

Laboratorio

Plataforma de Agentes Jade

Ulises Cortés

Ignasi Gómez-Sebastià

Sergio Álvarez

SID2018

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: Clase

- HelloWorldAgent.java

```
package org.upc.edu.Behaviours;
```

```
/**  
 *  
 * @author igomez  
 */
```

```
import jade.core.Agent;
```

```
public class HelloWorldAgent extends Agent  
{
```

```
    protected void setup()  
    {
```

```
        String arguments = "";
```

```
        Object[] args = getArguments();  
        if (args != null) {  
            for (int i = 0; i < args.length; ++i)  
            {  
                arguments = arguments + " - " + args[i];  
            }  
        }
```

```
        System.out.println("Hallo World! my name is "+getAID().getName() + " and my arguments are " + arguments);  
    }
```

```
}
```

Agente: Clase

```
--Hello World Agent Windows
start java -cp ./lib/jade.jar;./dist/JadeApplication.jar jade.Boot -local-host 127.0.0.1 -gui
start java -cp ./lib/jade.jar;./dist/JadeApplication.jar jade.Boot -local-host 127.0.0.1 -container
HelloTio:org.upc.edu.Behaviours.HelloWorldAgent("Que","pasa","tio")

--Hello World Agent Linux
java -cp ./lib/jade.jar:./dist/JadeApplication.jar jade.Boot -local-host 127.0.0.1 -gui &
java -cp ./lib/jade.jar:./dist/JadeApplication.jar jade.Boot -local-host 127.0.0.1 -container
HelloTio:org.upc.edu.Behaviours.HelloWorldAgent\("Que","pasa","tio"\) &
```

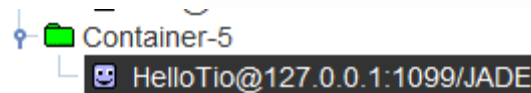
```
mar 13, 2019 11:02:50 AM jade.core.Runtime beginContainer
INFO: -----
This is JADE snapshot - revision 6357 of 2010/07/06 16:27:34
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
-----
Retrieving CommandDispatcher for platform null
mar 13, 2019 11:02:50 AM jade.imtp.leap.LEAPIMTPManager initialize
INFO: Listening for intra-platform commands on address:
- jicp://127.0.0.1:51000

mar 13, 2019 11:02:51 AM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
mar 13, 2019 11:02:51 AM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
mar 13, 2019 11:02:51 AM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
mar 13, 2019 11:02:51 AM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
mar 13, 2019 11:02:51 AM jade.core.messaging.MessagingService clearCachedSlice
INFO: Clearing cache
Hallo World! my name is HelloTio@127.0.0.1:1099/JADE and my arguments are - Que - pasa - tio
mar 13, 2019 11:02:51 AM jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Container-5@127.0.0.1 is ready.
-----
```

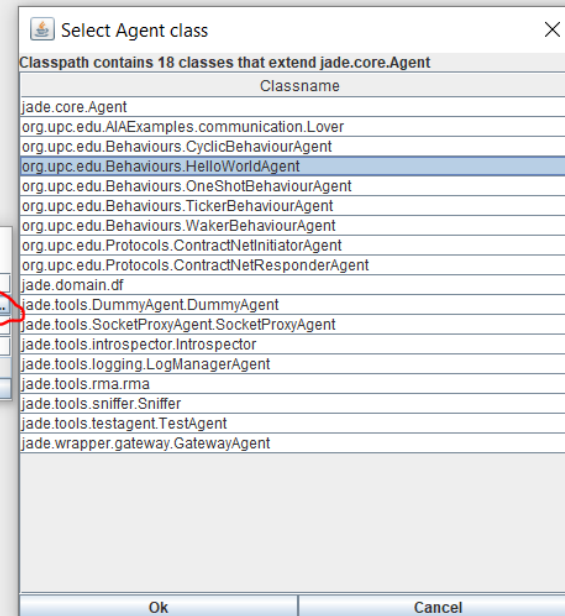
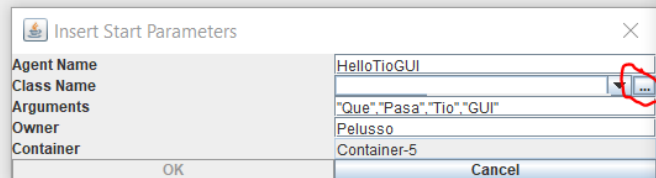
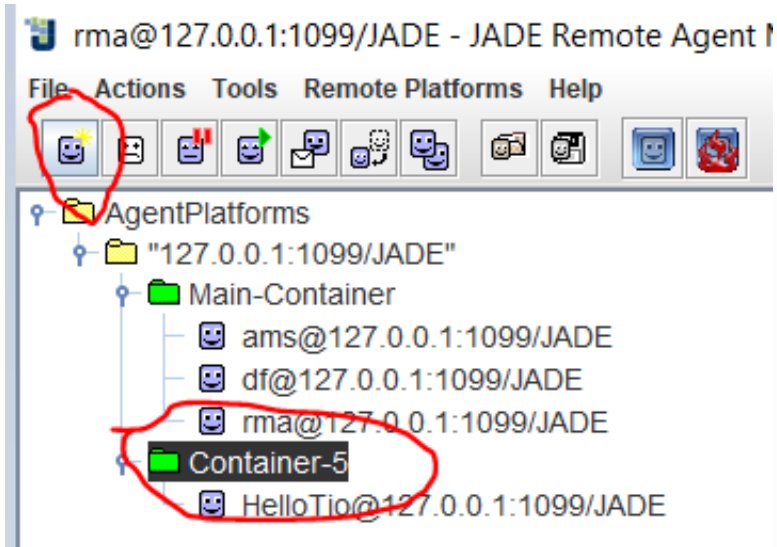
Agente: Clase

```
--Hello World Agent Windows
start java -cp ./lib/jade.jar;./dist/JadeApplication.jar jade.Boot -local-host 127.0.0.1 -gui
start java -cp ./lib/jade.jar;./dist/JadeApplication.jar jade.Boot -local-host 127.0.0.1 -container
HelloTio:org.upc.edu.Behaviours.HelloWorldAgent("Que","pasa","tio")

--Hello World Agent Linux
java -cp ./lib/jade.jar:./dist/JadeApplication.jar jade.Boot -local-host 127.0.0.1 -gui &
java -cp ./lib/jade.jar:./dist/JadeApplication.jar jade.Boot -local-host 127.0.0.1 -container
HelloTio:org.upc.edu.Behaviours.HelloWorldAgent\("Que","pasa","tio"\) &
```

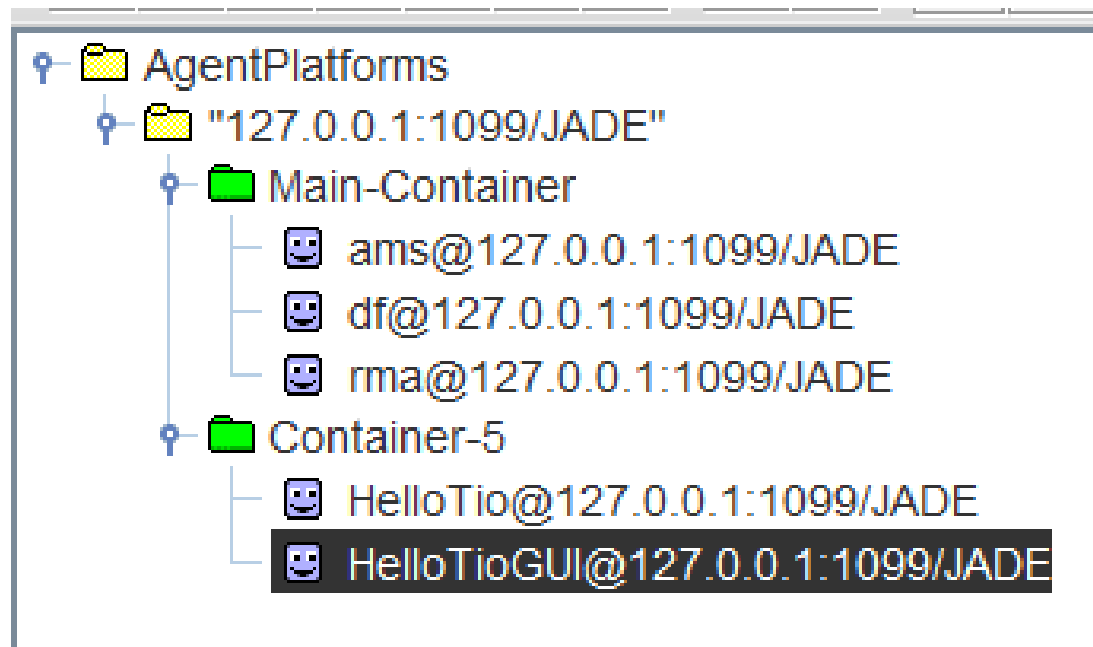


Agente: Clase



Agente: Clase

Hallo World! my name is HelloTioGUI@127.0.0.1:1099/JADE and my arguments are - "Que" - "Pasa" - "Tio" - "GUI"



Agente: Clase

- Ejercicios
 - Descargar plataforma desde el racó
 - Abrir con editor
 - Netbeans, Eclipse, etc.
 - Plataforma desde consola
 - Comprobar logs
 - Arrancar agente HelloWorldAgent desde consola
 - Comprobar logs
 - Arrancar agente HelloWorldAgent desde GUI
 - Comprobar logs
 - Se han pasado igual los parámetros?

Agente: Defunción

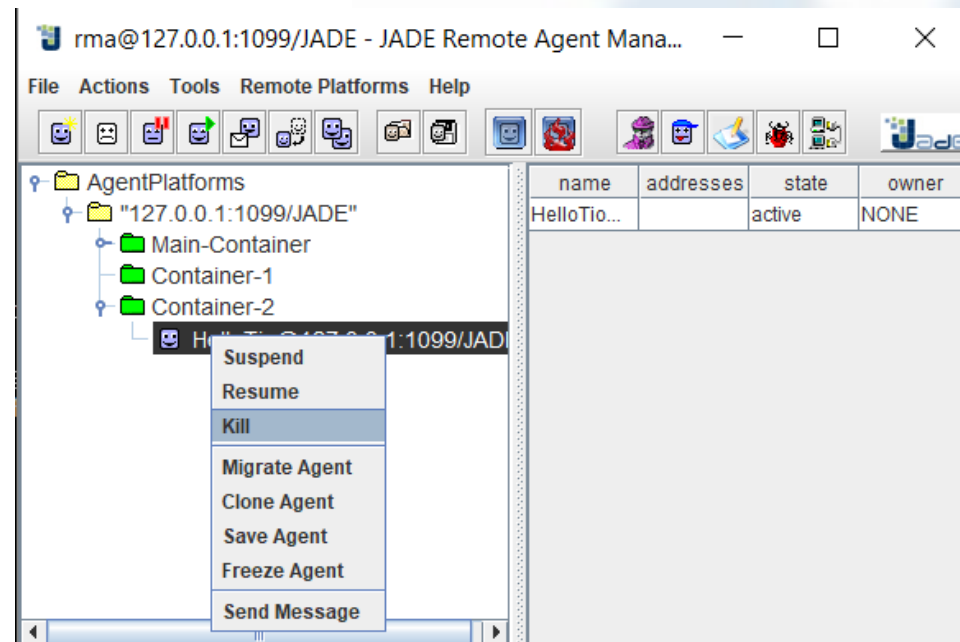
- El agente termina su ejecución cuando se llama su método *doDelete()*
- Se invoca el método *takeDown()* para realizar operaciones de limpieza
 - Cerrar ficheros y conexiones

```
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: Clase

- Ejercicios
 - Ampliar el agente HelloWorldAgent para añadir operación takeDown
 - Mostrar mensaje en los logs en la operación
 - Arrancar el agente y matarlo desde la GUI
 - Comprobar logs y ver que la operación es ejecuta
 - Ampliar el agente HelloWorldAgent para que se mate tras mostrar el mensaje con nombres y argumentos



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: Behaviours

```
public class CyclicBehaviourAgent extends Agent
{
    public class HelloWorldCyclicBehaviour extends CyclicBehaviour
    {
        String message;
        int count_chocula;

        public HelloWorldCyclicBehaviour()
        {
        }

        public void onStart()
        {
            this.message = "Agent " + myAgent + " with HelloWorldCyclicBehaviour in action!!" + count_chocula;
            count_chocula = 0;
        }

        public int onEnd()
        {
            System.out.println("I have done " + count_chocula + " iterations");
            return count_chocula;
        }

        public void action()
        {
            System.out.println(this.message + count_chocula);
            ++count_chocula;
        }
    }

    protected void setup()
    {
        HelloWorldCyclicBehaviour b = new HelloWorldCyclicBehaviour();
        this.addBehaviour(b);
    }
}
```

Agente: Behaviours

- Ejercicios
 - Ejecutar el agente CyclicBehaviourAgent desde consola
 - Comprobar los logs
 - Matar el agente desde GUI
 - Extender el behaviour con el método done
 - El behaviour termina al escribir 10 veces
 - Ejecutar el agente CyclicBehaviourAgent desde consola y mirar cambios en los logs

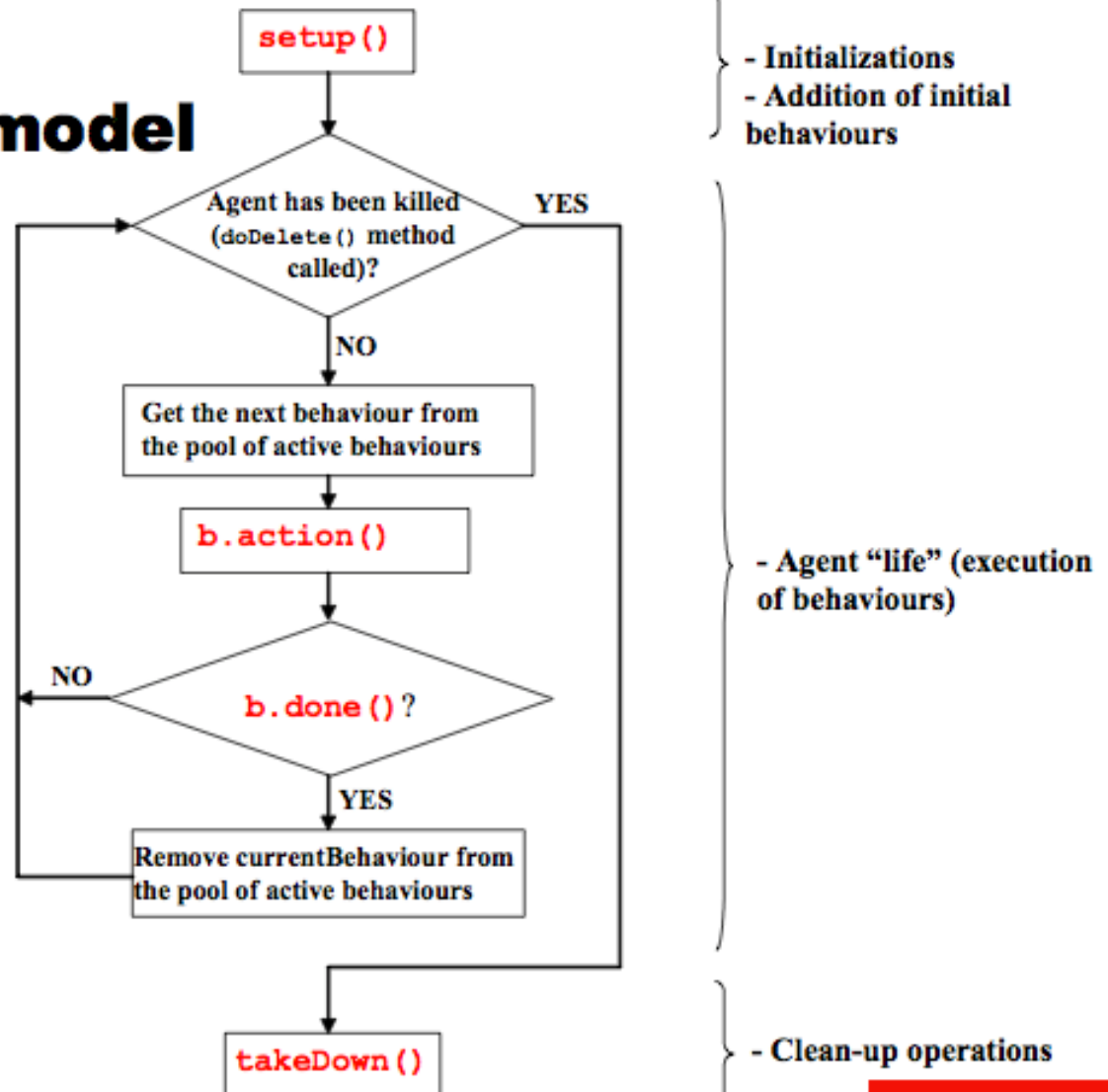
Agente: Concurrency

- 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

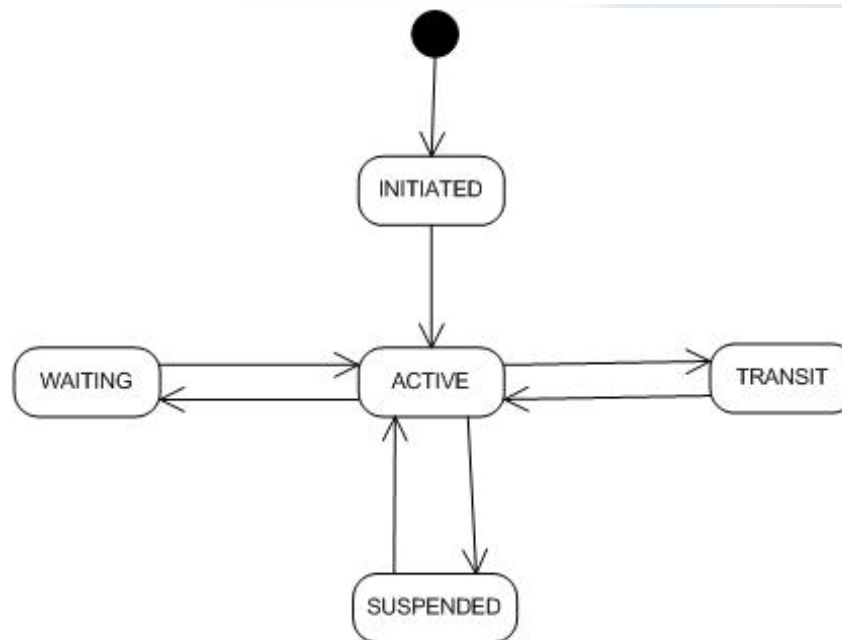
The agent execution model

Highlighted in red the methods that programmers have to/can implement



Agente: Ciclo de ejecución

State	Description
Initiated	The agent object is built, but hasn't registered itself with AMS, has neither name nor an address and 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 something. 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 internal 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()* periódicamente
 - Se para al ejecutar método *stop()*

Agente: Tipos de behaviour II

```
public class CyclicBehaviourAgent extends Agent
{
    public class HelloWorldCyclicBehaviour extends CyclicBehaviour
    {
        String message;
        int count_chocula;

        public HelloWorldCyclicBehaviour()
        {
        }

        public void onStart()
        {
            this.message = "Agent " + myAgent + " with HelloWorldCyclicBehaviour in action!!" + count_chocula;
            count_chocula = 0;
        }

        public int onEnd()
        {
            System.out.println("I have done " + count_chocula + " iterations");
            return count_chocula;
        }

        public void action()
        {
            System.out.println(this.message + count_chocula);
            ++count_chocula;
        }
    }

    protected void setup()
    {
        HelloWorldCyclicBehaviour b = new HelloWorldCyclicBehaviour();
        this.addBehaviour(b);
    }
}
```

Agente: Tipos de behaviour II

```
public class OneShotBehaviourAgent extends Agent
{
    public class HelloWorldOneShotBehaviour extends OneShotBehaviour
    {
        String message;
        int count_chocula;

        public HelloWorldOneShotBehaviour()
        {
        }

        public void onStart()
        {
            this.message = "Agent " + myAgent + " with HelloWorldOneShotBehaviour in action!!";
            count_chocula = 0;
        }

        public int onEnd()
        {
            System.out.println("I have done " + count_chocula + " iterations");
            return count_chocula;
        }

        public void action()
        {
            System.out.println(this.message + count_chocula);
        }
    }

    protected void setup()
    {
        HelloWorldOneShotBehaviour b = new HelloWorldOneShotBehaviour();
        this.addBehaviour(b);
    }
}
```

Agente: Tipos de behaviour II

```

public class TickerBehaviourAgent extends Agent
{
    public class HelloWorldTickerBehaviour extends TickerBehaviour
    {
        String message;
        int count_chocula;

        public HelloWorldTickerBehaviour(Agent a, long period)
        {
            super(a, period);
        }

        public void onStart()
        {
            this.message = "Agent " + myAgent + " with HelloWorldTickerBehaviour in action!!";
            count_chocula = 0;
        }

        public int onEnd()
        {
            System.out.println("I have done " + count_chocula + " iterations");
            return count_chocula;
        }

        public void onTick()
        {
            System.out.println(this.message + count_chocula);
            ++count_chocula;
        }
    }

    protected void setup()
    {
        HelloWorldTickerBehaviour b = new HelloWorldTickerBehaviour(this, 3000);
        this.addBehaviour(b);
    }
}

```

Agente: Tipos de behaviour II

```
public class WakerBehaviourAgent extends Agent
{
    public class HelloWorldWakerBehaviour extends WakerBehaviour
    {
        String message;
        int count_chocula;

        public HelloWorldWakerBehaviour(Agent a, long timeout)
        {
            super(a, timeout);
        }

        public void onStart()
        {
            this.message = "Agent " + myAgent + " with HelloWorldWakerBehaviour in action!!";
            count_chocula = 0;
        }

        public int onEnd()
        {
            System.out.println("I have done " + count_chocula + " iterations");
            return count_chocula;
        }

        public void onWake()
        {
            System.out.println(this.message + count_chocula);
            ++count_chocula;
        }
    }

    protected void setup()
    {
        HelloWorldWakerBehaviour b = new HelloWorldWakerBehaviour(this, 3000);
        this.addBehaviour(b);
    }
}
```


Agente: Tipos de behaviour II

- Ejercicios
 - Ejecutar los siguientes tipos de agentes y comprobar su comportamiento
 - OneShot
 - Ticker
 - Waker
 - Encajan con la descripción realizada?

Agente: Métodos

- onStart()
 - Se ejecuta una vez antes de ejecutar método *action()* por primera vez
 - 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 disponible 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 `receive()` es ineficiente
- 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");

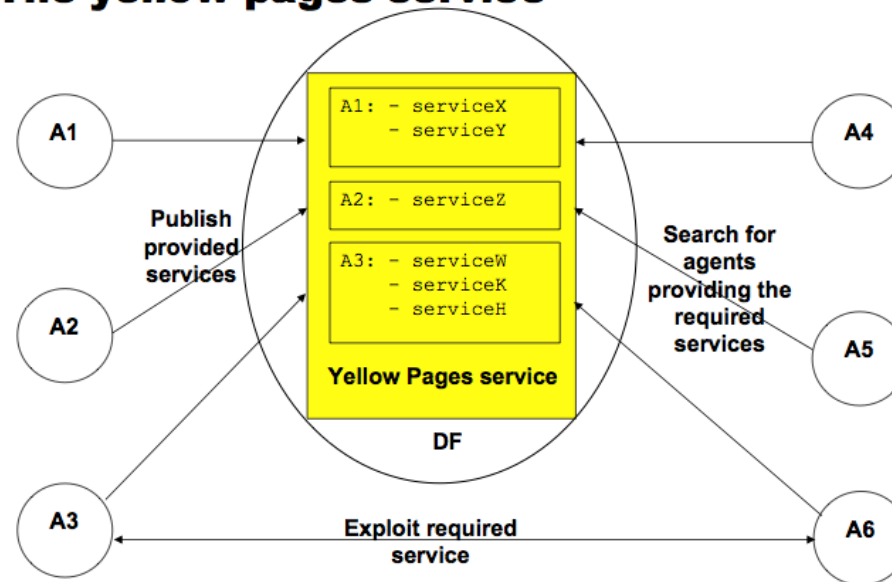
public void action() {
    ACLMessage msg = myAgent.receive(tpl);
    if (msg != null) {
        // Process the message
    }
    else {
        block();
    }
}
```

- Método *blockingReceive()* disponible
 - Pero peligroso

Agente: DF

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

The yellow pages service



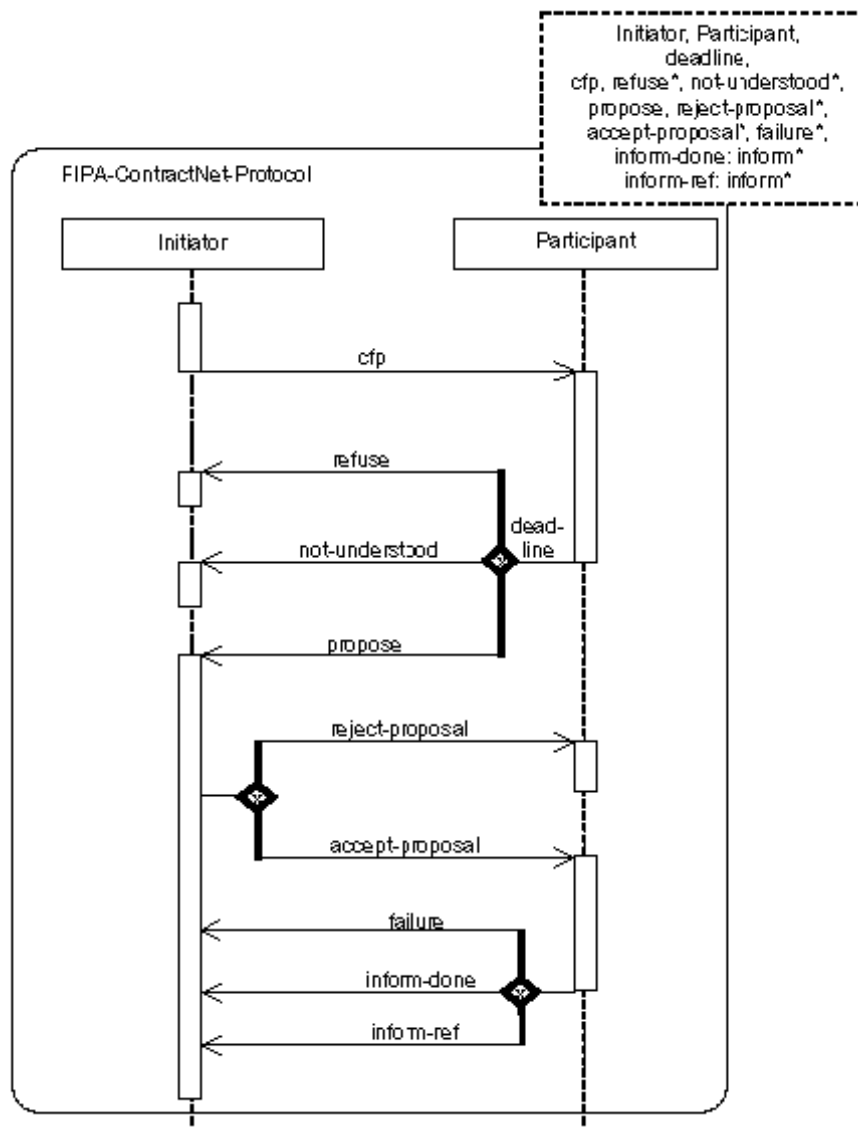
Tareas

- 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
 - Usar el Sniffer en la GUI
- Entender el uso Uso del DF

Performativas

performative	passing info	requesting info	negotiation	performing actions	error handling
accept-proposal			x		
agree				x	
cancel		x		x	
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propose			x		
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe		x			

Protocolos FIPA



Tareas

- Agentes con protocolos
 - ContractNetInitiatorAgent
 - ContractNetResponderAgent
- Ejecutar los agentes
 - Desde consola
 - Usar el Sniffer en la GUI para observarlos
 - Usar el introspector en la GUI para observarlos
 - Jugar con los parámetros del agente

```
--Complex agents based on protocols
java -cp /home/igomez/Jade/jade/lib/jade.jar:/home/igomez/NetBeansProjects/JadeApplication/dist/JadeApplication.jar jade.Boot
-local-host 127.0.0.1 -gui &
java -cp /home/igomez/Jade/jade/lib/jade.jar:/home/igomez/NetBeansProjects/JadeApplication/dist/JadeApplication.jar jade.Boot
-local-host 127.0.0.1 -container Timmy:org.upc.edu.Protocols.ContractNetResponderAgent\("YES","6"\) &
java -cp /home/igomez/Jade/jade/lib/jade.jar:/home/igomez/NetBeansProjects/JadeApplication/dist/JadeApplication.jar jade.Boot
-local-host 127.0.0.1 -container Jimmy:org.upc.edu.Protocols.ContractNetResponderAgent\("YES","4"\) &
java -cp /home/igomez/Jade/jade/lib/jade.jar:/home/igomez/NetBeansProjects/JadeApplication/dist/JadeApplication.jar jade.Boot
-local-host 127.0.0.1 -container Winnie:org.upc.edu.Protocols.ContractNetResponderAgent\("NO","0"\) &
java -cp /home/igomez/Jade/jade/lib/jade.jar:/home/igomez/NetBeansProjects/JadeApplication/dist/JadeApplication.jar jade.Boot
-local-host 127.0.0.1 -container Sesilu:org.upc.edu.Protocols.ContractNetInitiatorAgent\("Timmy","Jimmy","Winnie"\)
```