Multi-Agent Systems (MAS)

Adil Hussain --Presented by Xiuyi Fan (xf309)

Developing Multi-Agent Systems with JADE

- Section 7.2.3 of Wooldridge, Intro MAS, 2009
- http://jade.tilab.com/doc/
    - JADE Programmer's Guide
    - Descriptions of the Examples
- Developing Multi-agent Systems with JADE, Wiley Series in Agent Technology

Introduction

- JADE (Java Agent DEvelopment framework)
  - Agent-orientated middleware written completely in Java
- Agents
  - autonomous, social, reactive, proactive etc
- Agent Management
  - Agent, Directory Facilitator (DF), Agent Management System (AMS), Message Transport Service (MTS)

Demo

- Load up JADE administration GUI
- Launch agents
- The Dummy Agent
- The Sniffer Agent

Creating Agents

- Define a class that extends the **jade.core.Agent** class
- Implement the **setup()** method of the Agent class
  - Include agent initializations, e.g.
    - Show GUI
    - Open connection to database
    - Register services in yellow pages catalogue
    - Start the initial ‘behaviours’
  - Actual job of agent is carried out within ‘behaviours’

Agent Identifiers

- Each agent instance is identified by an 'agent identifier'
  - Agent identifier is instance of the jade.core.AID class
  - An AID object
    - includes a globally unique name (GUID) of the form <local-name>@<platform-name>
    - Provides methods to retrieve local name (getLocalName()) and GUID (getName())
- Local name of an agent is assigned at start-up time by the creator
  - must be unique within that platform

Compiling and Launching Agents

- Command for compiling agent
  - javac -classpath <JADE-classes> HelloWorldAgent.java
- Command for launching JADE and agents
  - java -classpath <JADE-classes> jade.Boot agent1:HelloWorldAgent agent2:HelloWorldAgent
- Agents can also be launched in other ways
  - e.g. by means of the administration GUI

[See compileJade.sh and runJade.sh files]

Agent Termination

- Implement the takedown() method of the Agent class
  - Include clean-up operations, e.g.
    - Close GUI
    - Close connection to database
    - Deregister services from yellow pages catalogue

Passing Arguments to an Agent

- Agents can take start-up arguments
- Arguments retrieved as an array of Object
  - by means of the getArguments() method of the Agent class
- Arguments specified when launching an agent, e.g.

```
java –classpath <JADE-classes> jade.Boot
   agent1:HelloWorldAgent(arg1 arg2 arg3)
```

  - only String arguments can be specified

Agent Tasks (1)

- The actual job, or jobs, an agent has to do is carried out within ‘behaviours’
- A behaviour represents a task that an agent can carry out
  - implemented as an object that extends **jade.core.behaviours.Behaviour**
  - can be added to the agent at any time
    - by means of the addBehaviour() method of the Agent class

Agent Tasks (2)

- Class extending Behaviour must implement two abstract methods
  - **action()**
    - the operations to be performed when the behaviour is in execution
  - **done()**
    - returns a boolean value to indicate whether a behaviour has completed
    - if so, is removed from the agent's pool of behaviours
- Scheduling of behaviours in an agent is not pre-emptive (as for Java threads), but cooperative

Primary Behaviour Types

- ‘One-Shot’
  - done() method returns true
- ‘Cyclic’
  - done() method returns false
- Generic
  - embed a status trigger and execute different operations depending on the status value
  - complete when a given condition is met

Generic Behaviour Example

```
public class TwoStepBehaviour extends Behaviour  {
    private int step = 0;
    public void action() {
        switch(step) {
            case 0: // perform operation X
            step++; break;
            case 1: // perform operation Y
            step++; break; } }
    public boolean done() { return step == 2; }
}
```

Agent Communication

- Fundamental feature of JADE
- Based on *asynchronous message passing*
  - Each agent has a ‘mailbox’ (the agent message queue) where messages sent by other agents are posted
  - Whenever a message is posted in the mailbox the receiving agent is notified
  - When, or if, the agent picks up the message from the queue for processing is a design choice

Message Structure

- Each message includes
  - the sender of the message
  - the list of receivers
  - the communicative act (‘performative’)
  - the content
  - additional fields
- Implemented as an object of the **jade.lang.acl.ACLMessage** class

Sending Messages

- Simply fill out the fields of an ACLMessage object and call the **send()** method of the Agent class, e.g.

```
ACLMessage msg = new ACLMessage();
msg.setPerformative(ACLMessage.INFORM);
msg.addReceiver(new AID(“Peter”, AID.ISLOCALNAME));
msg.setLanguage(“English”);
msg.setContent(“Today it’s raining”);
send(msg);
```

Receiving Messages

- Message picked up from message queue by means of the **receive()** method, e.g.

```
ACLMessage msg = receive();
if (msg != null) { // Process the message }
else { block(); }
```

- **createreply()** method creates a new ACLMessage
  – automatically sets the receivers and necessary fields for controlling the conversation
  – e.g. `ACLMessage reply = msg.createReply();`

Selecting Messages to Receive

- Done by specifying ‘templates’ as instances of **jade.acl.MessageTemplate** class
  - the **receive(mt)** method then returns the first message matching it (if any)
  - ignores all non-matching messages in the message queue

e.g.
```
private MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.CFP);
ACLMessage msg = myAgent.receive(mt); …
```

Agent Discovery

- Any agent can both register (public) services and search for (discover) services with a DF agent
- In order to publish a service...
  - agent must create a proper description (instance of **DFAgentDescription** class)
  - and call **register()** static method of **DFService** class
- In order to search for agents providing some service...
  - agent must create a template description
  - and call **search()** static method of **DFService** class