the Sniffer Agent is a tool useful to sniff, monitor and debug conversation between agents, the Introspector Agent should be used to debug the behaviour of a single agent.

In fact, it allows an agent’s life cycle, and its queues of sent and received messages, to be monitored and controlled.
Admin & Debugging Tools

Log Manager Agent

- Is a tool that simplifies the dynamic and distributed management of the logging facility by providing a graphical interface that allows the logging levels of each component of the JADE platform to be changed at run-time.
Agent Cycle

1. setup()
2. Agent has been killed (doDelete() method called)?
   - YES: takeDown()
   - NO: Get the next behaviour (b) from the pool of active behaviours
     - b.action()
     - NO: b.done()? (Loop back)
     - YES: Remove b from the pool of active behaviour
import jade.core.Agent;
import java.util.Iterator;

@SuppressWarnings("serial")
public class HelloWorldAgent extends Agent {
    @SuppressWarnings("unchecked")
    protected void setup() {
        System.out.println("Hello World! I'm an agent");
        System.out.println("My local name is " + getAID().getLocalName());
        System.out.println("My GUID is " + getAID().getName());
        System.out.println("My addresses are: ");

        Iterator it = getAID().getAllAddresses();

        while (it.hasNext()) {
            System.out.println("- " + it.next());
        }

        System.out.println("My addresses are: ");
        Object[] args = getArguments();

        if (args != null) {
            for (int i = 0; i < args.length; i++) {
                System.out.println("- " + args[i]);
            }
        }
        doDelete();
    }
}
Agent Execution

- Compile the Agent `HelloWorldAgent` with the command
  `javac -cp jade.jar *.java`

- Run the `Main Container` with the command
  `java -cp .:jade.jar jade.Boot -gui`
Agent Execution

- Run the Agent with the command
  
  ```java
  ```
  
  ![Output of running agent with command 1]

- Run the Agent with the command
  
  ```java
  java -cp .:jade.jar jade.Boot -container Pet:HelloWorldAgent(arg1,arg2)
  ```
  
  ![Output of running agent with command 2]
Agent Behaviour

- Three primary behaviour types are available with JADE
  - **One-shot.** Designed to complete in one execution phase
  - **Cyclic-shot.** Designed to never complete
  - **Generic-shot.** Embed a status trigger and execute different operations depending on the status value
Example of Behaviour Implementation (From BOOK-TRADING Project)

```java
private class OfferRequestsServer extends CyclicBehaviour {
    public void action() {
        MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
        ACLMessage msg = myAgent.receive(mt);
        if (msg != null) {
            // CFP Message received. Process it
            String title = msg.getContent();
            ACLMessage reply = msg.createReply();

            Integer price = (Integer) catalogue.get(title);
            if (price != null) {
                // The requested book is available for sale. Reply with the
                // price
                reply.setPerformative(ACLMessage.PROPOSE);
                reply.setContent(String.valueOf(price.intValue()));
            } else {
                // The requested book is NOT available for sale.
                reply.setPerformative(ACLMessage.REJECT);
                reply.setContent("not-available");
            }
        }
        myAgent.send(reply);
    }
    block();
}
```
Example of Behaviour Definition (From BOOK-TRADING Project)

```java
// Add the behaviour serving queries from buyer agents
addBehaviour(new OfferRequestsServer());

// Add the behaviour serving purchase orders from buyer agents
addBehaviour(new PurchaseOrdersServer());
```
Agent Communication

- Agent communication is implemented in accordance with the FIPA specifications.
- The communication paradigm is based on asynchronous message passing:
  - A mailbox associated to each agent
  - An agent is notified whenever a message is posted in the mailbox

Messages

- Compliant with FIPA-ACL message structure
  - The sender of the message
  - The list of receiver
  - The communication act (or performative) indicating what the sender intends to achieve by sending the message
  - The content containing the actual information to be exchanged
  - The content language indicating the syntax used to express the content
  - The ontology indicating the semantic used to interpreter the content
  - Some additional fields...
Example of DF Usage: Registration (From BOOK-TRADING Project)

```java
// Register the book-selling service in the yellow pages
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("book-selling");
sd.setName("JADE-book-trading");
dfd.addServices(sd);

try
{
    DFService.register(this, dfd);
}
catch (FIPAException fe)
{
    fe.printStackTrace();
}
```
Example of DF Usage: Searching (From BOOK-TRADING Project)

```java
dfaeAgentDescription template = new DfaeAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("book-selling");
template.addServices(sd);
try {
    DfaeAgentDescription[] result = DfaeService.search(myAgent, template);
    System.out.println("Found the following seller agents: ");
    sellerAgents = new AID[result.length];
    for (int i = 0; i < result.length; ++i)
    {
        sellerAgents[i] = result[i].getName();
        System.out.println(sellerAgents[i].getName());
    }
} catch (FIPAException fe)
{ 
    fe.printStackTrace();
}
```
Example of Message Sending (From BOOK-TRADING Project)

// Send the cfp to all sellers
ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
for (int i = 0; i < sellerAgents.length; ++i)
{
    cfp.addReceiver(sellerAgents[i]);
}

for (int i = 0; i < sellerAgents.length; ++i)
{
    cfp.addReceiver(sellerAgents[i]);
}
cfp.setContent(targetBookTitle);
cfp.setConversationId("book-trade");
cfp.setReplyWith("cfp" + System.currentTimeMillis());  // Unique value

myAgent.send(cfp);
Example of Message Reception and Blocking (From BOOK-TRADING Project)

```java
private class OfferRequestsServer extends CyclicBehaviour {
    public void action() {
        MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
        ACLMessage msg = myAgent.receive(mt);
        if (msg != null) {
            // CFP Message received. Process it
            String title = msg.getContent();
            ACLMessage reply = msg.createReply();
            Integer price = (Integer) catalogue.get(title);
            if (price != null) {
                // The requested book is available for sale. Reply with the price
                reply.setPerformative(ACLMessage.PROPOSE);
                reply.setContent(String.valueOf(price.intValue()));
            } else {
                // The requested book is NOT available for sale.
                reply.setPerformative(ACLMessage.REFUSE);
                reply.setContent("not-available");
            }
            myAgent.send(reply);
        } else {
            block();
        }
    }
} // End of inner class OfferRequestsServer
```
JADE for Usage

- Homepage: http://jade.tilab.com/
- Download JADE-all-4.0.1.zip containing
  - bin
  - doc
  - examples
  - src
- We need jade.jar in bin, the documentation in doc and examples
It is possible to use Jade as communication infrastructure for Jason
It is possible to integrate Jade agents with Jason agents
See
http://jason.sourceforge.net/mini-tutorial/jason-jade/
Outline

1. JADE Platform
2. Programming with JADE – Basic Features
3. JADE Download
4. Exercises
   - Exercise 1
   - Exercise 2
Exercise 1

Thermostat Agent with JADE

Requirements

- Check the environment temperature $T$.
- Until $T$ is not: $> 18$ and $< 22$:
  - Decrease $T$ of one unit if the temperature is 22
  - Increase $T$ of one unit if the temperature is 18

Constraint

- ThermostatAgent interacts with the environment to sense and change the temperature
- The environment can be simulated by an agent
Outline

1. JADE Platform
2. Programming with JADE – Basic Features
3. JADE Download
4. Exercises
   - Exercise 1
   - Exercise 2
There are three agents:

- **ThermostatAgent** interacts with the environment to sense and change the temperature
- **ManagerAgent** publish the service **JADE-temperature-checking** in the DF
- **ThermostatAgent** searches the service **JADE-temperature-checking** in the DF and obtain the Agent ID (AID) of **ManagerAgent**
- **ThermostatAgent** asks to **ManagerAgent** the new value of the temperature
Java Agent Development Framework (JADE)

Laboratory of Multiagent Systems LM
Laboratorio di Sistemi Multiagente LM

Elena Nardini
elena.nardini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2010/2011