

La Plataforma de Agentes JADE  
*Java Agents Development Environment*  
Escuela de Primavera de Agentes  
Patrocinado por Agentcities.es  
Imaginática 2005, Universidad de Sevilla

Juan A. Botía

3 de Marzo de 2005

Introducción a JADE

Comportamientos en JADE

Comunicación entre agentes con JADE

Protocolos de interacción FIPA en JADE

Manejo de Ontologías en JADE

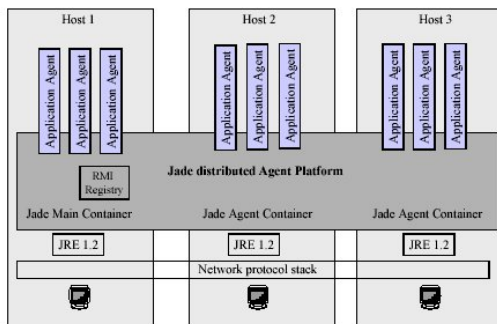
## La plataforma JADE

- ▶ JADE está compuesta de
  - ▶ Una plataforma FIPA para la ejecución de agentes
  - ▶ Un conjunto de paquetes para la programación de agentes FIPA
- ▶ Es 100 % Java (con el JDK 1.4 o superiores)
- ▶ Incluye
  - ▶ Creación básica de agentes
  - ▶ Programación del comportamiento de los agentes en base a *behaviors*
  - ▶ ACL FIPA para envío y recepción de mensajes
  - ▶ Clases útiles para programación de protocolos FIPA (y no FIPA)
  - ▶ Distintos codecs (SL, RDF, etc)
  - ▶ Manejo de información usando ontologías

## La plataforma JADE

- ▶ Plataforma FIPA (AMS, Facilitador de directorio y MTS)
- ▶ Puede ejecutarse en una o varias JVM
- ▶ Cada JVM es vista como un entorno en donde los agentes pueden ejecutarse concurrentemente e intercambiarse mensajes
- ▶ Organizada en contenedores
  - ▶ 1 principal: AMS, DF y el registro rmi están localizados ahí
  - ▶  $n$  containers no principales y conectados al principal

## La plataforma JADE (y II)



## Servicios básicos: directorio

El directorio, como en FIPA, es un servicio básico accesible a través de `jade.domain.DFService` (en realidad es un acceso al agente de páginas amarillas desde un interface) para los servicios

- ▶ `register`
- ▶ `deregister`
- ▶ `modify`
- ▶ `search`

## La clase Agent

Programar un agente en JADE consiste en definir una clase Java que representa al agente y:

- ▶ Determinar y codificar los comportamientos que va a *manifestar*
- ▶ Hacer que herede de la clase `jade.core.Agent`
- ▶ Programar sus métodos `setup()`, `takeDown`

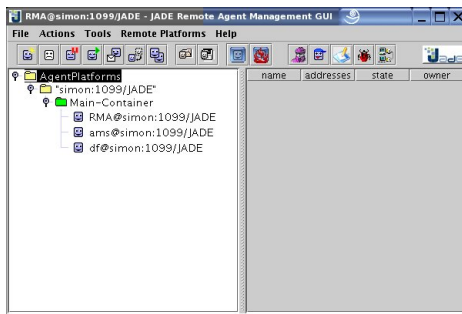
Para ejecutar el agente, podemos hacerlo desde el GUI de JADE o desde cualquier otro programa JAVA explícitamente

## Lanzar un agente desde el GUI

Desde una consola:

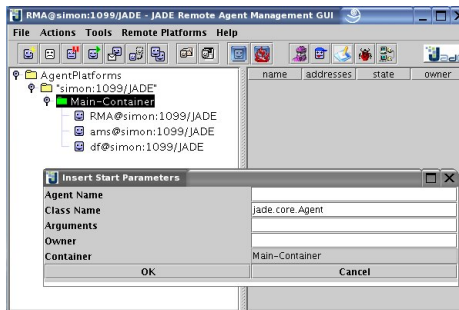
```
export CLASSPATH=/ruta/jade/lib/jade.jar:/ruta/jade/lib/iiop.jar:/ruta/jade/lib/http.jar  
java jade.Boot -gui
```

y tenemos



## Lanzar un agente desde el GUI (y II)

Seleccionando el botón de New Agent, posicionados en el contenedor principal



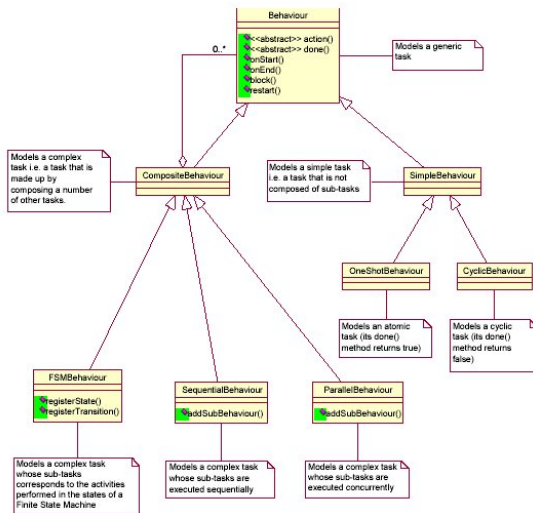
## Los Comportamientos de un agente JADE

- ▶ Los agentes deben poder ejecutar diferentes tareas al mismo tiempo
- ▶ JADE propone un modelo de agente *single threaded* y añade un nivel de scheduling sobre la única thread a nivel de comportamientos
- ▶ Programación basada en comportamientos:
  1. determinar qué debe ser capaz de hacer el agente
  2. asociar cada funcionalidad con un comportamiento
  3. escoger el tipo de comportamiento
  4. dejar a JADE la tarea del scheduling (un solo comportamiento se está ejecutando en cada instante)

## El scheduling de comportamientos

- ▶ Cada agente tiene para sí una cola de comportamientos activos
- ▶ El cuerpo de acciones de un comportamiento se programa redefiniendo el método `action()`
- ▶ Cuando el método anterior finaliza, y dependiendo del tipo de comportamiento, el scheduler lo saca de la cola o lo vuelve a colocar al final
- ▶ Un comportamiento puede bloquearse (`block()`) hasta que lleguen más mensajes al agente; el bloqueo significa que, cuando `action()` termina, se le coloca en una cola de bloqueados
- ▶ Cuando llega un nuevo mensaje, se le saca de esa cola y se coloca al final de la de comportamientos activos

## Los Comportamientos de un agente JADE (y II)



## El ACL estándar de FIPA en JADE

Ideas principales:

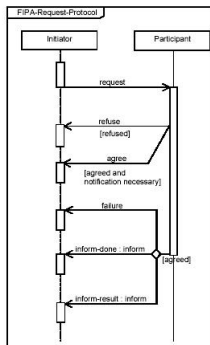
- ▶ La clase `jade.lang.acl.ACLMessage` es la base para composición de mensajes (métodos `set` y `get` para todos los parámetros de un mensaje FIPA)
- ▶ Los métodos `Agent.send(...)`, `Agent.receive(...)` y `Agent.blockingReceive()` para envío y recepción
- ▶ La clase `jade.lang.acl.MessageTemplate` es útil para hacer matching de mensajes

## Protocolos de Interacción

- ▶ FIPA define, como vimos ayer, protocolos de interacción estándares
- ▶ Filosofía de JADE: en lugar de programar el flujo del protocolo (i.e. la secuencia de intercambio de mensajes) programar qué hacer en cada situación (manejadores)
- ▶ Basado en las clases `jade.proto.AchieveREInitiator` y `jade.proto.AchieveREResponder` (suficientes para implementar FIPA-Request, FIPA-query, FIPA-Request-When, FIPA-recruiting, FIPA-brokering y FIPA-subscribe)
- ▶ Para el FIPA-contract-net disponemos de `jade.proto.ContractNetInitiator` y `jade.proto.ContractNetResponder`

## Programación de una interacción

Si queremos programar una interacción simple (i.e. 1 a 1), como por ejemplo `fipa-request`, echamos mano del estándar para el protocolo de interacción



Para programar el iniciador, nos fijamos en los mensajes entrantes y análogamente para el *responder*

## Programación de una interacción (y II)

Nos vamos al javadoc de  
`jade.proto.SimpleAchieveREInitiator`

Method Summary	
void	<b>action()</b> Runs the behaviour.
boolean	<b>done()</b> Check if this behaviour is done.
protected void	<b>handleAgree(ACLMessage msg)</b> This method is called every time an agree message is received, which is not out-of-sequence according to the protocol rules.
protected void	<b>handleAllResponses(java.util.Vector msgs)</b> This method is called when all the responses have been collected or when the timeout is expired.
protected void	<b>handleAllResultNotifications(java.util.Vector msgs)</b> This method is called when all the result notification messages have been collected.
protected void	<b>handleFailure(ACLMessage msg)</b> This method is called every time a failure message is received, which is not out-of-sequence according to the protocol rules.
protected void	<b>handleInform(ACLMessage msg)</b> This method is called every time an inform message is received, which is not out-of-sequence according to the protocol rules.
protected void	<b>handleNotUnderstood(ACLMessage msg)</b> This method is called every time a not-understood message is received, which is not out-of-sequence according to the protocol rules.
protected void	<b>handleOutOfSequence(ACLMessage msg)</b> This method is called every time a message is received, which is out-of-sequence according to the protocol rules.
protected void	<b>handleRefuse(ACLMessage msg)</b> This method is called every time a refuse message is received, which is not out-of-sequence according to the protocol rules.
void	<b>onStart()</b> This method is just an empty placeholders for subclasses.
protected ACLMessage	<b>prepareRequest(ACLMessage msg)</b> This method must return the ACLMessage to be sent.
void	<b>reset()</b> This method resets this behaviour so that it restarts from the initial state of the protocol with a null message.
void	<b>reset(ACLMessage msg)</b> This method resets this behaviour so that it restarts the protocol with another request message.

## Programación de una interacción (y III)

Nos vamos al javadoc de  
`jade.proto.SimpleAchieveREResponder`

Method Summary	
static <a href="#">MessageTemplate</a>	<a href="#">createMessageTemplate</a> (java.lang.String iprotocol) This static method can be used to set the proper message Template (based on the interaction protocol and the performative) into the constructor of this behaviour.
boolean	<a href="#">done</a> () This method checks whether this behaviour has finished or not.
protected <a href="#">ACLMessage</a>	<a href="#">prepareResponse</a> ( <a href="#">ACLMessage</a> request) This method is called when the Initiator's message is received that matches the message template passed in the constructor.
protected <a href="#">ACLMessage</a>	<a href="#">prepareResultNotification</a> ( <a href="#">ACLMessage</a> request, <a href="#">ACLMessage</a> response) This method is called after the response has been sent and only when one of the following two cases arise: the response was an agree message OR no response message was sent.
void	<a href="#">reset</a> () Reset this behaviour using the same MessageTemplate.
void	<a href="#">reset</a> ( <a href="#">MessageTemplate</a> mt) This method allows to change the MessageTemplate that defines what messages this FIPARequestResponder will react to and reset the protocol.

## Ontologías básicas en JADE

- ▶ JADE permite el manejo de ontologías para
  - ▶ Representar el dominio de aplicación mediante conceptos, predicados, acciones, agentes, etc.
  - ▶ Intercambiar elementos de la ontología entre agentes (i.e. en el cuerpo del mensaje FIPA)
- ▶ + Conceptualmente sencillo
- ▶ - Muy engorroso de manejar (definición tediosa de conceptos)
- ▶ Solución: podemos utilizar un sistema gestor de ontologías como Protégé2000 para el modelado y generación de código Java-JADE

## Manejo de ontologías desde Protégé2000

Los pasos a seguir en la programación de un sistema JADE sencillo, usando Protégé podrían ser los siguientes:

1. Conceptualización del problema
  - ▶ Definición de los elementos a participar en la ontología
  - ▶ Definición de los agentes
  - ▶ Definición de las interacciones que tendrán lugar entre los agentes (comprobando que la ontología sea adecuada para todos los casos)
2. Diseño de la ontología con Protégé
3. Programación de los protocolos de interacción con JADE (i.e. en forma de comportamientos), integrando el código generado por Protégé
4. Programación de los agentes
5. Y ya está

## Conclusiones

Para dominar JADE, tenemos que conocer

- ▶ Agentes y behaviours (mecanismos básicos de scheduling)
- ▶ Mensajes y plantillas para el matching
- ▶ Protocolos de interacción y su programación mediante clases Initiator y Responder
- ▶ Manejo de Ontologías con algún programa de apoyo (e.g. Protégé)

Manos a la obra...

## Agradecimientos

Algunas de las figuras usadas en esta presentación se han obtenido directamente del manual de Jade. Agradecemos a Fabio Bellifemine y a su equipo su expresa y desinteresada colaboración.