JADE – Java Agent DEvelopment Framework

Taller de sistemas multiagentes

Prof. Dr. Ariel Monteserin
amontese@exa.unicen.edu.ar
ISISTAN –Fac. Cs. Exactas – UNICEN
Tandil, Argentina

2017

1

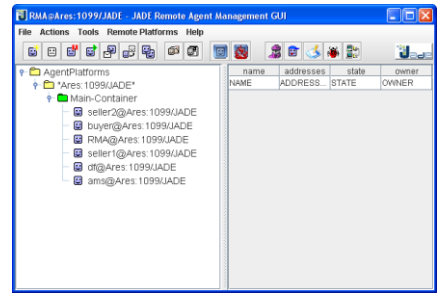
Agenda

- Java Agent DEvelopment Framework
- Plataforma multiagente
- Agentes en JADE
- Comportamientos
- Agentes con interfaz gráfica

2

Introducción

- Framework para el desarrollo de sistemas multi-agente en conformidad con las especificaciones de FIPA.
- JADE provee:
  - Plataforma multiagente conforme a FIPA.
  - Paquete para el desarrollo JAVA de agentes.
  - Set de herramientas gráficas para administrar y monitorear la ejecución de los agentes.



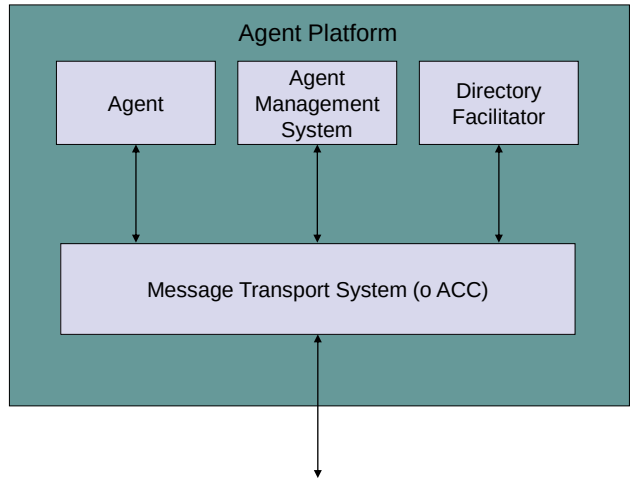
3

Características

- JADE ofrece al programador:
  - Plataforma distribuida.
  - Interfaces graficas para administración remota de agentes.
  - Herramientas de debugging.
  - Movilidad de agentes inter-plataforma.
  - Soporta la ejecución en paralelo de múltiples agentes.
  - Transporte de mensajes ACL dentro de la plataforma.
  - FIPA-complaint Agent Platform (incluye AMS, DF y ACC).
  - Librería de protocolos de interacción FIPA.
  - Servicio de nombres (GUID: Globally Unique Identifier).
  - Interface para que aplicaciones externas inicien agentes autónomos.

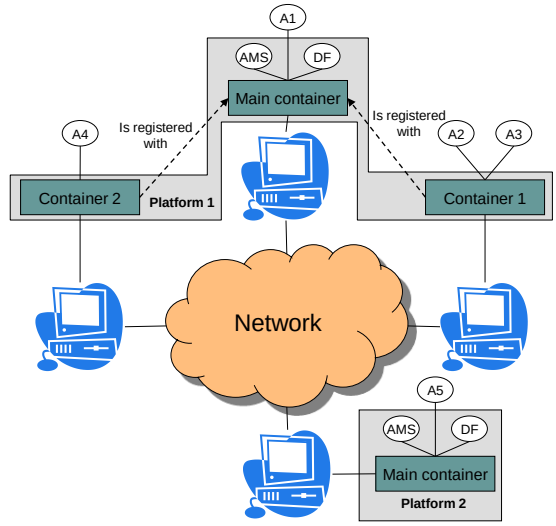
4

Plataforma multiagente



5

Plataforma multiagente distribuida



7

Agenda

- Java Agent DEvelopment Framework
- Plataforma multiagente
- **Agentes en JADE**
- Comportamientos
- Agentes con interfaz gráfica

8

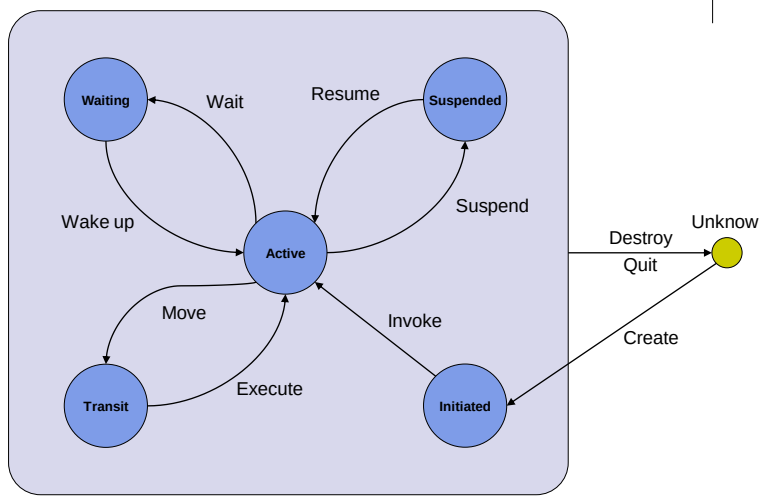
Creando un agente JADE

- La clase `Agent`
  - Incluida en el paquete `jade.core`
  - Define las características básicas para que el agente interactúe con la plataforma
    - registro
    - configuración
    - administración remota

    y los métodos para implementar el comportamiento del agente
    - envío y recepción de mensajes
    - uso de protocolos de interacción estándar
  - Modelo computacional multitarea.
  - Identificación
    - AID: local-name@platform-name

9

Ciclo de vida del agente



10

Creando un agente JADE

```
import jade.core.Agent;

public class MiAgente extends Agent {
  protected void setup() {
      System.out.println("El agente " +
  getAID().getName()+" está activo.");
  }
}
```

Descargar JADE e implementar el agente MiAgente

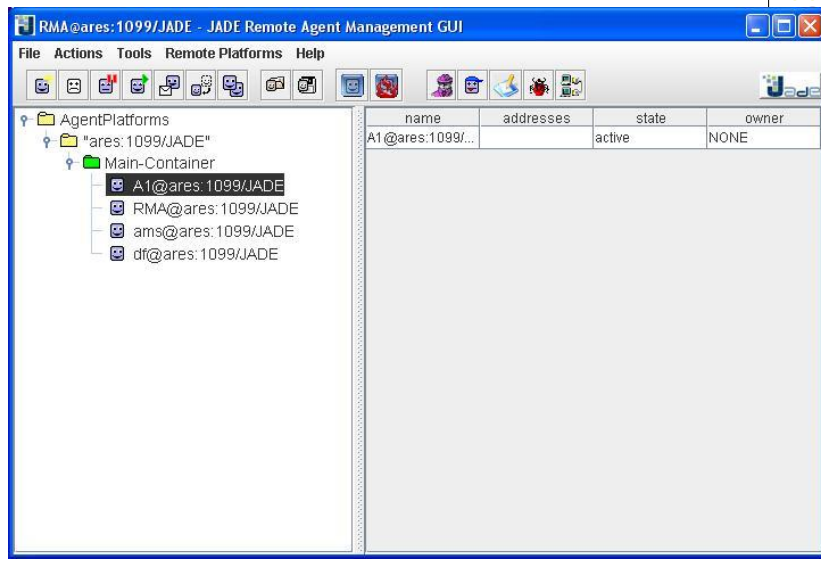
11

Ejecutando el agente

- Desde Eclipse
  - Run…
    - Java Application
      - Main Class: `jade.Boot`
      - Argumentos: `-gui nombre_agente:class_agente`

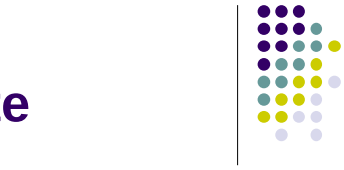
12

Interface de la plataforma JADE



13

Finalización del agente



- Finalizar la ejecución agente
  - `doDelete()`
  - Invoca a
    - `takeDown()`

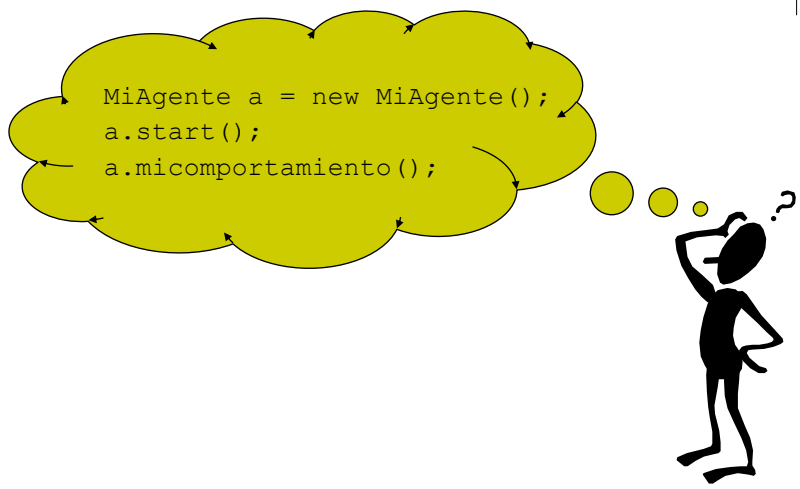
14

Pasaje de argumentos

- Por línea de comando.
  - java jade.Boot nombre:clase(arg1,arg2,arg3)
- Array de Object
- getArgument()

15

Crear un agente desde otro agente/aplicación



16

Crear un agente desde otro agente/aplicación

```
import jade.wrapper.*;
…
AgentContainer c = getContainerController();
   try {
       AgentController a = c.createNewAgent( "X1",
       "MiAgente", null );
       a.start();
   }
   catch (StaleProxyException e){}
…
```

- Nombre = “X1”
- Clase: “MiAgente”
- Argumentos: null

Implementar una agente que active una instancia de si mismo cuando sea eliminado

17

Agenda

- Java Agent DEvelopment Framework
- Plataforma multiagente
- Agentes en JADE
- **Comportamientos**
- Agentes con interfaz gráfica

18

Comportamientos del agente

- Un comportamiento es una tarea que el agente puede ejecutar.
- Son implementados como objetos de las clases que extienden
  - `jade.core.behaviours.Behaviour`
- Se agregan comportamientos al agente con
  - `addBehaviour()`
  - `removeBehaviour()`

19

Comportamientos del agente

- Cada clase que extiende `Behaviour` debe implementar
  - `action()` define la acción a ser ejecutada cuando se ejecute el comportamiento
  - `done()` determina si el comportamiento ha sido completado.
- El agente puede ejecutar varios comportamientos concurrentemente.
- Métodos `onStart()` y `onEnd()`

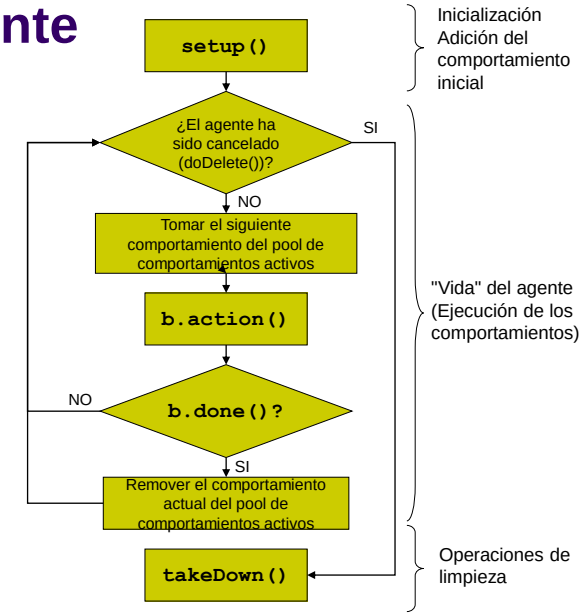
20

Comportamientos del agente

- Todos los comportamientos del agente son manejados por el mismo thread
  - Permite tener un único thread por agente.
  - Provee mejor performance ya que el switch de comportamiento es mas rápido que el switch de thread en Java.
  - Elimina los problemas de sincronización entre comportamientos concurrentes que acceden al mismo recurso.
  - Facilita la persistencia y la movilidad.

21

Path de ejecución del thread del agente



22

Comportamientos

```
public class MiBehaviour extends Behaviour {
  public void action() {
       System.out.println("El agente " +
  myAgent.getAID().getName()+" está activo.");
  }
  public boolean done() {
      return true;
  }
}
```

Implementar el agente MiAgente utilizando comportamientos

23

Comportamiento en un thread dedicado

- Desventaja de un único thread
  - Si el comportamiento se bloquea, se bloquea el agente.
- JADE permite que un comportamiento se ejecute en un thread dedicado.
  - `jade.core.behaviours.ThreadedBehaviourFactory`
  - `wrap()` permite *wrapear* un comportamiento normal en un `ThreadedBehaviour`

24

Consideraciones thread dedicado

- `removeBehaviour()` no funciona con comportamientos en thread dedicado.
  - `ThreadedBehaviourFactory.getThread(b)`
  - `interrupt()`
- Cuando el agente muere, se mueve o suspende, estos comportamientos deben ser explícitamente finalizados.
- Atención con los problemas de sincronización.

25

Tipos de comportamiento

- One-shot – para una única ocación
  - `OneShotBehaviour`
- Cíclico
  - `CyclicBehaviour`
- Genéricos

Comportamiento "One-shot"

- Se ejecuta una única vez
- La clase `OneShotBehaviour` ya implementa el método `done()`

```
public class MyOneShotBehaviour
  extends  OneShotBehaviour {
  public void action() {
       // Ejecuta la operación X
  }
}
```

Comportamiento cíclico

- Se mantiene activo tanto tiempo como esté activo el agente.
- La clase `CyclicBehaviour` ya implementa el método `done()`

```
public class MyCyclicBehaviour extends
      CyclicBehaviour {
  public void action() {
      // Ejecuta la operación Y
  }
}
```

28

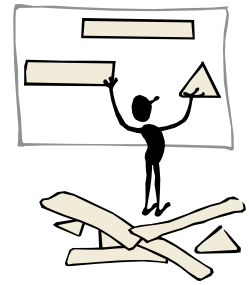
Comportamientos genéricos

- Mantienen un estado del agente y en base a él ejecutan diferentes operaciones.
- Finalizan cuando cierta condición es cumplida.

29

Comportamientos genéricos

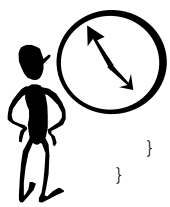
```
public class MyThreeStepBehaviour extends Behaviour {
   private int step = 0;
   public void action() {
        switch (step) {
        case 0: // perform operation X
                step++;
                break;
        case 1: // perform operation Y
                step++;
                break;
        case 2: // perform operation Z
                step++;
                break;
        }
   }
   public boolean done() {
        return step == 3;
   }
}
```



Comportamientos para programar operaciones

- `WakerBehaviour`
  - `action()` y `done()` ya implementados
  - Ejecuta `handleElapsedTimeout()` luego de alcanzado un timeout especificado en el constructor.

```
public class MyAgent extends Agent {
  protected void setup() {
```



```
addBehaviour(new WakerBehaviour(this, 10000) {
        protected void handleElapsedTimeout() {
                System.out.println("X");
        }
} );
        }
    }
```

Comportamientos para programar operaciones

- `TickerBehaviour`
  - `action()` y `done()` ya implementados
- Implementa un comportamiento cíclico que ejecuta periódicamente código definido en `onTick()`

```
public class MyAgent extends Agent {
  protected void setup() {
        addBehaviour(new TickerBehaviour(this, 1000) {
              protected void onTick() {
                    System.out.println("Y");
              }
        } );
  }
}
```

32

Comportamientos compuestos

- CompositeBehaviour
  - Modela comportamientos que están compuestos por otros comportamientos (hijos).
  - La política de selección de hijos está implementada en las subclases:
    - SequencialBehaviour
    - ParallelBehaviour
    - FSMBehaviour – Finite State Machine

33

SequencialBehaviour

- Ejecuta los hijos en forma secuencial.
- Finaliza cuando el último hijo finaliza
- Agregar hijos
  - addSubBehaviour(Behaviour b)

ParallelBehaviour

- Ejecuta sub-comportamientos concurrentemente.
- Se ejecuta hasta que cierta condición es alcanzada
  - Todos/uno/algunos de sus hijos finalizan

FSMBehaviour

- Permite especificar estados y transiciones
  - `registerFirstState(Behaviour b, String n)`
    - Registra un único comportamiento como estado inicial.
  - `registerLastState(Behaviour b, String n)`
    - Registra uno o más comportamientos como estados finales.
  - `registerState(Behaviour b, String n)`
    - Registra uno o más comportamientos como estados intermedios.

FSMBehaviour

- `registerTransition(String s1, String s2, int event)`
  - Registra una transición entre el estado s1 y s2. Event es el evento retornado por el método onEnd() del comportamiento s1.
- `registerDefaultTransition(String s1, String s2)`
  - Registra una transición entre el estado s1 y s2. La transición es ejecutada cuando s1 termina con un evento que no está explicitado para ninguna otra transición.

Behaviour - DataStore

- Útil para el intercambio de datos entre comportamientos
  - `setDataStore(DataStore ds)`
  - `DataStore getDataStore()`
- Extiende la clase `HashMap`
  - `put(Object key, Object value)`
- Los datos se pierden cuando el comportamiento es reseteado

38

Agenda

- Java Agent DEvelopment Framework
- Plataforma multiagente
- Agentes en JADE
- Comportamientos
- **Agentes con interfaz gráfica**

39

Agentes con interfaz gráfica

- Autonomía del agente vs. naturaleza reactiva de la interfaz.
- Clase `GuiAgent`
  - Maneja una cola de objetos `jade.gui.GuiEvent`
    - `GuiEvent(eventSource, eventType)`
    - `addParameter(Object)`
    - `Object getParameter(int)`
    - `Iterator getAllParameter()`
  - Método `onGuiEvent(GuiEvent)`
    - Toma un evento y lo procesa.
  - Método `postGuiEvent(GuiEvent)`
    - Envía el evento al GuiAgent

Agentes con interfaz gráfica

- **Agente**

```
public class MiAgenteGui extends
   GuiAgent {

private GuiAgentFrame gui;

public void setup() {
   gui = new GuiAgentFrame(this);
   gui.setVisible(true);

}

protected void onGuiEvent(GuiEvent
   ev) {
   // procesa los eventos
}
}
```

- **Interfaz gráfica**

```
public class GuiAgentFrame extends
   JFrame {
…
private MiAgenteGui myAgent;
…


// Listener de la interfaz
public void
   actionPerformed(ActionEvent evt){
   GuiEvent ev = new
   GuiEvent(this,0);
   ev.addParameter(//datos del
   evento);
   myAgent.postGuiEvent(ev);
}
```

Agentes con interfaz gráfica

OnGuiEvent()

```
07-sep-2007 14:10:44 jade.core.AgentContainerImpl
INFO: ----------------------------------------
Agent container Main-Container@ares is ready.
----------------------------------------------
Probando interfaz...
```

Behaviour.action()

Implementar MiAgente, pero ingresando el mensaje desde la interfaz gráfica

42

Agentes con interfaz gráfica

- Para modificar la interfaz desde el agente
  - Encapsular el acceso a la interfaz con un objeto `Runnable`
  - Utilizar `SwingUtilities.invokeLater()` para enviar el `runnable` al thread `EventDispatcher`.

```
// Método en la interfaz
public void contestar(final String s){
    Runnable addIt = new Runnable() {
          public void run() {
          // cambio en la interfaz
          }
    };
    SwingUtilities.invokeLater(addIt);
}
```

Mostrar en el cuadro de texto de la interfaz anterior, un mensaje del agente.

44

JADE – Java Agent DEvelopment Framework

Taller de sistemas multiagentes

Prof. Dr. Ariel Monteserin
amontese@exa.unicen.edu.ar
ISISTAN –Fac. Cs. Exactas – UNICEN
Tandil, Argentina