Laboratorio

Plataforma de Agentes Jade

Ulises Cortés
Ignasi Gómez-Sebastià

SID2015

Agente: Clase

- Una clase de agente se crea extendiendo la clase *jade.core.Agent*
  - Se redefine el método *setup()*
- Cada instancia de agente se identifica con un *AID*
  - Clase *jade.core.AID*
  - Método *getAID()* para obtener el AID del agente

```
import jade.core.Agent;

public class HalloWorldAgent extends Agent {

  protected void setup() {
    System.out.println("Hallo World! my name is "+getAID().getName());
  }
}
```

Agente: Nombre

- El nombre del agente es de la forma
  - *nombre-agente@nombre-plataforma*
  - El nombre del agente debe ser único localmente, el nombre completo debe ser único globalmente
- Se puede especificar el nombre de la plataforma al arrancarla usando la opción *–name*
- Dentro de una plataforma nos referimos al agente usando únicamente su nombre local

```
– AID id = new AID(localname, AID.ISLOCALNAME);
– AID id = new AID(name, AID.ISGUID);
```

Agente: Parámetros

- Paso de parámetros a un agente al arrancarlo

```
java jade.Boot .... a:myPackage.MyAgent(arg1 arg2)
```

- Recuperamos los parámetros usando el método *getArguments()*

```
protected void setup() {
  System.out.println("Hallo World! my name is "+getAID().getName());
  Object[] args = getArguments();
  if (args != null) {
    System.out.println("My arguments are:");
    for (int i = 0; i < args.length; ++i) {
      System.out.println("- "+args[i]);
    }
  }
}
```

Agente: Defunción

- El agente termina su ejecución cuando se llama su método *doDelete()*
- Se invoca el método *takeDown()* para realizar operaciones de limpieza
  - Cerrar ficheros y conexiones

```
protected void setup() {
  System.out.println("Hallo World! my name is "+getAID().getName());
  Object[] args = getArguments();
  if (args != null) {
    System.out.println("My arguments are:");
    for (int i = 0; i < args.length; ++i) {
      System.out.println("- "+args[i]);
    }
  }
  doDelete();
}

protected void takeDown() {
  System.out.println("Bye...");
}
```

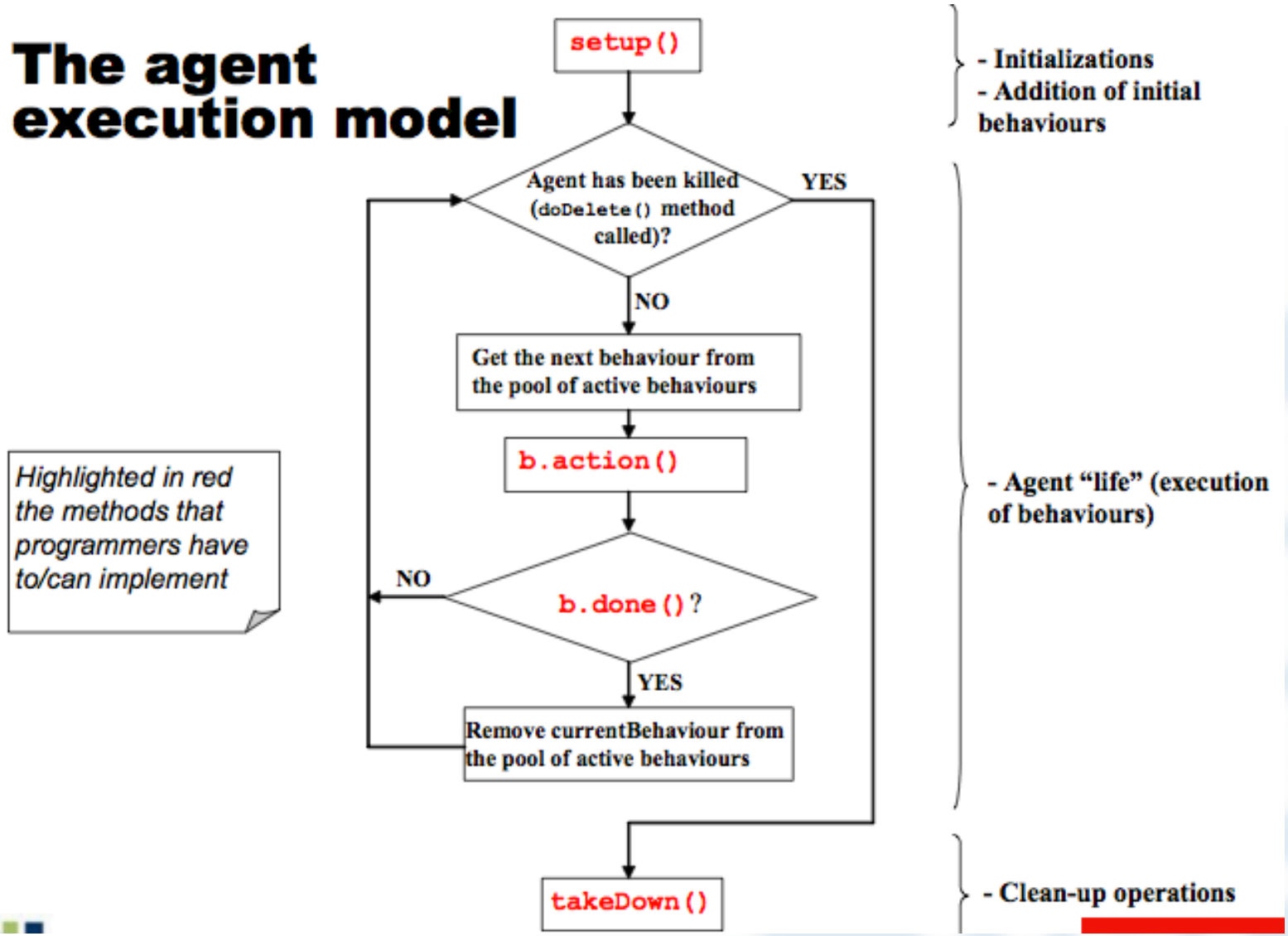
Agente: Behaviours

- Los agentes realizan sus tareas a partir de *behaviours*
  - Extensión de la clase *jade.core.behaviours.Behaviour*
- Creamos nuestra clase behaviour y la añadimos al agente usando la función *addBehaviour()*
  - Podemos compartir behaviours entre agentes
- Cada behaviour debe implementar:
  - Método *void action()*
    - Que hace el behaviour
  - Método *boolean done()*
    - Devuelve cierto si el behaviour ha finalizado

Agente: Concurrencia

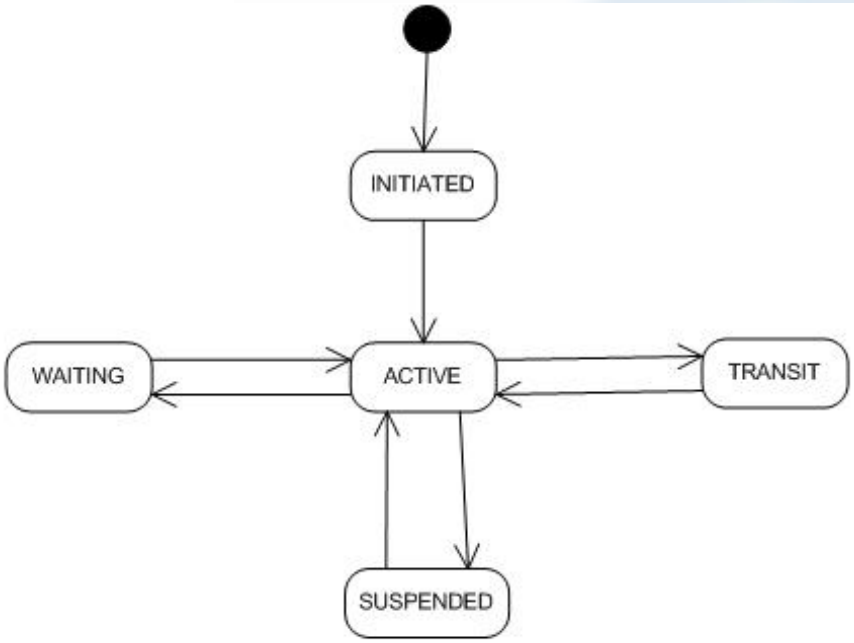
- Un agente puede tener varios comportamientos en paralelo
- El planificado no es *preemptive*
  - Se debe esperar a que termine un behaviour para seleccionar el siguiente
    - Método *action()* retorna
  - Todo ocurre dentro del mismo thread de Java
  - No es determinista a la hora de seleccionar un behaviour si hay varios disponibles

Agente: Ciclo de ejecución



Agente: Ciclo de ejecución

State	Description
Initiated	The agent object is built, but hasn't registered itself with AMS, has neither name nor an addressand can't communicate with other agents.
Active	The agent object is registered with AMS, has regular name and address, can access all the various JADE features.
Suspended	The agent object is currently stopped. Its internal thread is suspended and no agent behaviour is being executed.
Waiting	The agent is blocked, waiting for somethink. Its internal thread is sleeping on a JAVA monitor and will wake up when some conditions are met (for example when message arrives).
Deleted	The agent is definitely dead. The intternal thread has terminated its execution and the agent is no more registered with AMS.
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.



Agente: Tipos de behaviour I

- One shot
  - jade.core.behaviours.OneShotBehaviour
  - Action() se ejecuta una vez
  - Done() devuelve cierto
- Cyclic
  - jade.core.behaviours.CyclicBehaviour
  - Action() se ejecuta multiples veces
  - Done() devuelve falso
- Complex
  - Basados en máquinas de estados finitas

Agente: Tipos de behaviour II

- Waker
  - Espera un time-out y ejecuta método *onWake()*
  - El behaviour se completa
  - Despertador chino
- Ticker
  - Se ejecuta método *onTick()* periodicamente
  - Se para al ejecutar método *stop()*

Agente: Métodos

- onStart()
  - Se ejecuta una vez antes de ejecutar método *action()*
  - Prepara la ejecución (abrir ficheros y conexiones)
- onEnd()
  - Se ejecuta una vez después de que onEnd() devuelva cierto
  - Limpia la ejecución (cerrar ficheros y conexiones)
- removeBehaviour()
  - Elimina un behaviour dispnonible de la lista
  - No llama a onEnd()
- Si el agente no tiene behaviours disponibles está IDLE y el thread en sleep

Agente: Comunicación I

- Los mensajes son instancias de jade.lang.acl.ACLMessage
  - **get/setPerformative();**
  - **get/setSender();**
  - **add/getAllReceiver();**
  - **get/setLanguage();**
  - **get/setOntology();**
  - **get/setContent();**

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

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("Peter", AID.ISLOCALNAME));
msg.setLanguage("English");
msg.setOntology("Weather-Forecast-Ontology");
msg.setContent("Today it's raining");
send(msg);
```

Agente: Comunicación II

- El uso continuo de recieve() es infeficiente
- Uso de *block()*
- Bloquea el behaviour de la lista de behaviours del agente
  - No bloquea el agente
- Cada vez que llega un mensaje, se desbloquean los behaviours bloqueados y pueden leerlo

```
public void action() {
  ACLMessage msg = myAgent.receive();
  if (msg != null) {
    // Process the message
  }
  else {
    block();
  }
}
```

This is the strongly recommended pattern to receive messages within a behaviour

Agente: Comunicación III

- Es más seguro hacer uso de message templates

```
MessageTemplate tpl = MessageTemplate.MatchOntology("Test-Ontology");

public void action() {
  ACLMessage msg = myAgent.receive(tpl);
  if (msg != null) {
    // Process the message
  }
  else {
    block();
  }
}
```

Agente: Comunicación III

- Es más seguro hacer uso de message templates

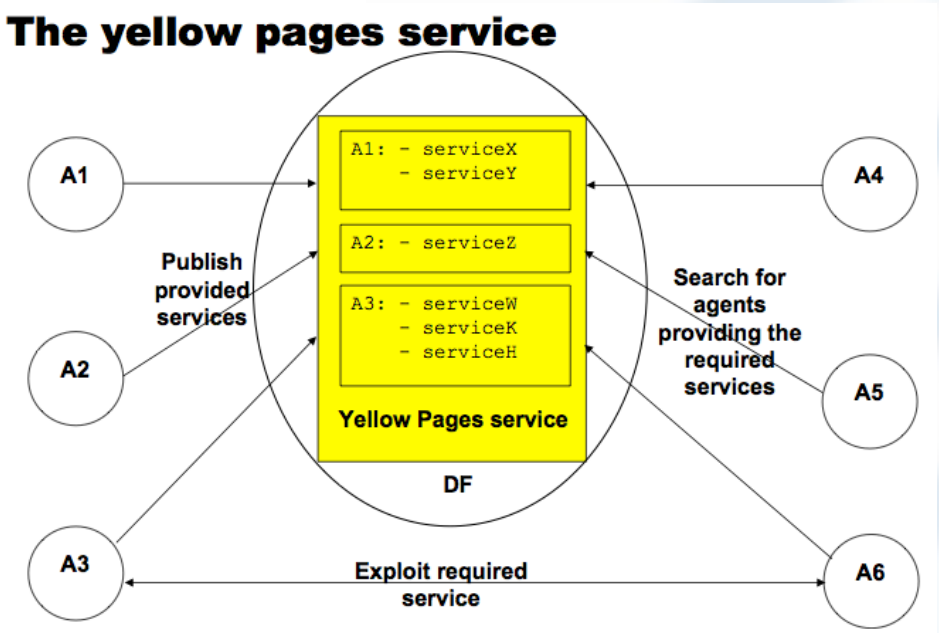
```
MessageTemplate tpl = MessageTemplate.MatchOntology("Test-Ontology");

public void action() {
  ACLMessage msg = myAgent.receive(tpl);
  if (msg != null) {
    // Process the message
  }
  else {
    block();
  }
}
```

- Método *blockingReceive()* disponible
  - Pero peligroso

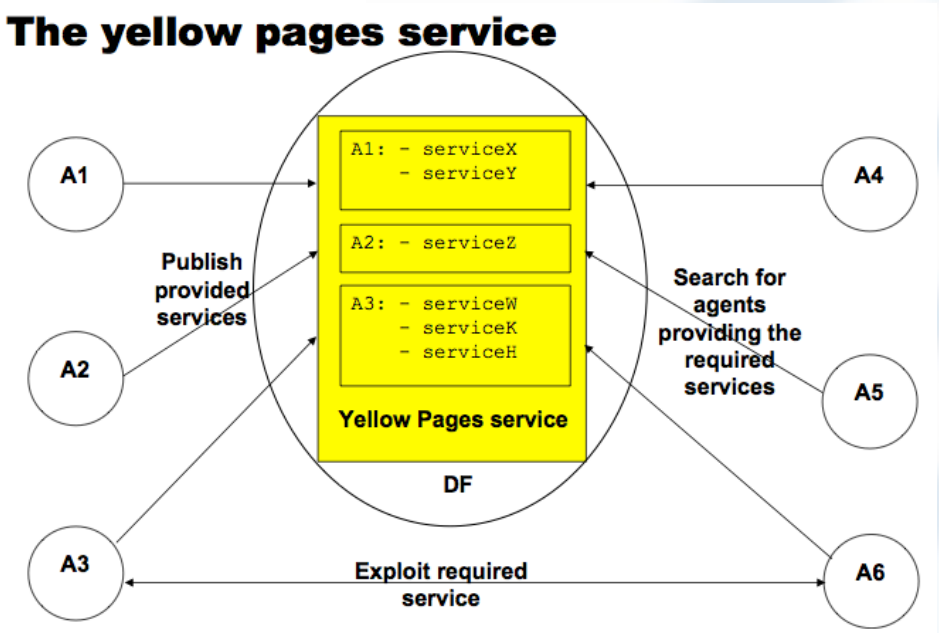
Agente: DF

- Clase jade.domain.DFService
  - Register()
  - Modify()
  - Deregister()
  - Search()



Agente: DF

- Clase jade.domain.DFService
  - Register()
  - Modify()
  - Deregister()
  - Search()



Tareas I

- Arrancar la plataforma de agentes
  - Ejemplo en jade.txt
    - GUI
    - Dummy Agent
    - Sniffer
    - Introspector

Tareas II

- Mirar HelloWorldAgent
  - Import
  - Extends
  - AID
  - Parámetros
  - Modificar y ejecutar
    - Desde GUI
    - Desde consola
    - Mirar resultados en consola
    - Usar sniffer en agente

Tareas III

- Behaviours
  - One shot
  - Waker
  - Cyclic/Ticker
    - Action
    - OnStart
    - OnEnd
    - Done
    - Setup
    - doDelete
    - takeDown
  - Ejecutar los 4 agentes y entender comportamiento

Tareas IV

- Agente Lover
  - Blocking behaviour
    - ¿Se bloquea el agente?
    - Recibir mensaje usando template
    - Enviar mensaje usando template
    - Uso de performativas
- Ejecutar el agente y comprobar su comportamiento
  - GUI
  - Consola (Sniffer)
- Uso del DF