Agent Programming with Jade

Weihong Yu
Outline

• Motivations for Agents
• What’s an Agent
• Multi-Agent System (MAS)
• Agent Programming
• JADE (Architecture, methods and examples, etc.)
Motivations for agents

• Large-scale, complex, distributed systems: understand, build, manage
• Open and heterogeneous systems - build components independently
• Distribution of resources
• Distribution of expertise
• Needs for personalization and customization
• Interoperability of pre-existing systems / integration of legacy systems
What’s an Agent

• The term *agent* is used frequently nowadays in:
  • Sociology, Biology, Cognitive Psychology, Social Psychology, and
  • Computer Science $\supset$ AI

• But there is **no universally accepted definition** of the term agent and there is a good deal of ongoing debate and controversy on this subject.
What’s an Agent

Many Definitions:

• "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.”

• "Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed."

• "Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires."
What’s an Agent---Internal and External Environment of an Agent

**Internal Environment:**
architecture, goals, abilities, sensors, effectors, profile, knowledge, beliefs, etc.

**External Environment:**
user, other humans, other agents, applications, information sources, their relationships, platforms, servers, networks, etc.

**Balance**
Multi-agent systems

A multi-agent system (MAS) is a system composed of multiple interacting **intelligent agents** within an environment.

Many entities (agents) in a common environment
MAS - many agents in the same environment

• Interactions among agents

• Interactions for
  - coordination
  - communication
  - organization

☐ Coordination

  ➔ collectively motivated / interested
  ➔ self interested

- own goals / indifferent
- own goals / competition / competing for the same resources
- own goals / competition / contradictory goals
- own goals / coalitions
Communication
  ➔ communication protocol
  ➔ communication language

Organizational structures
  ➔ centralized vs decentralized
  ➔ hierarchical/ markets
Agent Programming

• Platform
  – A place where agents live
  – Agent management
    • creation – termination
    • security
  – Agent communication services
  – Agent directory services
The **Foundation for Intelligent Physical Agents** (FIPA) is a body for developing and setting computer software standards for heterogeneous and interacting agents and agent-based systems. FIPA was founded as a Swiss not-for-profit organization in 1996 with the ambitious goal of defining a full set of standards for both implementing systems within which agents could execute (agent platforms) and specifying how agents themselves should communicate and interact.
JADE

JADE (Java Agent Development Framework)

- Framework aimed at developing multi-agent systems and applications conforming to FIPA standards for intelligent agents.
- Support to the execution of multiple, parallel and concurrent agent activities via the behaviour model.
- Full Java
- Runs on all JVM from J2EE to J2ME MIDP1.0
- Downloadable from http://jade.tilab.com
Containers and Platforms

- Each running instance of the JADE runtime environment is called a **Container** as it can contain several agents.
Containers and Platforms

- The set of active containers is called a **Platform**.
Containers and Platforms

– A single special *Main container must always be active* in a platform and all other containers register with it as soon as they start.
JADE

• Main container
• AMS (Agent Management System)
  – Provides the naming service and represents the authority in the platform.

• DF (Directory Facilitator)
  – Provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals.

• RMA (Remote Management Agent)
  – Acting as graphical console for platform management and control.
Agent Management System (AMS)
Agent Management System

• Provides the naming service
  – Ensures that each agent in the platform has a unique name

• Represents the authority in the platform
  – To create/kill agents on remote containers by requesting that to the AMS
Directory Facilitator
Directory Facilitator

• Provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals.
Remote Monitoring Agent

• Provide the GUI to control agents’ lifecycle
Agent’s Life Cycle

- Waiting
- Suspended
- Active
- Transit
- Initiated
- Wake Up
- Wait
- Resume
- Suspend
- Move
- Execute
- Invoke
- Destroy
- Quit
- Create
- Unknown
Message Transport System

- Agent Communication Channel (ACC)
  - Agent to Agent
  - Agent Platform to Agent Platform
JADE

• Agent identifier
  – <nickname>@<platform_name>
Install JDK

  - Download the latest version of J2SE Development Kit
    - The Java Runtime Environment (JRE)
    - Command-line development tools, such as compilers and debuggers, that are necessary or useful for developing applets and applications
Install JDK

Web Site
Install JADE

http://jade.tilab.com/

From this page you can now download JADE. We suggest to unzip the files by using the ‘jar xvf’ command rather than the winzip application because some incompatibilities have been reported in the past.

Current version of JADE is 4.2.0 (26/06/2012).

Note: All the binaries (lib/*.jar) have been generated by using the JDK1.5 compiler.

<table>
<thead>
<tr>
<th>JADE</th>
<th>~ File size</th>
<th>Description of the content</th>
</tr>
</thead>
<tbody>
<tr>
<td>jadeAll</td>
<td>16.1 MB</td>
<td>This file contains all JADE, i.e. it is just composed of the 4 files below. If it is too large for downloading, the 4 files below might be more convenient.</td>
</tr>
<tr>
<td>jadeBin</td>
<td>2.6 MB</td>
<td>This file contains JADE already compiled and ready to be used, i.e. a set of JAVA archive JAR files.</td>
</tr>
<tr>
<td>jadeDoc</td>
<td>11.2 MB</td>
<td>This file contains all the JADE documentation included the Administrator’s Guide and the Programmer’s Guide. NOTICE THAT all the documentation is also available on-line.</td>
</tr>
<tr>
<td>jadeSrc</td>
<td>2.2 MB</td>
<td>This file contains all the JADE source code.</td>
</tr>
<tr>
<td>jadeExamples</td>
<td>314 KB</td>
<td>This file contains the source code of the examples and a simple demo. All the examples and demo must be compiled.</td>
</tr>
</tbody>
</table>
JADE Package

- Extract JADE-ALL-4.2.0.ZIP
- Extract Sub Files

- JADE-doc
  - Document
- JADE-src
  - Source Code
- JADE-bin
  - Binary Code
- JADE-example
  - Example Code
Install eclipse or NetBeans

- http://netbeans.org/
Steps integrate JDEA with NetBeans ---

Step (1)
Step (2)
Step (3)
Step (4)
Step (5)

Choose Project:

Categories:
- Java
  - JavaFX
  - Java Web
  - Java EE
  - Java ME
  - Maven
  - PHP
  - Ruby
  - Groovy
  - C/C++
  - NetBeans Modules
  - Samples

Projects:
- Java Application
  - Java Desktop Application
  - Java Class Library
  - Java Project with Existing Sources
  - Java Free-Form Project

Description:

**Creates a new Java SE application** in a standard IDE project. You can also generate a main class in the project. Standard projects use an **IDE-generated Ant build script** to build, run, and debug your project.
Step (6)

New Java Application

Steps
1. Choose Project
2. Name and Location

Name and Location

Project Name: TestJade
Project Location: C:\Users\Administrator\Documents\NetBeansProjects\TestJade
Project Folder: Documents\NetBeansProjects\TestJade

Use Dedicated Folder for Storing Libraries
Libraries Folder: [field]

Create Main Class: testjade.Main
Set as Main Project:

Finish
Step (8)
Step (9)
JADE Platform Control/Status display
Implementation

1. Import \textit{jade.core.Agent} Library

2. \textbf{setup()} Initialize and register this agent to AMS, the current state is ACTIVE.

3. \textbf{addBehaviour()} Add behaviours to queue, the parameter is a behaviour class.

4. \textbf{action()} Define action of behaviour.

5. \textbf{Send or receive Message}
Implementation Example

• **Sensor Agent**
  – directly attached to an environmental device or sensor
  – get ambient light value every 5 seconds and
  – send it to the Decision Agent

• **Decision Agent**
  – receive the light value from the Sensor Agent
  – Make a decision(turn on/off) according to the light value and some rules
  – Send the decision to the Action Agent

• **Action Agent**
  – Receive the decision from the Decision Agent
  – Act(turn on/off light)
Sensor Agent

```java
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;

class Sensor extends Agent {
    protected void setup() {
        this.addBehaviour(new sensorbehaviour(this));
    }
}

class sensorbehaviour extends TickerBehaviour {
    Sensor xx;
    int light=(int)(Math.random()*1000+1);
    public sensorbehaviour(Sensor a) {
        super(a, 5000);
        xx=a;
    }
    protected void onTick() {
        ACLMessage msg=new ACLMessage(ACLMessage.INFORM);
        msg.addReceiver(new AID("decision",AID.ISLOCALNAME));
        msg.setContent(String.valueOf(light));
        xx.send(msg);
        light=(int)(Math.random()*1000+1);
    }
}
```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
public class Decision extends Agent{
    protected void setup() {
        this.addBehaviour(new decisionbehaviour(this));
    }
}
class decisionbehaviour extends CyclicBehaviour{
    Decision yy;
    String actioncommand="";
    int mylight;
    public decisionbehaviour(Decision a) {
        yy=a;
    }
    public void action() {
        ACLMessage mm=yy.receive();
        if(mm!=null){
            mylight=Integer.parseInt(mm.getContent());
        }
        if(mylight>300) {
            actioncommand="off";
        } else{
            actioncommand="on";
        }
        System.out.println("the current light is:"+mylight+";the light should be :"+actioncommand);
        ACLMessage mycommand=new ACLMessage(ACLMessage.INFORM);
        mycommand.addReceiver(new AID("actionagent",AID.ISLOCALNAME));
        mycommand.setContent(actioncommand);
        yy.send(mycommand);
        block();}
}
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import java.awt.Image;
import javax.swing.ImageIcon;
public class ActionAgent extends Agent {
    ConvertToBlack iii;
    @Override
    protected void setup() {
        iii=new ConvertToBlack(this);
        Behaviour loop=new CyclicBehaviour(){
            @Override
            protected void doOneCycle() {
                // Loop behavior logic
            }
        }
        addBehaviour(loop);
    }
}
public void action()
{
    ACLMessage msg = receive();
    String mystr="off";
    if(msg!=null) { mystr = msg.getContent(); }
    if(mystr.equals("off"))
    {
        ImageIcon imageIcon = new ImageIcon("c://x11.jpg");
        Image image = imageIcon.getImage();
        iii.changing(image);
        iii.setMytitle("OFF");
        iii.repaint();
    }
    if(mystr.equals("on"))
    {
        ImageIcon imageIcon = new ImageIcon("c://x11.jpg");
        Image image = imageIcon.getImage();
        iii.setImgOriginal(image);
        iii.setMytitle("ON");
        iii.repaint();
    }
    block();
}
this.addBehaviour(loop);