JADE: Java Agent DEvelopment Framework Advanced 2.0

Marco Fabbri  
(after Stefano Mariani)  
marco.fabbri28@unibo.it

Dipartimento di Informatica: Scienza e Ingegneria (DISI)  
ALMA MATER STUDIORUM—Università di Bologna

DS 2013/2014  
Distributed Systems – 27th of March 2014
OUTLINE

1. JADE Agents & Java Swing

2. JADE Agents Mobility
   • APIs
DISCLAIMER

All the material presented in these slides is rearranged by the author from a collection of documents kindly made available by the JADE team.

Then, credits for all the stuff (text & images) goes to the JADE team, in particular to Giovanni Caire.

Credits for all the mistakes goes to the author.
OUTLINE

1. JADE Agents & Java Swing

2. JADE Agents Mobility
   - APIs
Java Swing Troubles I

What’s the problem?

Whenever developing JADE agents which need to interact with a Java GUI, the thread-per-agent concurrency model of JADE agents must work together with the Swing Event Dispatcher Thread (EDT) concurrency model.
Java Swing Troubles II

More in detail

• as you should know, the Swing framework is not thread-safe, so any code that updates the GUI elements must be executed within the EDT

  • since modifying a model object triggers an update of the GUI, model objects too have to be manipulated just by the EDT

• the SwingUtilities class exposes two static methods to delegate execution of Runnable objects to the EDT

  • invokeLater() puts the Runnable into the System Event Queue (SEQ) (accessed by the EDT only) and returns immediately—asynchronous call

  • invokeAndWait() puts the Runnable into the SEQ and blocks waiting its completion—synchronous call
JADE Solution I

GuiAgent class

To develop JADE agents interacting with a GUI, simply extend GuiAgent class in package jade.gui

onGuiEvent(GuiEvent e)
may be viewed as the equivalent of the actionPerformed() method in Java Swing, that is, a callback invoked by JADE platform as soon as a GuiEvent is generated

postGuiEvent(GuiEvent e)
used by the agent’s GUI to queue GUI events for later processing —similar to queueing ACL messages in its mailbox
GuiEvent class

A GuiEvent object has:

- two mandatory attributes
  - `source` – the `Object` source of the event
  - `type` – an `integer` identifying the kind of event generated

- an optional list of parameters
  - `addParameter()` takes the `Object` to add as a `GuiEvent` parameter
  - `getParameter()` gets the i-th parameter
  - `getAllParameter()` returns an `Iterator` to browse all parameters
One final advice

From JADE Programmer’s Guide:

“In general, it is not a good thing that an external software component maintain a direct object reference to an agent, because this component could directly call any public method of the agent, skipping the asynchronous message passing layer and turning an autonomous agent into a server object, slave to its caller. The correct approach is that to gather all the external methods into an interface, implemented by the agent class, then an object reference of that interface will be passed to the external software component (e.g., a GUI) so that only the external methods will be available from event handlers.”

Check the ds.lab.jade.bookTrading.gui example carefully.
OUTLINE

1. JADE Agents & Java Swing

2. JADE Agents Mobility
   • APIs
What means “mobility” for JADE?

In JADE, *mobility* is the ability of an agent program to **migrate** or to **clone** (make a copy of) itself *across one or multiple network hosts*.

Which kind of “mobility”?  

As you may know, at least two different forms of mobility can be defined:

- **weak** only the program (agent) code is moved/cloned
- **strong** also the program (agent) state is moved/cloned along with its code—supposing to know what “state” means

JADE supports *some form* of strong mobility.
A JADE agent can:

- move/clone its state which means:
  1. stop its execution on the local container
  2. move/clone to a remote container resume its execution there from the exact point where it was interrupted
  3. resume its execution there from the exact point where it was interrupted

- move/clone its code, which means that if its code is not available on the destination container, then it is automagically retrieved by JADE platform

Keep in mind that...

! In order to be able to move, an agent must be **Serializable**.
OUTLINE

1. JADE Agents & Java Swing
2. JADE Agents Mobility
   • APIs
Location API

Where to move/clone to?

The jade.core.Location *interface* represents a place where agents can move / be clone to

- `getID()` to obtain the Location unique ID
- `getName()` to obtain the Location name
- `getAddress()` to get its address
- `getProtocol()` to know the exploited transport protocol
Location API II

Intra- vs. Inter- platform mobility

Two different classes implement the Location interface (both from its same package):

- **ContainerID** for *intra-platform* mobility

- Let cName be a container name, its ContainerID can be obtained with
  \[
  \text{new ContainerID(cName, null)}
  \]

- **PlatformID** for *inter-platform* mobility

  - requires the migration service add-on to be installed
  
  - it is developed and maintained by the Universitat Autònoma de Barcelona$^a$

$^a$tao.uab.cat/ipmp/
Finding destinations

To get a Location object, an agent must query the AMS by sending it an ACLMessage.REQUEST (thus, expecting an .INFORM back) storing either

- a new WhereIsAgentAction(AID aid) object → to get the Location where the given agent is
- a new QueryPlatformLocationsAction() object → to get all the Locations available within the JADE platform

In both cases, what you get is a Location object which hides a ContainerID.
What are such “actions”? jade.content.onto.basic.Action is the class representing a FIPA action, that is “an act to be carried out by an agent”—do you remember we defined ACL messages as communicative acts?
Action how-to

To create and request an Action

1. instantiate the Action object
   Action a = new Action()

2. decide who should perform the action—the AMS in our case
   a.setActor(getAMS())

3. choose the action to be performed
   a.setAction(new QueryPlatformLocationsAction())

4. embed the action into the request ACL message
   ACLMessage msg = new ACLMessage(ACLMessage.REQUEST)
   Agent.getContentManager().fillContent(msg, a))

5. send the message to the receiver—again, the AMS for us
   msg.addReceiver(getAMS()) send(msg)
Action API IV

Collecting AMS replies I

To collect AMS replies you can do something like

1. create a suitable data store for locations
   Map locations = new HashMap()

2. receive replies according to your preferred policy but using the correct MessageTemplate
   MessageTemplate mt = MessageTemplate.and(
     MessageTemplate.matchSender(getAMS()),
     MessageTemplate.matchPerformative(ACLMessage.INFORM))
   ACLMessage reply = blockingReceive(mt)

3. ...
Collecting AMS replies II

2. ...

3. *decode* the content of AMS reply—method dual to
   
   ```java
   Agent.getContentManager().fillContent(msg, a))
   Result res =
   (Result) getContentManager().extractContent(reply);
   ```

4. (in our case) collect all the Locations
   
   ```java
   Iterator it = res.getItems().iterator();
   while(it.hasNext()){
      Location l = (Location)it.next();
      locations.put(loc.getName(), l);
   }
   ```
doMove/doClone How-To I

Self-movement

In the case an agent autonomously decides to move itself to another (remote) container in the same JADE platform, it simply calls the method `doMove()` passing the destination Location as a parameter—either discovered thanks to the AMS or a-priori known.

Self-cloning

The case of cloning is similar, except that method to call is obviously `doClone()` and that a second parameter other than the target `Location` should be passed to the call: the new name of the cloned agent (a `String`).
doMove/doClone How-To II

Request for movement/cloning I

Instead, if any JADE agent wishes to make another agent move, it can only perform a [Move/Clone]Action request, hoping the destination agent will do it—nothing more should be expected as usual

1. create a MobileAgentDescription
   MobileAgentDescription mad = new MobileAgentDescription();

2. fill its mandatory fields
   mad.setName(aid);
   mad.setDestination(location);

3. embed it in a [Move/Clone]Action object
   MoveAction ma = new MoveAction();
   ma.setMobileAgentDescription(mad);

4. ...
3. ...

4. pack the ACL request message encoding the [Move/Clone]Action object
   ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
   Agent.getContentManager().fillContent(msg, ma));

5. send the move/clone request message
   msg.addReceiver(aid) send(msg);
Response for movement/cloning

The receiver agent, if agrees with the request

1. decodes the content of the ACL message conveying the action request
   ```java
   ContentElement content = getContentManager().extractContent(msg);
   ```

2. gets the [Move/Clone]Action
   ```java
   MoveAction ma = (MoveAction)(((Action)content).getAction());
   ```

3. gets the destination Location
   ```java
   Location loc = ma.getMobileAgentDescription().getDestination();
   ```

4. eventually, moves/clones itself
   ```java
   if(loc != null) doMove(loc);
   ```
doMove/doClone How-To V

The last note

To be able to call the above-used methods from the ContentManager object, the jade.content.lang.sl.SLCodec and the jade.domain.mobility.MobilityOntology must be registered with it.

• to do so, write in agents **setup()** method
  ```java
  getContentManager().registerLanguage(new SLCodec());
  getContentManager().registerOntology(
      MobilityOntology.getInstance()
  );
  ```

Not a FIPA standard

Notice that such ontology is not yet a FIPA standard, hence may be adapted in the future.
Are we done with JADE?

In lab., yes

In general, answer is no. JADE offers many other things in addition to what we’ve seen during lab. lessons:

- Fault Tolerance Service
- Persistent Message Delivery Service
- User-defined Ontologies Support
- ...

...feel free to experiment by yourselves and ask questions!