JADE API
Mobile Agents

Integrated Master's in Informatics Engineering

Intelligent Agents

2018/2019

Synthetic Intelligence Lab

Filipe Gonçalves

César Analide

ISLab

Mobile Agents

- A Mobile Agent is an executing program that can **migrate**, at **times** of its own choosing, from **machine to machine** in a **heterogeneous network**
- Mobile Agents are an effective paradigm for **distributed applications**, and are particularly attractive for partially **connected computing**
- Using JADE application developers can **build** mobile agents, which are able to **migrate** or **copy** themselves across **multiple network hosts**

Agent Life Cycle

- **INITIATED:** Agent object is built, but hasn't registered itself yet with the AMS
- **ACTIVE :** Agent object is registered with the AMS, has a regular name and address and can access all the various JADE features
- **SUSPENDED :** the Agent object is currently stopped since Its internal thread is suspended
- **WAITING :** the Agent object is blocked, waiting for an event
- **TRANSIT:** the mobile Agent enters this state while it is migrating to the new location. The system continues to buffer messages that will then be sent to its new location

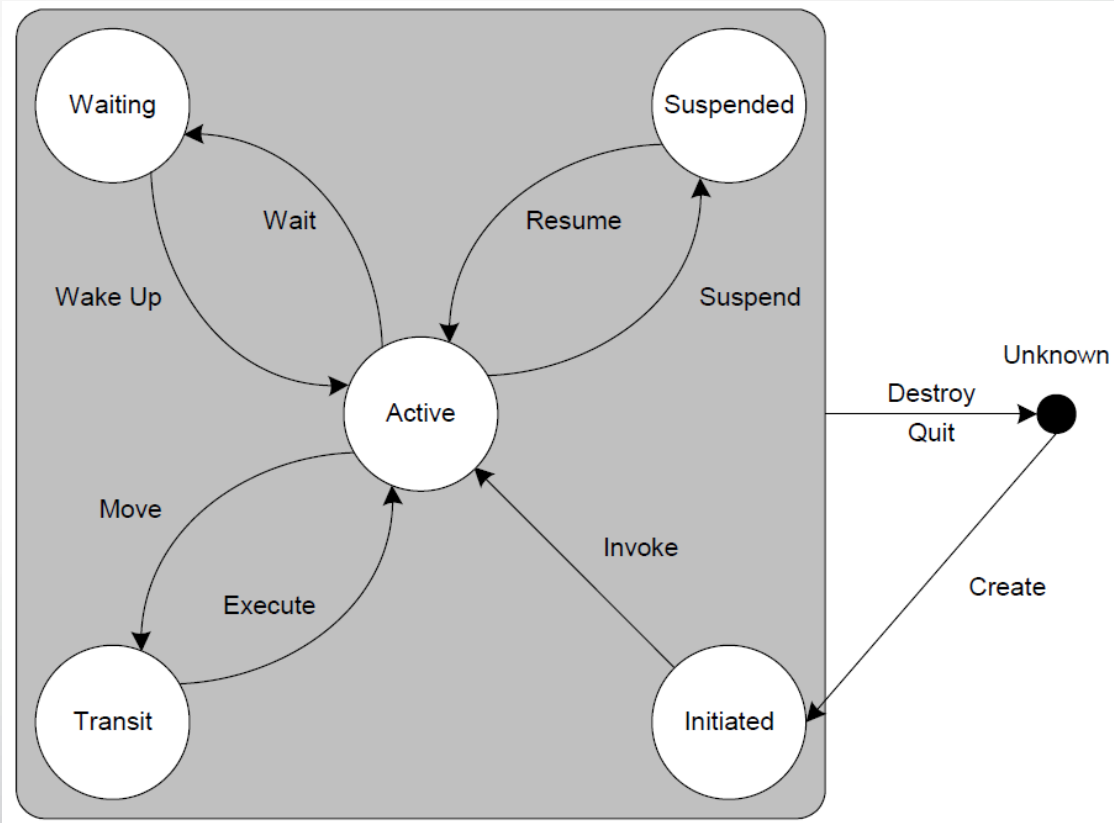

Figure 3 - Agent life-cycle as defined by FIPA.

Mobile Agents

- **Moving** or **cloning** is considered a **state transition** in the life cycle of the agent, which can be initiated either by the agent itself or by the Agent Management Service (AMS)
- **Moving** or **cloning** an agent involves sending its code, resources and state through a network channel
- Mobile agents need to be **location aware** in order to decide **when** and **where** to move
- ***JADE*** makes available a couple of matching methods in the Agent class for resource management and provides a proprietary ontology (***jade-mobility-ontology***) holding the necessary concepts and actions, contained within the ***jade.domain.mobility*** package

JADE API for Agent Mobility

- **Jade Methods**:
  - ***doMove():*** allows a JADE Agent class to migrate elsewhere
    - **INPUT**:
      - **jade.core.Location** single parameter representing the intended destination
  - ***doClone():*** allows a JADE Agent class to spawn a remote copy of itself under a different name
    - **INPUT**:
      - **jade.core.Location** parameter (intended destination)
      - **String** containing the name of the new cloned Agent

- **NOTE**: ***jade.core.Location*** is an abstract interface. Applications agents must ask the AMS for the list of available locations and choose one or request another agents location.

JADE Mobile Agent Patterns

- The ***Location*** class implements ***jade.core.Location*** interface, so that it can be passed to ***Agent.doMove()*** and ***Agent.doClone()*** methods

- A typical behaviour pattern for a JADE mobile agent is **to ask the AMS for locations** (either the complete list or through one or more where-is-agent actions); then the **agent** will be able to **decide if, where and when to migrate**

- **Every mobility related action** can be **requested** to the **AMS** through a FIPA-request protocol, with ***jade-mobility-ontology*** as ontology value and ***FIPA-SLo*** as language value

Move/Clone Code

- **doMove()**

```
String destination = "Container-1";
System.out.println();
Location dest = new jade.core.ContainerID(destination, null);
System.out.println("They requested me to go to " + destination);
// Set reply sentence
replySentence = "\"OK moving to " + destination+" \"";
// Prepare to move
//((MobileAgent)myAgent).nextSite = dest;
myAgent.doMove(dest);
```

- **doClone()**

```
String destination = "Container-1";
System.out.println();
Location dest = new jade.core.ContainerID(destination, null);
System.out.println("They requested me to clone myself to " + destination);
// Set reply sentence
replySentence = "\"OK cloning to " + destination+" \"";
// Prepare to move
//((MobileAgent)myAgent).nextSite = dest;
myAgent.doClone(dest, "clone"+((MobileAgent)myAgent).cnt+"of"+myAgent.getName());
```

JADE API for Agent Mobility

- **doMove() sub-methods:**
  - **beforeMove():** called at the starting location normally to release any local resources used by the original instance
  - **afterMove():** called at the destination location and executed when the agent transition is completed
- **doClone() sub-methods** - similar to beforeMove() and afterMove():
  - **beforeClone()**
  - **afterClone()**
- **AMS Requests:**
  - **query-platform-locations**
  - **where-is-agent**

AMS Requests

- ***query-platform-locations*** action takes no arguments, but its result is a set of all the Location objects available in the current JADE platform.
- **Example**: request message to ask for the location where the agent Peter resides

```
( REQUEST
    :sender (agent-identifier :name Johnny)
    :receiver (set (Agent-Identifier :name AMS))
    :content (( action (agent-identifier :name AMS)
                ( query-platform-locations ) ))
    :language FIPA-SL0
    :ontology JADE-Agent-Management
    :protocol fipa-request
)
```

```
( INFORM
    :sender (Agent-Identifier :name AMS)
    :receiver (set (Agent-Identifier :name Johnny))
    :content (( Result ( action (agent-identifier :name AMS)
                        ( query-platform-locations ) )
                    (set (Location
                        :name Container-1
                        :transport-protocol JADE-IPMT
                        :transport-address IOR:000….Container-1 )
                        (Location
                        :name Container-2
                        :protocol JADE-IPMT
                        :address IOR:000….Container-2 )
                        (Location
                        :name Container-3
                        :protocol JADE-IPMT
                        :address IOR:000….Container-3 )
                    )))
    :language FIPA-SL0
    :ontology JADE-Agent-Management
    :protocol fipa-request
)
```

QueryPlatformLocationsAction Example

```
 private ACLMessage request;

public GetAvailableLocationsBehaviour(MobileAgent a) {
  // call the constructor of FipaRequestInitiatorBehaviour
  super(a, new ACLMessage(ACLMessage.REQUEST));
  request = (ACLMessage)getDataStore().get(REQUEST_KEY);
  // fills all parameters of the request ACLMessage
  request.clearAllReceiver();
  request.addReceiver(a.getAMS());
  request.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
  request.setOntology(MobilityOntology.NAME);
  request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
  // creates the content of the ACLMessage
  try {
    Action action = new Action();
    action.setActor(a.getAMS());
    action.setAction(new QueryPlatformLocationsAction());
    a.getContentManager().fillContent(request, action);
  }
  catch(Exception fe) {
      fe.printStackTrace();
  }
  // creates the Message Template
  // template = MessageTemplate.and(MessageTemplate.MatchOntology(MobilityOntology.NAME),template);
  // reset the fiparequestinitiatorbheaviour in order to put new values
  // for the request aclmessage and the template
  reset(request);
}
```

AMS Requests

- ***where-is-agent*** action has a single AID argument, holding the identifier of the agent to locate. This action has a result, namely the location for the agent, that is put into the content slot of the inform ACL message that successfully closes the protocol
- **Example**: request message to ask for the location where the agent Peter resides

```
(REQUEST
    :sender (agent-identifier :name da1@Zadig:1099/JADE)
    :receiver (set (agent-identifier :name ams@Zadig:1099/JADE))
    :content (
        (action
        (agent-identifier :name ams@Zadig:1099/JADE)
        (where-is-agent (agent-identifier :name Peter@Zadig:1099/JADE))
        )
    )
    :language FIPA-SL0
    :ontology JADE-Agent-Management :protocol fipa-request
)
```

```
(INFORM
    :sender (agent-identifier :name ams@Zadig:1099/JADE)
    :receiver (set (agent-identifier :name da1@Zadig:1099/JADE))
    :content ((result
        (action
        (agent-identifier :name ams@Zadig:1099/JADE)
        (where-is-agent (agent-identifier :name Peter@Zadig:1099/JADE))
        )
        (set (location
            :name Container-1
            :protocol JADE-IPMT
            :address Zadig:1099/JADE.Container-1
            )
        )
    ))
    :reply-with da1@Zadig:1099/JADE976984777740
    :language FIPA-SL0
    :ontology JADE-Agent-Management
    :protocol fipa-request
)
```

WhereIsAgentAction Example

```
public GetAvailableLocationsBehaviour(MobileAgent a) {
  // call the constructor of FipaRequestInitiatorBehaviour
  super(a, new ACLMessage(ACLMessage.REQUEST));
  request = (ACLMessage)getDataStore().get(REQUEST_KEY);
  // fills all parameters of the request ACLMessage
  request.clearAllReceiver();
  request.addReceiver(a.getAMS());
  request.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
  request.setOntology(MobilityOntology.NAME);
  request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
  // creates the content of the ACLMessage
  try {
    Action action = new Action();
    action.setActor(a.getAMS());
    WhereIsAgentAction whereisagentaction = new WhereIsAgentAction();
    whereisagentaction.setAgentIdentifier(new AID("Customer1"));
    action.setAction(whereisagentaction);
    a.getContentManager().fillContent(request, action);
  }
  catch(Exception fe) {
      fe.printStackTrace();
  }
```

Running Mobile Agents

- A singleton instance of the JADE Runtime can be obtained via the static method ***jade.core.Runtime.instance()***, *which provides two methods:*
  - ***JADE main-container***
  - ***JADE remote container*** (i.e. a container that joins to an existing main-container, forming a distributed agent platform)
- **INPUT**: ***jade.core.Profile*** object parameter that keeps the configuration options (e.g. hostname, port number)
- **Notice that,** having created the agent, it still needs to be started via the method start()

Remote Container Code Example

```
import jade.core.Runtime;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.wrapper.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class JavaApplication2 {

    public static void main(String[] args) {
        // Get a hold on JADE runtime
        Runtime rt = Runtime.instance();
        // Create a default profile
        Profile p = new ProfileImpl();
        // Create a new non-main container, connecting to the default;
        // main contianer (i.e. on this host, port 1099)
        ContainerController cc = rt.createAgentContainer(p);
        // Create a new agent, a DummyAgent
        // and pass it a reference to an Object
        Object reference = new Object();
        Object argsl[] = new Object[1];
        argsl[0] = reference;
        AgentController dummy;
        // Fire up the agent
        try {
            dummy = cc.createNewAgent("inProcess","jade.tools.DummyAgent.DummyAgent", argsl);
            dummy.start();
        } catch (StaleProxyException ex) {
            Logger.getLogger(JavaApplication2.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

}
```

Bibliography

- Bellifemine, F., Poggi, A., & Rimassa, G. (1999, April). JADE–A FIPA-compliant agent framework. In *Proceedings of PAAM* (Vol. 99, No. 97-108, p. 33).
- Bellifemine, F., Caire, G., Trucco, T., & Rimassa, G. (2002). Jade programmer's guide. *Jade version*, *3*.
- Bellifemine, F., Poggi, A., & Rimassa, G. (2001, May). JADE: a FIPA2000 compliant agent development environment. In *Proceedings of the fifth international conference on Autonomous agents* (pp. 216-217). ACM.

- Important Link: https://www.iro.umontreal.ca/~vaucher/Agents/Jade/Mobility.html

JADE API
Mobile Agents

Integrated Master's in Informatics Engineering

Intelligent Agents

2018/2019

Synthetic Intelligence Lab

Filipe Gonçalves

César Analide

ISLab