# PROYECTO FINAL DE CARRERA


# Estudio de la implementación de agentes en dispositivos móviles

***Alexandre Viejo Galicia***
Ingeniero técnico en Informática de Sistemas

Directores de proyecto:
**Dr. Antonio Moreno Ribas**
**Dra. Aïda Valls Mateu**

Escola Técnica Superior d'Enginyeria (ETSE)
Universitat Rovira i Virgili (URV)

# ÍNDICE

**1. INTRODUCCIÓN ..... 4**
1.1. FIPA ..... 5
1.2. Agentcities ..... 5
1.3. Grupo de investigación BANZAI - GruSMA ..... 6
1.4. El ejemplo implementado ..... 6
1.5. Objetivos del proyecto ..... 7
1.6. Estructura del documento ..... 7
1.7. Recursos utilizados ..... 8

**2. AGENTES Y SISTEMAS MULTI-AGENTES ..... 9**
2.1. Propiedades de los agentes ..... 9
2.2. Arquitectura de los agentes ..... 10
2.3. Tipos de agentes ..... 10
2.4. Sistemas Multi-Agente ..... 10
2.4.1. Ventajas de un Sistema Multi-Agente ..... 11
2.4.2. Gestión de un Sistema Multi-Agente ..... 11
2.4.3. Lenguajes de comunicación entre agentes ACL ..... 12
2.4.3.1. Elemento de un mensaje ..... 12
2.4.3.2. Protocolos de comunicación ..... 14

**3. JADE ..... 15**
3.1. Paquetes de JADE ..... 15
3.2. Componentes principales de JADE ..... 16
3.3. Agentes y Comportamientos (Behaviours) ..... 16
3.3.1. Comportamientos Simples ..... 17
3.3.2. Comportamientos Compuestos ..... 17
3.4. Mensajes ..... 18
3.5. Ontología ..... 19
3.6. Protocolos de comunicación ..... 20
3.7. Tipos de protocolos ..... 21
3.7.1. FIPA REQUEST ..... 21
3.7.2. FIPA CONTRACT-NET ..... 23
3.8. Herramientas de JADE ..... 25
3.9. Como actualizar código de JADE2.x a JADE3.0b. ..... 30

**4. LEAP 3.0 PARA JADE 3.0b ..... 33**
4.1. Entorno de ejecución JADE-LEAP ..... 34
4.2. Ejecución de JADE-LEAP en estaciones (PC o Servidor) ..... 35
4.3. Ejecución de JADE-LEAP en dispositivos con recursos limitados ..... 36
4.3.1. PDAs con PersonalJava ..... 37
4.3.2. Moviles con MIDP1.0 ..... 37
4.4. Ejemplo final del entorno JADE-LEAP ..... 38
4.5. Diferencias entre JADE y JADE-LEAP para J2SE ..... 38
4.6. Limitaciones en JADE-LEAP para pjava y midp ..... 39
4.7. JADE-LEAP y la conexión sin cables (wireless) ..... 40
4.7.1. Conexión Bluetooth entre PDA y PC ..... 41

**5. DESCRIPCIÓN DEL SMA DE EJEMPLO: GESTIÓN DE TAXIS ..... 46**
5.1. Descripción general del problema ..... 46
5.2. Justificación en la utilización de un SMA ..... 46
5.3. Arquitectura del sistema ..... 47
5.4. Agente Cliente ..... 48
5.5. Agente Centralita ..... 48
5.6. Agente Taxi ..... 49
5.7. Agente Visualizador ..... 50
5.8. Agente Tráfico ..... 51
5.9. La simulación ..... 51
5.10. La Interficie Grafica del AgenteCliente ..... 53

**6. COMUNICACIÓN ENTRE LOS AGENTES ..... 56**
6.1. Intercambio de mensajes entre los agentes ..... 56
6.2. Ontología ..... 59
6.2.1. Los conceptos de la ontología ..... 60
6.2.2. Las acciones de la ontología ..... 61
6.2.3. Relación: Ontología - Servicios Agentes – DF Agent ..... 63

**7. IMPLEMENTACIÓN ..... 64**
7.1. Clases y directorios ..... 64
7.2. Package AGENTES ..... 64
7.2.1. AgenteCliente.java ..... 64
7.2.2. AgenteCentralita.java ..... 65
7.2.3. AgenteTaxi.java ..... 66
7.2.4. AgenteTrafico.java ..... 67
7.2.5. AgenteVisualizador.java ..... 68
7.3. Package ONTOLOGY ..... 68

**8. DEMOSTRACIÓN PRÁCTICA ..... 70**
8.1. Ejemplo con mapa clásico ..... 71
8.2. Ejemplo con mapa alternativo ..... 79

**9. CÓDIGO FUENTE ..... 81**
9.1. Package ONTOLOGY ..... 81
9.1.1. TaxiOntology.java ..... 81
9.1.2. TaxiType.java ..... 84
9.1.3. RespuestaPeticion.java ..... 86
9.1.4. RecorridoTaxi.java ..... 88
9.1.5. Posicion.java ..... 89
9.1.6. Peticion.java ..... 89
9.1.7. ListaCalles.java ..... 90
9.1.8. EstimacionRecorridoLlegada.java ..... 91
9.1.9. DibujarTaxi.java ..... 91
9.1.10. DibujarRecorridoTaxi.java ..... 92
9.1.11. DibujarPersona.java ..... 93
9.1.12. Desocupado.java ..... 93
9.1.13. DameCallesAnuladas.java ..... 93
9.1.14. BuscarTaxi.java ..... 94

9.1.15. BorrarTaxi.java ..... 94
9.1.16. BorrarRecorridoTaxi.java ..... 95
9.1.17. BorrarPersona.java ..... 96
9.2. Package AGENTES ..... 96
9.2.1. AccionARealizar.java ..... 96
9.2.2. AgenteCentralita.java ..... 97
9.2.3. AgenteCliente.java ..... 108
9.2.4. AgenteTaxi.java ..... 113
9.2.5. AgenteTrafico.java ..... 132
9.2.6. AgenteVisualizador.java ..... 138
9.2.7. CoordCliente.java ..... 146
9.2.8. Coordenadas2.java ..... 148
9.2.9. Coordenadas.java ..... 149
9.2.10. DatosMapa.java ..... 150
9.2.11. InfoMap.java ..... 153
9.2.12. Mapa.java ..... 153
9.2.13. MapaCliente.java ..... 158
9.2.14. Ventana.java ..... 163
9.2.15. VentanaCliente.java ..... 164
9.2.16. VentanaMapa.java ..... 170

**10. VALORACIÓN Y CONCLUSIONES ..... 172**

**11. FUTUROS PROYECTOS ..... 173**

**12. AGRADECIMIENTOS ..... 174**

**13. BIBLIOGRAFÍA ..... 175**

# 1. INTRODUCCIÓN

El mundo en que vivimos, nos exige de forma imperativa, una mayor facilidad a la hora de realizar cualquier tipo de tarea, sea cual sea el ámbito donde nos movemos. Nos exige facilidad si, pero también comodidad en el trato, rapidez en la resolución y sobretodo, eficiencia. Estos tres factores que a priori serían antagónicos, son el desafío de las nuevas tecnologías, que responde aportándonos el concepto de *asistente personal electrónico*, el cometido del asistente, será ahorrar tiempo y dinero para conseguir que las personas tengamos una vida más cómoda y llevadera.

Podemos imaginar cientos de situaciones, donde tener lo que llamaríamos un asistente informatizado, nos sería de gran utilidad, centrándonos en un ámbito del ocio, podríamos pensar en las facilidades que obtendríamos al planear un fin de semana, siendo el asistente el encargado de realizar todo el trabajo más pesado: buscar espectáculos, reservar entradas, billetes etc. Y todo ello dándonos un servició a nuestra medida, totalmente personalizado, dado que el asistente personal nos conoce, sabe nuestros gustos y lo que estamos dispuestos a gastar. Si por el contrario, quisiéramos enfocar nuestro asistente a un entorno de trabajo, las ventajas también son evidentes, tanto a la hora de buscar información, como planificar sistemas de alarma, detección... los campos son múltiples, solo haría falta aplicar el asistente personal a ellos, con el software adecuado.

No obstante, el concepto de agente personal, se encuentra con un grave problema, y es el de la movilidad, habremos visto estaciones y ordenadores portátiles ofrecernos su ayuda en los entornos comentados anteriormente, e incluso volverse indispensables en la mayoría de las situaciones cuotidianas. Sin duda son eficientes y naturalmente muy rápidos, pero no presentan facilidad en su uso y por descontado no son cómodos de utilizar, incluso en el caso del portátil más ligero, su carga nos impide considerarlo el asistente personal que nos tiene que acompañar allí donde vayamos.

Es en este punto donde pretendemos mejorar las condiciones existentes, y darle un verdadero impulso al agente personal. Por ello, nuestra pretensión con este trabajo, sería proveer a los dispositivos móviles, agendas electrónicas (PDAs) o teléfonos móviles, que proliferan por nuestras ciudades y que la mayoría de nosotros llevamos en nuestros bolsillos, de un entorno de ejecución de software, que ayude a las personas en aquellos campos, en los que la aportación del asistente informatizado sea necesaria.

Esta plataforma que pretendemos conseguir, nos presenta diferentes alternativas, incluyendo entornos realizados con VisualC++ o VisualBasic. Estas opciones, aunque interesantes, no nos aportan las soluciones que nosotros buscamos, y de hecho, se alejan del concepto de asistente personal, y es que nuestro asistente, tiene que ser capaz de comunicarse y colaborar con muchos otros asistentes y herramientas provistas por empresas y gobiernos. Es en este marco donde podemos encontrar una nueva tecnología emergente, nacida de la mano de la Inteligencia Artificial, que trabaja con los llamados *Sistemas Multi-Agente (SMA),* sistemas que engloban un conjunto de programas independientes y autónomos, también llamados *agentes*, diseñados para trabajar juntos para una finalidad común.

Es en esta definición donde podemos ver reconocido, el modelo de asistente que deseamos, por tanto, es por este camino por donde avanzaremos, topándonos de inmediato con la problemática de la movilidad, y es que aunque pensados para entornos distribuidos, con los agentes ejecutándose en diferentes maquinas, los SMA, requieren suficientes recursos como para complicar su funcionamiento en los pequeños dispositivos que son el objeto de nuestro estudio. Es por ello, que el objetivo prioritario de nuestro trabajo, será conseguir mantener un SMA en un dispositivo móvil, en nuestro caso una PDA.

Como ya hemos indicado, uno de los puntos que más nos llama la atención de los agentes es su comunicación elaborada, que les permite trabajar correctamente juntos para la resolución de una tarea complicada. Para garantizar que cualquier agente, desarrollado por uno de tantos programadores del mundo, se entienda con otro agente totalmente independiente y desarrollado a su vez por una tercera persona, debemos trabajar con unos mecanismos estándar, que garanticen la interoperabilidad entre los sistemas. Para conseguir esto se necesita una entidad encargada de definir y certificar estos estándares aplicados, que velarán para que diversos SMAs puedan interactuar entre si.

## 1.1. FIPA

Para definir estándares (tanto de arquitectura, lenguajes, protocolos…) se creó la FIPA[1], una fundación sin animo de lucro, con sede en Ginebra (Suiza), nacida con la obligación de establecer las reglas para el diseño e implementación de un SMA para garantizar la interoperabilidad que comentábamos en el apartado anterior. Esta organización, inició su andadura en el 1997 haciendo pública su primera especificación, y desde entonces se ha ido imponiendo hasta conseguir ser el estándar seguido en todo el mundo.

Partiendo de estas especificaciones definidas se han desarrollado todo un conjunto de proyectos que desean explotar las ventajas de los agentes, siguiendo el modelo impuesto por la FIPA [FIPA03]. En el caso de nuestro estudio realizado, nos hemos valido de una herramienta de desarrollo de SMAs, que cumple las especificaciones FIPA., por tanto nos extenderemos en los comentarios sobre estas reglas tanto de diseño como de implementación.

## 1.2. Agentcities

*AgentCities*[2] [AgentCities03] es un ambicioso proyecto europeo que pretende la construcción de una red mundial pública, que ofrezca servicios similares a los de una ciudad real. Las plataformas creadas en este entorno, siguen las especificaciones definidas por la FIPA, lo que garantiza la comunicación entre SMAs creados por diversos desarrolladores.

Los agentes que ya han sido integrados en esta red se centran en ofrecer servicios sobre cines, restaurantes, hoteles, taxis, turismo…en la actualidad podemos encontrar cerca de 50 plataformas activas.

El futuro contempla la creación de servicios complejos que integren un conjunto de funcionalidades simples como las descritas anteriormente, como por ejemplo la planificación de un fin de semana, en cuyo caso sería interesante la participación del asistente personal que venimos comentando y que sirve de punto de inicio para este estudio.

Podemos destacar que el ejemplo práctico que hemos desarrollado estaría encuadrado dentro del proyecto *AgentCities* como gestor del servicio de taxi de una ciudad. Esta basado en un SMA ya creado y que mostró su operatividad y utilidad dentro del entorno AgentCities.

[1] *Foundation for intelligent Physical Agent. Organización impulsora del estándard a nivel mundial para la creación de SMAs*

[2] *Iniciativa europea que pretende ofrecer mediante la utilización de Sistemas Multi-Agente, los servicios que una ciudad realiza.*

## 1.3. Grupo de investigación BANZAI - GruSMA

El grupo de investigación en el campo de la inteligencia artificial llamado BANZAI [Banzai03], forma parte de la Universidad Rovira i Virgili, situada en Tarragona (España). Es uno de los equipos de investigación enmarcados dentro del departamento de ingeniería informática y matemáticas (DEIM) y esta formado por diversos profesores y estudiantes de Doctorado.

BANZAI tiene como una de sus líneas de investigación el diseño de aplicaciones de IA y dentro de este, encontramos el grupo de trabajo *GruSMA*[3] [GRUSMA03] que se encarga de desarrollar diversos proyectos basados en SMAs.

GruSMA proporciona un entorno de trabajo en el cual los alumnos de Informática de la ETSE[4], interesados en las nuevas tecnologías de la Inteligencia Artificial, donde pueden compartir conocimientos y experiencias en el campo de los Sistemas MultiAgente. Dicho grupo trabaja en el desarrollo de sistemas utilizando la herramienta para el desarrollo de SMAs, llamada JADE, la cual cumple con las especificaciones de la FIPA, y por tanto es indicada para crear agentes para el entorno AgentCities. Por todo ello, este documento, además de hacer una introducción a cómo se desarrollan este tipo de sistemas, hará también una introducción a cómo se implementan utilizando esta plataforma.

## 1.4. El ejemplo implementado

El ejemplo tratado se basa en un sistema de gestión de taxis creado preliminarmente por Juan Infante García [Infante02] sobre JADE2.61, ese SMA, finalizado en febrero del 2002, utilizaba un conjunto de agentes, para que partiendo de la petición de un cliente, el cual pedía un taxi para realizar un determinado recorrido, se lanzasen una serie de procesos con el objetivo de generar el mejor trayecto posible de entre los múltiples posibles, este sistema, cada agente en este sistema se encargaba de una tarea especifica: recopilar información de trafico, generar trayectorias, decidir la mejor trayectoria…lo cual nos daba un comportamiento ágil y muy interesante dado su comportamiento distribuido.

Este gestor de taxis, nos pareció idóneo para demostrar las posibilidades de los agentes en sistemas móviles, así que en el caso que presentamos podremos ver una de las primeras aplicaciones de lo que entendemos como asistente personal, sin olvidar que el gestor de taxis no deja de ser un servicio sencillo, que deberá estar integrado en un conjunto mucho mas complejo para ofrecer múltiples servicios al cliente.

La adaptación que hemos realizado para certificar la viabilidad de nuestro estudio, presenta a un cliente que utiliza una PDA para solicitar los taxis dentro de una ciudad dada. Este seleccionará su ubicación actual y su destino seleccionado utilizando una interficie grafica creada específicamente para esta aplicación, y enviará su petición de servicio, que será procesada por el resto de agentes del SMA. Posteriormente el cliente recibirá en su PDA el resultado de la petición con los detalles del servició que recibirá, incluyendo trayecto, precio, tiempo de espera…datos que pueden ser ampliados en posteriores revisiones del SMA sin ningún tipo de problema.

El SMA ha sido implementado según la plataforma JADE 3.0b, la mas actualizada a día de hoy. No obstante, solo con JADE estándar, estaríamos incapacitados para trabajar con agentes en dispositivos tan limitados como PDAs o móviles. Es por ello que hemos aprovechado, la nueva vertiente de trabajo que nos ofrece JADE y que se basa en una plataforma mas sencilla que puede ser ejecutada en entornos con recursos muy limitados. Este nuevo enfoque, que nos libera de las pesadas y poco

*[3] Grupo de Sistemas Multi-Agente de la Universitat Rovira i Virgili.*
*[4] Escola tècnica superior de Enginyeria de la URV.*

manejables estaciones de trabajo, se denomina LEAP [5] y es uno de los llamados add-ons de la plataforma JADE, que permite trabajar con unas librerías menos exigentes en cuanto a recursos se refiere.

## 1.5. Objetivos del proyecto

El objetivo principal de este proyecto era realizar un estudio sobre la operatividad de los agentes sobre dispositivos con recursos limitados, lo que pretendíamos conseguir era tener un SMA funcionando en un entorno distribuido, en el cual, el cliente intervendría utilizando una PDA o un teléfono móvil, en nuestro caso trabajábamos con una PDA, pero tenemos que pensar que el código seria totalmente trasladable de un dispositivo a otro.

Una vez tuviéramos agentes funcionando en la PDA, el siguiente objetivo, era demostrar que podíamos establecer una conexión sin cables entre el dispositivo móvil y la estación o estaciones para permitir la comunicación entre el agente personal del cliente y el resto de los agentes del sistema. Esto era de vital importancia, puesto que el primer principio del asistente personal es la portabilidad, y esta solo puede ser garantizada si no dependemos de cables que entorpezcan nuestra actividad.

Tras estos dos objetivos prioritarios y verdaderos motores del proyecto, venia la creación de un ejemplo que demostrase la viabilidad de forma practica. Para eso recurrimos al gestor de taxis, el cual requería un trabajo exhaustivo para adaptarlo al nuevo entorno, para ello le proporcionamos una interficie para comunicarse con la persona, y realizamos ciertas mejoras en la estructura y funcionamiento partiendo de la base provista por Juan Infante García.

## 1.6. Estructura del documento

Este documento pretende explicar con detalle todo el proceso seguido para conseguir los objetivos propuestos, y la forma en que subsanamos los problemas encontrados durante el tiempo de estudio y implementación. Dado que para entender correctamente el trabajo realizado se necesitan unos conocimientos básicos, dedicaremos los capítulos preliminares ha introducir todos aquellos conceptos que creemos indispensables entender correctamente lo aquí expuesto.

La documentación se estructura de la siguiente forma: en el capítulo 2 se hace una introducción a los conceptos básicos sobre Agentes y los Sistemas Multi-Agentes. En el capítulo 3 se explica la herramienta de desarrollo y programación de Sistemas Multi-Agente utilizada (JADE). En el capitulo 4 nos extenderemos en la aplicación en sistemas móviles del addon LEAP para JADE, haciendo énfasis en el proceso de comunicación entre la estación principal y la PDA o móvil, en nuestro caso el medio elegido ha sido Bluetooth[6], pero se comentarán otras opciones [Bluetooth]. A partir de aquí nos centraremos en el ejemplo realizado para mostrar la operatividad de JADE-LEAP. En el capitulo 5 analizaremos la descripción del sistema. En el capítulo 6 se explica la comunicación que existe entre los diferentes agentes. En el capítulo 7 está dedicado a la explicación de las funciones y las principales características de cada uno de los módulos que forman la implementación del sistema. El capítulo 8 está destinado a la prueba final del sistema. En el capítulo 9 encontramos todo el código fuente del sistema y finalmente destinaremos el capítulo 10 ha una valoración del trabajo realizado, conclusiones y futuros proyectos,

[5] *Lightweight Extensible Agent Platform. Addon para JADE que nos permite tener agentes en dispositivos con fuertes restricciones de recursos como moviles o pdas.*

[6] *Bluetooth es una tecnología orientada a la conectividad inalámbrica entre dispositivos utilizando radiofrecuencia.*

## 1.7. Recursos utilizados

Para finalizar esta introducción, hagamos una breve referencia al material que de una u otra forma ha intervenido en la creación de este proyecto, tanto en su fase de estudio, como en la de implementación:

HARDWARE:

- PDA Fujitsu-Siemens Pocket LOOX 600, dotada de Windows PocketPC 2002
- PC con procesador AMD 900 mhz,256m ram con windows XP
- PC con procesador intel P4 2.6ghz,256m ram con windows XP
- móvil Siemens S55
- Conceptronic Bluetooth Dongle

SOFTWARE:

- J2sdk1.4.1, J2ME de Sun microsystems
- PersonalJava1.1 de Sun microsystems
- Jeode Virtual Machine de Insignia
- JADE (2.5, 3.0b), LEAP (2.1, 3.0) de Telecom Italia Lab
- ActiveSync 3.6 de microsoft
- Plugfree 2.0 para windows de Fujitsu-Siemens
- Pockethost para pocket PC 2002 de Marc Zimmermann
- me4se de Kobjects
- Palm OS Emulator de PalmSource
- Microsoft Embedded Visual Tools

# 2. AGENTES Y SISTEMAS MULTI-AGENTE

Antes de entrar a definir que son los Sistemas Multiagente, hemos de centrarnos en los elementos que lo forman: los AGENTES.

Podemos encontrar diversas definiciones de agente, todas ellas recogidas en el *Desenvolupament de Sistemes MultiAgent en JADE, Report de Recerca* [Isern et al., 2001]. De todos modos, nos quedaremos con la explicación ofrecida por el señor Wooldridge :

> ➢ Un agente inteligente es un proceso computacional capaz de realizar tareas de forma autonoma y que se comunica con otros agentes para resolver problemas mediante cooperación,coordinación y negociación. Habitan en un entorno complejo y dinamico con el cual interaccionan en tiempo real para conseguir un conjunto de objetivos. [Wooldridge02]

## 2.1. Propiedades de los Agentes

Las propiedades indispensables para que un agente sea digno de llevar ese nombre son:

- **Autonomía:** es la capacidad de operar sin la intervención directa de los humanos, y tener algún tipo de control sobre las propias acciones y el estado interno.

- **Sociabilidad/Cooperación:** los agentes han de ser capaces de interactuar con otros agentes a través de algún tipo de lenguaje de comunicación.

- **Reactividad:** los agentes perciben su entorno y responden en un tiempo razonable a los cambios detectados.

- **Pro-actividad o iniciativa:** deben ser capaces de mostrar que pueden tomar la iniciativa en ciertos momentos.

Otras propiedades destacables serian:

- **Movilidad:** posibilidad de moverse a otros entornos a través de una red.

- **Continuidad temporal:** los agentes están continuamente ejecutando procesos.

- **Veracidad:** un agente no comunicará información falsa premeditadamente.

- **Benevolencia:** es la propiedad que indica que un agente no tendrá conflictos, y que cada agente intentará hacer lo que se le pide.

- **Racionalidad:** el agente ha de actuar para conseguir su objetivo.

- **Aprendizaje:** mejoran su comportamiento con el tiempo.

- **Inteligencia:** usan técnicas de IA para resolver los problemas y conseguir sus objetivos.

Los agentes del ejemplo implementado tendrán las propiedades de: Autonomía, Sociabilidad, Continuidad Temporal, Veracidad, Racionalidad e Inteligencia.

## 2.2. Arquitectura de los agentes

Las arquitecturas utilizadas en el proceso de implementación de los agentes pueden ser de tres tipos.

- **Deliberativas:** dada una representación del mundo real, los agentes razonan según el modelo BDI *(Beliefs Desires and Intentions)*.
- **Reactivas:** estructura donde las acciones se llevan a cabo respondiendo a estímulos simples. La combinación de varios agentes reactivos proporcionan un comportamiento inteligente.
- **Híbridas:** estructuras que mezclan diferentes tipos.

Los agentes del SMA implementado siguen el modelo deliberativo, de modo que tienen unos objetivos a cumplir y por eso realizan una serie de acciones y no otras.

## 2.3. Tipos de Agentes

Existen diferentes tipos de agentes, podemos clasificarlos en:

- **Información:** gestionan y manipulan datos. Pueden responder a requisitos del usuario u otros agentes. Un ejemplo muy común son los buscadores de información en Internet.

- **Interficie:** sistema dónde los agentes colaboran con el usuario para responder a un problema. La interacción con el individuo permite al agente desarrollar un aprendizaje basado en las acciones que se desarrollan.

- **Colaboración:** las características principales es la comunicación y cooperación con otros agentes para resolver un problema común. Utilizan técnicas de IA (Inteligencia Artificial) distribuida.

- **Móviles:** su característica principal es la posibilidad de poder moverse por una red electrónica recogiendo información o interactuando con otros hosts.

- **Reactivos:** su comportamiento se basa en la respuesta a estímulos según un patrón del estado actual en que se encuentra. Son muy simples, pero la combinación de diversos agentes reactivos puede generar comportamientos complejos.

- **Híbridos:** son combinaciones de diversos tipos anteriores intentando reunir las ventajas de unos y otros.

- **Heterogéneos:** se refiere a un conjunto integrado de al menos dos agentes que pertenecen a dos o más clases diferentes.

Los agentes creados para el ejemplo serán de tipo *colaborativo*, ya que, sus características principales son las de comunicación y cooperación para resolver un problema común.

## 2.4. Sistemas MultiAgente

Llamaríamos Sistema MultiAgente o MAS a aquel en el que un conjunto de agentes cooperan, coordinan y se comunican para conseguir un objetivo común.

### 2.4.1. Ventajas de un sistema MultiAgente

Las principales ventajas de la utilización de un sistema MultiAgente son:

– Modularidad: se reduce la complejidad de trabajar con unidades más pequeñas, permiten una programación más estructurada
– Eficiencia: la programación distribuida permite repartir las tareas entre los agentes, consiguiendo paralelismo (agentes trabajando en diferentes máquinas).
– Fiabilidad: el hecho de que un elemento del sistema deje de funcionar no tiene que significar que el resto también lo hagan, además, se puede conseguir más seguridad replicando servicios críticos y así conseguir redundancia.
– Flexibilidad: se pueden añadir y eliminar agentes dinámicamente.

### 2.4.2. Gestión de un sistema MultiAgente (SMA)

La administración de agentes establece el modelo lógico para la creación, registro, comunicación, movilidad y destrucción de agentes. En este proyecto vamos a seguir los estándares establecidos por la FIPA (*Foundation for Intelligent Physical Agents*, organización que promueve el desarrollo de aplicaciones basadas en agentes ). En la figura 1 se puede ver el modelo para este entorno de administración de agentes:

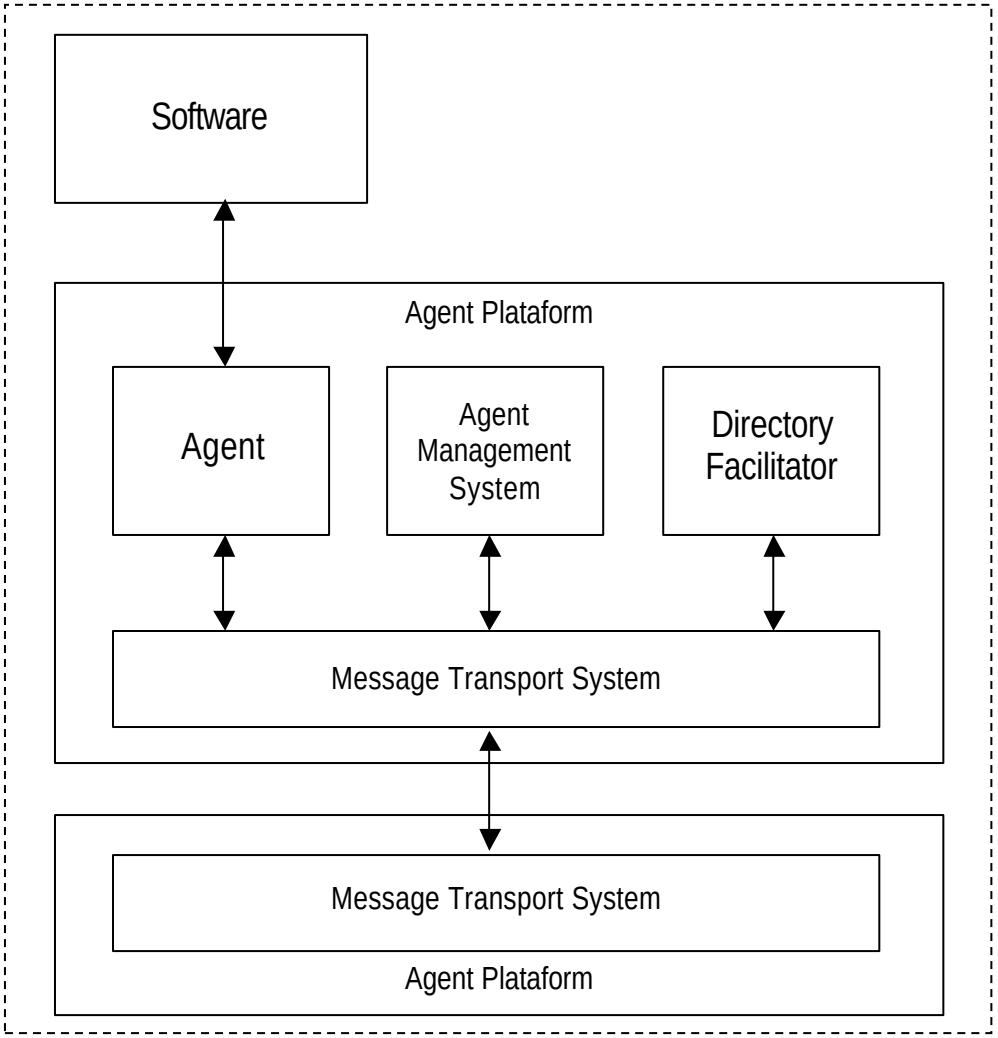



Fig. 1 – Interior de un Sistema MultiAgente

Los componentes que forman parte de la *figura 1* son:

– Agente: unidad básica. Se podría definir como un programa que contiene y ofrece una serie de servicios.
– Directory Facilitator (DF): agente que proporciona un servicio de *páginas amarillas* dentro del sistema, es decir, conoce los diferentes servicios que el resto de agentes de la plataforma ofrecen. Los agentes se han de registrar en el DF para ofrecer sus servicios.
– Agent Management System (AMS): es el agente que controla el acceso y uso de la plataforma. Almacena las direcciones de los agentes, ofreciendo un servicio de *páginas blancas*.
– Message Transport System (MTS): facilita la comunicación entre los agentes de diferentes plataformas.
– Agent Platform (AP): proporciona la infraestructura básica para crear y ejecutar agentes.
– Software: cualquier programa accesible desde un agente.

### 2.4.3. Lenguaje de comunicación entre agentes (ACL)

Los agentes individualmente proporcionan una funcionalidad interesante, pero lo que les hace tan adecuados para ciertos sistemas es su capacidad de cooperar para resolver problemas. Para poder hacerlo, los agentes se han de comunicar entre si, utilizando un lenguaje común: ACL *(Agent Comunication Language)*. Para garantizar la homogeneidad y compatibilidad entre los diversos agentes, la FIPA determina que forma ha de tener un mensaje[7] y su utilización, para ello esta organización elabora las *FIPA Specifications*[8] [specs03].

#### 2.4.3.1. Elementos de un mensaje

Todo mensaje estará compuesto por una serie de campos (o slots) que son los siguientes:

| Slot | Categoría que pertenece |
|---|---|
| Performative | Definición del tipo de mensaje |
| Sender | Participante de la comunicación |
| Receiver | Participante de la comunicación |
| Reply-to | Participante de la comunicación |
| Content | Contenido |
| Language | Descripción del contenido |
| Encoding | Descripción del contenido |
| Ontology | Descripción del contenido |
| Protocol | Control de la conversación |
| Conversation-id | Control de la conversación |
| Reply-with | Control de la conversación |
| In-reply-to | Control de la conversación |
| Reply-by | Control de la conversación |

[7] *Unidad básica de comunicación según la FIPA.*
[8] *Especificaciones presentadas por la FIPA, que forman el estándar para la creacion de agentes i SMAs*

- Definición del tipo de mensaje: indica que tipo de comunicación deseamos hacer.

o **accept-proposal:** aceptamos una propuesta hecha anteriormente para realizar una acción
o **agree:** aceptación de una acción
o **cancel:** cancelación de una acción
o **cfp:** para proponer una acción
o **confirm:** el emisor informa al receptor que una proposición es cierta, cuando el receptor dudaba
o **disconfirm:**el emisor informa al receptor que una proposición es falsa, cuando el receptor pensaba que era cierta.
o **Failure:** indica que la acción pedida anteriormente ha fallado por algún motivo.
o **Inform:** el emisor informa al receptor que una acción se ha realizado correctamente.
o **Not-understood:** el emisor informa al receptor que no ha entendido una petición realizada anteriormente.
o **Propose:** acción de proponer la realización de una acción, dadas ciertas precondiciones.
o **Query-if:** acción de preguntar a otro agente si un hecho es cierto o no.
o **Refuse:** negación a la realización de una acción dada.
o **Request:** el emisor pide al receptor la realización de una acción.
o **Subscribe:** el emisor pide ser avisado en el momento que se cumpla una condición.

- Participante de la comunicación: son los identificadores de los agentes. Hay 3: **sender** (quien envía el mensaje), **receiver** (quien lo recibe), **reply-to** (a quien tiene que ir destinado el siguiente mensaje de la conversación).

- Contenido: existen cuatro tipos de contenidos predefinidos que se utilizan en función de las necesidades que se tengan.

o **FIPA-CCL *(Constrant Choice Language)*:** define una semántica que permite especificar predicados con restricciones.
o **FIPA-SL *(Semantic Language)*:** permite formar expresiones lógicas, intercambiar conocimientos de forma óptima y expresar acciones a realizar. Es el lenguaje más general de todos y puede ser aplicado a muchos dominios diferentes. Es el que ha sido utilizado en el SMA creado.
o **FIPA-KIF *(Knowledge Interchange Format)*:** permite expresar objetos como términos y proposiciones como sentencias.
o **FIPA-RDF *(Resource Description Framework)*:** permite expresar objetos, interacciones y acciones entre ellos.

- Descripción del contenido: permite que el agente que reciba el mensaje pueda identificar qué se le está enviando y en qué formato. Se definen 3 descriptores:

o **Language:** en qué está escrito el contenido, nosotros hemos utilizado FIPA-SL.
o **Enconding:** indica si se utiliza algún tipo de codificación especial.
o **Ontology:** es el campo más importante, ya que, define la ontología utilizada para dar significado al contendido del mensaje.

- Control de la conversación: identifica el tipo de conversaciones que se mantienen. Sus campos son:

o **Protocolo** utilizado en el mensaje.
o **Conversation-id**: identificador de la conversación.
o **Reply-with**: identificador del mensaje que se utiliza para seguir los diferentes pasos de la conversación.
o **In-reply-to**: identifica una acción pasada a la cual este mensaje es la respuesta.
o **Reply-by**: marca de *time-out*: fecha/hora de caducidad del mensaje.

#### 2.4.3.2. Protocolos de comunicación

Como hemos indicado anteriormente en *control de la conversación*, existe un campo que identifica el protocolo utilizado. Dicho protocolo es el que define una serie de reglas o pasos que se ha de seguir para desarrollar una conversación. Existen diferentes tipos dependiendo de su finalidad:

- FIPA Request: permite pedir la realización de acciones y devolver los resultados
- FIPA Query: se utiliza para hacer preguntas.
- FIPA Contract Net: es un protocolo de negociación (se proponen y se evalúan propuestas).
- FIPA Iterated Contract Net: variante de la anterior que permite la renegociación.
- FIPA Brokering: gestiona los servicios disponibles para los agentes.
- FIPA Recruiting: variante de la anterior.
- FIPA Subscribe: permite la notificación de hechos importantes.
- FIPA Propose: simplificación del contract net.

En este proyecto hemos usado los protocolos REQUEST y CONTRACT-NET.

# 3. JADE

JADE[9] (Java Agent Developement Enviroment) es una herramienta de programación que contiene un conjunto de librerías escritas en JAVA [Java03] para el desarrollo de Sistemas Multiagente [Java]. Además de proporcionar las funciones de manejo de agentes a bajo nivel y las interficies que facilitan la programación, también nos presenta un entorno de ejecución donde los agentes podrán ejecutarse e interactuar [Isern et al., 2001].

Para la realización del proyecto hemos trabajado con la ultima versión disponible a día de hoy, la 3.0b. Esta distribución aunque en estado de beta, es ciertamente muy estable, y no ha presentado ningún inconveniente a lo largo de la realización del trabajo. Una de sus mayores ventajas reside en la integración de LEAP[10] como *addon* para JADE, lo que resuelve gran parte de problemas encontrados en su antecesor: JADE2.5 con LEAP2.1 [JADE03].

Las diferencias en las librerías de JADE2.61 y JADE3.0b son mínimas, aun así destinaremos un apartado a enumerarlas, para así facilitar la promoción del código de una versión a otra. Cabe destacar que todo lo comentado aquí, puede ser profundizado leyendo el *JADE Programmer's guide*, que encontraremos junto a mucha mas información en la página oficial de JADE [guide03].

## 3.1. Paquetes de JADE

JADE está compuesto por una serie de paquetes en Java. Los principales son los siguientes:

- ***JADE.core*** es el núcleo del sistema. Proporciona la clase JADE.core.Agent que es imprescindible para la creación de SMA. También contiene el paquete JADE.core.behaviour que incluye las clases de los comportamientos que ha de tener todo agente.

- ***JADE.content*** contiene las clases específicas para soportar un lenguaje de comunicación entre agentes (ACL). Está formado por los paquetes JADE.content.lang.acl y JADE.content.lang.sl. En éste mismo encontramos todas las clases necesarias para la realización e implementación de la ontología (JADE.content.onto).

- ***JADE.domain*** contiene todas las clases para representar la plataforma de agentes y los modelos del dominio, tales como entidades para la administración de agentes, lenguajes y ontologías (AMS, DF, ACC…).

- ***JADE.proto*** paquete que contiene los protocolos de interacción entre agentes FIPA.

- ***JADE.tools*** encontramos algunas herramientas que facilitan el desarrollo de aplicaciones y la administración de la plataforma:

  - *Remote Management Agent* (RMA): es el agente que se encarga de la interficie gráfica de JADE para la administración y control del sistema.
  - *Dummy Agent*: es una herramienta de monotorización y depuración, compuesta por una interficie gráfica y un agente JADE. Permite la creación de mensajes ACL y enviarlos a otros agentes pudiendo visualizarlos.
  - *Sniffer*: agente que intercepta mensajes ACL y los visualiza siendo muy útil para el seguimiento de una conversación.

[9] *Java Agent Developement Enviroment*
[10] *Lightweight Extensible Agent Platform. Addon para JADE que nos permite tener agentes en dispositivos con fuertes restricciones de recursos como moviles o pdas.*

## 3.2. Componentes principales de JADE

Para JADE, los agentes residen en un entorno predefinido llamado plataforma. Cada plataforma está dividida en contenedores y cada uno de ellos contendrá agentes. Los contenedores podrían asemejarse al dominio de los agentes.

La plataforma se enciende en una máquina determinada, mientras que los agentes podrían ejecutarse en cualquier estación, esto nos ofrece un comportamiento distribuido de el SMA que será la base para la incorporación del LEAP.

Tal como se define en la FIPA, en cada plataforma se inician dos agentes que tienen unas funciones vitales para el funcionamiento del sistema:

1. *DOMAIN FACILITATOR (DF):* podría definirse como un servicio de "páginas amarillas", es decir, contendrá todos los servicios que proporcionan los agentes que se han dado de alta en la plataforma. Será de especial utilidad a la hora de requerir algún servicio, puesto que nos informará de quien puede proporcionárnoslo.

2. *AGENT MANAGEMENT SYSTEM (AMS):* se podría considerar como un servicio de "páginas blancas", donde se registran las direcciones de los agentes que se han dado de alta. Dado que JADE se fundamenta en entornos distribuidos, será básico para conocer la localización de los agentes.

Gracias a estos agentes, la comunicación en la plataforma es muy sencilla de realizar y permite una gran flexibilidad y transparencia. Por ejemplo, si un agente A quiere enviar un mensaje a un agente B que desarrolla la tarea T, el agente A preguntará al DF qué agentes me pueden proporcionar dicho servicio T. Cuando se quiera conocer la dirección física de un agente determinado, preguntará al AMS. Para que este sistema funcione correctamente, será necesario que, durante el proceso de inicialización del agente, éste informe al DF de qué servicios dispone y el tipo de agente qué es (para poder identificarlo), al hacerlo, AMS le dará una dirección física (dependiendo de dónde se ejecute), para que otro se puedan poner en contacto con él.

## 3.3. Agentes y comportamientos (Behaviours)

Para poder crear agentes en JADE, deberemos extender la clase **JADE.core.Agent** que ya está definida y deberemos rellenar los campos y las funciones básicas de las cuales nos proporciona la interficie. Dos métodos muy importantes que se han de implementar son **setup()** y **takedown()** que inicializan y finalizan el agente respectivamente [JADE Api].

Como inciso de interés comentaremos la existencia de la clase **JADE.gui.GuiAgent**, una extensión de la clase agente estándar, creada para facilitar la implementación de los agentes con GUI[11], que no ha sido utilizada en este proyecto dado que se fundamenta en la librería grafica *swing* y nosotros, debido a las limitaciones impuestas por LEAP, hemos utilizado la librería *awt*.

Una vez hayamos creado el agente deberemos darle funcionalidad. Eso se consigue a través de la definición de los ***comportamientos (behaviours)*** que son los que controlan sus acciones. Los behaviours son los encargados de darle dinamismo al agente y que reaccione a estímulos (mensajes) externos.

[11] *Graphic user interface*

La figura 2 muestra la estructura de un agente:

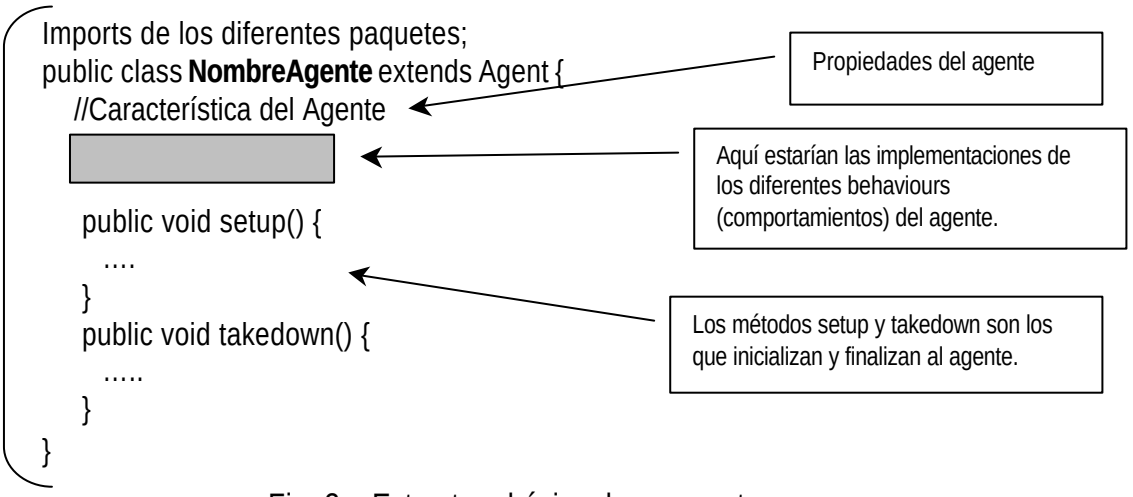



Fig. 2 – Estructura básica de un agente

Ahora vamos a profundizar un poco más en el tema de los Behaviours. JADE nos proporciona un conjunto de comportamientos básicos que permiten implementar diferentes protocolos de comunicación: *SIMPLES* y *COMPUESTOS*.

### 3.3.1. Comportamientos SIMPLES

Son los comportamientos que no pueden tener hijos. Estos comportamientos derivan de la clase *JADE.core.behaviours.Behaviour*. Hay una serie de métodos que se pueden sobrecargar, como por ejemplo: **action()** (que engloba el conjunto de sentencias a ejecutar), **done()** (que devuelve un boolean que nos indica si el comportamiento ha finalizado) y el **reset()** (que reinicia el comportamiento). Hay tres clases de comportamientos simples:

- **SimpleBehaviour:** representa un comportamiento atómico que se ejecuta sin interrupciones. Podemos diferenciar entre:
  - OneShotBehaviour: solo se ejecuta una vez.
  - CyclicBehaviour: el comportamiento se ejecuta cíclicamente.

- **ReceiverBehaviour:** comportamiento que espera la recepción de un mensaje.

- **SenderBehaviour:** es equivalente al anterior pero desde el punto de vista de quien envía el mensaje.

### 3.3.2. Comportamientos COMPUESTOS

Son los comportamientos que pueden tener hijos, es decir, podemos asociar uno o más *subbehaviours* a un comportamiento complejo, que serán gestionados por un ***scheduler*** independiente del agente. Estos comportamientos derivan de la clase *JADE.core.behviours.ComplexBehaviour*.
Los métodos que se pueden sobrecargar son **onStart**() i **onEnd**(). El primero en ejecutarse será el **onStart**(), que será dónde añadiremos los subbehaviours. Cuando finaliza la ejecución de este método

se lanzarán todos los subbehaviours, y cuando todos hayan acabado se ejecutará el método **onEnd**(). Cabe destacar que en la antigua versión 2.61 estos métodos eran llamados *preAction()* y *postAction()* respectivamente, apunte que será de interés a aquellos mas curtidos en la versión antigua.

Hay dos tipos de comportamientos complejos:

- SequentialBehaviour: ejecuta los hijos de manera secuencial.
- ParallelBehaviour: ejecuta los hijos según una política de Round Robin.

Como apunte de interés, podemos indicar que el *ParallelBehaviour* viene a sustituir al antiguo *NonDeterministicBehaviour*.

A continuación se muestra un ejemplo muy sencillo de la implementación y la utilización de un comportamiento simple (OneShotBehaviour):

```
import JADE.core.*;
import JADE.core.behaviours.*;

public class AgentePrueba extends Agent {

      class PruebaBehaviour extends OneShotBehaviour {
          String texto;
          public EnviarPeticionBehaviour(Agent a,String  texto) {
                    super(a);
                    this.texto=texto;
          }
          public void action() {
                    System.out.println(texto);
          }
      }

      public void setup() {
          Behaviour b = new PruebaBehaviour(this, “prueba SIMPLE”);
          addBehaviour(b));
      }
}
```

## 3.4. Mensajes

JADE también sigue las especificaciones de la FIPA en cuanto al formato de los mensajes, y nos proporciona la clase JADE.lang.acl.ACLMessage para su creación y utilización. Los mensajes se dividen en *slots* (ver apartado 2.4.3.1) y sobre éstos, se definen funciones de consulta **(get)** y escritura **(set)**.

Los métodos básicos de qué disponemos para el uso de los mensajes entre agentes son:

- send (ACLMessage): utilizado para enviar mensajes un vez se hayan rellenado todos sus parámetros.
- ACLMessage receive(): recibe un mensaje de la cola del agente. Se puede especificar qué tipo de mensaje se quiere recibir a través de los patrones de búsqueda *(MessageTemplate)*.

El lenguaje que utilizaremos en nuestro caso para escribir los mensajes será el FIPA-SL *(Semantic Language)*, que es el que tiene un ámbito más general. Permite formar expresiones lógicas, intercambiar información entre agentes y expresar acciones a realizar. La notación utilizada es parecida al Lisp: se define un conjunto de predicados genéricos y cada uno con una serie de atributos y parámetros.

## 3.5. Ontología

JADE también nos permite crear ontologías. Éstas son muy importantes, ya que, definen el vocabulario utilizado en nuestro SMA. Es por eso que antes de definir los agentes, hay que diseñar la ontología que se adapte a nuestras necesidades.
El elemento básico de una ontología es el **frame**. Éste a su vez se divide en **slots**, que pueden ser de diferentes tipos: SIMPLES (integer, string, boolean...) o OTROS FRAMES.
Para la creación de una ontología completa se han de tener 3 elementos en cuenta:

1. CONCEPTOS o OBJETOS (frames): define los elementos básicos de la ontología.
2. PREDICADOS: define una serie de propiedades o características que pueden cumplir los objetos. Ejemplo: “edad mayor de 18”, “que vivan en la ciudad de TGN”, …
3. ACCIONES: define las acciones que se podrán pedir a los agentes del sistema.

En la versión 3.0b de JADE la implementación de la ontología es prácticamente igual al utilizado en la versión 2.61, se han eliminado los viejos contenidos a los que aun se daba soporte, pero en general las ontologías que cumplan las condiciones de JADE2.61 deberían funcionar perfectamente en su versión 3.0b, de todas maneras mas adelante haremos especial referencia a las diferencias generales entre ambas versiones, comentando la parte correspondiente a ontologías. De todas formas, no esta de menos comentar que estas dos versiones presentan fuertes diferencias respecto a las versiones anteriores. Ahora la realización de dicha tarea es más sencilla y eficiente. Las clases están incluidas en el package *JADE.content*. A continuación se muestran los pasos más significativos en la realización de la ontología:

1. Los nuevos packages que se han de importar son:

```
Import JADE.content.onto.*;
Import JADE.content.schema.*;
```

2. Método de inicialización

```
private static Ontology theInstance = new MiOntología();
public static Ontology instance() { return theInstance; }
private MiOntología()
{
        super(NAME, BasicOntology.getInstance());
        …
}
```

3. La especificación de CONCEPTOS, PREDICADOS Y ACCIONES

**CONCEPTOS**

```
add(new ConceptSchema(NAME), Name.class);
```

**PREDICADOS**
add(new PredicateSchema(NAME), Name.class);
**ACCIONES**
add(new AgentActionSchema(NAME), Name.class);

4. La definición de SLOTS

**CONCEPTOS**
ConceptSchema cs= (ConceptSchema)getSchema(NAME);
cs.add(SLOT_NAME,type,cardinality,optionality);
**PREDICADOS**
PredicateSchema ps= (PredicateSchema)getSchema(NAME);
ps.add(SLOT_NAME,type,cardinality,optionality);
**ACCIONES**
AgentActionSchema as= (AgentActionSchema)getSchema(NAME);
as.add(SLOT_NAME,type,cardinality,optionality);

**PARÁMETROS DE LA FUNCIÓN ADD**

1. SLOT_NAME: string que indica el nombre del Slot
2. TYPE: indica el tipo del slot. (integer, string,boolean,float, date, byte_seqüence, aid, y también otro CONCEPTO creado por nosotros).
3. CARDINALITY: es un parámetro opcional e indica el número mínimo y máximo de elementos.
4. OPTIONALITY: también es opcional e indica si el slot puede contener el valor NULL (OPTIONAL) o no (MANDATORY).

5. Clases java asociadas a cada elemento (CONCEPT,PREDICATE, ACTION)

Public class CONCEPT_NAME implements Concept { . . . }
Public class PREDICATE_NAME implements Predicate { . . . }
Public class ACTION_NAME implements AgentAction { . . . }

Para cada slot de la clase, tendremos que definir una propiedad y su correspondiente método SET y GET.

## 3.6. Protocolos de comunicación

Una vez definida e implementada la ontología, deberemos escoger el protocolo de comunicación entre agentes que más se adapte a nuestras necesidades. Para cada conversación, JADE distingue dos roles:

- El que inicia la conversación: INITIATOR
- El que la sigue: RESPONDER

JADE 3.0b en su condición de plataforma que cumple las especificaciones de la FIPA, nos proporciona los elementos necesarios para definir los protocolos descritos por esta organización. Los protocolos entre los que podemos elegir son: FIPA-Request, FIPA-Propose, FIPA-Query, FIPA-Request-

When, FIPA-Recruiting, FIPA-Brokering y FIPA-Contract-Net.

Las nuevas clases disponibles en *JADE.proto* package son:

- **AchiveREInitiator:** reemplaza a la clase FIPAXXXinitiatorBehaviour (dónde XXX es el nombre del protocolo) que ofrece la funcionalidad al role de INITIATOR de la comunicación.

- **AchiveREResponder:** reemplaza a la clase FIPAXXXResponderBehaviour (dónde XXX es el nombre del protocolo) que ofrece la funcionalidad al role de RESPONDER de la comunicación.

Algunos de los métodos de las versiones antiguas han sido eliminados o han sido reemplazados por otras funciones que no son compatibles (principalmente en el rol del responder).

Algunos de los cambios más importantes son:

1. Con los protocolos en los que se piden acciones (ex. FIPA-Request), el mensaje que envía el receptor de AGREE, yo no es obligatorio y se puede enviar directamente el resultado de la acción.
2. Hay métodos que recopilan todas las propuestas (*handleAllProposes*) para el Initiator y todas los resultados finales (*handleAllResultNotifications*) para el responder.
3. Con las nuevas clases, es posible registrar la llegada de algunos tipos de mensajes de un usuario definido en el behaviour.

Destacamos que estos cambios solo serán aplicables a las versiones anteriores a JADE2.61, puesto que en este tema la versión 3.0b i la 2.61 son prácticamente exactas.

## 3.7. Tipos de protocolos

Aunque como hemos indicado, la variedad de protocolos que se pueden implementar es importante, para la realización de este proyecto sólo hemos utilizado *FIPA-Request* y *FIPA-Contract-Net*.

A continuación explicaremos con más detalle los dos protocolos utilizados.

### 3.7.1. FIPA REQUEST

Se utiliza para pedir la realización de una acción. El comportamiento que tendrán los agentes que utilicen este protocolo será el siguiente:

- El agente *iniciador* envía un mensaje de tipo request al agente *receptor* para que realice una acción en concreto.

- El receptor del mensaje lee el contenido del mensaje y obtiene la acción. El agente estudiará la petición y determinará si la puede hacer o no. Si la puede hacer, enviará un mensaje de **agree** al iniciador y empezará a hacer la acción. Si por alguna razón no puede realizarla, enviará un mensaje de **refuse**.

- En cualquier caso, si el iniciador no ha formulado bien el mensaje y, por tanto, no se puede descifrar correctamente, se enviará un mensaje de **not-understood**.

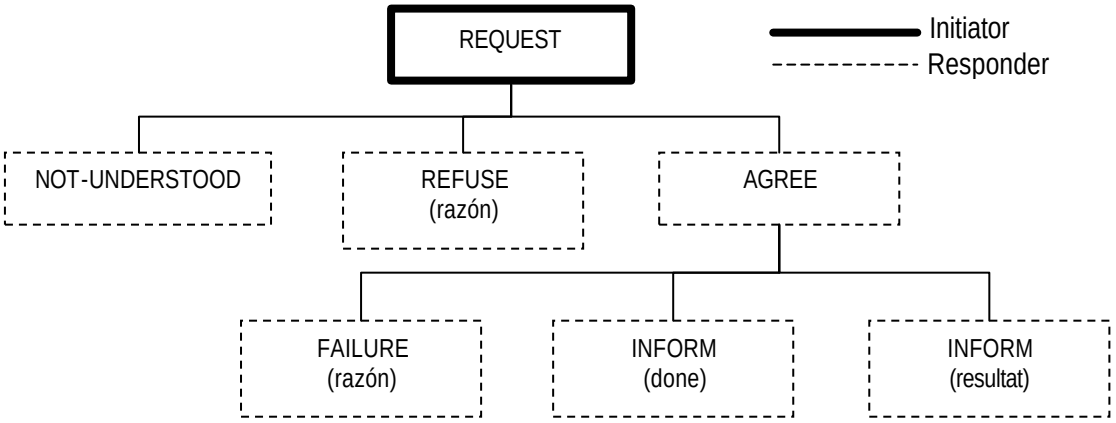



Fig. 3 – Protocolo FIPA Request

- En el caso de haber enviado un mensaje de **agree**, el receptor habrá comenzado a realizar la acción. Cuando acabe, enviará un mensaje del tipo **inform**. Este puede ser de dos tipos: **inform-done** que solo notifica que la acción ha sido finalizada e **inform-ref**, que devuelve un resultado. Si la acción no se ha finalizado correctamente, se enviará un mensaje de **failure** indicando la razón.

A continuación les mostraremos la estructura de los métodos que se han de implementar en la nueva versión del JADE para la utilización del protocolo FIPA Request.

**INITIATOR**

```
Class RequestInitiator extends [Simple]AchieveREInitiator
{
     public RequestInitiator (Agent myAgent, ACLMessage requestMsg)
     {
          Super(myAgent, requestMsg);
     }
     protected void handleAgree(ACLMessage msg) { … }
     protected void handleInform(ACLMessage msg) { … }
     protected void handleNotUnderstood(ACLMessage msg) { … }
     protected void handleFailure(ACLMessage msg) { … }
     protected void handleRefuse(ACLMessage msg) { … }
}
```

**RESPONDER**

```
Class RequestResponder extends [Simple]AchieveREResponder
{
     public RequestResponder (Agent myAgent, MessageTemplate mt)
     {
          Super(myAgent, mt);
     }
     protected ACLMessage prepareResponse(ACLMessage request) { … }
     protected ACLMessage prepareResultNotification (ACLMessage
     request, ACLMessage response) { … }
}
```

**NOTA:** Con la implementación del protocolo no es necesario hacer el *send(msg),* ya que, el protocolo lo hace automáticamente.

### 3.7.2. FIPA CONTRACT-NET

Este protocolo se utiliza cuando queremos realizar una negociación entre diversos agentes para encontrar el que la puede realizar mejor según nuestros intereses. Los pasos que se han de segurir son los siguientes:

- El agente *iniciador* envía una petición de propuesta de acción.

- El agente *receptor* evaluará la acción recibida y, en función de cual sea la acción y unas premisas (que también se pueden enviar) propondrá realizar o no la acción. En el primer caso, enviará una propuesta de acción con unas condiciones determinadas, en el segundo, un mensaje de rechazo. Con la nueva versión del JADE, para saber cuando hemos recibido todas las propuestas existe un método **handleAllResponses** que se ejecuta cuando las ha recibido todas, o cuando el *time-out* (que se sabe ha través del campo *in-reply-by*) ha expirado.

- Si se ha enviado una propuesta, el agente *iniciador* la evaluará, y podrá o no aceptarla enviando al agente *receptor* el mensaje correspondiente.

- Finalmente, el agente *receptor* comenzará a realizar la acción, enviando un mensaje de finalización (**inform**) o un fallo (**failure**) en el caso de que se haya producido un error.

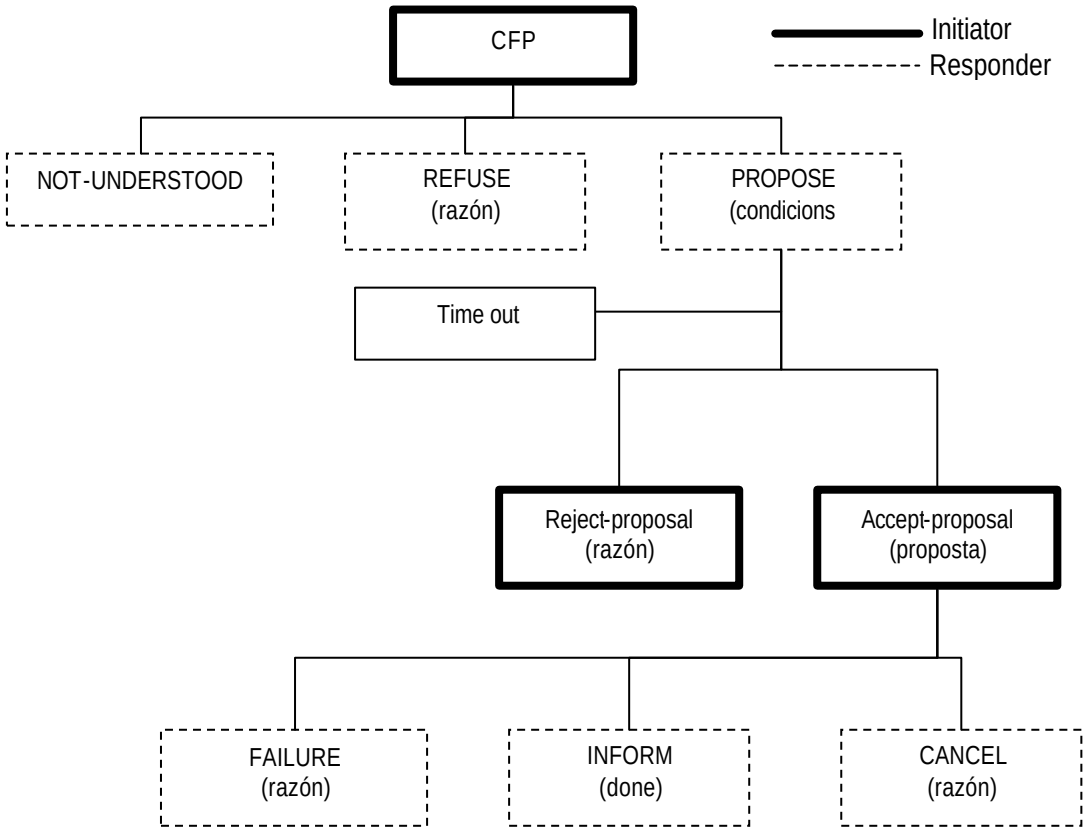



Fig. 4 – Protocolo FIPA Contract-Net

A continuación les mostraremos la estructura de los métodos que se han de implementar en la nueva versión del JADE para la utilización del protocolo FIPA Request.

**INITIATOR**

```
Class CNInitiator extends ContractNetInitiator
{
     public CNInitiator (Agent myAgent, ACLMessage CfpMsg)
     {
          Super(myAgent, CfpMsg);
     }

     protected vector prepareCfps(ACLMessage msg) { … }
     protected void handlePropose(ACLMessage msg, Vector acceptances)
     { … }
     protected void handleAllResponses(Vector proposes,Vector
     acceptances) { … }
     protected void handleNotUnderstood(ACLMessage msg) { … }
     protected void handleInform(ACLMessage msg) { … }
     protected void handleFailure(ACLMessage msg) { … }
}
```

Hagamos una somera explicación de los tres métodos en cursiva del cuadro anterior:

- **prepareCfps:** este método se utiliza para enviar bs mensajes a los destinatarios solicitados. Es lo primero que se ejecuta de la comunicación CFP. Recibe el mensaje que ha sido especificado en el constructor y devuelve un vector con la lista de mensajes que serán enviados a cada receptor, es decir, primero hemos de guardar el mensaje (sin receptores) en el vector y posteriormente ir añadiendo receptores al mensaje. El método automáticamente sabe qué es lo que debe hacer y envía el mensaje a cada uno de los agentes responder. Para verlo más claro les mostraremos un ejemplo.

```
Vector v = new Vector();
v.add(CfpMsg);
while (HayanReceptores()) {
    CfpMsg.addReceiver(AID_agentes_receptores);
}
return v;
```

- **handlePropose:** método que se ejecuta cada vez que recibe una petición del agente receptor.

- **handleAllResponses:** método que se ejecuta cuando recibe todas las peticiones de los agentes receptores. También se ejecuta cuando finaliza el tiempo máximo.

**RESPONDER**

```
Class CNResponder extends ContractNetResponder
{
     public CNResponder (Agent myAgent, MessageTemplate mt)
     {
          Super(myAgent, mt);
     }
     protected ACLMessage prepareResponse(ACLMessage request) { … }
     protected ACLMessage prepareResultNotification (ACLMessage
     cfp, ACLMessage propose, ACLMessage accept) { … }
     protected void handleRejectProposal (ACLMessage cfp, ACLMessage
     propose, ACLMessage reject) { … }
}
```

**NOTA:** como pasaba con el protocolo Request, no hay que hacer el send(msg), ya que, el protocolo automáticamente lo realiza automáticamente cuando haces un *return.*

## 3.8. Herramientas de JADE

JADE ofrece una interficie gráfica para la administración de la plataforma, así como herramientas para facilitar la depuración y testeo de las aplicaciones.

Para iniciar una sesión y crear los agentes, deberemos escribir la siguiente línea de comandos:

*java JADE.Boot [options] –platform [llista d'agents]*

En el campo de opciones podemos especificar que deseamos visualizar la interficie gráfica de JADE utilizando el parámetro *–gui*.

Para la lista de agentes debemos colocar una cadena de texto donde cada uno de ellos estará separado espacios en blanco y tendrán el siguiente formato: *<NombreAgente>:ClaseAgenteJava* , en el caso que las clases las tengamos dentro de un package deberemos colocar la siguiente línea de texto: *<NombreAgente>:PackageClase.ClaseAgenteJava*. Cabe destacar, que este formato solo es valido para JADE en estado puro. Si utilizásemos JADE con LEAP, como de hecho haremos en el ejemplo final, el formato variaría ligeramente: cada agente seria separado por punto y coma (;), y tampoco podemos olvidar que en caso que las clases requieran parámetros para su ejecución, estos serán introducidos usando paréntesis al final de cada clase, veamos un rápido ejemplo:

*<NombreAgente>:Package.ClaseAgente(param); <NombreAgente2>:Package.ClaseAgente(p1,p2)*

De todas formas, existe la posibilidad de iniciar el entorno gráfico de JADE sin ningún agente y posteriormente con las herramientas que te ofrece ir cargando los agentes que se deseen y pudiendo visualizar su comportamiento, nosotros no obstante, por comodidad, los cargaremos todos de inicio, utilizando además un fichero BAT, que agilizará el procedimiento.

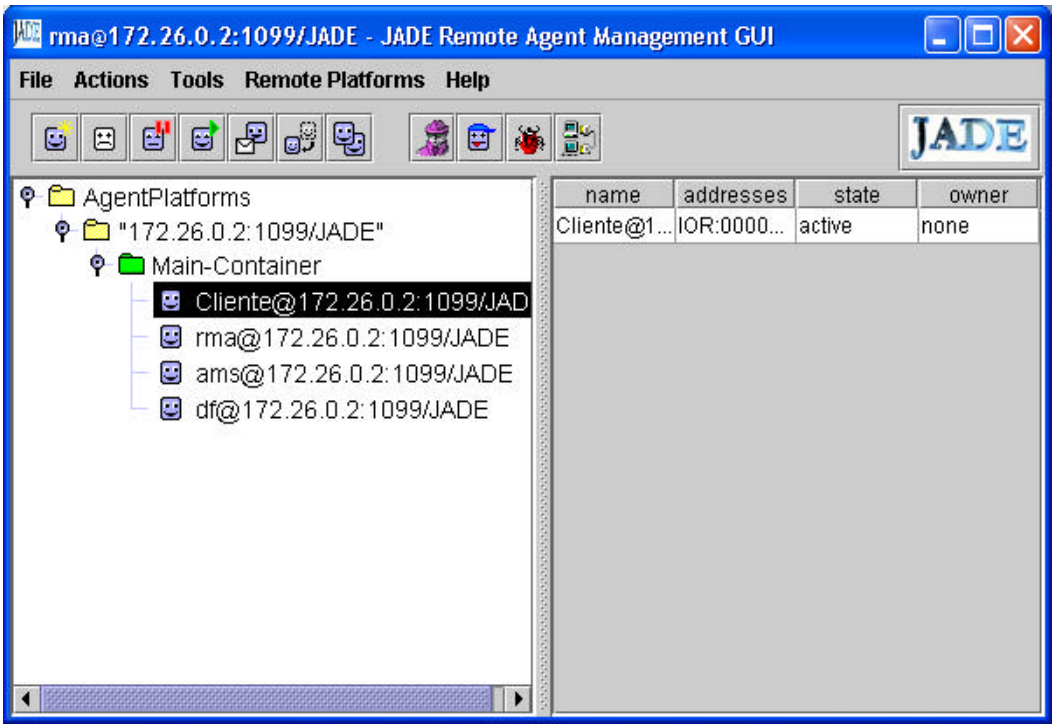



Fig. 5 – Pantalla principal de JADE

Cuando iniciamos el entorno gráfico, tal como se muestra en la figura 5, aunque no inicialicemos ningún agente, se crean algunos automáticamente, que son los agentes básicos de la plataforma, encargados de que todo funcione correctamente. Aunque ya han sido comentados en apartados anteriores, refresquemos un poco la memoria, para seguir correctamente el hilo de la explicación:

- *Agente RMA*: será el encargado de controlar la interficie gráfica de JADE.
- *Agente DF*: donde se subscribirán los servicios de los agentes. Es muy importante, ya que, se utiliza para buscar agentes con unas características indicadas (páginas amarillas).
- *Agente AMS*: donde se guardan las direcciones de los agentes de la plataforma.

Además de la lista de agentes que hay en el contenedor y la información sobre el que hemos seleccionado, disponemos de una serie de opciones que nos facilitarán el manejo y testeo de los agentes:

- Crear un nuevo agente: deberemos indicar el nombre, la clase que lo implementa
- Eliminar los agentes seleccionados
- Ejecutar los agentes seleccionados
- Paralizar los agentes seleccionados
- Continuar los agentes que estaban paralizados
- Enviar un mensaje a los agentes seleccionados

En la figura 6 se muestra la ventana que se ha de rellenar para el envío de un mensaje ACL con todos sus correspondientes campos.

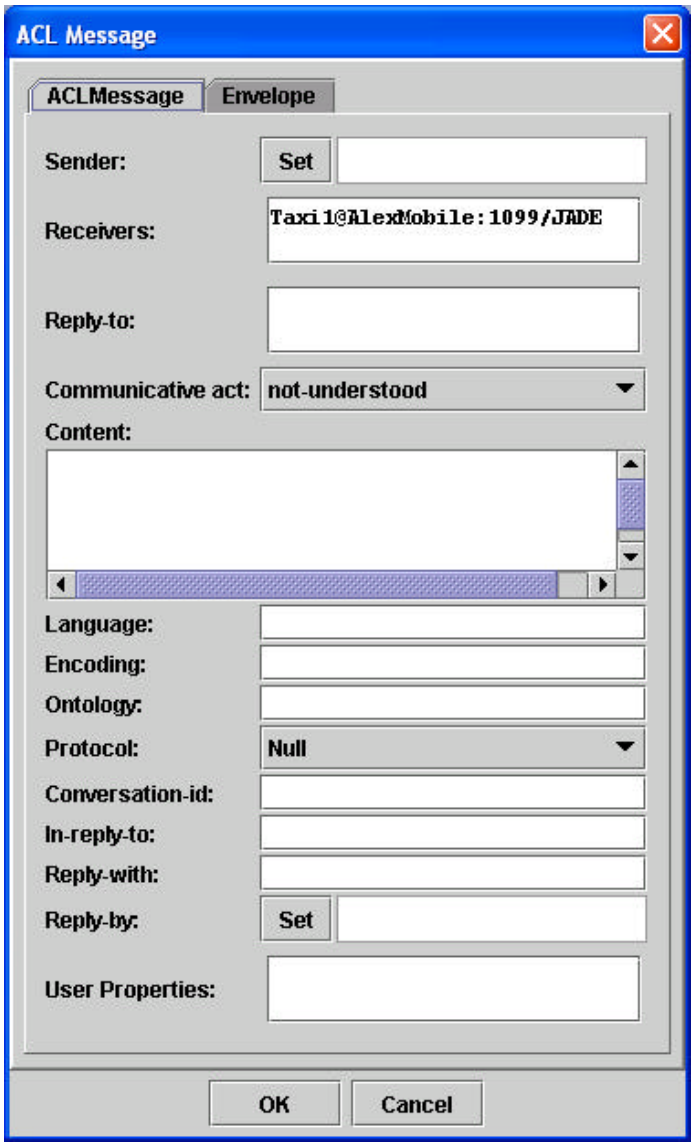



Fig. 6 – Ventana para envío de mensajes

**AGENTE SNIFFER:** es una aplicación JAVA creada para rastrear el intercambio de mensajes entre los agentes JADE. Cuando lo activas se abre una ventana y solo podrás observar que existe un agente llamado **others**. Para que el agente funcione correctamente y te muestre el intercambio de mensajes que se puede mostrar en la figura 7 debes indicarle qué agentes quieres visualizar.
Esta herramienta es muy útil para el testeo del correcto funcionamiento del paso de mensajes. Si haces doble click en las flechas se te abrirá una ventana como en la figura 7, donde tendrás toda la información necesaria sobre ese mensaje en concreto. Remarcar que otra posibilidad que nos ofrece es guardar los mensajes y posteriormente poderlos abrir.

No obstante, hay que destacar el serio inconveniente que presenta a los desarrolladores que usamos JADE-LEAP, y es que no nos permite hacer un seguimiento de los agentes situados fuera del *contenedor*[12] principal. De todos modos esto será tratado en el capitulo dedicado a LEAP.

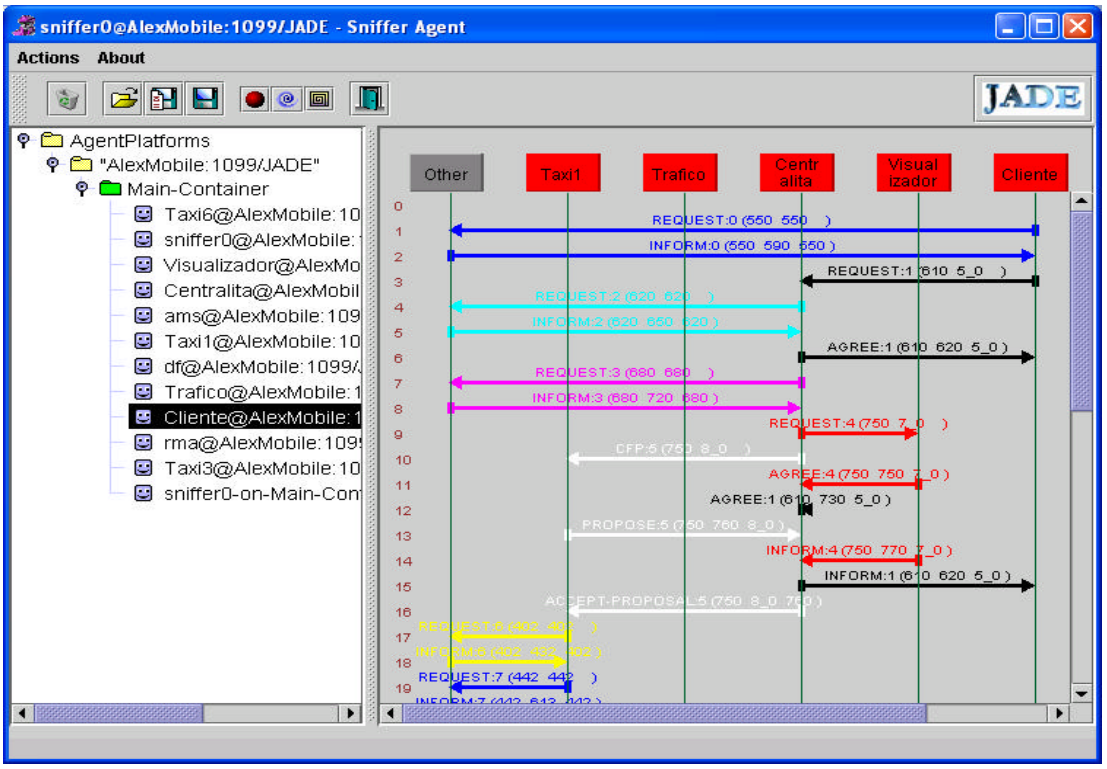



Fig. 7 – Ventana del sniffer

**AGENTE DUMMY *(Dummy-Agent)*:** éste permite enviar mensajes ACL entre los agentes, recibirlos, inspeccionarlos, leer y salvarlos en un fichero (figura 8).

[12] *en LEAP, los agentes se situan en contenedores, donde cada contenedor representa una estacion diferente, que puede ser un PC, una PDA, un movil…*

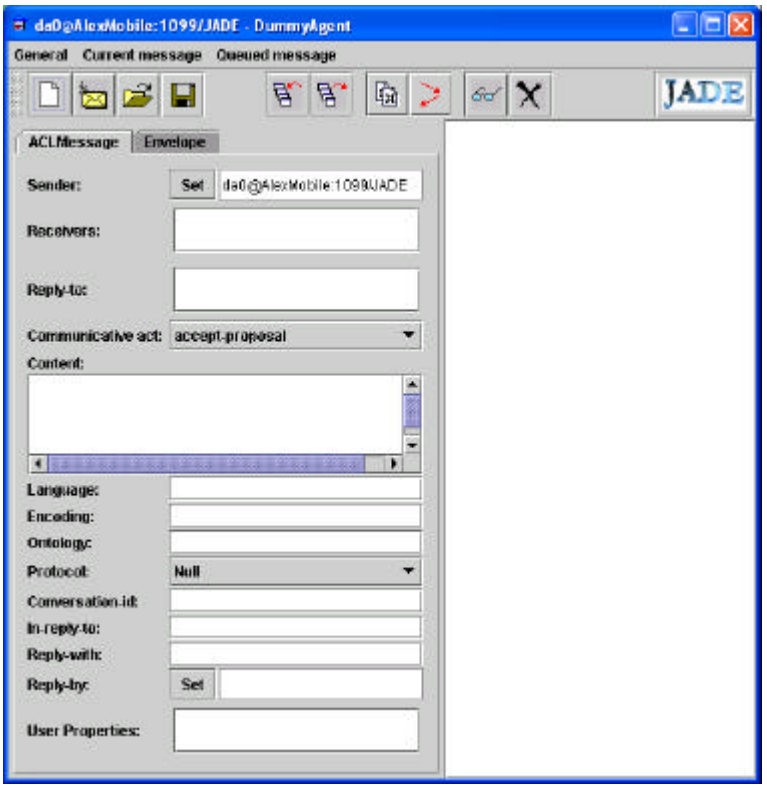



Fig. 8 – Ventana de *dummy-agent*

Otra herramienta muy útil para la realización de este tipo de sistemas, es la activación de la interficie del agente DF (menú *tools*), donde podemos observar los agentes que se han registrado y qué servicio ofrecen, tal como podemos observar en la figura 9.

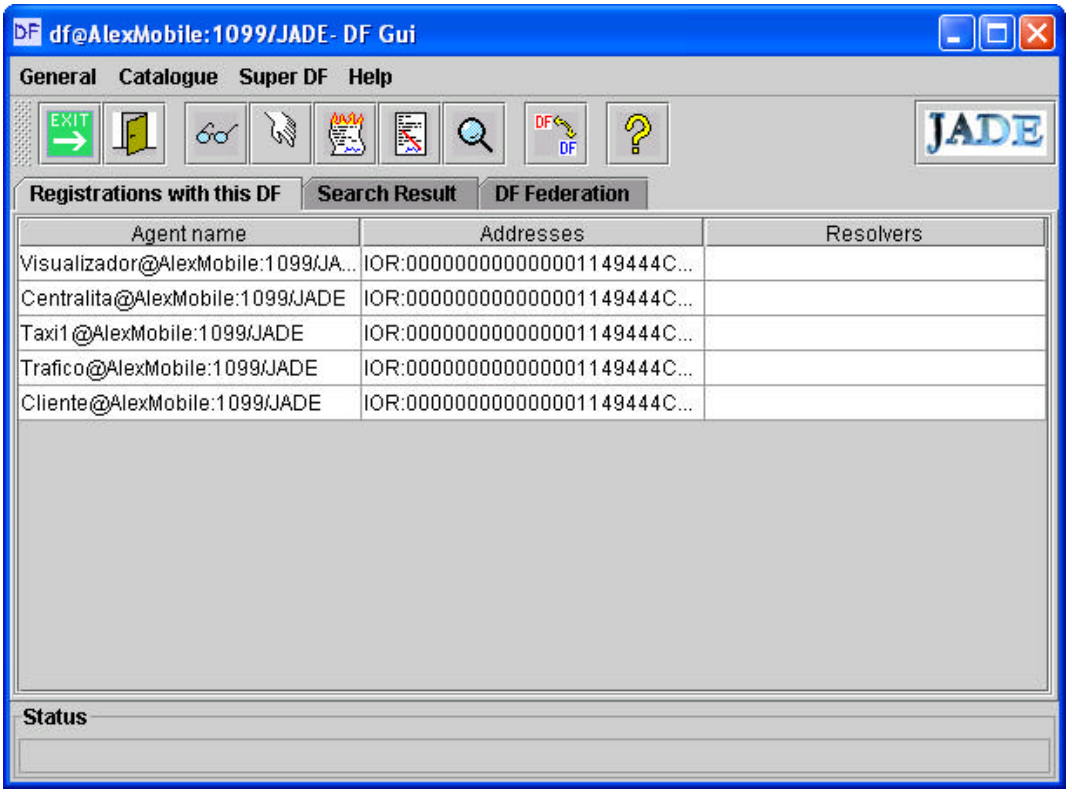



Fig. 9 – Int.gráfica del agente DF

## 3.9. Como actualizar código de JADE2.x a JADE3.0b

Como se ha indicado al inicio del capitulo, las diferencias entre ambas versiones son mínimas, prácticamente se reduce a la eliminación total de aquellos métodos y clases considerados deprecated[13] en la versión 2.61, y a ligeros cambios en los elementos que destacaremos a continuación, de todos modos, aquí trataremos tanto la versión 2.61 como anteriores, por tanto, hablaremos de ciertas modificaciones que probablemente ya hayan sido aplicadas a código en JADE2.61. Aunque estas eventualidades serán indicadas en la medida de lo posible, suponemos que el lector no tendrá ningún problema en percatarse por si mismo de estos detalles.

Toda esta información ha sido extraída de la documentación de JADE, en especial de su apartado dedicado a la promoción a 3.0b de agentes realizados en JADE 2.x.[JADE.2xto3_03]

**Acceso a los servicios ofrecidos por el agente DF i el agente AMS**

Los métodos *register()* y *modify()* de el package *JADE.domain.DFService*, ahora devuelven el DFAgentDescription que ha sido guardado por el agente DF.

En la búsqueda del DF, tenemos que definir el objeto *SearchConstraints*, para indicarle entre otras cosas, el número máximo de resultados que esperamos recibir del DF. Si no especificamos este límite, por defecto se considerará 1, a diferencia del código en JADE2.x donde se consideraba infinito. Si queremos un comportamiento similar a las versiones anteriores, tendremos que especificar -1(infinito).

Las clases deprecated, utilizadas para acceder a los servicios de AMS y DF:

*JADE.domain.AMSServiceCommunicator*
*JADE.domain.DFServiceCommunicator*
*JADE.domain.FIPAServiceCommunicator*

han sido substituidas por:

*JADE.domain.AMSService*
*JADE.domain.DFService*
*JADE.domain.FIPAService*

Todos estos cambios muy probablemente deberán ser realizados en código incluso de la versión 2.61, puesto que incluso en el caso de las clases deprecated, su utilización era ciertamente habitual.

**Ontologias**

Los cambios a nivel de ontologías solo se aprecian en versiones anteriores a 2.61. La definición de la *Agent Plaform description* (ap-description) ha cambiado en las nuevas especificaciones de la FIPA, y por lo tanto las clases del package *java.domain.FIPAAgentManagement* ha sido modificado, para ajustarse a ellas. El *ap-transport-description* y el *mtp-descriptions* ya no forman parte la definición ap-description, como consecuencia las clases que los representaban: *ApTransportDescription* y *MTPDescription* han sido eliminadas.

Lenguajes y ontologías basadas en el paquete *JADE.onto* y los métodos: *registerLanguage()*;*registerOntology()*;*fillContent()* y *extractContent()* han sido eliminados. Los programadores están obligados a utilizar las nuevas opciones incluidas en el paquete *JADE.content*.

[13] *elementos que estan en vias de desaparición por quedarse anticuados.*

Las ontologías incluidas en JADE:

*JADE.domain.FIPAAgentManagement.FIPAAgentManagementOntology*
*JADE.domain.JADEAgentManagement.JADEAgentManagementOntology*
*JADE.domain.MobilityOntology*
*JADE.domain.introspection.JADEIntrospectionOntology*

han sido substituidas por:

*JADE.domain.FIPAAgentManagement.FIPAManagementOntology*
*JADE.domain.JADEAgentManagement.JADEManagementOntology*
*JADE.domain.mobility.MobilityOntology*
*JADE.domain.introspection.IntrospectionOntology*

Todas las clases que representan elementos de estas ontologías han sido modificados, eliminando los métodos: *get_0()/set_0(), get_1()/set_1()*...

**Protocolos**

Las clases deprecated, utilizadas para los protocolos:

*FipaRequestInitiatorBehaviour*
*FipaRequestResponderBehaviour*
*FipaQueryInitiatorBehaviour*
*FipaQueryResponderBehaviour*
*FipaContractNetInitiatorBehaviour*
*FipaContractNetResponderBehaviour*

han sido substituidas por:

*AchieveREInitiator*
*AchieveREResponder*

Podemos destacar que este cambio ya fue adoptado por JADE2.61.

**Varios**

- *JADE.core.behaviours.NonDeterministicBehaviour* substituye *JADE.core.behaviours.ParallelBehaviour*.
- el método *setArguments(String)* del *JADE.core.Age*nt ha sido eliminado. Los programadores deberán llamar explícitamente *Object[] getArguments().*
- El método *toText()* de la clase *JADE.core.AID* ha sido eliminado. Los programadores deberán usar *toString().*
- los métodos *preAction()* y *postAction()* de la clase *JADE.core.behaviours.CompositeBehaviour* han sido eliminados. Los programadores deberán usar *onStart()* and *onEnd().*
- Los métodos *toText()*, *getAllPerformatives()*, *setReplyBy(String)* de la clase *JADE.lang.acl.ACLMessage* han sido substituidos por *toString()*, *getAllPerformativeNames()*, *setReplyByDate(Date).*

- Los métodos *MatchReceiver(java.util.List)*, *MatchReplyTo(java.util.List)* and *MatchReplyBy(String)* de la clase *JADE.lang.acl.MessageTemplate* han sido eliminados. los programadores deberán usar: *MatchReceiver(AID[])*, *MatchReplyTo(AID[])* and *MatchReplyByDate(Date)*.
- El método *delete()* de la clase *JADE.wrapper.Agent* ha sido eliminado. Los programadores deberán usar *kill()*.
- El método *createAgent()* de la clase *JADE.wrapper.AgentContainer* ha sido eliminado. Los programadores deberán utilizar *createNewAgent()*.

Invitamos al lector que desee más detalles sobre el tema de la promoción de código que estudie los trabajos correspondientes expuestos en la página oficial de JADE.

# 4. LEAP 3.0 PARA JADE 3.0b

LEAP (*Lightweight Extensible Agent Platform)* es un addon para JADE, que nos permite obtener un entorno de ejecución para agentes que cumplan las especificaciones FIPA y se ejecuten en dispositivos de recursos limitados, que soporten java.

Hasta la versión 2.1 LEAP fue desarrollado dentro del LEAP IST project [LEAP02]. El hecho de desarrollarse de forma externa al proyecto JADE, provocaba ciertos problemas e inconvenientes, tanto a la hora de instalar el entorno JADE-LEAP[14], como a la hora de ejecutar los agentes en el, básicamente encontrábamos ciertas limitaciones que se convertían en dificultades añadidas. Las cosas cambiaron radicalmente desde la llegada de JADE3.0 que vino acompañada de su propio addon LEAP3.0. Esta nueva versión de LEAP, usada en nuestro proyecto, se ha destapado como sencilla de instalar y muy cómoda de utilizar, facilitándonos nuestro acceso al mundo de los dispositivos móviles.

La necesidad de LEAP, viene dada por la incapacidad de JADE de ejecutarse correctamente en condiciones extremas de recursos, situación que nos encontramos claramente al trabajar en móviles o PDAs. JADE requiere un mínimo de JDK1.2 [15], problema añadido puesto que la mayoría de estos dispositivos solo soportan el llamado PersonalJava[16] o MIDP[17]. Por ultimo también hay que destacar la naturaleza de las conexiones de estos dispositivos: bajo ancho de banda, conexión intermitente, IPs dinámicas…todo esto debe ser tratado de forma especial, es por ello que LEAP nació. Su base principal es crear un contenedor para cada dispositivo, donde sus agentes se ejecutarán. Estos contenedores, supeditados al contenedor principal (main container), formaran parte de una misma plataforma, que permitirá a todos los agentes trabajar conjuntamente.

Antes de entrar de forma más exhaustiva, en la plataforma JADE-LEAP, hagamos ciertos comentarios, que nos ayudaran a comprender el tema más fácilmente:

El proyecto que hemos creado dio sus primeros pasos por el mundo wireless (sin cables), de la mano de JADE2.5+LEAP2.1, para ello utilizamos una maquina virtual de java, de reciente creación que nos dio unos fantásticos resultados, la Jeode Virtual Machine (PersonalJava1.2) [JeodeVM03].

Esta MV, pensada para PDAs y mas explícitamente aquellas armadas con el sistema operativo PocketPC 2002[18], nos resolvió todos los problemas que nos daba su versión gratuita mas antigua, la Personaljava1.1 de Sun, desarrollada para WindowsCE2.11, una versión ciertamente más antigua que la utilizada por nuestra PDA. No obstante los problemas de LEAP2.1 eran numerosos, y en las primeras pruebas nos decantamos por desarrollar los agentes dentro de MIDlets, generados con el J2MicroEdition de Sun [J2ME03]. La JeodeVM se mostró incapaz de trabajar con midlets, así que tuvimos que añadirle una librería adicional creada por Kobjects, la *me4se* [KOBJECTS03]. Con todo ello nuestra flamante MV, empezó a trabajar correctamente con los midlets generados.

No obstante, con la llegada de LEAP3.0, todo lo realizado anteriormente pasó a segundo plano. Nos decantamos por la utilización del entorno para pjava(PersonalJava) aportado por JADE-LEAP y olvidarnos así de los MIDlets. Como resultado la JeodeVM respondió perfectamente a los agentes ejecutados y cabe destacar las importantes ventajas que recibimos, por pasar de J2ME a PersonalJava, y es que la JeodeVM se comporta como si de un entorno JDK1.1 se tratase, esto nos ofrece una API mucho mas completa, que la aportada para J2ME, y entre otras cosas nos ofrece la librería grafica AWT, librería algo olvidada desde la irrupción del Swing, pero que se convierte en una bocanada de aire fresco, cuando la comparas con la arcaica librería grafica del J2ME, la

[14] *JADE powered by LEAP*
[15] *Java Development Kid. A dia de hoy se utiliza la version 1.4.1*
[16] *Maquina virtual para PDAs. A dia de hoy se utiliza su version 1.2 (JeodeVM)*
[17] *Entorno facilitado por el J2ME de sun. Es el utilizado por los moviles. Las aplicaciones que ejecuta se llaman midlets.*
[18] *Tambien llamado Microsoft Windows CE 3.0*

*javax.microedition.lcdui*, pensada únicamente para aportar entorno grafico a los móviles y desaprovechando, por tanto, las mejores aptitudes de las PDAs.

Dicho esto, recordar que el addon LEAP, puede ser descargado gratuitamente en la pagina oficial de JADE (http://JADE.cselt.it), en la cual también encontraremos un tutorial muy acertado sobre LEAP, que ayudara a profundizar al lector que se muestre interesado por esta plataforma.

## 4.1. Entorno de ejecución JADE-LEAP

LEAP, al ser combinado con JADE, substitute ciertas partes de kernel de JADE, creando un entorno modificado que identificamos como JADE-LEAP, que puede funcionar correctamente en dispositivos muy limitados desde el punto de vista de recursos. LEAP nos ofrece 3 versiones diferentes pensadas para cada uno de los 3 tipos posibles de entornos Java disponibles:

- **j2se** (Simple Edition): la indicada para ejecutarse en estaciones con jdk1.2 o superior
- **pjava** (PersonalJava) : idónea para dispositivos con PersonalJava como las PDAs.
- **midp** (Micro Edition): para dispositivos que soportan MIDP1.0 como los móviles.

Aunque internamente muy diferentes, las 3 versiones ofrecen la misma API a los programadores, solo presentando ciertas diferencias de las versiones j2se y pjava respecto a la midp. Cabe destacar que estas incompatibilidades están relacionadas con las clases de java no soportadas por MIDP. De todos modos, aunque realmente pocas, estas diferencias serán comentadas más adelante.

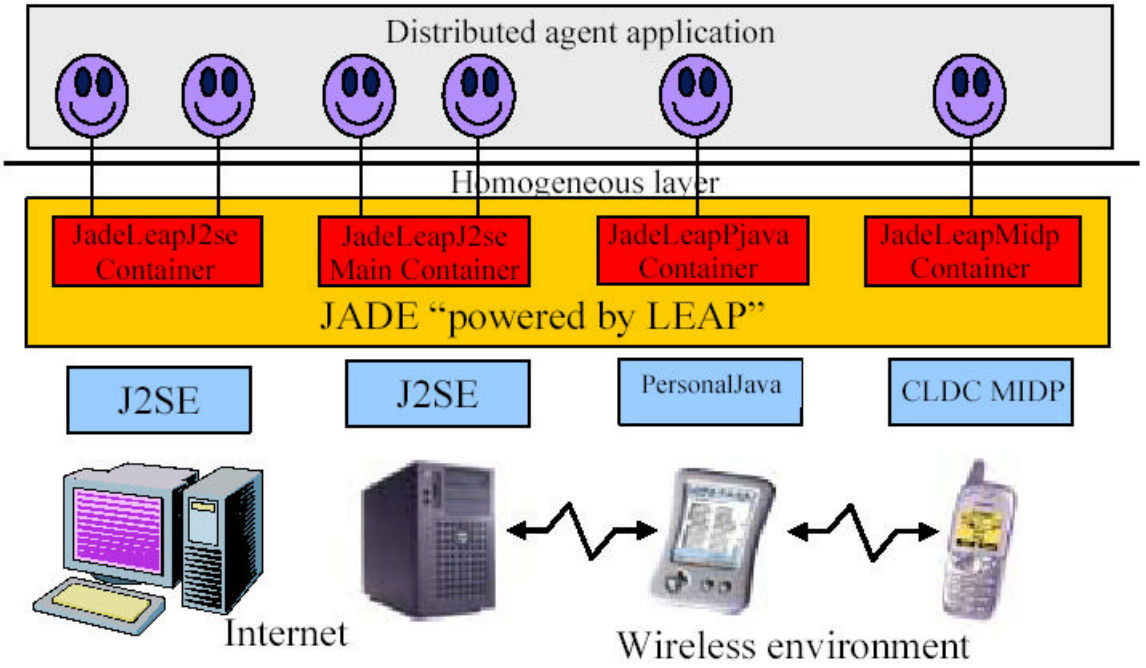



Fig. 10 – Entorno ejecución de JADE-LEAP

Desde el punto de vista del desarrollador de aplicaciones, JADE-LEAP para j2se es prácticamente igual a JADE, tanto en términos de APIs, como de ejecución, solo teniendo en mente unas ligeras diferencias, que en nada afectan al código de la aplicación. No podemos dejar de comentar que, aunque sean tan similares, no podremos mezclar los contenedores de JADE con los contenedores de JADE-LEAP en una misma plataforma.

## 4.2. Ejecución de JADE-LEAP en estaciones (PC o Servidor)

Como ya se ha dicho, en este tipo de entornos la versión de LEAP idónea será la j2se. Sobre ella podremos actuar como si de JADE normal se tratase. Por lo tanto, asumiendo que tenemos las clases de JADE-LEAP en el classpath, la forma de arrancar la plataforma sería:

*java JADE.Boot [options] [agents specification]*

Donde la lista de agentes seria algo similar a:

*local-name:package.agentclass(arg1,arg2);local-name2:package.agentclass2(arg1,arg2)*

Mención especial para la separación por punto y coma, a diferencia de la usada en JADE (usando espacio en blanco).

Las opciones posibles serían:

- **-main <true|false>:** indica si la plataforma albergará el contenedor principal (por defecto true)
- ***-container* :** equivale a "– main false"
- **-host <host adress> :** indica la dirección del main container. Si es el main container lo ignora.
- **-port <port-number> :** indica el puerto donde escucha el main container. Si es el maincontainer lo ignora.
- **-local-port <port-number>** : indica el puerto donde este container escuchara.
- **-gui :** activa la RMA GUI
- **-name <nom> :** nombre que será asignado a la plataforma.Ignorado si es un contenedor normal.
- **-container-name <nom> :** nombre del container.
- **-mtps :** activa el MTPS
- **-agents <lista de agentes separados por ; > :** activa los agentes indicados.
- **-conf :** lee la configuración del fichero especificado. (heredado de LEAP2.1)

Para la ejecución de nuestro proyecto, utilizamos la siguiente línea desde MSDOS:

*java -cp ".\j2se\lib\JADELEAP.jar;.\pfc" JADE.Boot -gui Centralita:Agentes.AgenteCentralita(DatosCalles.txt);Trafico:Agentes.AgenteTrafico;Visualizador:Agentes.AgenteVisualizador;Taxi1:Agentes.AgenteTaxi(DatosCalles.txt);Taxi6:Agentes.AgenteTaxi(DatosCalles.txt);Taxi3:Agentes.AgenteTaxi(DatosCalles.txt)*

## 4.3. Ejecución de JADE-LEAP en dispositivos con recursos limitados

Dentro de este apartado distinguiremos las versiones para PersonalJava y para J2ME, pero antes comentaremos una de las importantes novedades de LEAP3.0, y es la incorporación de dos modos de ejecución: Stand-alone y Split.

STAND-ALONE

Es el modo de ejecución normal en que un contenedor completo se ejecuta en el dispositivo. Es el modo que nosotros utilizaremos en nuestro proyecto.

SPLIT

En este modo, el contenedor se separa en dos partes llamadas FrontEnd (que se ejecuta en el dispositivo móvil) y BackEnd (que se ejecuta en un j2se host), unidas por un link a través de una conexión permanente. Las ventajas de este modelo son claras aprovechando que:

- el FrontEnd es claramente más sencillo que un contenedor completo
- la fase de bootstrap es más rápida.
- hay menos envío de datos en la transmisión entre ambas partes.

Naturalmente el programador no deberá preocuparse del modo de ejecución que presentará el container donde se ejecute su agente dado que las APIs son exactamente las mismas. No obstante sí tendrá que tener en cuenta que un Main container no puede ser Split. La movilidad de agentes y la clonación no son soportadas por un split container y además al lanzar un Split container, un j2se container tiene que estar activo en el host donde el BackEnd debe ser creado.

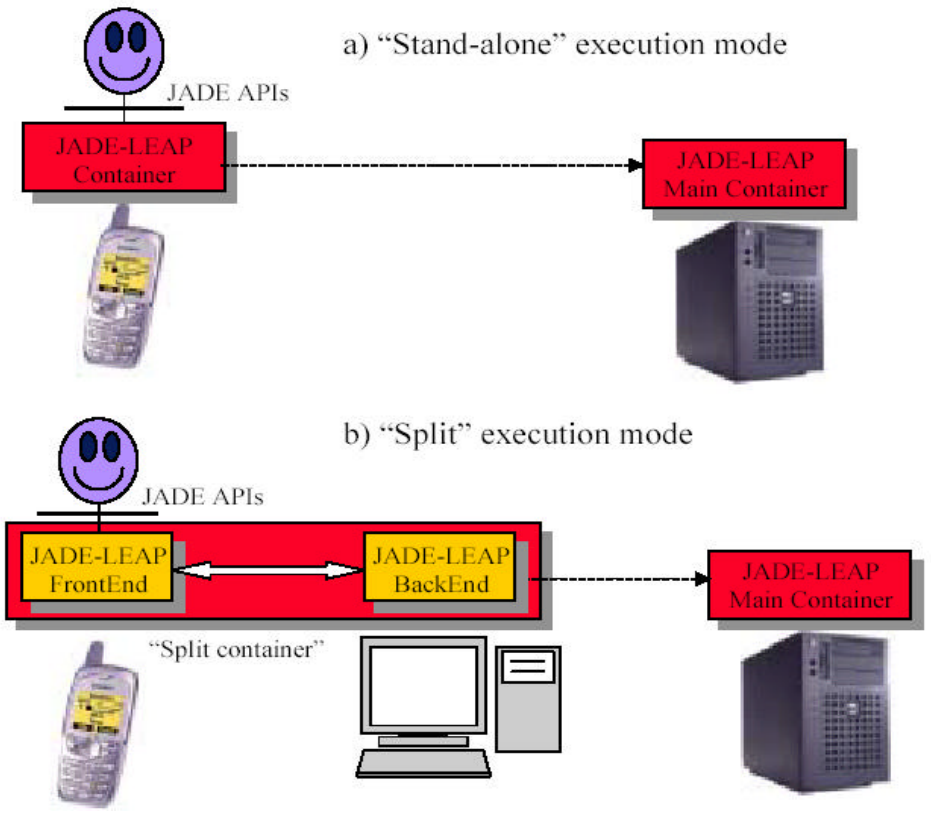



Fig. 11 – Modos de ejecución

### 4.3.1. PDAs con PersonalJava

En este medio, la versión de LEAP idónea será la *pjava*.

EJECUCIÓN STAND-ALONE

Un contenedor stand-alone sobre pjava es arrancado exactamente como un j2se container:

*java JADE.Boot [options] [agents specification]*

Aplicaremos las mismas opciones que en el caso j2se, recordando que la opción *–gui* deberá ser excluida,dado que las herramientas de JADE requieren jdk1.2 para funcionar.

EJECUCION SPLIT

Un contenedor split sobre pjava es arrancado tecleando:

*java JADE.MicroBoot [options] [agents specification]*

Con las siguientes opciones disponibles:

- **-host <host address> :** indica el host donde se encuentra el BackEnd
- **-port <port number> :** indica el puerto donde escucha el BackEnd
- **-agents <lista de agentes separados por ; > :** activa los agentes indicados
- **-exitwhenempty :** el container acabara cuando no tenga más agentes ejecutándose.
- **-conf :** lee la configuración del fichero especificado.

Para aquellos que se pregunten cómo se puede realizar la llamada a la MV de una PDA pasándole la línea para arrancar el JADE-LEAP, tenemos la respuesta: se realiza un acceso directo a la MV, en nuestro caso la JeodeVM (*\Windows\evm.exe*) y con un editor de texto se introduce la línea indicada. Como ejemplo:

*18#"\Windows\evm.exe" -Djeode.evm.console.local.keep=TRUE -cp "/java/JADELEAPPjava.jar;/java/pfc/" JADE.Boot -container -host 192.168.0.1 -port 1099 Cliente:Agentes.AgenteCliente*

### 4.3.2. Móviles con MIDP1.0

En este medio, la versión ha utilizar de LEAP será la midp. Para conseguir introducir la plataforma LEAP en dispositivos MIDP, configuraremos un MIDlet Suite[19] con los siguientes midlets:

- **JADE.Boot :** carga este midlet para iniciar un stand-alone container
- **JADE.MicroBoot :** este iniciara un split container
- **JADE.util.LEAP.Config :** midlet para editar manualmente la configuración.
- **JADE.util.LEAP.OutputViewer:** midlet para hacer una sencillo depuración de las ejecuciones.

Las opciones son las mismas que en el caso del pjava, con la misma distinción para el *–host* y *-port* si estamos en modo Stand-alone o en modo Split.

[19] *Conjunto de varios MIDlets que trabajan juntos.*

Lo que si debemos recordar en este caso, es que estas opciones de configuración pueden ser introducidas de dos maneras:

- como propiedades en el JAD del midlet (novedad de LEAP3.0)
- utilizando el JADE.util.LEAP.Config midlet (la manera clásica de LEAP2.1)

Para más información sobre este punto consultar el *"LEAP User Guide"* , disponible en la página de JADE.

## 4.4. Ejemplo final del entorno JADE-LEAP

Para finalizar este apartado dedicado a la ejecución del entorno JADE-LEAP, presentamos un último ejemplo final representado por la figura 12, que incluye un Stand-alone container funcionando sobre PersonalJava en una PDA, un Split container funcionando sobre un MIDP móvil, y una estación con j2se que hace la función de contenedor principal. Como detalle de interés podemos ver que muestra las propiedades de configuración para cada contenedor.

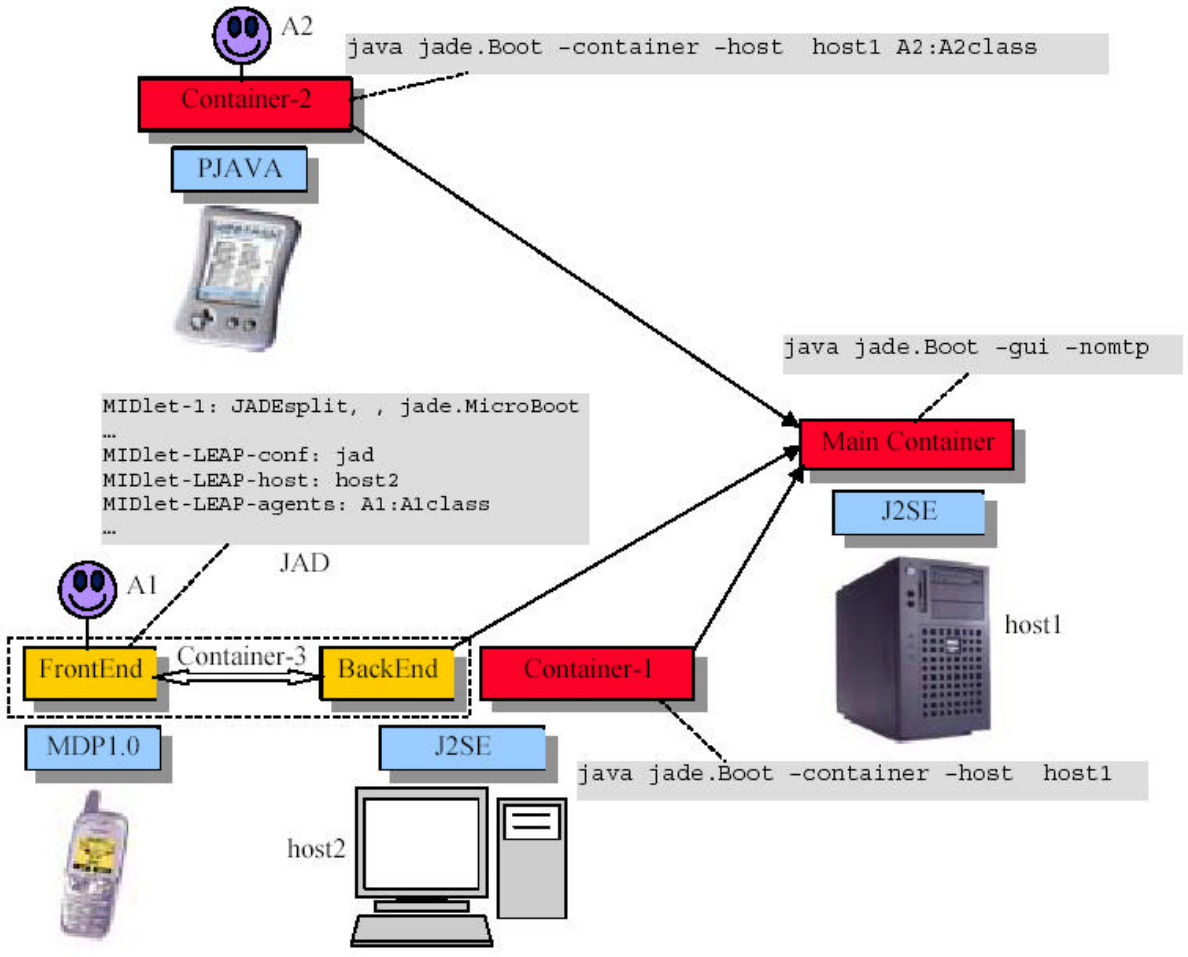



Fig. 12 – Ejemplo final

## 4.5. Diferencias entre JADE y JADE-LEAP para j2se

Vamos a indicar las diferencias entre ambas plataformas para facilitar el trabajo de los desarrolladores; como ya hemos dicho, los dos entornos son prácticamente calcados, pero veamos cuales son esas pequeñas excepciones:

JAR FILES

El JADELEAP.jar incluye las clases relacionadas con las herramientas de administración y el IIOP MTP; en JADE, éstas se encuentran en JADETools.jar y iiop.jar respectivamente.

LINEA DE COMANDOS

- En la especificación de agentes usaremos punto y coma en vez de espacio en blanco como en JADE. Además, no habrá espacios entre los argumentos de un agente.
- En JADE al lanzar un main container *–host* y *–port* indican el local host. En LEAP, ignoramos estas opciones, en su lugar usaríamos *-local-host* y *–local-port*.
- En LEAP siempre hay que especificar la opción *–agents* cuando tengamos que introducir agentes a ejecutar. Sino, lo interpretaría como un fichero de configuración.
- La opción *–dump* no esta disponible en LEAP.

IN-PROCESS INTERFACE

La clase ProfileImpl disponible en JADE-LEAP es diferente respecto a la de JADE. La diferencia radica en que ofrece un constructor que en LEAP devuelve un objeto *JADE.util.LEAP.Properties*, mientras que en JADE devuelve un objeto *JADE.util.BasicProperties.*

SEGURIDAD

El addon JADE-S, encargado de la seguridad en la plataforma JADE, a día de hoy no funciona en el entorno JADE-LEAP.

## 4.6. Limitaciones en JADE-LEAP para pjava y midp

Como prometimos anteriormente, aquí presentamos una lista de las limitaciones de LEAP en sus dos versiones para dispositivos de mano, tanto PDAs como teléfonos móviles.

PJAVA

- Todas las herramientas de administración de JADE, tienen GUIs basadas en Swing. Dado que pjava solo soporta AWT, estas herramientas no pueden ser ejecutadas en un pjava container.
- No es posible usar el Sniffer sobre un agente ejecutándose en un *pjava stand-alone container*. No obstante, sí es posible hacer el sniff sobre un agente en un *pjava split container*.

MIDP

- Las mismas limitaciones que se aplican a pjava, también se aplican a midp.
- No se puede realizar movilidad o clonación de agentes.
- Los reflective introspectors: *JADE.content.onto.ReflectiveIntrospector* y *JADE.content.onto.BCReflectiveIntrospector*, no son soportados por midp.
- El paquete *JADE.wrapper* y los métodos de la clase *JADE.core.Runtime* que se refieren a clases en ese paquete, no están disponibles en midp.

## 4.7. JADE-LEAP y la conexión sin cables (wireless)

Hemos hablado sobre la aplicación de LEAP en entornos distribuidos, formados por estaciones, PDAs y móviles, cada uno con sus propios agentes, y todos ellos trabajando juntos en una misma plataforma. Ahora nos es obligado abordar el problema de la conectividad, puesto que estos dispositivos, para poder comunicarse, deben estar conectados a través de algún medio. De esta parte JADE-LEAP no se ocupa. Este entorno de ejecución trabaja a más alto nivel, y simplemente establece conexiones TCP/IP entre los containers, sin preocuparse del medio por donde estos se producen. Por lo tanto seremos nosotros los que deberemos facilitarle esa conexión.

Teniendo en mente que WindowsCE 3.0 (el sistema operativo de nuestra PDA) soporta el llamado PPP[20], miembro del grupo de protocolos TCP/IP, parece lógico avanzar por este camino e intentar establecer una conexión PPP entre la estación y la PDA [WCE3.0+PPP03].

Una vez teniendo claro el protocolo, debemos pensar en el tipo de conexión. Las opciones del mercado son muchas y variadas, desde vía cable utilizando línea serie, hasta las mas sofisticadas y modernas conexiones sin cable, que dada la naturaleza de LEAP, con dispositivos móviles trabajando juntos, será la decisión idónea, puesto que no seria lógico tanto esfuerzo para tener agentes en pequeños dispositivos, si finalmente coartamos su libertad conectándolos con cables.

Las *wireless networks* (redes sin cables) son por lo tanto el tema que realmente nos interesa, y por el que podemos apostar en un futuro próximo. Podremos seleccionar el tipo deseado dentro de un surtido elenco: GPRS, UMTS, WLAN, Wi-fi, Bluetooth, infrarrojos...estas quizás sean las opciones mas famosas y por tanto las que cuentan con mas apoyo. De todas ellas, dado el nivel tecnológico actual, la más interesante seria la GPRS, la conexión utilizada por los teléfonos móviles para acceder a Internet. Dado que ya esta implantada a nivel mundial, y es accesible desde cualquier punto, con la única ayuda de un proveedor y un dispositivo compatible, podríamos decir que es la opción a elegir. Sin embargo, en términos experimentales como los que nosotros trabajamos y sin proveedores que nos ofrezcan ofertas satisfactorias, decidimos descartar, al menos por el momento, la conexión GPRS por resultar económicamente inviable. En vez de eso, decidimos aprovechar el modulo Bluetooth integrado en la PDA, para establecer una conexión Bluetooth con una estación, a la cual también le añadimos un modulo Bluetooth.

Para nosotros este tipo de conexión vía Bluetooth nos ofrecía muchas ventajas. Su condición de gratuita nos daba margen para realizar tantas pruebas como quisiéramos, sin preocuparnos de temas económicos. El hecho de que la PDA fuera compatible, era un motivo de peso más. No obstante, también tenemos en cuenta que nuestro proyecto vía Bluetooth no es factible. Las limitaciones de este sistema son demasiado dramáticas, ya que el alcance estándar seria 10 metros, aunque con el material adecuado podríamos llegar hasta los 100 metros, o incluso 200. Cifras que, siendo optimistas, ya se muestran del todo insuficientes, si lo que quisiéramos fuese implantar nuestro sistema en la sociedad. Pero apartándonos de su vertiente más comercial, la configuración actual nos ha permitido entrar en el mundo de los agentes en entornos wireless, y demostrarnos a nosotros mismos, que realmente podemos avanzar por este camino, sobretodo teniendo en cuenta que JADE-LEAP no tiene en cuenta el medio por donde se establece el PPP. Por tanto, podríamos pasar íntegramente nuestro SMA de Bluetooth a GPRS sin ningún inconveniente, con lo cual podemos dar por valido el sistema Bluetooth para ámbitos como la investigación, sin embargo extrapolar este ámbito a un uso mas cotidiano nos llevaría a la necesidad de poner infinidad de módulos Bluetooth a lo largo de por ejemplo una ciudad, posibilidad que ciertamente se nos antoja impracticable.

[20] *point-to-point protocol*

### 4.7.1. Conexión Bluetooth entre PDA y PC

Llegados a este punto pasaremos a explicar el procedimiento realizado para establecer la conexión PPP vía Bluetooth entre nuestros dos dispositivos, una *PDA Fujitsu Siemens Pocket LOOX 600* y un *PC con windowsXP*.

**Elementos utilizados:**

*ActiveSync 3.6*

*Conceptronic BT dongle*

*Fujitsu-Siemens Pocket LOOX 600*

*Windows XP Pro*

*PlugFree 2.0*

*Protocolo Netbeui*

*Pockethost*

Nuestra idea a la hora de realizar la conexión fue la necesidad de obtener el llamado RAS[21] entre los dos dispositivos, que nos permitiría conseguir la conexión PPP deseada. Para conseguir esto varias opciones se nos plantearon. La aplicación *Mocha Win 32 PPP* [Mocha03] presentaba aparentes buenos resultados en Palms, tanto usando cable serie como Bluetooth, no obstante, a la hora de trabajar con una PDA dotada de PocketPC2002 mostró sus debilidades, que resultaron ser insalvables. Tras este primer fracaso optamos por trabajar con la aplicación *ActiveSync3.6* de Microsoft [ActiveSync03]; las pruebas con cable serie resultaron totalmente satisfactorias, pero las complicaciones llegaron al intentar la conexión vía Bluetooth, puesto que ActiveSync, aunque pretende poder realizar este tipo de sincronismo, en realidad no esta preparado para conseguirlo, al menos no por sus propios medios. Es en este punto donde entran a colación varias herramientas, drivers y configuraciones, que fueron utilizadas para permitir a ActiveSync obtener la conexión vía Bluetooth. Pasemos a describir por pasos el proceso a seguir.

**1) Instalar Plugfree 2.0 en el PC:**

http://support.fujitsu-siemens.de/DriverCD/Start_GB_LIFEBOOK.htm?uri=/drivercd/_DriverSteuerung/GB/LIFEBOOK/SSeries/LIFEBOOK_S6010_WinXP.htm&menu=/DriverCD/_1st_Start/GB/MenuListe_LIFEBOOK_SSeries.htm

En la página indicada anteriormente encontraremos la aplicación Plugfree 2.0 y los drivers necesarios para la correcta instalación del plugfree. Procederemos desinstalando todo lo relacionado con el BT de nuestra propiedad (drivers y software) e instalaremos los drivers nuevos que son totalmente compatibles con el *Conceptronics BT dongle*, así como con todos aquellos que porten el chip CSR. La idea principal es que tanto el PC como la PDA usen plugfree para gestionar la conexión, lo cual facilitará el establecimiento del link PPP que queremos realizar. No es descartable que haya más posibilidades, pero esta es la única que ha funcionado de todos los intentos realizados.

[21] *Remote Access Service*

**2) Realizamos una primera conexión entre los dispositivos**

Al establecer la primera conexión conseguiremos que ambos dispositivos se conozcan, estableciendo passwords de acceso y almacenando la dirección Bluetooth de cada uno, muy útil para posteriores conexiones en red.

**3) Instalamos Netbeui en el PC**

Cogemos el CD del WindowsXP, buscamos la carpeta *Valueadd\MSFT\Net\NetBEUI* y copiamos el fichero *Nbf.sys* a la carpeta *C:\Windows\System32\Drivers* y el fichero *Netnbf.inf* a la carpeta *C:\Windows\Inf*.

Una vez realizado esto podemos añadir NetBEUI a nuestra configuración de red que utilizara el Bluetooth. Esto se realizaría desde el menú *conexiones de red*, seleccionando las propiedades del ítem equivalente a la conexión Bluetooth.

**4) Instalamos Pockethost en la PDA**

Lo bajamos de: http://www.handango.com/PlatformProductDetail.jsp?productId=54305 y lo instalamos según el procedimiento habitual. [Zimm02]

**5) Usamos las siguientes configuraciones**

**En el PC:**

Mis sitios de red(conexiones de red) > propiedades > la red creada para el BT:

Ip-settings BT-Dongle:
IP-address: 192.168.0.1
Subnetmask: 255.255.255.0

En la figura 13, podemos ver en la práctica, esta configuración.

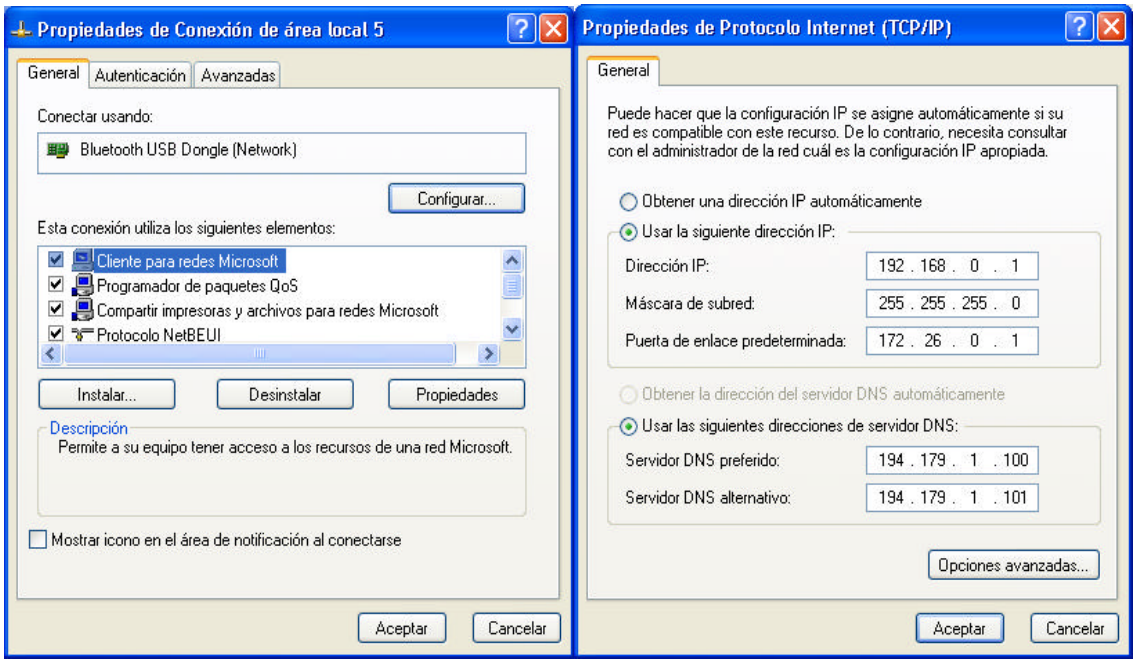



Fig. 13 – Config. de red en el PC

**En la PDA:**

In network adapters choose Rappore Technologies adapter:
IP-address: 192.168.0.10
Subnetmask: 255.255.255.0
Default Gateway: 192.168.0.1
Nameserver: 192.168.0.1

En la figura 14, podemos ver en la práctica esta configuración.


Fig. 14– Config. de red en la PDA

**Conexiones:**

Todas en modo *work*

Pockethost en la PDA:
<computername> 192.168.0.1
el *computername* tiene que ser exactamente el mismo nombre que el seleccionado en el panel *options* en el activesync de la PDA.

Podemos ver en la figura 15 como tenemos seleccionado el host ALEXMOBILE. También podemos ver otro host, ALEX1, que no se encuentra seleccionado; esto nos permite cambiar de host según con cual de ellos deseemos sincronizarnos.

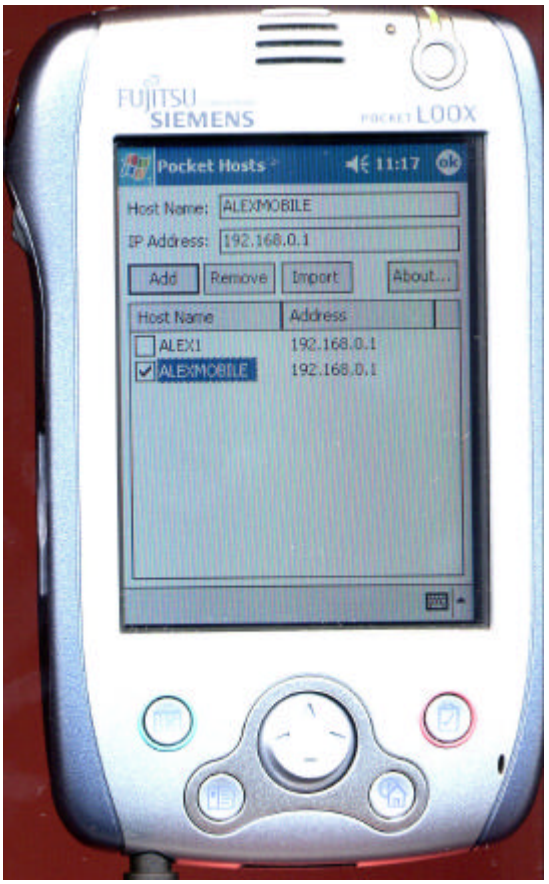



Fig. 15– PocketHost

**6) Ponemos a punto los ActiveSyncs de PC y PDA**

El primer paso será establecer una partnership entre la PDA y el PC, para ello usaremos el **cable serie**, y siguiendo los pasos que nos da Activesync, obtendremos la partnership entre los dos dispositivos. En las connection settings del activesync del PC, seleccionaremos *Allow USB* y *Allow network and RAS*, y dejaremos desactivada la opción *Allow Serial cable*.

En el Activesync de la PDA, iremos a *Tools*, a *options*, etiqueta *PC* y en *Include PC when sincro*, seleccionamos el nombre del PC partnership al que nos conectamos. En *Enable Syncro when cradled using*, seleccionamos la LAN que corresponde al Bluetooth del PC. Podemos ver en la figura 16 la imagen de la configuración correspondiente a la etiqueta *PC*.

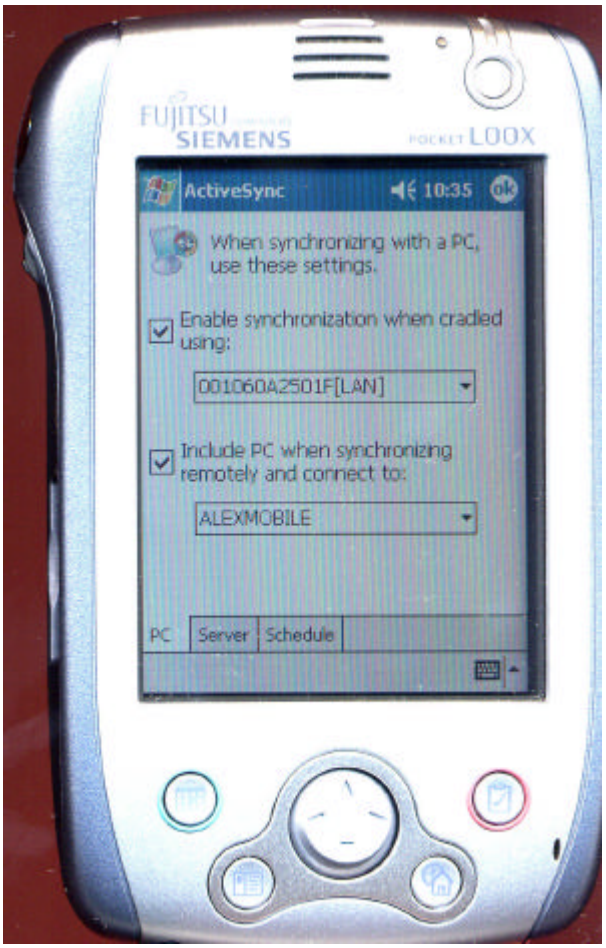



Fig. 16– Config de ActiveSync

**7) Realización de la conexión**

Una vez seguidos todos los pasos procederemos a realizar el activesync. Simplemente arrancamos el plugfree en la PDA y en el PC. Desde el PC establecemos una conexión *Personal Network*. Si se produciera algún tipo de problema, procederíamos haciendo reset a la PDA, abriendo de nuevo el plugfree de la PDA, y una vez establecido el entorno wireless, realizaríamos la conexión *Personal Network* desde el PC. Cabe tener en cuenta que en nuestro caso siempre hay que resetear la PDA antes de conseguir establecer la red entre la PDA y el PC. Una vez establecida la conexión, el activesync arrancará automáticamente tanto en el PC como en la PDA, y comenzarán la sincronización, en la PDA probablemente saldrá un mensaje de esperar varios minutos que puede ser cerrado libremente. A partir de este momento ambos dispositivos estarán conectados en red como deseábamos con el protocolo PPP vía Bluetooth funcionando. Será éste el momento de arrancar la plataforma JADE-LEAP en el PC y en la PDA.

**Comentario final:**

Como se puede observar conseguir esta conexión no es de ningún modo trivial. Sería preferible un modo más sencillo, y cabe destacar que para otros modelos de PDA se han encontrado maneras mas rápidas y fáciles. Con la PDA que nos ocupa este ha sido el único método, de todos los probados, que finalmente ha dado un resultado satisfactorio. De todos modos, entendemos que pueden existir múltiples métodos de conseguir lo mismo, simplemente nosotros no hemos sido capaces de descubrirlos.

# 5. DESCRIPCIÓN DEL SMA DE EJEMPLO: GESTIÓN DE TAXIS

## 5.1. Descripción general del problema

Para la realización del ejemplo que demostrará las aptitudes de JADE-LEAP, nos hemos decantado por un SMA para la gestión de los taxis de una ciudad. Este SMA, inicialmente creado por Juan Infante [Infante03] y pensado para funcionar en una única plataforma, ha sido modificado para su correcto funcionamiento en un entorno distribuido formado por un PC y una PDA utilizando JADE-LEAP. Las modificaciones realizadas incluyen: promoción de JADE2.6 a JADE3.0, adaptación a LEAP, creación de todo el AgenteCliente y su interficie grafica asociada y ciertas mejoras en el funcionamiento del sistema, que habían quedado pendientes en su primera concepción.

El objetivo de nuestro sistema es la utilización de las ventajas que proporciona la implementación de un SMA para optimizar de forma sustancial todo el proceso que conlleva las demandas de taxis en una ciudad. En concreto, nuestro sistema se encarga de facilitar la gestión de los taxis de una compañía que opere en una ciudad. Así pues, se gestionan de forma automática las peticiones de taxis de los clientes y se pretende mejorar el proceso de selección de taxis basándose en las propuestas realizadas por los propios taxistas. De este modo, se podría sustituir el sistema actual de radio por un sistema informático que operaría con ordenadores de abordo.

A continuación haremos una breve descripción del funcionamiento del sistema. El cliente realizará una petición mediante su agente personal que en nuestro caso se estará ejecutando en una PDA, pero que podría adaptarse a un móvil, simplemente cambiando la interficie gráfica. Cuando la centralita reciba una petición se pondrá en contacto con todos los taxis que estén trabajando en ese turno. Los taxis que estén realizando su servicio enviarían el recorrido que deben realizar para llegar hasta el cliente junto con más información a la centralita y ésta escogería al más adecuado teniendo en cuenta el taxi más cercano en términos de tiempo, en otras palabras, el taxi elegido puede no ser el más cercano, pero si será el que tardará menos tiempo en llegar hasta el cliente (esto es debido a que tenemos en cuenta el nivel de tráfico de las calles).

Como en toda ciudad, siempre hay alguna calle cortada, ya sea por congestión circulatoria, por accidente o por obras. Para incrementar la eficiencia de nuestro sistema hemos creado un agente tráfico que se encargaría de darnos en cada turno laboral, información sobre el estado de las calles. Así, los taxis tendrían siempre en cuenta el estado del tráfico para realizar con mucha más eficacia y anticipación sus propuestas de recorrido.

Una vez escogido el taxi apropiado, la centralita enviaría una respuesta al cliente dándole una información muy detallada sobre el servicio que le van a ofrecer.

## 5.2. Justificación de la utilización de un SMA

Las principales ventajas que nos proporciona el hecho de utilizar un Sistema Multiagente para este caso son:

- MODULARIDAD: El sistema ha de resolver un problema complejo de coordinación de los taxis, la centralita, el tráfico y los clientes. Por lo tanto, se descompone en tareas que pueden realizar diferentes actores (que son responsabilidad de diferentes personas) y así obtener la modularidad y mantener la independencia de los diferentes actores, que pueden actuar por sí solos según su propia conveniencia.

- EFICIENCIA: permite que los diferentes agentes se puedan ejecutar paralelamente (en diferentes máquinas) o concurrentemente (en la misma máquina), mejorando el tiempo de cálculo.

- FLEXIBILIDAD: Se pueden incorporar y eliminar fácilmente los diferentes agentes del sistema. Esto permite que el problema sea escalable, es decir, se pueden ir incorporando más taxis o muchos clientes sin tener que modificar el código.

- COOPERACIÓN: La cooperación que se consigue con este tipo de sistemas hace que se puedan solventar problemas con más complejidad. En nuestro sistema se ve claramente dicha propiedad en el momento de la negociación entre taxistas y la centralita. En futuras actualizaciones se podrían incluir nuevas técnicas de negociación.

## 5.3. Arquitectura del sistema

A continuación se muestra en esta figura la arquitectura completa del sistema, representando cada agente y los diferentes mensajes entre ellos. Recordamos que al estar trabajando en un entorno distribuido los elementos se repartirán entre varias estaciones. En nuestro caso, el AgenteCliente se encuentra en una PDA y el resto de agentes en un ordenador convencional. En un caso más real, cada AgenteTaxi se encontraría en cada uno de los taxis que recorren la ciudad, el AgenteTrafico se situaría en alguna estación perteneciente a Dirección de Tráfico, el AgenteCentralita y el AgenteVisualizador se encontrarían en la empresa de taxis, y como ya hemos comentado el AgenteCiente se encontraría en la PDA o teléfono móvil del cliente que solicita un taxi.

*R=Respuesta de la comunicación (INFORM, FAILURE, NOT_UNDERSTOOD,…)*

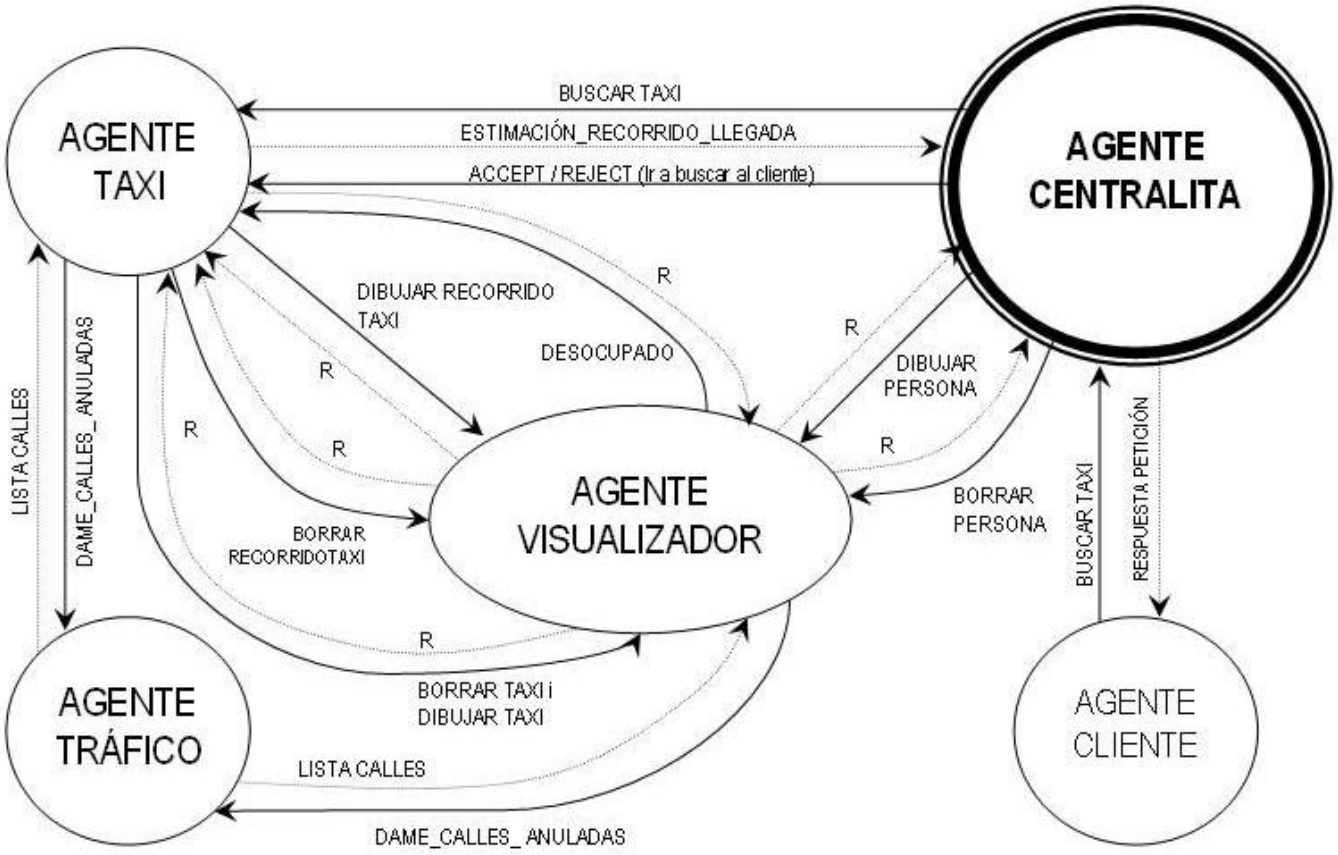



Fig. 17 – Arquitectura del sistema

## 5.4. Agente Cliente

El agente CLIENTE representa a una persona que pide un taxi en una ciudad. Es el iniciador de todo el proceso y a continuación les explicaremos con más detalle qué funciones y características tiene:

Su función principal es la de proporcionar toda la información necesaria sobre su petición al agente CENTRALITA. Para ello se pone en contacto con el DF para localizar al agente CENTRALITA de su Sistema Multi-Agente, es decir, la centralita de su ciudad.

La información enviada en una petición está formada por la siguiente información: el número de personas, la posición del cliente y el destino. El cliente, a partir del mapa de la ciudad que posee su PDA, seleccionará con el Stylus[22], el origen y el destino.

Una vez enviada la petición, será procesada por la centralita y esta dará una respuesta al Cliente, dándole los datos del taxi que vendrá a recogerlo, junto con el tiempo que tardara el taxi en recoger al cliente, el precio del viaje y el trayecto, que será visualizado en el mapa de la PDA.

Como detalle de interés podemos indicar que el mapa a utilizar se carga vía fichero de texto, por lo tanto, al desplazarnos por otra ciudad simplemente tendríamos que cargar el fichero de texto correspondiente. Dicho fichero, podría ser descargado automáticamente, de algún servidor de la empresa de taxis, para facilitar aun más el proceso.

## 5.5. Agente Centralita

El agente CENTRALITA es uno de los más importantes del sistema, ya que, es el coordinador de las diferentes peticiones de los clientes. Su funcionamiento es primordial para obtener un resultado adecuado.

La función principal es la de coordinar las peticiones y escoger el taxi que mejor se adapte a las necesidades del cliente, aunque también se encarga de actualizar la *cache* de los taxis y avisar al cliente del resultado de su demanda.

La cache es una lista que se actualiza cada 8 horas y contiene los taxis que están trabajando en el turno actual (los que están registrados en el DF). Así sabemos a qué taxis hemos de avisar, ya que, en la plataforma se dan de alta todos los taxis de cada turno.

Puesto que en la revisión actual, hemos introducido la opción de utilizar varios mapas por el cliente, es obvio que la Centralita también deberá ser capaz de utilizar una ciudad u otra en función de donde se encuentre. Para facilitar estos cambios, se le indicara por parámetro el fichero de texto que representa la ciudad, y sabiendo esto, la Centralita cargará los datos de la ciudad idónea.

Una vez recibida la petición, le enviamos la información recibida a cada taxi (lo sabemos por la cache de taxis) y esperamos la recepción de sus propuestas, que proporcionan la siguiente información: el recorrido hasta llegar al cliente, el recorrido del servicio que le realizará al cliente y si el taxi está ocupado. Al recibirlas, escoge el más adecuado, a través de unos criterios que aunque se podrían mejorar en futuras revisiones, en la actualidad ya están bastante perfeccionados, puesto que aparte de tener en cuenta la distancia que separa el taxi del cliente (criterio elegido por el señor Infante), nosotros hemos tenido en cuenta el coste de circular por cada tramo. Por lo tanto, el taxi seleccionado no tiene porque ser el más cercano, puesto que ahora tenemos en cuenta que un taxi más alejado podría llegar antes gracias a la fluidez del tráfico.

[22] *Estilete que sirve para pinchar sobre la pantalla de la PDA.*

Una vez hemos escogido al taxi, la centralita avisaría al cliente proporcionándole la siguiente información:

- Respuesta afirmativa o negativa.
- Identificador del taxi que le ofrecerá el servicio.
- El precio que le costará el viaje.
- Tiempo que tardará el taxi en recoger al Cliente
- Trayecto que realizará el taxi para efectuar el servicio requerido por el Cliente.

## 5.6. Agente Taxi

El agente TAXI es el que representa a un taxi de la ciudad. Como hemos explicado anteriormente, substituiríamos los sistemas de radiofrecuencia, por ordenadores de abordo que facilitarían la tarea del taxista, ofreciéndole información muy valiosa para su trabajo. A continuación les explicaremos las características y funciones con más detalles.

De cada taxi necesitamos saber una serie de características para poder identificarlos:

IDTaxi: número para identificar a cada Taxi.
Propietario: nombre el propietario del Taxi.
NumLicencia: Número de la licencia de cada taxista.
Matricula: Matrícula del vehículo.
Marca: Marca del vehículo.
Modelo: Modelo del vehículo.
Turno: Turno al que pertenece dicho taxi.
Posactual: Posición del mapa donde se encuentra situado el Taxi.
Ocupado: Nos indica si el Taxi está o no ocupado.

Toda esta información puede ser utilizada en futuras actualizaciones para mejorar la calidad y la información que se obtenga del sistema. Con esto conseguimos, por ejemplo, que si un cliente desea ser atendido por una marca o modelo en concreto de vehículo, se le pueda ofrecer dicho servicio.

Al igual que en el caso de la Centralita, el hecho que el Cliente pueda cambiar de mapa, obliga a los taxis a tener los datos de la ciudad correcta. De forma similar a como se ha resuelto en el caso de la Centralita, se le indicara por parámetro el fichero de texto que representa la ciudad, y sabiendo esto, el Taxi cargará los datos de la ciudad idónea.

La función principal del agente Taxi es la de ofrecer un recorrido, tanto de búsqueda del cliente, como de realización de servicio, lo más óptima posible. En nuestro sistema el taxi calcula siempre el camino más corto desde su posición actual, hasta llegar a la del cliente. Para hacerlo utiliza algoritmos de Inteligencia Artificial basados búsqueda heurística, en concreto aplica el algoritmo A* con una función heurística basada en la distancia de Manhattan.

Una vez ha calculado el recorrido, el agente taxi se lo envía a la Centralita, que posteriormente se encarga de seleccionar el Taxi que sea mas idóneo para el servicio requerido.

## 5.7. Agente Visualizador

El agente VISUALIZADOR, sirve para poder monitorizar el sistema y estudiar el prototipo. En un entorno real, podriamos situarlo en la empresa encargada de la gestión de los taxis, para llevar un seguimiento de los servicios realizados. En el trabajo realizado, su uso ha quedado limitado al uso del mapa cuadriculado, siendo incapaz de dar resultados coherentes con cualquier otro mapa creado. Por lo tanto, podriamos pensar en mejorar este Agente en futuras modificaciones para que se adapte a cualquier mapa que deseemos usar.

Los detalles sobre la forma de dibujar edificios y trayectos, serán explicados, en el apartado dedicado a la interficie gráfica del AgenteCliente. Esto es así porque ambos funcionan de manera análoga utilizando una misma matriz de nodos donde, cada nodo representa un punto de origen/destino y de igual forma, ambos utilizan un sistema similar por matrices con valores de píxel, para dibujar correctamente en pantalla. Siempre salvando las distancias entre un mapa de 1024x711 pixels, utilizado por el AgenteVisualizador y otro de 240x300 utilizado por el AgenteCliente. Podemos ver un ejemplo de la interficie en la *figura 18* y una imagen de la leyenda en la *figura 19,* con los colores utilizados para mostrar los resultados obtenidos.

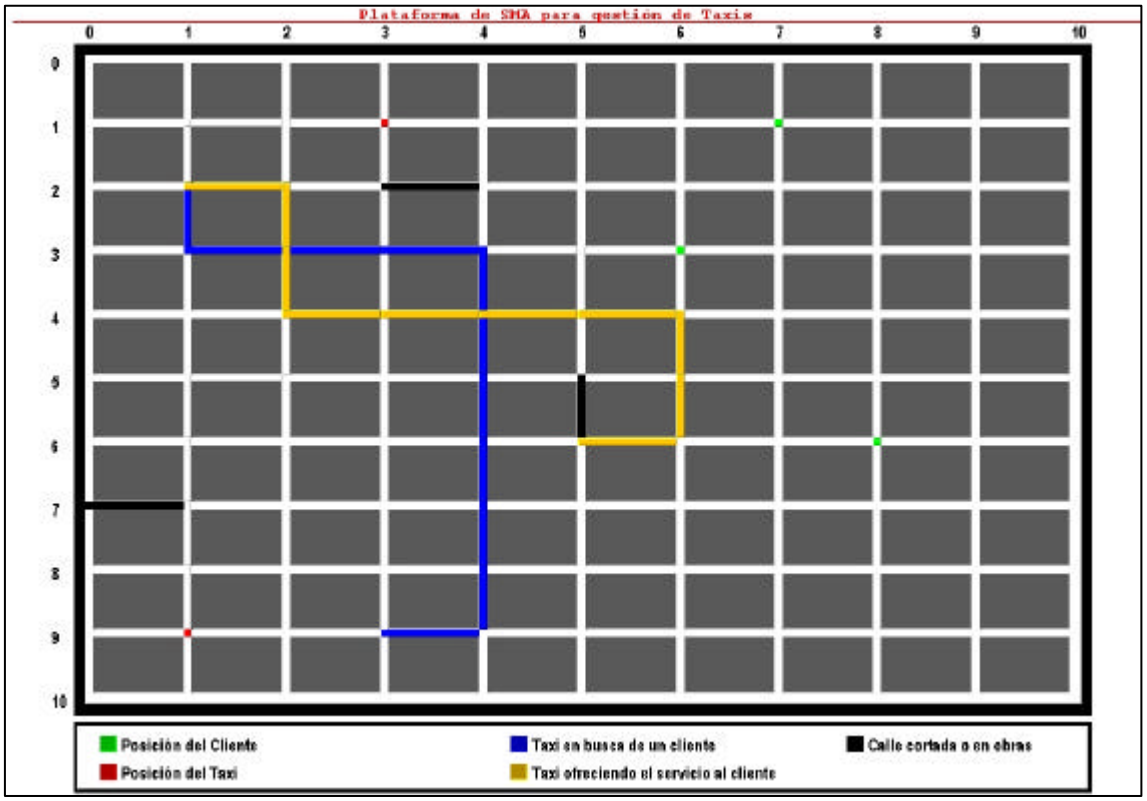



Fig. 18 – Ejemplo de la interficie gráfica

| Color | Nombre | Descripción |
|---|---|---|
| | Posición del cliente | Indica la posición de un cliente que ha pedido un taxi. |
| | Posición del taxi | Indica la posición actual de un taxi. |
| | Taxi en busca de un cliente | Representa el recorrido que realiza el taxi hasta llegar al cliente. |
| | Taxi ofreciendo el servicio al cliente | Representa el recorrido que realiza el taxi para llevar al cliente a su destino. |
| | Calle cortadas o en obras | Indica qué calles son las que, por algún motivo, están cortadas. |

Fig. 19– Leyenda de la interficie

## 5.8. Agente Tráfico

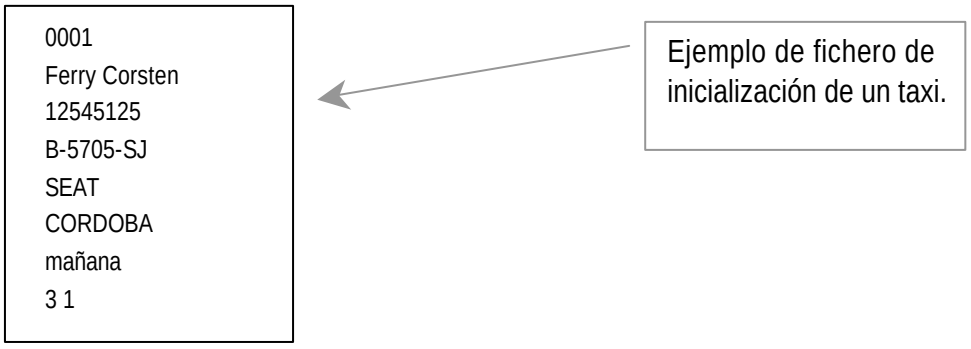
El agente TRÁFICO es el encargado de informar sobre el estado de las calles. Actualmente en el sistema, nos indica qué calles son las que están cortadas por una serie de motivos que anteriormente hemos comentado, aunque en futuras actualizaciones se podría realizar una conexión remota a la DGT (Dirección General de Tráfico) y poder estar completamente informados en todo momento del estado de las calles.

Otro detalle muy interesante sería la posibilidad que el AgenteTrafico informe en tiempo real, de la fluidez del tráfico en cada calle, para permitir un control más realista del coste de circular por cada tramo. Coste, incluido en el AgenteCentralita como factor para decidir la selección de un taxi u otro. En la actualidad, el coste de cada tramo se toma al cargar el sistema, del fichero de texto que porta los datos de la ciudad para taxis y centralita. Proponemos pues, que en una aplicación real del programa, la DGT fuera la encargada de ir actualizando el valor del coste, en función de los problemas circulatorios que pueda tener cada tramo.

## 5.9. La simulación

A continuación explicaremos cuales son los pasos más importantes para la correcta ejecución del sistema.

La creación de los *Taxis* se hace a través de ficheros de inicialización, los cuales nos proporcionan toda la información necesaria para la creación de dichas entidades. El fichero, con formato de texto, nos facilita la siguiente información: Identificador del taxi, propietario, número de licencia, matrícula, marca, modelo, turno, posición actual. Toda esta información debe estar en líneas diferentes dentro del fichero, como muestra el siguiente ejemplo:




Como hemos explicado anteriormente, a la hora de realizar la simulación todos los taxis se inicializan al ejecutar el sistema y dependiendo de la hora en que se este ejecutando (lo averiguamos a través del reloj del sistema), el simulador sabrá qué taxis de todos los cargados, deben comenzar a ofrecer sus servicios, es decir, si es ejecutado a las 12:34h los taxis que comenzarán su turno serán los de MAÑANA, si es ejecutado a las 17:21h los taxis que comenzarán su turno serán los de TARDE y si es ejecutado a las 1:21h los taxis que comenzarán su turno serán los de NOCHE.

Eso lo conseguimos a través de procesos automáticos (threads) que se ejecutan cada 8 horas (suponemos turno de trabajo estándar) aproximadamente. Con la ejecución de dichos threads conseguimos que cada 8 horas (cada turno) los taxis automáticamente se den alta/baja en el sistema. Los taxis que no trabajan en el sistema, es decir, no es su turno laboral, se eliminan del DF y así conseguimos que los únicos taxis dados de alta en el DIRECTORY FACILITATOR sean los que en ese momento realizan su turno. Gracias a esto,a la centralita le será más sencillo averiguar qué taxis, de todos los cargados en el sistema, son los que realizan su servicio en ese momento.

La creación de los *Clientes* se realiza lanzando el container JADE-LEAP, con su consiguiente AgenteCliente, desde la PDA. El container creado se unirá a la plataforma principal y el agente estará preparado para enviar sus peticiones a la centralita. Cabe destacar, que una vez finalizado el servicio requerido, el AgenteCliente no es eliminado, como ocurría en la versión del señor Infante. En nuestro caso, el Agente permanecerá en la plataforma hasta que el Cliente decida cerrar la aplicación.

La centralita al recibir una petición, pregunta a los taxis que en ese momento realizan sus servicios (lo sabemos porque son los que están dados de alta en el DF) y de las propuestas recibidas elige el que se ajuste a su criterio de selección.

Es muy probable que un cliente pida un taxi, y en ese momento, no haya ninguno libre que le pueda ofrecer el servicio. En ese caso, hemos implementado la centralita de tal forma que realice peticiones automáticas, es decir, avisaría al cliente de que en ese momento no hay ningún taxi libre, pero volvería a lanzar la petición hasta que fuese servida.

Para la representación del mapa de la ciudad hemos utilizado una estructura de datos *grafo dirigido*[23]*.* Dicha estructura es cargada a través de un fichero de inicialización donde obtenemos los diferentes nodos calles, direcciones y el coste asignado a cada calle, este coste será el utilizado para tener en cuenta la fluidez circulatoria de cada tramo. El ' * ' indica que la información de dicho nodo ha finalizado. Este fichero de inicialización obviamente será diferente para cada ciudad que pretendamos utilizar, y será cargado por el AgenteTaxi (tantos agentes taxi como hayan) y por el AgenteCentralita.

A continuación mostramos un pequeño ejemplo de dicho fichero:

```
0 1 0*
1 2 0*
2 3 0 13 1*
…
```

Del nodo 0 hay una calle hacia el nodo 1 (indica su dirección) con coste 0.
Del nodo 1 hay una calle hacia el nodo 2 (indica su dirección) con coste 0.
Del nodo 2 hay una calle hacia el nodo 3 (indica su dirección) con coste 0.
hay una calle hacie el nodo 13 (indica su dirección) con coste 1.

La obtención de las calles cortadas, por el agente Tráfico, también se realiza a través de un fichero de inicialización. El formato del fichero es de texto y la información que representa son las calles cortadas que en ese momento hay en la ciudad. Para saber el final de la línea lo indicamos a través de una marca ' * '. En este tipo de ficheros no indicamos la coordenada del mapa, sino, el número del vértice dentro del grafo.

Ejemplo: 25 26 71 60 78 77*

Observamos que del vértice 25 al vértice 26 hay una calle cortada, del 71 al 60 otra y del 78 al 77 otra calle en las mismas condiciones.

Para poder realizar una simulación más completa, hemos decidido que dependiendo de la hora en que se ejecute el sistema, el agente obtendrá calles cortadas diferentes (mañana **8h – 16h**, tarde **16h – 24h** y noche **0h – 8h**). Cada 60 segundos (tiempo que se puede modificar, es un ejemplo), el agente comprobará otra vez la hora del sistema para saber si ha de cambiar las calles cortadas. Esto se realiza a través de procesos (threads) automáticos.

Tanto el agente *Taxi* como el agente *Visualizador* , actualizan sus calles cortadas al mismo tiempo, así conseguimos que haya coherencia en los datos y que dispongan de la misma información.

[23] *El grafo dirigido nos permite que las calles puedan ser de un solo sentido.*

## 5.10. La Interficie Gráfica del AgenteCliente

Preliminarmente, hablaremos sobre los mecanismos de representación del mapa y las trayectorias, mecanismos que comparten tanto el AgenteVisualizador como el AgenteCliente.

Tanto el mapa cuadriculado (el inicial creado por Infante) como cualquier nuevo mapa que generemos, comparten una misma estructura de datos, estamos hablando de un grafo dirigido, donde cada nodo representa un punto seleccionable del mapa. Los nodos están numerados del 0 al 120, y están introducidos en una matriz para conocer rápidamente el número de nodo, a partir de las coordenadas del mapa. Esto es especialmente útil, a la hora de coger los datos del mapa (calles que unen nodos, coste de circular por un tramo…)por parte de la Centralita y de los Taxis. La utilización de este grafo, nos da una importante libertad a la hora de dibujar cualquier mapa que deseemos, solo tendremos que tener en cuenta la distribución de las conexiones entre nodos. Por ejemplo, si un nodo esta situado en medio de un edificio, obviamente este nodo estará aislado, por tanto a la hora de introducir los datos de la ciudad, dejaremos ese nodo isolado. Cabe estacar, como interesante novedad respecto a la versión de Infante, que en la actualidad, permitimos carreteras diagonales, o sea un nodo puede estar conectado con cualquier nodo, este horizontal, vertical o diagonal. Podemos ver la representación de este grafo en la figura 20. Al observarlo podemos constatar que cada nodo esta unido a sus adyacentes, que esta conexión exista o no dependerá del mapa que queramos representar.

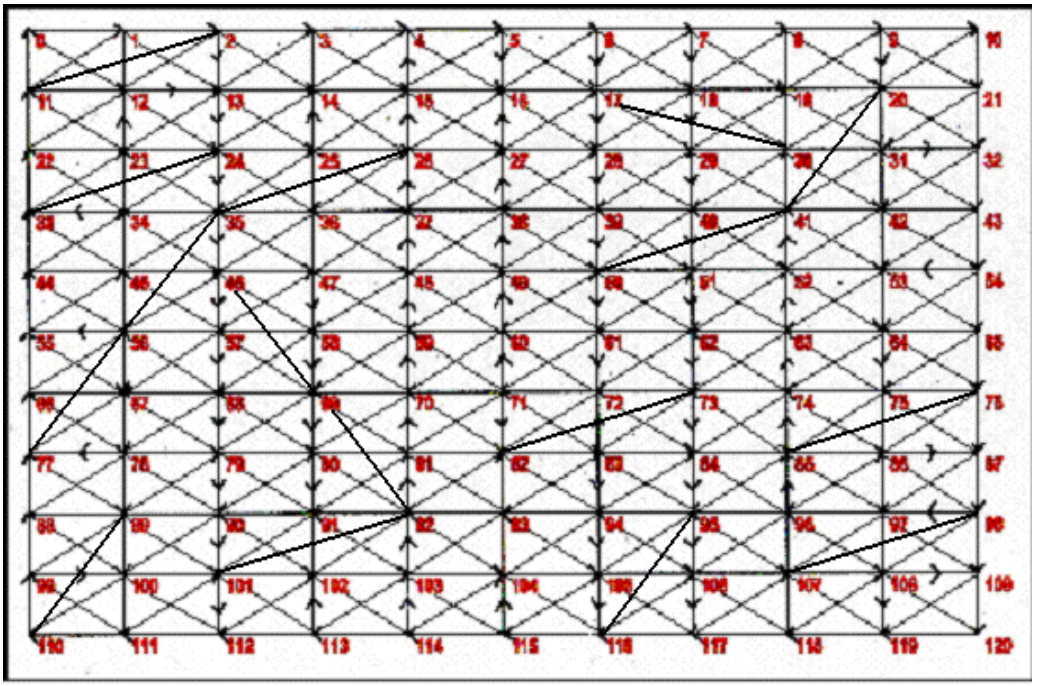


Fig. 20 – el grafo dirigido

Una vez tenemos clara la forma como el SMA entiende el mapa, y realiza los itinerarios sobre el, es obligado comentar, como representa en pantalla edificios y trayectos, para que puedan ser observados por los usuarios. La idea principal, es ya que tenemos el mapa representado por coordenadas (filas,columnas), tener una matriz que contendrá el valor en pixels de cada coordenada.Obviamente dado que la pantalla del PC y la pantalla de la PDA son muy diferentes, deberemos tener dos matrices, una para cada dispositivo. Una vez tengamos estas matrices, dibujar será tan fácil como saber las coordenadas donde tenemos que dibujar, y preguntar a las matrices para saber la posición real en pixels, valor que le pasaremos a las rutinas encargadas de dibujar. Para ser mas claros podemos observar las Figuras 21 y 22.

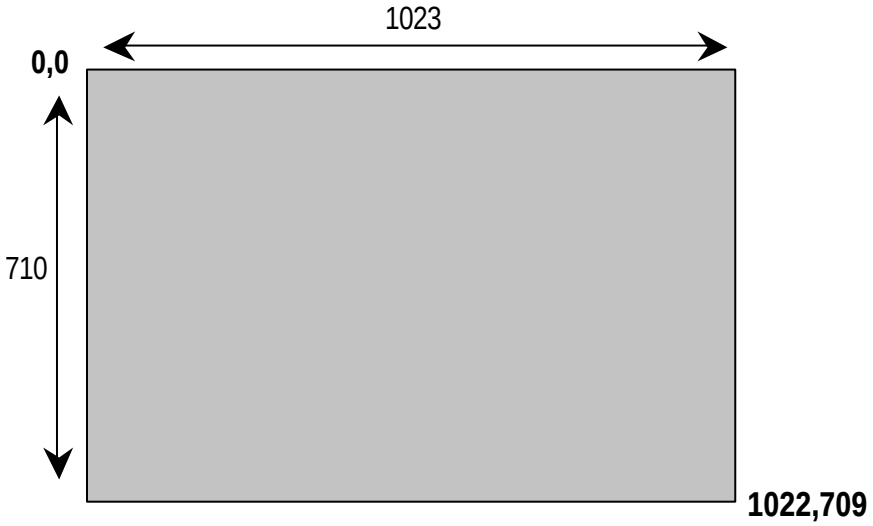



Fig. 21 – Rango de píxeles para PC

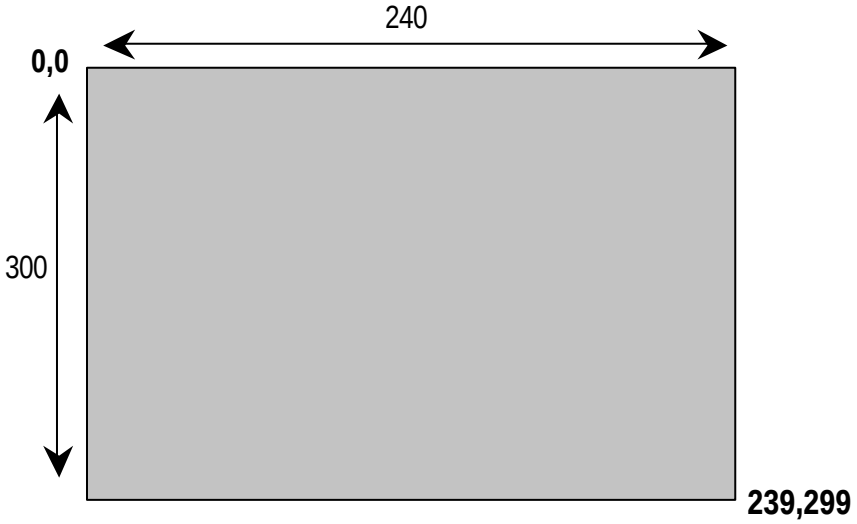



Fig. 22 – Rango de píxeles para PDA

Una vez vistas las ideas puestas en práctica para llevar acabo todo el motor grafico, comentemos el trabajo realizado en el AgenteCliente.

Su estructura ha sido creada para permitir al usuario interactuar con facilidad para poder enviar la petición de servicio a la centralita. La interfice cuenta con 2 ventanas y un Canvas[24]. La primera ventana, es la encargada de contener los datos a enviar y mostrar los datos recibidos. Además incluye unos sencillos menús, para cargar los ficheros de texto que representan la ciudad, también desde el menú, es posible informarse sobre la versión de producto utilizando un clásico *About*.

La segunda ventana será la encargada de servir de base para el canvas, que será el objeto donde dibujaremos el mapa. Al canvas se le ha añadido un *listener*[25] que captará las pulsaciones que realice el usuario con el Stylus sobre el mapa, de esta forma sabremos origen y destino del servicio deseado. Una vez la petición sea servida, recibiremos los datos requeridos, junto con el trayecto, que será dibujado en el mapa para que el Cliente pueda ver los tramos que recorrerá en su servicio.

Podemos, destacar, que junto con los datos del mapa, el fichero de texto que representa la ciudad, trae los nombres de las calles que pasan por el nodo seleccionado. Estos serán mostrados en el mapa, al pulsar con el Stylus, facilitando así, la orientación del cliente, por el entramado de calles.

[24] *Objeto utilizado en las librerias graficas como soporte para dibujar.*
[25] *Objeto java que nos permite cazar eventos.*

Llegados a este punto y para acabar de representar bien la forma de trabajar de la interficie, mostraremos un corte del fichero de texto que se utiliza para representar la ciudad, con sus edificios y sus nombres de calles. La idea es mostrar el formato del fichero, para facilitar en la medida de lo posible que terceras personas generen sus propios mapas.

El fichero se separa en dos partes, la primera esta destinada a los nombres de las calles que pasan por cada nodo, el formato es sencillo, primero se indican las coordenadas del nodo y luego se introduce un String con los nombres de las calles que se dan cita en ese nodo, acabamos la línea con el carácter especial #.

```
0 0 calle1,calle2#
0 1 calle2,calle3#
0 2 calle2,calle4,calle5#
0 3 calle4,calle6#
.....
```

en el nodo de coordenadas 0,0 las calles son: calle1 y calle2
en el nodo de coordenadas 0,1 las calles son: calle2 y calle3
en el nodo de coordenadas 0,2 las calles son: calle2 ,calle4 y calle5
en el nodo de coordenadas 0,3 las calles son: calle4 y calle6

La segunda parte del fichero se diferencia por que cada línea comienza con el carácter especial %. Esta parte será la que dice a la rutina que dibuja, las posiciones y el tipo de edifico que es. El formato es el siguiente: tras el carácter especial %, un entero nos dice el número de vértices que tendrá el edificio y los enteros sucesivos separados por el carácter especial $, serán las coordenadas de cada vertice, según los pixels de la pantalla de la PDA. Estas coordenadas se ponen por parejas: primero las X y luego las Y. La línea acaba con el carácter especial #.

```
%4$9$9$28$9$28$28$9$28$#
%3$9$31$28$31$28$50$#
.....
```

edificio de 4 vertices: (9,9) (28,9) (28,28) (9,28)
edificio de 3 vertices: (9,31) (28,31) (28,50)

Dado que trabajamos directamente con los píxeles de la pantalla, este mapa creado, solo podrá ajustarse al modelo de PDA que estamos usando. Aunque esto presenta un claro inconveniente a la hora de trasladarlo a otros dispositivos, podemos pensar que el objetivo final, en el caso de la generación de mapas es tener un editor que los construya, por lo tanto podemos suponer que será el editor el encargado de adaptar el mapa al dispositivo que nos interese.

Como último detalle respecto al formato de los datos entrados, cabe destacar que la posición de las parejas no es aleatoria, tiene que seguir un orden, empezando por el vértice inicial y acabando con el vértice final, sino la figura dibujada podría no parecerse a lo que deseamos.

# 6. COMUNICACIÓN ENTRE LOS AGENTES

## 6.1. Intercambio de mensajes entre los agentes

A continuación se muestran los diferentes diagramas temporales del proceso de petición de un taxi y como los agentes interactúan entre ellos.

1. Al iniciarse un Taxi
2. Al iniciarse el Visualizador
3. Cuando el Taxi finaliza el servicio
4. Cuando un cliente pide un taxi
5. Borrar Cliente
6. Borrar Taxi

**1. Al iniciarse un taxi:** Todos y cada uno de los taxis que estarán en el sistema tienen un horario de trabajo de 8 horas. Al comenzar dicha jornada se deberán informar del estado de las calles, para ello le preguntarán al agente TRÁFICO. Pasada la jornada laboral el taxi se dará de baja del DF, ya que, no trabajará hasta el día siguiente. Así conseguimos que los únicos agentes TAXI que estarán activos en el DF serán los que estén haciendo su turno de trabajo.

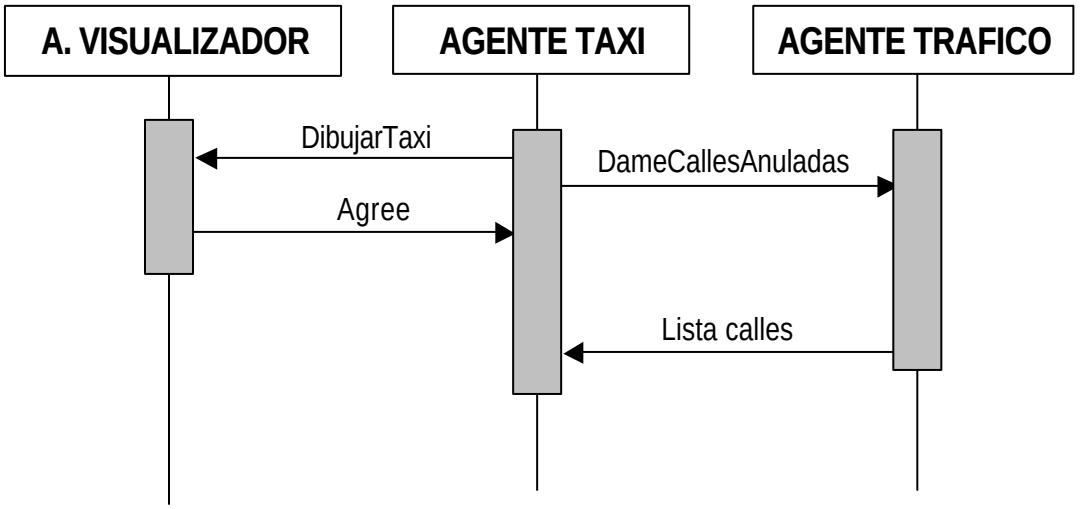



Fig. 23

**2. Al iniciarse el agente Visualizador**: En el momento de la inicialización, dicho agente, necesitará saber toda la información referente al estado de las calles de la ciudad. Para ello, enviará un mensaje al agente tráfico para actualizar su información. Este proceso se realizará cada vez que se cambie de turno (8 horas), así nos aseguramos que la información que tienen los taxis es la misma que dispone el Visualizador, para evitar posibles incoherencias, de todos modos, recordemos que todo lo relacionado con el entorno grafico presentado por el Visualizador solo puede ser aplicable en el caso del mapa cuadriculado (el mapa inicial creado por Infante). En caso de utilizar mapas alternativos, podemos dar por seguro que el visualizador dará resultados poco reales.


Fig. 24

**3. Cuando el taxi finaliza el servicio:** En el momento en el que el agente visualizador dibuja tanto el recorrido para ir a buscar al cliente, como el recorrido del servicio realizado por el taxista, el agente visualizador, envía un mensaje al taxi en concreto para avisarle que ya ha finalizado y así saber que ya no está ocupado.

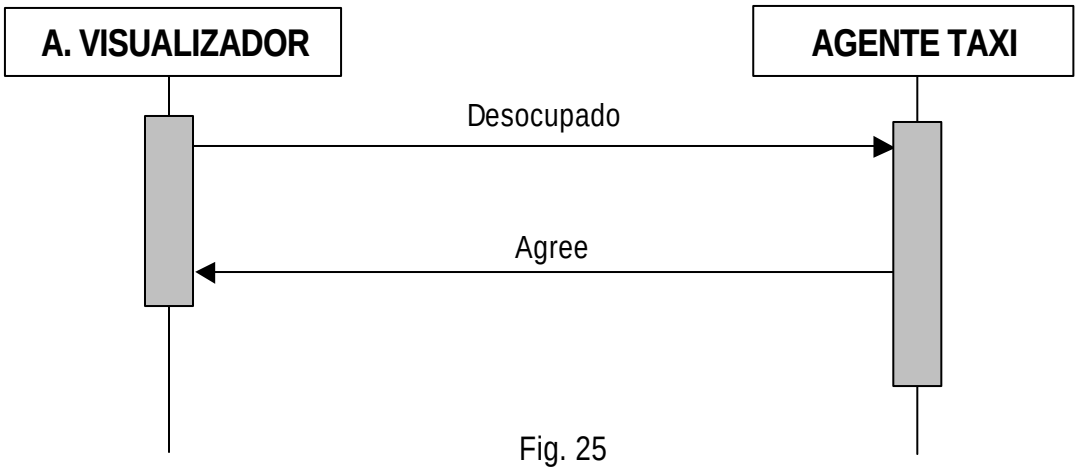



Fig. 25

4. **Cuando un cliente pide un taxi:** En este diagrama podemos observar todo el proceso temporal desde que un cliente pide un taxi hasta que se le sirve la petición

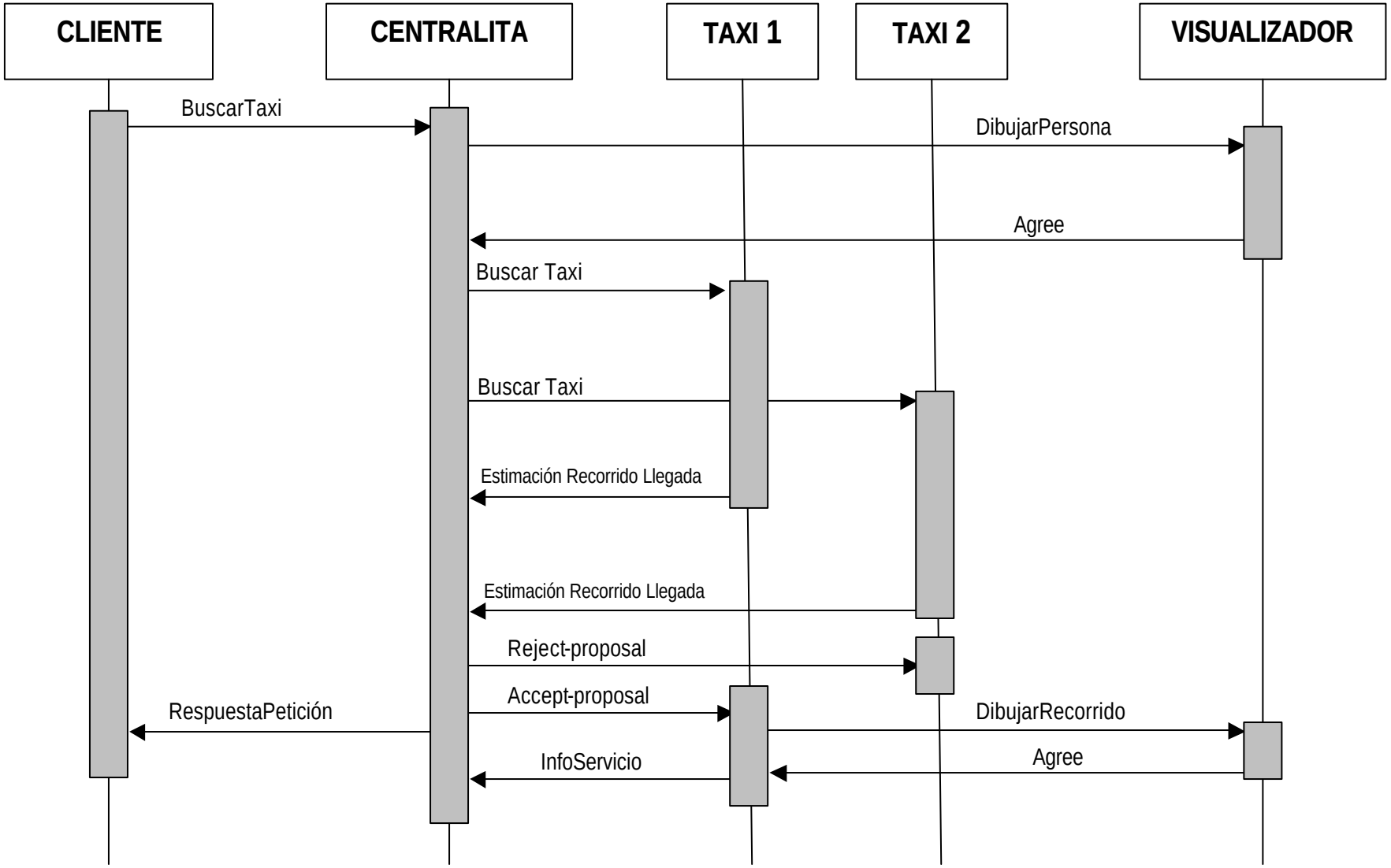



Fig. 26

5. **Borrar cliente:** Una vez el taxi ha realizado el servicio al cliente indicado, éste sería borrado del sistema.

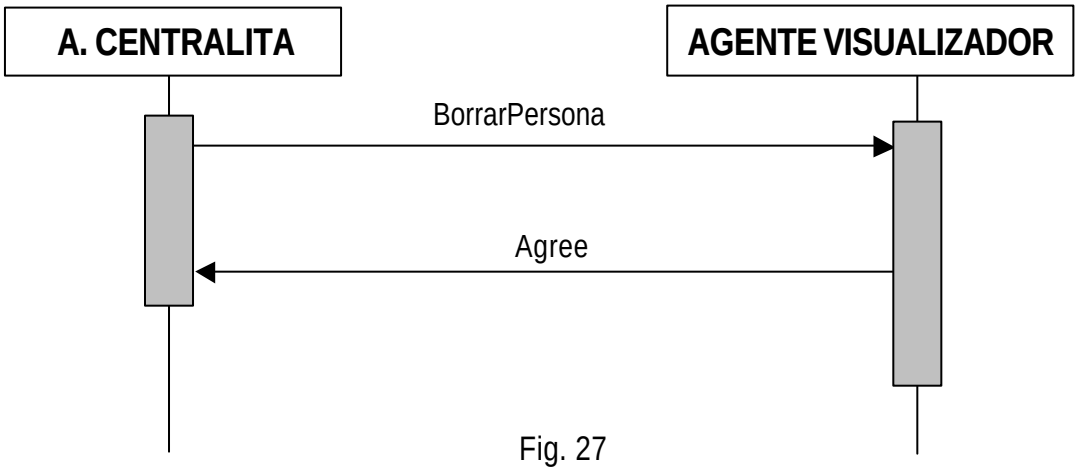



Fig. 27

6. **Borrar taxi:** Como hemos explicado anteriormente, los taxis trabajan por turnos de 8 horas. Una vez se ha realizado dicho turno, el taxi se elimina del sistema.

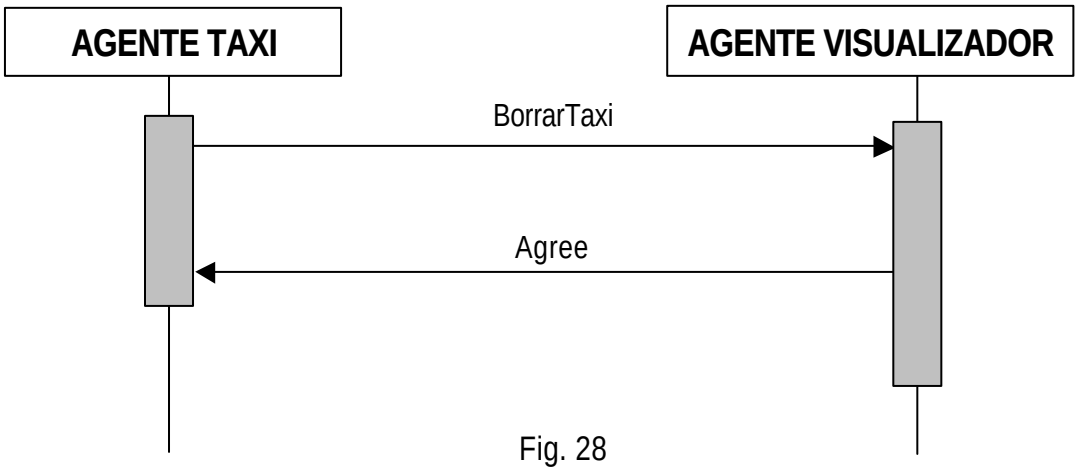



Fig. 28

## 6.2. Ontología

Como se ha comentado anteriormente la base de la comunicación entre agentes la aporta la ontología, que definirá el vocabulario utilizar en los mensajes que se enviarán unos a otros.

A continuación se muestran los CONCEPTOS, PREDICADOS y ACCIONES que se han utilizado para definir la ontología que usaremos en nuestro sistema:

### 6.2.1. Los Conceptos de la ontología

Para una buena comunicación entre los agentes, se utilizan los mensajes, pero dichos mensajes debe tener un contenido. Dicho contenido es el que se ha de diseñar e implementar en la ontología, en el caso que nos trata, los *Conceptos* son los objetos que se incluyen en los mensajes. En nuestro caso tenemos los conceptos: Posición, Petición, TaxiType, RespuestaPeticion, ListaCalles y RecorridoTaxi.

- **POSICIÓN:** Este objeto indica la posición exacta de una persona o de un taxi. Serán las coordenadas del mapa.

| Nombre Slot | Tipo |
|---|---|
| X | Integer |
| Y | Integer |

- **PETICIÓN:** Objeto que se utiliza para indicar que un cliente necesita un taxi. Incluye los datos entrados por el cliente usando el AgenteCliente: numero de personas, posición origen y posición destino.

| Nombre Slot | Tipo |
|---|---|
| Numpersonas | Integer |
| PosCliente | Posición |
| Destino | Posición |

- **TAXITYPE:** Objeto que nos proporciona toda la información disponible sobre un taxi en concreto.

| Nombre Slot | Tipo |
|---|---|
| IdTaxi | String |
| Propietario | String |
| NumLicencia | String |
| Matricula | String |
| Marca | String |
| Modelo | String |
| PosActual | Posición |
| Ocupado | Boolean |

- **RESPUESTAPETICIÓN:** Objeto que se utiliza para responder al cliente una vez se ha seleccionado el taxi que le ofrecerá el servicio, incluye el identificador del taxi, el precio del servicio, la lista de calles que forman el trayecto para permitir su visualización en la PDA y por ultimo el tiempo estimado de llegada del taxi hasta donde se encuentra el Cliente.

| Nombre Slot | Tipo |
|---|---|
| Respuesta | Boolean |
| IdTaxi | String |
| Precio | Integer |
| ListaCallesServicio | ListaCalles |
| Tiempo | Integer |

Cabe destacar, que en este objeto hemos añadido los atributos *ListaCallesServicio* y *Tiempo* respecto a la implementación original del Sr. Infante.

- **LISTACALLES:** Objeto que indica el recorrido que un taxi realiza.

| Nombre Slot | Tipo |
|---|---|
| ListaCalles | Lista de String |

- **RECORRIDOTAXI:** Este objeto se utiliza para dibujar el recorrido en la interficie gráfica. Será intercambiado entre el agente TAXI y el agente VISUALIZADOR. Con él obtenemos toda la información necesaria para poder indicar de forma precisa cada recorrido de los servicios.

| Nombre Slot | Tipo |
|---|---|
| CosteBuscar | Integer |
| CosteServir | Integer |
| Listacallesbuscar | ListaCalles |
| ListaCallesServir | ListaCalles |

### 6.2.2. Las Acciones de la ontología

Las acciones de la ontología son enviadas por un agente y recibidas por otro. Las acciones tienen un nombre y unos argumentos, siempre que estos sean necesarios). Las líneas de código dónde realmente indican qué hace dicha acción las tendrían los *agentes receptores*. En nuestro caso las acciones implementadas son: BuscarTaxi, EstimacionRecorridoLlegada, DameCallesAnuladas, Desocupado, BorrarPersona, DibujarPersona, BorrarRecorridoTaxi, DibujarRecorridoTaxi, DibujarTaxi, BorrarTaxi.

- **BUSCARTAXI:** Esta acción la inicia el cliente. Indica cuando la centralita debe informar a sus taxis para que les den sus recorridos y luego él escoger al que más le convenga. Los argumentos de dicha acción serían los siguientes:

| Nombre Argumento | Tipo |
|---|---|
| Petición | Petición |

- **ESTIMACIONRECORRIDOLLEGADA:** Esta acción es la que emite el taxi a la centralita una vez ha hecho todos los cálculos de recorridos necesarios para el servicio. Una vez recibidos por la centralita, se escogería el taxi que más convenga para el cliente.

| Nombre Argumento | Tipo |
|---|---|
| ListaCallesCamino | ListaCalles |
| ListaCallesServicio | ListaCalles |
| Ocupado | Boolean |

- **DAMECALLESANULADAS:** Esta es una acción que inicia el agente Taxi y el agente Visualizador para informarse de cuales son las calles que están anuladas. Esta acción no necesita ningún argumento.

- **DESOCUPADO:** Esta es una acción que inicia el agente Visualizador para comunicarle al Taxi que ya ha realizado la visualización pertinente se su recorrido y deja de estar ocupado. Esta acción no necesita ningún argumento.

- **BORRARPERSONA:** Está acción la inicia la centralita y la recibe el visualizador. Consiste en borrar a un cliente de la interficie gráfica representada por el agente visualizador. Los argumentos son:

| Nombre Argumento | Tipo |
|---|---|
| Posición | Posición |

- **DIBUJARPERSONA:** Es una acción que nos presenta en la interficie gráfica un cliente. También la inicia la centralita y la recibe el agente visualizador. Los argumentos son:

| Nombre Argumento | Tipo |
|---|---|
| Posición | Posición |

- **BORRARRECORRIDOTAXI:** Está acción la inicia el agente taxi y la recibe el visualizador. Consiste en borrar el recorrido de un taxi que ha finalizado su servicio. Los argumentos son:

| Nombre Argumento | Tipo |
|---|---|
| RecorridoTaxi | RecorridoTaxi |

- **DIBUJARRECORRIDOTAXI:** Es la acción contraria a borrarrecorridotaxi. Nos dibuja el recorrido del servicio de un taxi. Los argumentos son:

| Nombre Argumento | Tipo |
|---|---|
| RecorridoTaxi | RecorridoTaxi |

- **DIBUJARRTAXI:** Acción que inicia el agente taxi y recibe el agente visualizador. Dibuja en la interficie un taxi. Los argumentos son:

| Nombre Argumento | Tipo |
|---|---|
| Posición | Posición |

- **BORRARTAXI:** Acción que inicia el agente taxi y recibe el agente visualizador. Borra un taxi en la interficie, ya que, su turno de trabajo ha acabado. Los argumentos son:

| Nombre Argumento | Tipo |
|---|---|
| Posición | Posición |

### 6.2.3. Relación: Ontología – Servicios Agentes – DF Agent

En la implementación de la Ontología también se deben definir los Servicios y propiedades de cada agente. Esta información es muy útil en la plataforma cuando tengamos que buscar a un agente en concreto o conjunto de los mismos, puesto que informando al DF sobre los servicios que cada agente ofrece, posteriormente al requerir la realización de un cierto trabajo, solo tendremos que consultar el DF para saber que agente puede resolver con mayor eficiencia la tarea deseada, es a través de estos parámetros como podremos diferenciar un agente de otro.

A continuación se muestra el caso de TaxiOntology:

```
public static final String
  SERVICE_TYPE = "agent-cities.taxi.services",
  TAXI_SERVICE = "taxi",
  CENTRALITA_SERVICE = "centralita",
  VISUALIZADOR_SERVICE = "visualizador",
  TRAFICO_SERVICE = "trafico",
  CLIENTE_SERVICE = "cliente";
```

Para realizar el registro en el DF se necesita indicar el TYPE y el NAME del servicio que el agente realiza. Para dejar mas claro este punto, mostraremos un caso práctico: Registrar un agente taxi.

En este caso los parámetros que registraremos serán el TYPE=SERVICE_TYPE y el NAME=TAXI_SERVICE. Si necesitásemos indicar más información para ser más concretos, podríamos crear propiedades de los distintos servicios, en este caso incluiremos como propiedad la ciudad donde opera el taxi.

```
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
      sd.setName(TaxiOntology.TAXI_SERVICE);
      sd.setType(TaxiOntology.SERVICE_TYPE);
Property pp=new Property();
      p.setname(TaxiOntology.CIUDAD_PROPERTY);
      p.setValue("L'Hospitalet del Llobregat");
sd.addProperties(p);
dfd.addServices(sd);
dfd.setName(getAID());

try {
  DFService.register(this, dfd);
  System.out.println("Registrado en el DF");
}catch (JADE.domain.FIPAException e) {
  System.out.println("No se ha podido registral al DF");
  doDelete();
}
```

# 7. IMPLEMENTACIÓN

## 7.1. Clases y directorios

Para estructurar todo el sistema descrito anteriormente, el código se ha dividido en dos bloques bien diferenciados: AGENTES y ONTOLOGY, cada uno de los cuales corresponde a un package. Pasemos ahora a describir estos dos módulos.

## 7.2. Package AGENTES

En el package *Agentes*, que se encuentra en el directorio con su mismo nombre, encontramos todas las clases referentes al código fuente de cada uno de los agentes, ficheros de inicialización, ficheros de información sobre los mapas y todas las clases encargadas de la interficie gráfica. A continuación haremos una breve descripción de las clases de este package:

### 7.2.1. AgenteCliente.java

Este es el agente asociado a los clientes del sistema. Su cometido es enviar una petición a la centralita requiriendo un taxi para realizar un trayecto, para ello, tomará los datos entrados por el cliente a través de la interficie grafica y una vez realizadas las operaciones pertinentes recibir una respuesta de la Centralita y mostrará por la pantalla de la PDA, los detalles del servicio. Para conseguir esto, el agente utilizara el protocolo de comunicación FIPA-Request y interactuará con el cliente a través de la GUI creada.

Este agente utiliza para su funcionamiento las clases definidas en los siguientes ficheros: *AgenteCliente.java, VentanaCliente.java, VentanaMapa.java, MapaCliente.java y CoordCliente.java.*

El código principal del agente se encuentra en *AgenteCliente.java*, es aquí donde esta definido su comportamiento, incluyendo desde registrarse al DF, la realización del FIPA-Request, o incluso los pasos a realizar al finalizar su ejecución.

Los otros 4 ficheros se encargarían de la GUI creada, que estructuralmente esta formada por 2 ventanas y un canvas para dibujar. Una de las ventanas sería la llamada ventana principal (VentanaCliente.java), donde se introducen los valores de la petición y se reciben los detalles del servicio. La otra ventana (VentanaMapa.java) se ocupa de sustentar el canvas (MapaCliente.java), que es donde dibujaremos el mapa, con los trayectos. *El CoordCliente.java* se encarga de la matriz donde traduciremos coordenadas por número de pixels en la pantalla de la PDA.

Para la realización del agente se han tomado ciertas decisiones que son necesarias comentar, inicialmente el agente arranca con un SimpleBehaviour como comportamiento definido. Este behaviour, al constatar que el Cliente tiene una petición para enviar, lanzara el FIPA-Request con el mensaje idóneo a la Centralita, cabe destacar que una vez completado el ciclo del Request y por tanto lo que sería el funcionamiento del agente, este no es eliminado del DF, como ocurría en la implementación inicial a cargo del Sr. Infante. Mas al contrario, el agente permanece latente en espera de nuevas peticiones por parte del usuario.

La realización de la interface ha sido íntegramente implementada utilizando las librerías AWT, por ser estas las únicas que soporta la maquina virtual de la PDA (JeodeVM). La ventana principal, la forma un juego de menús, etiquetas y botones, con ActionListeners asociados. La ventana de apoyo al canvas realiza de puente entre la ventana principal y el canvas. Posee varios métodos que hacen de

interprete entre ambos, en un intento de independizar al máximo las clases entre si. Por ultimo el canvas, es posiblemente la parte mas sofisticada de la GUI, dado que tiene que gestionar toda la parte de dibujo. Tiene un MouseListener asociado, que cazará los eventos producidos por el Stylus del usuario. A cada evento, decidirá que hacer, desde marcar Origen, destino o borrar marcas anteriores. El método *Paint*, que es el que dibuja, ha sido equipado con un entramado de booleans para realizar lo que llamamos, *dibujo selectivo*. En otras palabras, cada vez que se llama al paint dibujamos solo lo que nos interesa, sea marca de origen/destino, trayectoria, los edificios… De esta forma conseguimos reducir el trabajo del procesador, que no tiene que repintar cada vez todos los elementos, con todos los problemas que ello comporta en un dispositivo ciertamente limitado como es una PDA.

Además hemos implementado un gestor de dibujo de mapas, el cual a partir de los datos de una ciudad introducidos vía fichero de texto, dibuja la ciudad incluyendo los nombres de las calles que se cruzan en cada nodo. Este fichero, se carga a través del menú situado en la Ventana principal de la aplicación, cosa que facilita el cambio de mapa por parte del cliente, que no necesita ser un avezado conocedor del tema. El principal problema que esta modificación presenta, se encuentra precisamente en la creación de ese fichero de texto con los datos asociados a una ciudad o entramado de calles. En la actualidad, es una tarea ciertamente costosa, que solo puede ser realizada por alguien que conozca perfectamente la distribución por pixels de la pantalla de la PDA y la posición de los nodos en esa pantalla. Una opción para solventar esto, sería la creación de un editor de mapas visual, que se ocupase de todo el trabajo duro y permitiese a un usuario cualquiera, poder construir su propio mapa sin dificultad.

### 7.2.2. AgenteCentralita.java

Este es el agente encargado de gestionar las peticiones de los clientes. Tiene un comportamiento simple que controla la recepción de las peticiones de los clientes y otro que gestiona dichas peticiones mediante la información obtenida de cada taxi. Utiliza el protocolo de comunicación FIPA-Request para comunicarse con el Visualizador y con el cliente, y FIPA-Contract Net para comunicarse con los taxis de la plataforma.

Dado que hemos introducido la posibilidad de intercambiar los mapas por parte del cliente, hemos de actuar en consecuencia con la Centralita, para que pueda consultar la ciudad correcta y no presente información incoherente al cliente. Al entender la Centralita como un servidor situado en la sede de la empresa de taxis, suponemos que el mapa, cambiará en contadas ocasiones, por tanto hemos decidido, que el fichero con los datos de la ciudad, serán pasados por parámetro y cargados al iniciar el servidor. Esto significa que para cambiar el mapa deberíamos reiniciar el servidor, pero como ya hemos dicho esto pasaría en contadas ocasiones.

Otro de los puntos clave en este agente, es el proceso de selección del taxi idóneo para el servicio, al método *longitud_camino* que a todos los efectos era el encargado de dar los valores a tener en cuenta en la selección final, se le ha añadido el factor del coste por tramo recorrido, de esta forma, no solo valoraremos la distancia a la que esta el taxi del cliente, sino que añadimos el tiempo que tardara el taxi en circular por los tramos que lo separan del cliente.

Los métodos más importantes son:

**RequestResponder extends AchieveREResponder:** método que espera la recepción de un mensaje REQUEST. Se ejecuta cuando recibe una petición de un cliente.
**CNInitiator extends ContractNetInitiator:** método que se encarga de la comunicación del protocolo Contract Net. Se ejecuta cuando la centralita recibe una petición y se encarga de escoger la propuesta más apropiada para el cliente.

**RequestInitiator extends AchieveREInitiator:** método que se encarga de iniciar una comunicación REQUEST. Se utiliza para enviar al Visualizador las acciones a realizar.
**void RegistrarDF():** método que registra en el DF.
**ACLMessage EnviarMsgVisualizador(String accion,Posicion pos):** método que crea un mensaje donde indicaremos la acción que deseamos realizar al visualizador.
**ACLMessage EnviarMsgTaxis(Action a):** método que te crea un mensaje para luego enviarlo a todos los taxis.
**CacheUPDAte extends Thread implements Runnable:** thread que se autoejecuta cada cierto tiempo. Lo utilizamos para la actualización de la Cache de taxis.
**searchTaxis extends OneShotBehaviour:** Behaviour relacionado con el Thread *CacheUPDAte* (explicado anteriormente), que actualiza la cache de los taxis.
**List LlenarCacheTaxis(String type,String name):** método que te llena una lista con todos los taxis que encuentra en el DF, es decir, todos los taxis que en ese momento están trabajando.
**String getAgentDF(String type,String name):** método que se encarga de buscar en el DF al agente que interese.

### 7.2.3. AgenteTaxi.java

Este es el objeto encargado de controlar los comportamientos del agente Taxi. Implementa una serie de comportamientos que definirán su manera de actuar, en este caso son cuatro:

**RequestInitiator extends AchieveREInitiator:** behaviour que se encarga de iniciar una comunicación del tipo REQUEST. Se utiliza para la petición de calles anuladas al agente tráfico.
**CNResponder extends ContractNetResponder:** este behaviour controla la recepción de los mensajes Contract Net. Se utiliza para la recepción de las peticiones que envía la centralita y para enviar las propuestas de cada taxi.
**RequestResponder extends AchieveREResponder:** behaviour que controla la recepción de mensajes REQUEST. Se utiliza para saber cuando el taxi deja de estar ocupado, ya que, nos avisa el agente Visualizador mediante el envío de un mensaje.
**RegistroTurno extends OneShotBehaviour:** comportamiento simple que va actualizando el turno de cada taxi. Se utiliza para saber cuando finaliza un turno y comienza otro.

Al igual que ocurría con la Centralita, el agente taxi tomará por parámetro el fichero de texto con los datos de la ciudad por donde se mueve. En este caso también suponemos que no es habitual que el taxi cambie de mapa, de todos modos, para seleccionar otro simplemente debería reiniciar el sistema.

Los métodos más importantes son:

**void RegistrarDF():** método que registra en el DF.
**TaxiType readInfoTaxi(String pathname):** método que extrae de un fichero toda la información necesaria para la inicialización del taxi.
**ACLMessage EnviarMsgDameCalles():** método que crea el mensaje que se encarga de pedir las calles anuladas al agente tráfico.
**ACLMessage EnviarMsgVisualizador(String accion,Posicion pos):** método que crea un nuevo mensaje donde le indicaremos las acciones que deseamos realizar al visualizador (DibujarTaxi o BorrarTaxi).
**ACLMessage EnviarMsgVisRec(String accion,int costebuscar,int costeservir,ListaCalles buscar,ListaCalles servir):** método que crea un nuevo mensaje donde le indicaremos las acciones que deseamos realizar al visualizador (DibujarRecorridoTaxi o BorrarRecorridoTaxi). Los parámetros son toda la información

necesaria para que el agente visualizador pueda dibujar/borrar todo el recorrido.
**turnoUPDAte extends Thread implements Runnable:** thread que se autoejecuta cada cierto tiempo. Lo utilizamos para saber cuando un turno finaliza y otro comienza.
**void DeRegistrarDF():** método encargado de elimnar el taxi del DF y enviar un mensaje al Visualizador para eliminar al taxi de la interficie.
**String getAgentDF(String type,String name):** método que se encarga de buscar en el DF al agente que interese.

A continuación explicaremos los métodos más importantes para la implementación del algoritmo A* y el de la distancia de Manhattan.

**int[] Aestrella(InfoMap[][] mapa,int ninici,int nfinal):** método que implementa el algoritmo A*. Los parámetros que necesita son: *mapa* es el grafo de la ciudad, con todos sus vértices y arestas, *ninici* es el nodo donde inicará la búsqueda y *nfinal* es el nodo donde finalizará la búsqueda del recorrido.
**int g(int inici_x,int inici_y,int final_x, int final_y):** método que nos da la distancia de Manhattan.
**int[] primer(int[] c):** método que te da el primer recorrido del camino.
**int[] reste(int[] c):** método que te da el resto del recorrido del camino.
**boolean objectiu(int[] c,int nfinal):** método que nos indica si el final del camino es el nodo final que buscamos.
**int[] successors(int[] c,int[] lc):** método que te crea los sucesores del camino actual y también te los concatena con el camino ya realizado.
**int[] eliminar_redundants(int[] c):** método que te elimina los caminos redundantes.
**int[] ordenar(int[] l):** método que te ordena los caminos.

### 7.2.4. AgenteTrafico.java

Este es el objeto encargado de controlar los comportamientos del agente Tráfico. Tiene dos comportamientos que definirán su manera de actuar:

**RequestResponder extends AchieveREResponder:** este behaviour es el que controla la recepción de los REQUEST. Se ejecuta cuando recibe un mensaje de *DameCallesAnuladas*, ya sea del agente *Taxi* o del agente *Visualizador*.
**DarCalles extends OneShotBehaviour:** comportamiento que se encarga de coger las calles anuladas de un fichero dependiendo de la hora del sistema.

Los métodos más importantes son:

**horaUPDAte extends Thread implements Runnable:** thread que se autoejecuta cada cierto tiempo. Lo utilizamos para saber qué fichero de calles anuladas ha de coger.
**void RegistrarDF():** método que registra en el DF.
**JADE.util.LEAP.List readCallesAnuladas(String pathname):** método que se encarga de leer las calles anuladas de un fichero.

### 7.2.5. AgenteVisualizador.java

Este es el objeto encargado de controlar los comportamientos del agente Visualizador. Tiene tres comportamientos simples que son los que definirán su forma de actuar:

**RequestResponder extends AchieveREResponder:** este behaviour es el que controla la recepción de los REQUEST. Se ejecuta cuando recibe un mensaje con acciones como BorrarPersona, DibujarPersona, BorrarTaxi, DibujarTaxi, BorrarRecorridoTaxi o DibujarRecorridoTaxi.
**RequestInitiator extends AchieveREInitiator:** este behaviour es el que controla el inicio de la comunicación de un protocolo FIPA-Request. Se ejecuta cuando desearmos enviar un mensaje de DameCallesAnuladas a Tráfico.
**DarCalles extends OneShotBehaviour:** comportamiento simple que nos avisa de cuando hemos de actualizar las calles anuladas y hemos de enviar un mensaje al agente tráfico.

Los métodos más importantes son:

**void RegistrarDF():** método que registra en el DF.
**ACLMessage EnviarMsgDameCalles():** método que crea el mensaje para enviar la acción DameCallesAnuladas a Trafico.
**String getAgentDF(String type,String name):** método que se encarga de buscar en el DF al agente que interese.
**ACLMessage EnviarMsgDesocupado(ACLMessage req):** método que crea el mensaje para informar al taxi de que ya no está ocupado.
**horaUPDAte extends Thread implements Runnable:** thread que se autoejecuta cada cierto tiempo. Lo utilizamos para saber cuando debemos actualizar las calles anuladas.
**String getAgentDF(String type,String name):** método que se encarga de buscar en el DF al agente que interese.

## 7.3. Package ONTOLOGY

En el package *Ontology*, el cual se encuentra en el directorio con su mismo nombre, encontramos todas las clases referentes a la ontología del sistema que ya fue comentada en el apartado 6.2, hagamos ahora una breve descripción de las clases que podemos encontrar en este paquete:

1. **TaxiOntology.java:** clase java donde encontramos la definición completa de la ontología del sistema.
2. **TaxiType.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos TaxiType.
3. **RespuestaPeticion.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos RespuestaPeticion.
4. **RecorridoTaxi.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos RecorridoTaxi.
5. **Posicion.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos Posición.
6. **Peticion.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos Petición.
7. **ListaCalles.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos ListaCalles.
8. **EstimacionRecorridoLlegada.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos EsimaciónRecorridoLlegada.

9. **DibujarTaxi.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos DibujarTaxi.
10. **DibujarRecorridoTaxi.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos DibujarRecorridoTaxi.
11. **DibujarPersona.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos DibujarPersona.
12. **Desocupado.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos Desocupado.
13. **DameCallesAnuladas.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos DameCallesAnuladas.
14. **BuscarTaxi.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos BuscarTaxi.
15. **BorrarTaxi.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos BorrarTaxi.
16. **BorrarRecorridoTaxi.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos BorrarRecorridoTaxi.
17. **BorrarPersona.java:** clase java que contiene los métodos get y set para las propiedades del tipo de datos BorrarPersona.

Para el lector que desee más información sobre las ontologias implementadas, le proponemos una lectura del apartado 6.2.

# 8. DEMOSTRACIÓN PRÁCTICA

Realizaremos una demostración del SMA creado, utilizando para ello, los dos mapas de que disponemos. El primero de ellos será el mapa clásico del Sr. Infante que representa el eixample de bcn, el segundo mapa, es un entramado de calles ficticio que nos mostrará edificios de diferentes formas y calles diagonales. Para el ejemplo clásico, proveeremos pantallas de muestra del AgenteVisualizador, pero no así para el segundo caso, dado que mostraría resultados incoherentes que no nos aportarían nada.

Para los dos ejemplos, trabajaremos con los mismos taxis y los mismos ficheros de calles cortadas, por lo tanto los describiremos en este apartado preliminar.

### FICHEROS CALLES CORTADAS

**mañana.txt:**25 26 71 60 78 77*
**tarde.txt:**18 29 50 51 101 102*
**noche.txt:**2 13 86 87 91 92*

En la figura 29 podemos ver esto reflejado sobre el mapa de nodos, consideraremos el color verde para las calles cortadas de mañana, rojo para las de tarde y amarillo para las de mañana.

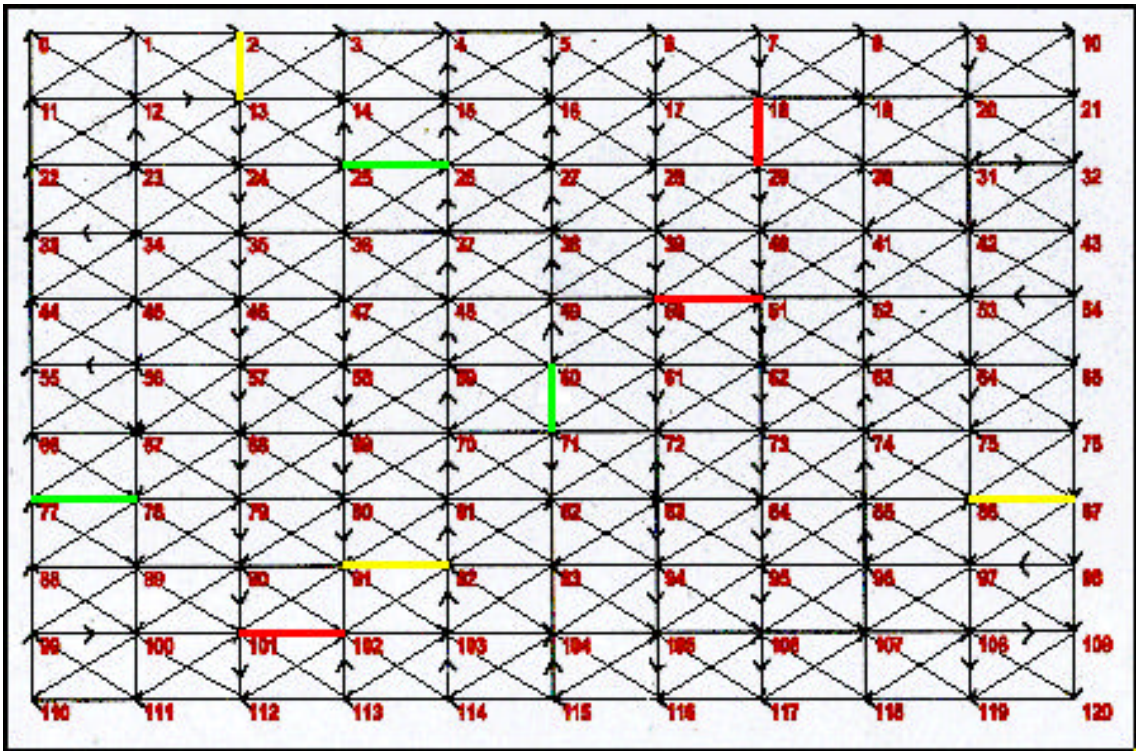


Fig. 29 Calles anuladas sobre mapa de nodos

**TAXIS CARGADOS EN EL SISTEMA**

**taxi1.txt:**
0001
Ferry Corsten
12545125
B-5705-SJ
SEAT
CORDOBA
mañana
3 1

**taxi2.txt:**
0002
Armin van Buuren
11237658
B-4418-LM
RENAULT
CLIO
noche
8 5

**taxi3.txt:**
0003
Tiesto
39446688
B-1234-OM
FORD
SCORT
tarde
9 7

**taxi4.txt:**
0004
Shasha
97347168
B-8912-SW
PEUGEOT
206
tarde
3 7

**taxi5.txt:**
0005
Pascal Feos
33224567
M-1354-SJ
FORD
MONDEO
mañana
3 9

**taxi6.txt:**
0006
Paul Oakenfold
19436759
L-1234-SW
WOLKSWAGEN
BEATTLE
noche
7 2

**taxi7.txt:**
0007
Johnny Vicious
19436759
L-1234-SW
WOLKSWAGEN
BEATTLE
noche
1 1

**taxi8.txt:**
0008
Jan Johnston
12545125
B-5705-SJ
SEAT
CORDOBA
mañana
5 4

**taxi9.txt:**
0009
Iio Vaiio
11237658
B-4418-LM
RENAULT
CLIO
tarde
7 6

## 8.1. Ejemplo con mapa clásico

El fichero de texto donde encontramos los detalles de la ciudad para este caso será el *DatosCallesPARACLASICO.txt*. Invitamos al lector a que lo abra en un editor y lo analice por sí mismo (su formato ya ha sido explicado en el apartado 5.9). Nosotros nos quedaremos con lo verdaderamente importante, que es el grafo dirigido que representa. Podemos comprobar en la figura 30 que este grafo es algo diferente al genérico que mostramos en la explicación del apartado 5.10. En este caso no existen conexiones diagonales entre los nodos debido a que el mapa clásico es totalmente cuadriculado, y sólo hay calles horizontales y verticales.

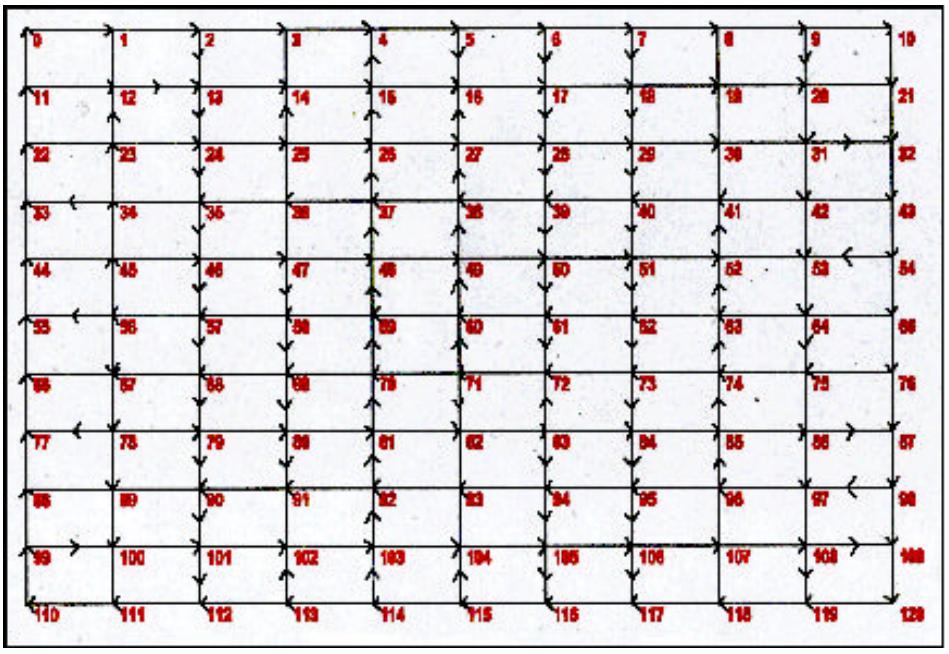


Fig. 30 – Grafo dirigido del mapa Clásico

Mostremos ahora un ejemplo de la pantalla grafica ofrecida por el AgenteVisualizador para irnos familiarizando con el entorno. La figura 31, nos muestra la pantalla, una vez iniciado el sistema en horario de mañana.

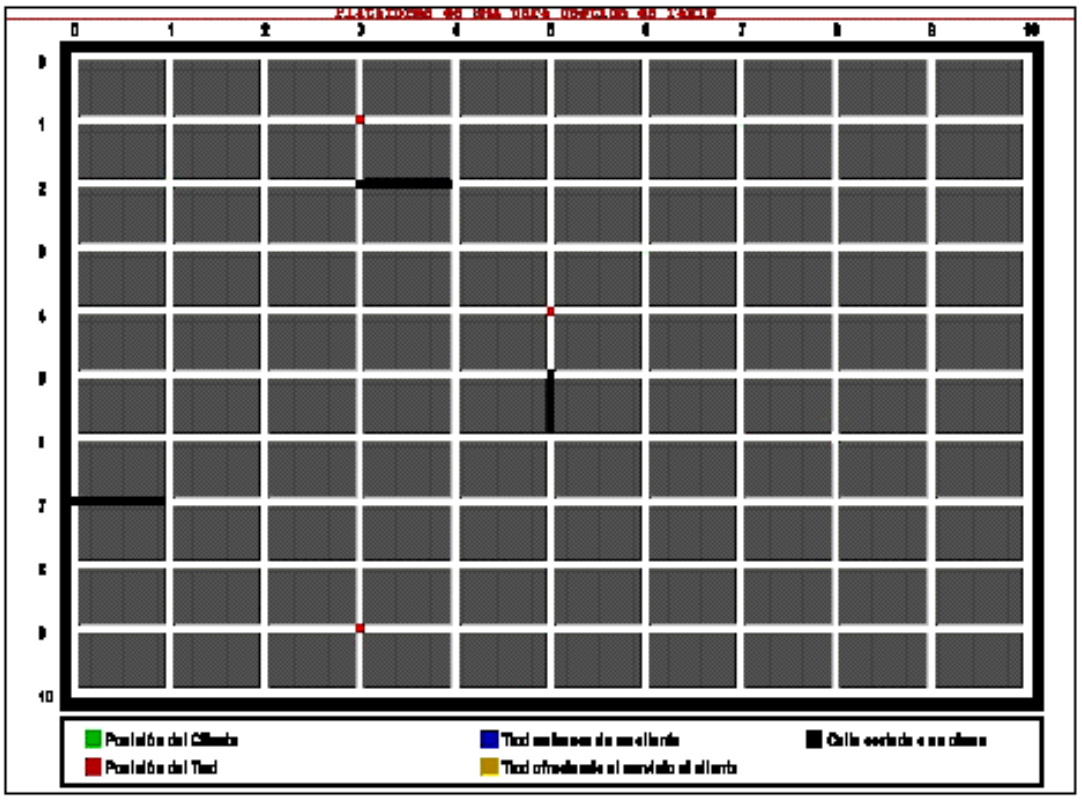


Fig. 31 – Interficie del AgenteVisualizador iniciado por la mañana

Pasemos ahora a la prueba en si. Procederemos arrancando el Main container, que estará situado en el PC, con el siguiente comando en MS-DOS:

*java -cp ".\j2se\lib\JADELEAP.jar;.\pfc" JADE.Boot -gui Centralita:Agentes.AgenteCentralita(DatosCallesPARACLASICO.txt);Trafico:Agentes.AgenteTrafico;Visualizador:Agentes.AgenteVisualizador;Taxi1:Agentes.AgenteTaxi(DatosCallesPARACLASICO.txt);Taxi6:Agentes.AgenteTaxi(DatosCallesPARACLASICO.txt);Taxi3:Agentes.AgenteTaxi(DatosCallesPARACLASICO.txt)*

A continuación explicaremos qué hace dicha instrucción con sus correspondientes parámetros:

**-gui:** especifica que deseamos visualizar el entorno gráfico de JADE.

**Centralita: Agentes.AgenteCentralita(DatosCallesPARACLASICO.txt):** Indica que deseamos cargar en la plataforma un agente denominado CENTRALITA que será controlado a través de la clase java AgenteCentralita que se encuentra en un paquete llamado Agentes e indicamos el parámetro que le pasamos.

**Trafico:Agentes.AgenteTrafico:** Indica que deseamos cargar en la plataforma un agente denominado TRAFICO que será controlado a través de la clase java AgenteTrafico que se encuentra en un paquete llamado Agentes.

**Visualizador:Agentes.AgenteVisualizador:** Indica que deseamos cargar en la plataforma un agente denominado VISUALIZADOR que será controlado a través de la clase java AgenteVisualizador que se encuentra en un paquete llamado Agentes.

**Taxi1:Agentes.AgenteTaxi (DatosCallesPARACLASICO.txt):** Indica que deseamos cargar en la plataforma un agente denominado TAXI1 que será controlado a través de la clase java AgenteTaxi que se encuentra en un paquete llamado Agentes e indicamos el parámetro que le pasamos.

Podemos constatar que los 3 taxis que cargamos corresponden a diferentes franjas horarias, por tanto solo uno de ellos entrará en servicio. La figura 32 nos muestra la pantalla inicial en el PC.

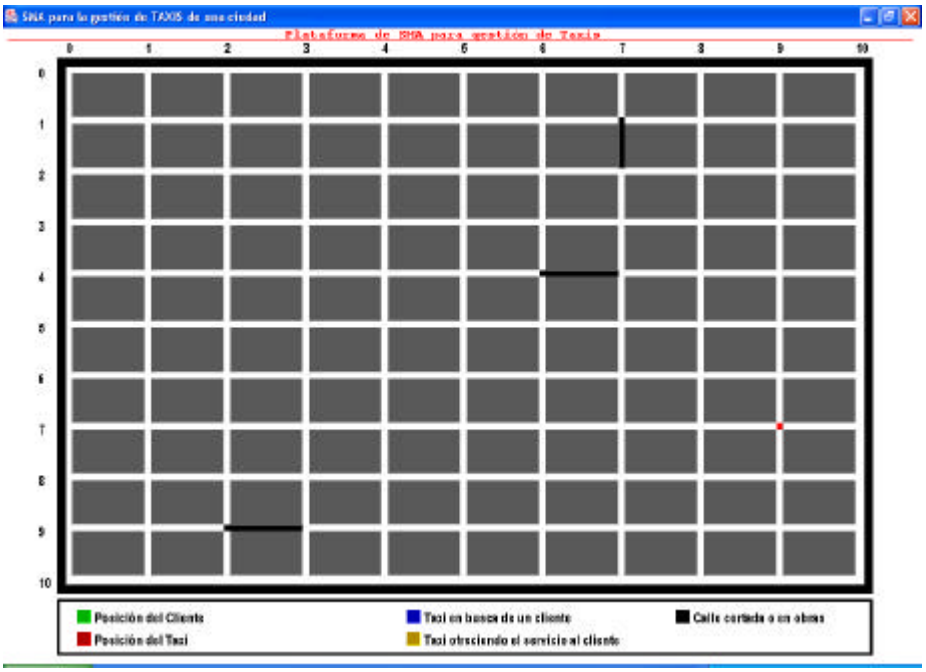



Fig. 32 – Interficie del AgenteVisalizador al iniciar la prueba

Una vez la Estación principal está funcionando, cargaremos el AgenteCliente en la PDA, que iniciará su propio contenedor y se conectará a la plataforma JADE-LEAP creada en el PC.

En la PDA ejecutaremos la siguiente instrucción, de la forma que ya comentamos en el apartado 4.3.1.

*18#"\Windows\evm.exe" -Djeode.evm.console.local.keep=TRUE -cp "/java/JADELEAPPjava.jar;/java/pfc/" JADE.Boot -container -host 192.168.0.1 -port 1099 Cliente:Agentes.AgenteCliente*

Una vez lanzada la aplicación se conectará a la plataforma JADE-LEAP que tenemos funcionando en el PC. Al conseguirlo, lanzará el AgenteCliente y podremos ver la pantalla principal del agente en nuestra PDA que corresponde a la figura 33.


Fig. 33 – Pantalla principal del AgenteCliente

Un vistazo a la gui de la plataforma JADE-LEAP nos permitirá ver los dos containers, el principal y el creado en la PDA, con los respectivos agentes cargados en ellos (ver figura 34).

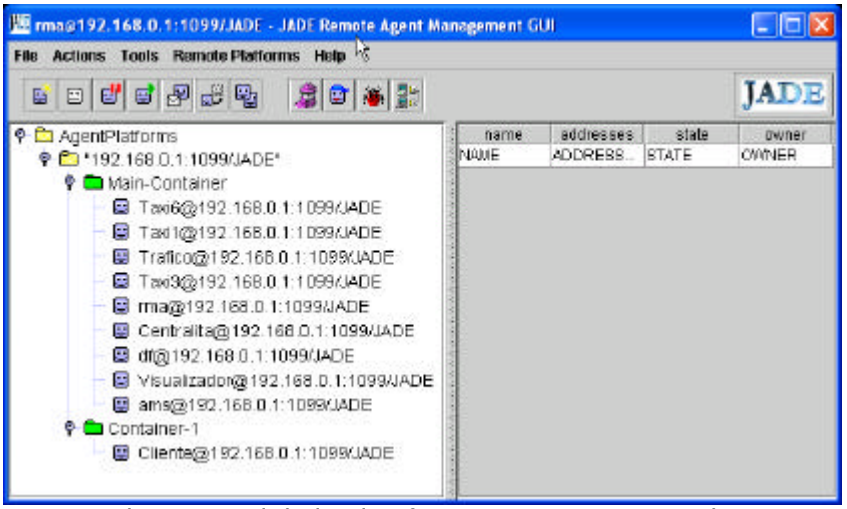



Fig. 34 – gui de la plataforma JADE-LEAP creada

Una vez cargado el programa, procederemos seleccionando el mapa a utilizar mediante el menú de la pantalla principal. Para esta prueba seleccionaremos el fichero *clasico.txt*, que contiene el mapa cuadriculado. Con el fichero cargado, abriremos el mapa, pulsando el botón *mapa*. Y pulsaremos con el Stylus el origen y el destino deseados quedando la pantalla del mapa como muestra la figura 35.


Fig. 35 – Mapa del AgenteCliente con origen/destino marcados

Una vez entrados todos los datos necesarios pulsaremos *enviar*, para lanzar el request a la centralita y esperaremos la respuesta con los datos del servicio y el trayecto. A continuación podemos ver los mensajes enviados por los agentes cargados en el main container, situado en el PC:

*C:\JADE\add-ons\LEAP>java -cp ".\j2se\lib\JADELEAP.jar;.\pfc" JADE.Boot -gui Cen tralita:Agentes.AgenteCentralita(DatosCalles.txt);Trafico:Agentes.AgenteTrafico; Visualizador:Agentes.AgenteVisualizador;Taxi1:Agentes.AgenteTaxi(DatosCalles.txt );Taxi6:Agentes.AgenteTaxi(DatosCalles.txt);Taxi3:Agentes.AgenteTaxi(DatosCalles .txt)*

*This is JADE 3.0b1 - 2003/03/19 17:08:01*
*downloaded in Open Source, under LGPL restrictions,*
*at http://JADE.cselt.it/*

*Listening for intra-platform commands on address:*
*jicp://192.168.0.1:1099*
*IOR:000000000000001149444C3A464950412F4D54533A312E3000000000000000010000000000000064000102000000000C3139322E3136382E302E3100044000000000019AFABCB0000000002F5154B39000000080000000000000000A000000000000010000000100000020000000000001000100000002050100010001002000010109000000100010100*
*Agent container Main-Container@JADE-IMTP://192.168.0.1 is ready.*
*[Visualizador]: SE HA REGISTRADO EN EL DF*

*[Centralita]:SE HA REGISTRADO EN EL DF*
*[Trafico]: SE HA REGISTRADO EN EL DF*
*[Taxi3]: SE HA REGISTRADO EN EL DF*
****************SE HA USADO EL FICHERO DE CALLES TARDE.TXT***********
*[Trafico]: HA RECIBIDO UN REQUEST DE DAMECALLESANULADAS Y VA A ENVIAR LAS CALLES ANULADAS*
*[Trafico]:EL ENVIO DE LA LISTA DE CALLES SE HA HECHO CORRECTAMENTE*
*[Trafico]: HA RECIBIDO UN REQUEST DE DAMECALLESANULADAS Y VA A ENVIAR LAS CALLES ANULADAS*
*[Trafico]:EL ENVIO DE LA LISTA DE CALLES SE HA HECHO CORRECTAMENTE*
*[Visualizador]: HA RECIBIDO LAS CALLES ANULADAS DE TRAFICO*
*[Taxi3]: HA RECIBIDO LAS CALLES ANULADAS DE TRAFICO Y ACTUALIZO EL MAPA*
*[Centralita]: HA RECIBIDO UN REQUEST DE BUSCARTAXI Y VA A INICIAR UN CFP*
*[Centralita]:buscando los agentes taxis...*
*[Centralita]:ha encontrado 1 agentes TAXI.*
*[Taxi3]: HA RECIBIDO EL CFP CORRECTAMENTE Y VAMOS A ENVIAR UNA PROPUESTA*
*[Taxi3]: CAMINO ENCONTRADO!!!!!!!!!*
*[Taxi3]: CAMINO ENCONTRADO!!!!!!!!!*
*[Centralita]:HA RECIBIDO TODAS LAS PROPUESTAS HABIDAS Y POR HABER*
*[Centralita]:HA ESCOGIDO UNA PROPUESTA DE TODAS LAS ENVIADAS.....*
*.....*
*[Taxi3]:HA RECIBIDO UN ACCEPTPROPOSAL//////////////////////////////////////*

Una vez aceptada la proposición del Taxi3, la centralita envía la respuesta final al cliente. En la figura 36 podemos ver el trayecto dibujado, y en la figura 37 los datos referentes al servicio recibidos.


Fig. 36 – Mapa del AgenteCliente con el trayecto dibujado


Fig. 37 – Pantalla principal del AgenteCliente con los datos del servicio

Por ultimo podemos comprobar las trayectorias de llegada del taxi al cliente (azul) y del servicio realizado (naranja) en la pantalla del AgenteVisualizador, tal y como mostramos en la figura 38.

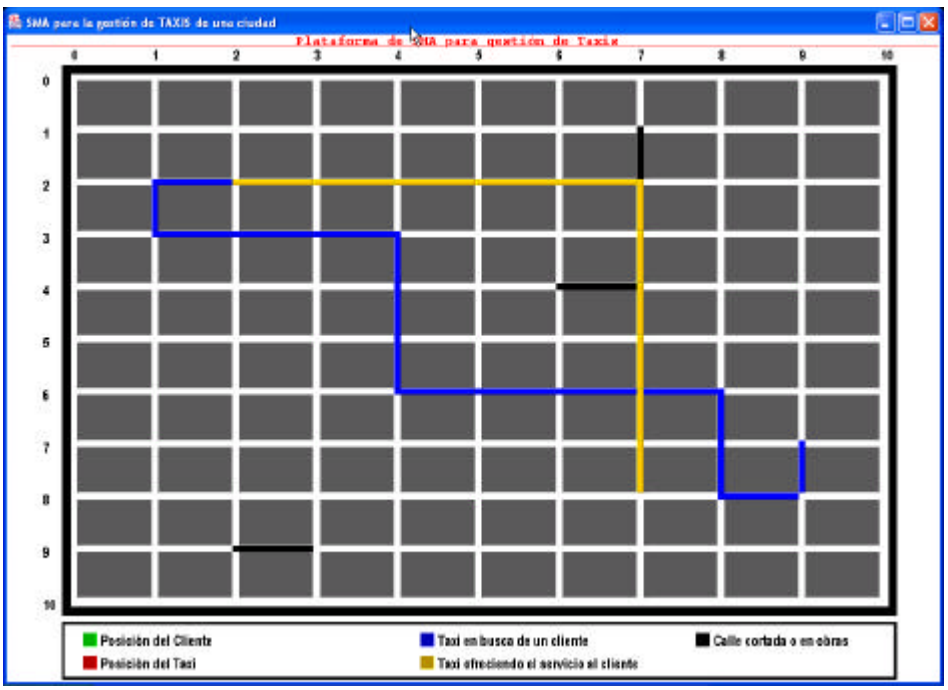



Fig. 38 – Pantalla del AgenteVisualizador una vez completada la petición de servicio

Vamos ha comentar la lista de posibles mensajes que pueden aparecer por la consola MSDOS, donde se ejecuta el main container y por tanto la gran parte de los agentes cargados en el sistema.

**AGENTECENTRALITA**

***[Centralita]: SE HA REGISTRADO EN EL DF:*** Indica que se ha iniciado en el sistema.
***[Centralita]: HA RECIBIDO UN REQUEST DE BUSCARTAXI Y VA INICIAR UN CFP:*** La Centralita ha recibido una petición de un cliente y va a avisar a los diferentes taxis que estén realizando su turno en esos momentos.
***[Centralita]: BUSCANDO LOS AGENTES TAXIS…:*** La Centralita está accediendo al DF para saber qué taxis son los que en ese momento trabajan.
***[Centralita]: HA ENCONTRADO 3 AGENTES TAXI:*** Centralita ha encontrado 3 agentes taxi en el DF.
***[Centralita]: HA RECIBIDO TODAS LAS PROPUESTAS HABIDAS Y POR HABER:*** Indica que la centralita ha recibido todas las propuestas de todos los taxis que en ese momento trabajan.
***[Centralita]: HA ESCOGIDO UNA PROPUESTA DE TODAS LAS ENVIADAS….:*** Indica que Centralita ya ha escogido una de las propuestas enviadas por los taxis del sistema.

**AGENTETAXI**

***[TaxiN]: SE HA REGISTRADO EN EL DF:*** El TaxiN se ha registrado en el DF, es decir, es su turno de trabajo.
***[TaxiN]: EL AGENTE HA SIDO ELIMINADO DEL DF:*** El agente ha sido eliminado del DF, es decir, no debía trabajar porque no era su turno laboral.
***[TaxiN]: HA RECIBIDO UN CFP CORRECTAMENTE Y VAMOS A ENVIAR UNA PROPUESTA:*** El TaxiN ha recibido el CFP de Centralita indicándole que hay un cliente que desea un taxi y éste le va a enviar una propuesta a la Centralita para que luego ella elija el que más le convenga para el cliente.
***[TaxiN]: CAMINO ENCONTRADO !!!!!!!!:*** El agente TaxiN ha encontrado el camino que le había pedido la Centralita.
***[TaxiN]: HA RECIBIDO UN ACCEPT PROPOSAL !!!!!!!!!!!!!:*** La Centralita ha escogido al TaxiN para que realice un servicio.
***[TaxiN]: HA RECIBIDO UN REJECT:*** La Centralita no lo ha escogido para realizar un servicio a un cliente.
***[TaxiN]: HA RECIBIDO LAS CALLES ANULADAS DE TRÁFICO:*** Ha recibido correctamente las calles anuladas que tráfico le ha enviado.

**AGENTETRAFICO**

***[Trafico]: SE HA REGISTRADO EN EL DF:*** Indica que se ha iniciado en el sistema.
***[Trafico]: HA RECIBIDO UN REQUEST DE DAMECALLESANULADAS Y VA A ENVIAR LAS CALLES ANULADAS:*** algún agente Taxi o el agente Visualizador le están pidiendo que le envíen las calles anuladas.
***[Trafico]: EL ENVÍO DE LA LISTA DE CALLES SE HA HECHO CORRECTAMENTE:*** El TaxiN ha recibido las calles anuladas de Tráfico correctamente.
***[Trafico]: FICHERO CALLES CORTADAS/OBRAS ACTUALIZADO. Próxima actualización: Thu Jan 09 21:42:21 CET 2003:*** Este mensaje indica que tráfico ha mirado la hora del sistema y ha escogido el fichero que le tocaba y a su vez indica la hora y el día de la próxima actualización.

**AGENTEVISUALIZADOR**

***[Visualizador]: SE HA REGISTRADO EN EL DF:*** Indica que se ha iniciado en el sistema.
***[Visualizador]: HA RECIBIDO LAS CALLES ANULADAS DE TRÁFICO:*** Ha recibido correctamente las calles anuladas que tráfico le ha enviado.

## 8.2. Ejemplo con mapa alternativo

Una vez analizado el funcionamiento con el mapa clásico mostraremos por ultimo, el funcionamiento utilizando un mapa alternativo, con calles diagonales y edificios de diferentes dimensiones. Puesto que la idea es muy similar a la demostración anterior nos limitaremos a mostrar los resultados, ya que no nos aportaría nada extendernos mas en este punto. Recordemos que el AgenteVisualizador no esta preparado para adaptarse a mapas alternativos así no mostraremos sus resultados, puesto que serían incorrectos.

Para esta simulación cargaremos desde el AgenteCliente el fichero *callejero.txt*, el cual incluye los datos sobre el mapa alternativo, cabe destacar que los agentes centralita y taxi deberán cargar el fichero *DatosCallesPARACALLEJERO.txt*. En la figura 39 mostramos el esquema del grafo dirigido que representa las calles de esta ciudad.

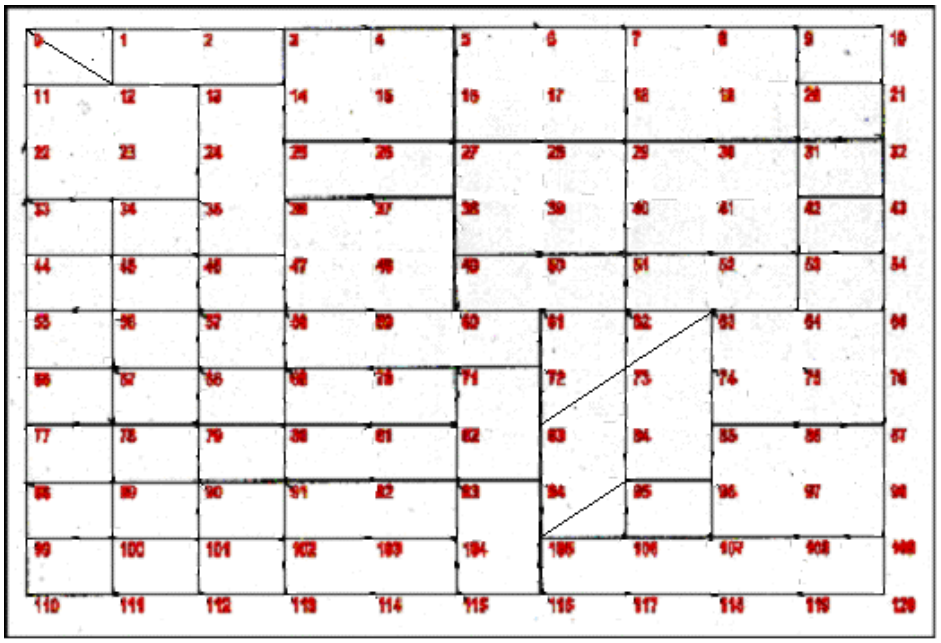


Fig. 39 – grafo dirigido del mapa alternativo

Una vez tenemos el mapa idóneo procederemos como en el ejemplo anterior, seleccionaremos origen y destino, número de personas y enviremos la petición. En la Figura 40 podemos observar el nuevo mapa con las selecciones realizadas.


Fig. 40 – mapa del agentecliente con el mapa alternativo.

Una vez nuestra petición será procesada, nos contestaran con los datos habituales, tiempo de espera, precio del servicio, identificador del taxi y el trayecto del servicio. En la figura 41 podemos observar el camino seleccionado y como detalle relevante, como el trayecto toma tanto direcciones horizontales,verticales y diagonales. La pantalla con el resto de datos no la mostraremos por considerar que no aporta nada, es muy similar al caso anterior.


Fig. 41 – trayecto dibujado en el mapa alternativo

# 9. CÓDIGO FUENTE

A continuación se incluye el código fuente JAVA de todas las clases JAVA que forman el sistema.

## 9.1. PACKAGE ONTOLOGY

### 9.1.1. TAXIONTOLOGY.JAVA

```
/**************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en codigo de Juan Infante García(16-10-2002)

ONTOLOGÍA
**************************************************/
package Ontology;

import java.io.*;
import JADE.content.*;
import JADE.content.onto.*;
import JADE.content.abs.*;
import JADE.content.schema.*;
import JADE.content.lang.*;

public class TaxiOntology extends Ontology {

   public static final String NAME = "Taxi-Ontology";

   private static Ontology theInstance = new TaxiOntology(BasicOntology.getInstance());

   public static Ontology instance() {
      return theInstance;
   }

   public String getName() {
      return NAME;
   }

   //*********************************
   //    TEMPLATES
   //*********************************
   public static final String
     SERVICE_TYPE = "agent-cities.taxi.services",
     TAXI_SERVICE = "taxi",
     CENTRALITA_SERVICE = "centralita",
     VISUALIZADOR_SERVICE = "visualizador",
     TRAFICO_SERVICE = "trafico",
     CLIENTE_SERVICE = "cliente";
```

```
//**********************************
//    VOCABULARIO (CONCEPTOS)
//**********************************
public static final String POSICION        = "posicion";
public static final String PETICION        = "peticion";
public static final String TAXITYPE       = "taxitype";
public static final String RESPUESTAPETICION= "respuestapeticion";
public static final String LISTACALLES   = "listacalles";
public static final String RECORRIDOTAXI = "recorridotaxi";

//**********************************
//    VOCABULARIO (ACCIONES)
//**********************************
public static final String BUSCARTAXI="buscartaxi";
public static final String ESTIMACIONRECORRIDOLLEGADA ="estimacionrecorridollegada";
public static final String DAMECALLESANULADAS="damecallesanuladas";
public static final String BORRARPERSONA ="borrarpersona";
public static final String DIBUJARPERSONA="dibujarpersona";
public static final String BORRARTAXI="borrartaxi";
public static final String DIBUJARTAXI   ="dibujartaxi";
public static final String BORRARRECORRIDOTAXI="borrarrecorridotaxi";
public static final String DIBUJARRECORRIDOTAXI="dibujarrecorridotaxi";
public static final String OCUPADO           ="ocupado";
public static final String LISTACALLESCAMINO ="listacallescamino";
public static final String LISTACALLESSERVICIO  ="listacallesservicio";
public static final String DESOCUPADO           ="desocupado";

private TaxiOntology(Ontology base) {
   super(NAME,base);
   try {

     PrimitiveSchema stringSchema  = (PrimitiveSchema)getSchema(BasicOntology.STRING);
     PrimitiveSchema integerSchema = (PrimitiveSchema)getSchema(BasicOntology.INTEGER);
     PrimitiveSchema booleanSchema = (PrimitiveSchema)getSchema(BasicOntology.BOOLEAN);
PrimitiveSchema dateSchema    = (PrimitiveSchema)getSchema(BasicOntology.DATE);

//**********************************
//    CONCEPTOS
//**********************************

     add(new ConceptSchema(POSICION),Posicion.class);
     add(new ConceptSchema(PETICION),Peticion.class);
     add(new ConceptSchema(TAXITYPE),TaxiType.class);
     add(new ConceptSchema(LISTACALLES),ListaCalles.class);
     add(new ConceptSchema(RESPUESTAPETICION),RespuestaPeticion.class);
     add(new ConceptSchema(RECORRIDOTAXI),RecorridoTaxi.class);

       ConceptSchema cs = (ConceptSchema)getSchema(POSICION);
cs.add("x",integerSchema);
cs.add("y",integerSchema);

cs = (ConceptSchema) getSchema(TAXITYPE);
```

```
            cs.add("idtaxi",stringSchema);
            cs.add("propietario",stringSchema);
            cs.add("numlicencia",stringSchema);
            cs.add("matricula",stringSchema);
            cs.add("marca",stringSchema);
            cs.add("modelo",stringSchema);
            cs.add("posactual",(ConceptSchema) getSchema(POSICION));
            cs.add("ocupado",booleanSchema);

            cs = (ConceptSchema)getSchema(PETICION);
            cs.add("numpersonas",integerSchema);
            cs.add("poscliente",(ConceptSchema) getSchema(POSICION));
                 cs.add("destino",(ConceptSchema) getSchema(POSICION));

            cs = (ConceptSchema)getSchema(LISTACALLES);
            cs.add("listacalles",integerSchema,0,ObjectSchema.UNLIMITED);

                 cs = (ConceptSchema)getSchema(RESPUESTAPETICION);
            cs.add("respuesta",booleanSchema);
            cs.add("idtaxi",stringSchema);
            cs.add("precio",integerSchema);
            cs.add("tiempo",integerSchema);
            cs.add("listacallesservicio",(ConceptSchema) getSchema(LISTACALLES));


            cs = (ConceptSchema)getSchema(RECORRIDOTAXI);
            cs.add("costebuscar",integerSchema);
            cs.add("costeservir",integerSchema);
            cs.add("listacallesbuscar",(ConceptSchema) getSchema(LISTACALLES));
            cs.add("listacallesservir",(ConceptSchema) getSchema(LISTACALLES));

            //**********************************
            //    ACCIONES
            //**********************************
                 add(new AgentActionSchema(BUSCARTAXI),BuscarTaxi.class);
                 add(new
AgentActionSchema(ESTIMACIONRECORRIDOLLEGADA),EstimacionRecorridoLlegada.class);
                 add(new AgentActionSchema(DAMECALLESANULADAS),DameCallesAnuladas.class);
                 add(new AgentActionSchema(DESOCUPADO),Desocupado.class);
                 add(new AgentActionSchema(BORRARPERSONA),BorrarPersona.class);
                 add(new AgentActionSchema(DIBUJARPERSONA),DibujarPersona.class);
                 add(new AgentActionSchema(BORRARTAXI),BorrarTaxi.class);
                 add(new AgentActionSchema(DIBUJARTAXI),DibujarTaxi.class);
                 add(new AgentActionSchema(BORRARRECORRIDOTAXI),BorrarRecorridoTaxi.class);
                 add(new AgentActionSchema(DIBUJARRECORRIDOTAXI),DibujarRecorridoTaxi.class);

            AgentActionSchema as = (AgentActionSchema)getSchema(BUSCARTAXI);
            as.add(PETICION,(ConceptSchema) getSchema(PETICION));

            as = (AgentActionSchema)getSchema(ESTIMACIONRECORRIDOLLEGADA);
            as.add(LISTACALLESCAMINO,(ConceptSchema) getSchema(LISTACALLES));
            as.add(LISTACALLESSERVICIO,(ConceptSchema) getSchema(LISTACALLES));
```

```
        as.add(OCUPADO,booleanSchema);

        as = (AgentActionSchema)getSchema(DAMECALLESANULADAS);

        as = (AgentActionSchema)getSchema(DESOCUPADO);

        as = (AgentActionSchema)getSchema(BORRARPERSONA);
        as.add(POSICION,(ConceptSchema) getSchema(POSICION));

        as = (AgentActionSchema)getSchema(DIBUJARPERSONA);
        as.add(POSICION,(ConceptSchema) getSchema(POSICION));

        as = (AgentActionSchema)getSchema(BORRARRECORRIDOTAXI);
        as.add(RECORRIDOTAXI,(ConceptSchema) getSchema(RECORRIDOTAXI));

        as = (AgentActionSchema)getSchema(DIBUJARRECORRIDOTAXI);
        as.add(RECORRIDOTAXI,(ConceptSchema) getSchema(RECORRIDOTAXI));

        as = (AgentActionSchema)getSchema(BORRARTAXI);
        as.add(POSICION,(ConceptSchema) getSchema(POSICION));

        as = (AgentActionSchema)getSchema(DIBUJARTAXI);
        as.add(POSICION,(ConceptSchema) getSchema(POSICION));

             } catch(OntologyException oe) { oe.printStackTrace(); }

        }
    }
```

### 9.1.2. TAXIONTYPE.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en codigo de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
*****************************************************/
package Ontology;

import JADE.content.*;

public class TaxiType implements Concept {

   private String IDTaxi;
   private String Propietario;
   private String NumLicencia;
   private String Matricula;
   private String Marca;
   private String Modelo;
   private String Turno;
   private Posicion posactual;
```

```
            private boolean ocupado;

            public  TaxiType(String  IDTaxi,String  Propietario,String  NumLicencia,String
    Matricula,String Marca,String Modelo,String Turno,Posicion posactual,boolean ocupado) {
                this.IDTaxi=IDTaxi;
                this.Propietario=Propietario;
                this.NumLicencia=NumLicencia;
                this.Matricula=Matricula;
                this.Marca=Marca;
                this.Modelo=Modelo;
                this.Turno=Turno;
                this.posactual=posactual;
                this.ocupado=ocupado;
            }

            public String getIDTaxi() {
                return IDTaxi;
            }

            public String getPropietario(){
                return Propietario;
            }

            public String getNumLicencia(){
                return NumLicencia;
            }

            public String getMatricula(){
                return Matricula;
            }

            public String getMarca(){
                return Marca;
            }

            public String getModelo(){
                return Modelo;
            }

            public String getTurno(){
                return Turno;
            }

            public Posicion getPosActual(){
                return posactual;
            }

            public boolean getOcupado(){
                return ocupado;
            }

            public void setIDTaxi(String ID){
```

```
        IDTaxi=ID;
    }

    public void setPropietario(String prop){
        Propietario=prop;
    }

    public void setNumLicencia(String num){
        NumLicencia=num;
    }

    public void setMatricula(String mat){
        Matricula=mat;
    }

    public void setMarca(String marca){
        Marca=marca;
    }

    public void setModelo(String mod){
        Modelo=mod;
    }

    public void setTurno(String tur){
        Turno=tur;
    }

    public void setPosActual(Posicion posactual){
        this.posactual=posactual;
    }

    public void setOcupado(boolean ocupado){
        this.ocupado=ocupado;
    }

}
```

### 9.1.3. RESPUESTAPETICION.JAVA

```
/**************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
**************************************************/
package Ontology;

import JADE.content.*;

public class RespuestaPeticion implements Concept {
```

```
    private boolean Respuesta;
    private String IDTaxi;
    private int Precio;
    private ListaCalles ListaCallesServicio; // para pasarle el recorrido al cliente
    private int Tiempo;

    public RespuestaPeticion() { }

    public boolean getRespuesta() {
        return Respuesta;
    }

    public void setRespuesta(boolean Resp) {
        Respuesta=Resp;
    }

    public String getIDTaxi() {
        return IDTaxi;
    }

    public void setIDTaxi(String taxi) {
       this.IDTaxi=taxi;
    }

    public int getPrecio() {
        return Precio;
    }

    public void setPrecio(int precio) {
       this.Precio=precio;
    }

    public ListaCalles getListaCallesServicio() {return ListaCallesServicio;}

    public void setListaCallesServicio(ListaCalles l){
   this.ListaCallesServicio=l;
    }

    public int getTiempo() {
        return Tiempo;
    }

    public void setTiempo(int tiempo) {
       this.Tiempo=tiempo;
    }

}
```

### 9.1.4. RECORRIDOTAXI.JAVA

```
/**************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
**************************************************/
package Ontology;

import JADE.content.*;
import JADE.util.LEAP.*;

public class RecorridoTaxi implements Concept {

    private int costebuscar;
    private int costeservir;
    private ListaCalles listacallesbuscar;
    private ListaCalles listacallesservir;

    public RecorridoTaxi() {}

    public int getCosteBuscar() {
        return costebuscar;
    }

    public void setCosteBuscar(int cb) {
        costebuscar=cb;
    }

    public int getCosteServir() {
        return costeservir;
    }

    public void setCosteServir(int cs) {
        costeservir=cs;
    }

    public ListaCalles getListaCallesBuscar() {
        return listacallesbuscar;
    }

    public void setListaCallesBuscar(ListaCalles l) {
        this.listacallesbuscar=l;
    }

    public ListaCalles getListaCallesServir() {
        return listacallesservir;
    }

    public void setListaCallesServir(ListaCalles l) {
```

```
        this.listacallesservir=l;
    }
}
```

### 9.1.5. POSICION.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
*****************************************************/
package Ontology;

import JADE.content.*;

public class Posicion implements Concept {

   private int x;
   private int y;

   public int getx() { return x;}

   public void setx(int x){
      this.x=x;
   }

   public int gety() { return y;}

   public void sety(int y){
      this.y=y;
   }
}
```

### 9.1.6. PETICION.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
*****************************************************/
package Ontology;

import JADE.content.*;

public class Peticion implements Concept {

   private int numpersonas;
   private Posicion poscliente;
```

```
        private Posicion destino;

    public int getnumpersonas(){return numpersonas;}

    public void setnumpersonas(int num){
       this.numpersonas=num;
    }

    public Posicion getposcliente() {return poscliente;}

    public void setposcliente(Posicion pos){
       this.poscliente=pos;
    }

    public Posicion getdestino() {return destino;}

    public void setdestino(Posicion dest){
       this.destino=dest;
    }

}
```

### 9.1.7. LISTACALLES.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
*****************************************************/
package Ontology;

import JADE.content.*;
import JADE.util.LEAP.*;

public class ListaCalles implements Concept {

    private List ListaCalles;

    public List getListaCalles() { return ListaCalles;}

    public void setListaCalles(List LCalles) {
       this.ListaCalles=LCalles;
    }

}
```

### 9.1.8. ESTIMACIONRECORRIDOLLEGADA.JAVA

```
/**************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
**************************************************/
package Ontology;

import JADE.content.*;

public class EstimacionRecorridoLlegada implements AgentAction {

    private ListaCalles ListaCallesCamino;
    private ListaCalles ListaCallesServicio;
    private boolean ocupado;

    public EstimacionRecorridoLlegada() {}

    public boolean getOcupado() {return ocupado;}

    public void setOcupado(boolean o){
   this.ocupado=o;
    }

    public ListaCalles getListaCallesCamino() {return ListaCallesCamino;}

    public void setListaCallesCamino(ListaCalles l){
   this.ListaCallesCamino=l;
    }

    public ListaCalles getListaCallesServicio() {return ListaCallesServicio;}

    public void setListaCallesServicio(ListaCalles l){
   this.ListaCallesServicio=l;
    }


}
```

### 9.1.9. DIBUJARTAXI.JAVA

```
/**************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
**************************************************/
package Ontology;
```

```
import JADE.content.*;

public class DibujarTaxi implements AgentAction {

    private Posicion pos;

    public DibujarTaxi() {}
    public DibujarTaxi(Posicion p) {
   this.pos=p;
    }

    public Posicion getposicion() {return pos;}

    public void setposicion(Posicion p){
   this.pos=p;
    }

}
```

### 9.1.10. DIBUJARRECORRIDOTAXI.JAVA

```
/*******************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
*******************************************************/
package Ontology;

import JADE.content.*;
import JADE.util.LEAP.*;

public class DibujarRecorridoTaxi implements AgentAction {

    private RecorridoTaxi r;

    public DibujarRecorridoTaxi() {}
    public DibujarRecorridoTaxi(RecorridoTaxi r) {
   this.r=r;
    }

    public RecorridoTaxi getRecorridoTaxi() {return r;}

    public void setRecorridoTaxi(RecorridoTaxi r){
   this.r=r;
    }

}
```

### 9.1.11. DIBUJARPERSONA.JAVA

```
/**************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
**************************************************/
package Ontology;

import JADE.content.*;

public class DibujarPersona implements AgentAction {

    private Posicion pos;

    public DibujarPersona() {}

    public Posicion getposicion() {return pos;}

    public void setposicion(Posicion p){
   this.pos=p;
    }
}
```

### 9.1.12. DESOCUPADO.JAVA

```
/**************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
**************************************************/
package Ontology;

import JADE.content.*;

public class Desocupado implements AgentAction {

    public Desocupado() { }

}
```

### 9.1.13. DAMECALLESANULADAS.JAVA

```
/**************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)
```

```
ONTOLOGÍA (clases de java asociadas)
*****************************************************/
package Ontology;

import JADE.content.*;

public class DameCallesAnuladas implements AgentAction {

    public DameCallesAnuladas() { }

}
```

### 9.1.14. BUSCARTAXI.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
*****************************************************/
package Ontology;

import JADE.content.*;

public class BuscarTaxi implements AgentAction {

    private Peticion p;

    public BuscarTaxi() {}

    public BuscarTaxi(Peticion p) {
   this.p=p;
    }

    public Peticion getpeticion() {return p;}

    public void setpeticion(Peticion pet){
       this.p=pet;
    }
}
```

### 9.1.15. BORRARTAXI.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
*****************************************************/
package Ontology;
```

```
import JADE.content.*;
import JADE.util.LEAP.*;

public class BorrarTaxi implements AgentAction {

    private Posicion pos;

    public BorrarTaxi() {}

    public Posicion getposicion() {return pos;}

    public void setposicion(Posicion p){
   this.pos=p;
    }
}
```

### 9.1.16. BORRARRECORRIDOTAXI.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
*****************************************************/
package Ontology;

import JADE.content.*;
import JADE.util.LEAP.*;

public class BorrarRecorridoTaxi implements AgentAction {

    private RecorridoTaxi r;

    public BorrarRecorridoTaxi() {}
    public BorrarRecorridoTaxi(RecorridoTaxi r) {
   this.r=r;
    }

    public RecorridoTaxi getRecorridoTaxi() {return r;}

    public void setRecorridoTaxi(RecorridoTaxi r){
   this.r=r;
    }
}
```

### 9.1.17. BORRARPERSONA.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

ONTOLOGÍA (clases de java asociadas)
*****************************************************/
package Ontology;

import JADE.content.*;
import JADE.util.LEAP.*;

public class BorrarPersona implements AgentAction {

    private Posicion pos;

    public BorrarPersona() {}
    public BorrarPersona(Posicion p) {
   this.pos=p;
    }

    public Posicion getposicion() {return pos;}

    public void setposicion(Posicion p){
   this.pos=p;
    }
}
```

## 9.2. PACKAGE AGENTES

### 9.2.1. ACCIONAREALIZAR.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)
*****************************************************/
package Agentes;

import java.awt.Color;

public class AccionARealizar{

   public int accion;
   public int col;
   public int fil;
   public Color color;
   public JADE.util.LEAP.List calles;
```

```
   public   AccionARealizar(int   accion,   int   col,   int   fil,Color   color,
   JADE.util.LEAP.List calles){
          this.accion=accion;
          this.col=col;
          this.fil=fil;
          this.color=color;
          this.calles=calles;
   }
}
```

### 9.2.2. AGENTECENTRALITA.JAVA

```
/*******************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

AGENTE CENTRALITA
*******************************************************/
package Agentes;

import Ontology.*;

import java.io.*;

import java.util.*;
import JADE.core.*;
import JADE.core.behaviours.*;
import JADE.proto.*;
import JADE.lang.acl.*;
import JADE.domain.FIPAAgentManagement.*;
import JADE.domain.*;
import JADE.content.*;
import JADE.content.onto.basic.*;
import JADE.content.onto.*;
import JADE.content.lang.*;
import JADE.content.lang.sl.*;
import JADE.content.lang.sl.*;
import JADE.content.lang.Codec.*;
import JADE.content.onto.basic.*;

public class AgenteCentralita extends Agent {

    Agent myCentralita=this;

    private ContentManager manager  = (ContentManager) getContentManager();
    private Codec codec=new SLCodec();
    private Ontology ontology=TaxiOntology.instance();

    private final long MILLIS_PER_8HOURS = 120 * 1000;
    Date deathLine = new Date(System.currentTimeMillis() + MILLIS_PER_8HOURS);  //Para
   poder actualizar la cache de Taxis cada 8 HORAS
```

```
 JADE.util.LEAP.List lista_clientes=new JADE.util.LEAP.ArrayList();
 int PRECIOPASO=60; //céntimos
 int seg_segmento=30; //30 segundos en recorrer un segmento
 DatosMapa dm;

 List TaxiCache = new ArrayList();
 Iterator TaxiCacheIterator;

//*************************************************************************************
****
 //Metodo que registra en el DF
 void RegistrarDF() {
//REGISTRAMOS la ontología y el lenguaje
manager.registerLanguage(codec,codec.getName());
manager.registerOntology(ontology,ontology.getName());

     //Preparación para dar de alta en el DF
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setName(TaxiOntology.CENTRALITA_SERVICE);
sd.setType(TaxiOntology.SERVICE_TYPE);
dfd.addServices(sd);
dfd.setName(getAID());

try {
    DFService.register(this, dfd);
    System.out.println("["+getLocalName()+"]:"+"SE HA REGISTRADO EN EL DF");
    }
         catch (JADE.domain.FIPAException e) {
         System.out.println("["+getLocalName()+"]:"+"No se ha podido registrar en el
DF");
         doDelete();
         }

 }//Fin de RegistrarDF


//*************************************************************************************
*****************************
 //Método que se encarga de buscar todos los agentes TAXI de la plataforma y te los
devuelve en una lista "cache"
 private List LlenarCacheTaxis(String type,String name) {
System.out.println("["+this.getLocalName()+"]:"+"buscando los agentes taxis...");
List agentsFound = new ArrayList();
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(type);
sd.setName(name);
dfd.addServices(sd);

try {
       SearchConstraints c = new SearchConstraints();
```

```
        c.setMaxResults(new Long(-1));  //buscaremos infinitos resultados (estilo
JADE2.x)
  DFAgentDescription[] DFAgents = DFService.search(this,dfd,c);

  int i=0;
  while ((DFAgents != null) && (i<DFAgents.length)) {
    DFAgentDescription agent = DFAgents[i];
    i++;
    Iterator services = agent.getAllServices();
    boolean found = false;
    ServiceDescription service = null;
    while (services.hasNext() && !found) {
      service = (ServiceDescription)services.next();
      found = (service.getType().equals(type) && service.getName().equals(name));
    }
    if (found) {
          agentsFound.add(agent.getName());    //Introduzco en agentsFound el AID
de un agente TAXI
    }
  }
} catch (FIPAException e) {
    System.out.println("["+getLocalName()+"]:"+"Se  ha  producido  un  error:
"+e.getMessage());
       }

System.out.println("["+this.getLocalName()+"]:"+"ha encontrado "+agentsFound.size()
+" agentes TAXI.");
return agentsFound;
 }


//*************************************************************************
*****************************
 //Método que se encarga de buscar los agentes que deseo en el DF
 //TaxiOntology.SERVICE_TYPE,TaxiOntology.TAXI_SERVICE
 private String getAgentDF(String type,String name) {
String agentFound = new String();
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(type);
sd.setName(name);
dfd.addServices(sd);

try {
  SearchConstraints c = new SearchConstraints();
       c.setMaxResults(new Long(-1));  //buscaremos infinitos resultados (estilo
JADE2.x)
  DFAgentDescription[] DFAgents = DFService.search(this,dfd,c);

  int i=0;
  while ((DFAgents != null) && (i<DFAgents.length)) {
    DFAgentDescription agent = DFAgents[i];
```

```
    i++;
    Iterator services = agent.getAllServices();
    boolean found = false;
    ServiceDescription service = null;
    while (services.hasNext() && !found) {
      service = (ServiceDescription)services.next();
      found = (service.getType().equals(type) && service.getName().equals(name));
    }
    if (found) {
      AID aux=agent.getName();
           agentFound=aux.getName();    //Introduzco en agentsFound el AID de un
agente TAXI
    }
  }
} catch (FIPAException e) {
    System.out.println("["+getLocalName()+"]:"+"Se  ha  producido  un  error:
"+e.getMessage());
     }
return agentFound;
 }


//*************************************************************************************
*****************************
 //método que simplemente te monta TaxiCache y TaxiCacheIterator. Luego lanza el
UPDAteCache
 private class searchTaxis extends OneShotBehaviour {
searchTaxis(Agent agent) {
   super(agent);
}

public void action() {
   try {
       Thread.sleep(1000);
   } catch (Exception e) {}
   //Construyo la cache del Agente Centralita que contendrá los Taxis que encuentre
en la plataforma
   TaxiCache                                                                    =
LlenarCacheTaxis(TaxiOntology.SERVICE_TYPE,TaxiOntology.TAXI_SERVICE);
   TaxiCacheIterator = TaxiCache.iterator();
   if                                          (!TaxiCacheIterator.hasNext())
{System.out.println("["+myAgent.getLocalName()+"]:"+" no ha encontrado ningún
TAXI.");}
   new CacheUPDAte(deathLine,(Agent)myAgent);
}
 }


//*************************************************************************************
*******************************
 //Thread que actualiza la "cache" de TAXIS
 class CacheUPDAte extends Thread implements Runnable {
```

```
     long millis;
Agent myAgent;

CacheUPDAte(Date deathLine, Agent agent) {
   super();
   millis = deathLine.getTime() - System.currentTimeMillis();
   if(millis < 0)
       {
          millis=0;
       }
   myAgent = agent;
   this.start();
}

public void run(){
   try {
       //Cuando entra aquí es que debe hacer una nueva actualización (ya ha pasado
el deathLine)
       Thread.sleep(millis);
       addBehaviour(new searchTaxis(AgenteCentralita.this));
       deathLine = new Date(System.currentTimeMillis() + MILLIS_PER_8HOURS);
       //System.out.println("["+myAgent.getLocalName()+"]:"+"CACHE ACTUALIZADA.
Proxima actualizacion: "+ deathLine + ".");
   } catch (Exception e) { }
}
 }


//*************************************************************************************
*******************************
 //Método que crea un mensaje para enviarlo de la Centralita a todos los Taxis
 ACLMessage EnviarMsgTaxis(Action a){

    //Creamos el mensaje y rellenamos los campos claves
    ACLMessage msg = new ACLMessage(ACLMessage.CFP);
    msg.setSender(getAID());
    msg.setLanguage(codec.getName());
    msg.setOntology(ontology.getName());
    msg.setProtocol(JADE.proto.FIPAProtocolNames.FIPA_CONTRACT_NET);

    BuscarTaxi b=(BuscarTaxi)a.getAction();

    try{
  //Rellenamos el contenido (SL requires actions to be included into the action
construct)
  Action ac=new Action(getAID(),b);
  manager.fillContent(msg,ac);
    }catch(Exception e) {
       e.printStackTrace();
    }

    return msg;
```

```
 }// Fin de EnviarMsgTaxis


//*************************************************************************************
*********************************
 //te devuelve la lista sin el primer elemento
 private JADE.util.LEAP.List EliminaPrimero(JADE.util.LEAP.List l) {
JADE.util.LEAP.List laux=new JADE.util.LEAP.ArrayList();
int i=0;ACLMessage k;boolean fet=false;
while(l.get(i)!=null && !fet){
       k=(ACLMessage)l.get(i);
       if(i!=0){
           laux.add(l.get(i));
       }
       i++;if(i==l.size()){fet=true;i=0;}
}
return laux;
 }


//*************************************************************************************
*********************************
 //Te da la longitud del camino para que la centralita escoga el más corto
 private int longitud_camino(JADE.util.LEAP.List l) {
       int n=0,dist=0;
       int num=0;
       if (l==null) {  //esta vacia
         dist=0;
         num=0;
       }
       else {  //hay recorrido
  Iterator it=l.iterator();
  int nodo_or,nodo_de;
  Long num1,num2;
  int c1,f1,c2,f2;  //col fil
  Coordenadas2 co2=new Coordenadas2();
  InfoMap im=new InfoMap();
  boolean trobat=false;
  while(it.hasNext()){
    num1=(Long)it.next();
    num2=(Long)it.next();
    dist+=2;

    f1=co2.getFil(num1.intValue());f2=co2.getFil(num2.intValue());
    c1=co2.getCol(num1.intValue());c2=co2.getCol(num2.intValue());
    nodo_or=dm.traduccion[f1][c1]; //cogemos nodo origen
    nodo_de=dm.traduccion[f2][c2]; //cogemos nodo destino
    trobat=false;
    n=0;
    while (!trobat && n<4) {
      im=dm.mapa[nodo_or][n];
      if (im.succ==nodo_de) {
```

```
          trobat=true;
          num=num+im.coste; //acumulamos coste
        }
        else
          n++;
      }
    }
  }
  dist=(dist/2)+num; //sumem distancia amb bonus de temps
  return dist;
   }
```

```
/*   private int longitud_camino(JADE.util.LEAP.List l) {
  Iterator it=l.iterator();
  int num=0;

  while(it.hasNext()){
        it.next();
        num++;
  }
  num=(num/2);
  return num;
   }
*/
```

```
  //*************************************************************************
  ******************************
   //Clase que inicia una comunicación CFP
   class CNInitiator extends ContractNetInitiator {

  public CNInitiator (Agent myAgent, ACLMessage CfpMsg) {
       super(myAgent,CfpMsg);
  }

  //Este método se ejecuta automáticamente (es el primer paso de la comunicación)
  //IMPORTANTE: Añadir al vector el CfpMsg y luego añadir a CfpMsg los receivers.
  protected Vector prepareCfps (ACLMessage CfpMsg) {
        Vector v = new Vector();
        TaxiCache                                                    =
  LlenarCacheTaxis(TaxiOntology.SERVICE_TYPE,TaxiOntology.TAXI_SERVICE);
        TaxiCacheIterator = TaxiCache.iterator();
          v.add(CfpMsg);
        while (TaxiCacheIterator.hasNext()) {
            CfpMsg.addReceiver((AID)TaxiCacheIterator.next());
        }
```

```
      return v;
     }

protected void handleAllResponses (Vector proposes, Vector acceptances) {
    System.out.println("["+getLocalName()+"]:"+"HA RECIBIDO TODAS LAS PROPUESTAS
HABIDAS Y POR HABER");
    int longmin=500;int numpropuesta=999;
    EstimacionRecorridoLlegada est;
    ListaCalles l=new ListaCalles();
    ListaCalles lista_servicio=new ListaCalles();
    int tiempo_empleado=0;
    boolean ocupado=false;

    try{
      int i=0;
      ACLMessage respuesta;
      ACLMessage msg;
      while (i<proposes.size()) {
            msg=(ACLMessage)proposes.get(i);
            Action a=(Action)manager.extractContent(msg);
            est=(EstimacionRecorridoLlegada)a.getAction();
            l=est.getListaCallesCamino();
            ocupado=est.getOcupado();
            if(longitud_camino(l.getListaCalles())<longmin && !ocupado){
                  longmin=longitud_camino(l.getListaCalles());
                  numpropuesta=i;
                  lista_servicio=est.getListaCallesServicio();  //guardamos el
servicio del taxi seleccionado
                  tiempo_empleado=seg_segmento*longmin;  //numero  segundos
totales para q llegue el taxi
            }
            i++;
      }
      //Aqui es donde aviso al taxi que me interesa y al resto le rechazo la
propuesta
      i=0;
      while(i<proposes.size()){
            msg=(ACLMessage)proposes.get(i);
             respuesta=msg.createReply();
            if(i==numpropuesta){
                System.out.println("["+getLocalName()+"]:"+"HA ESCOGIDO UNA
PROPUESTA DE TODAS LAS ENVIADAS.........................");
                respuesta.setPerformative(ACLMessage.ACCEPT_PROPOSAL);
                acceptances.add(respuesta);
            }else{
                respuesta.setPerformative(ACLMessage.REJECT_PROPOSAL);
                send(respuesta);
            }
            i++;
      }
      ACLMessage info;
      info=(ACLMessage)lista_clientes.get(0);
```

```
        lista_clientes=EliminaPrimero(lista_clientes);
        if(numpropuesta!=999){ //SERVICIO REALIZADO CORRECTAMENTE
                info.setPerformative(ACLMessage.INFORM);
                msg=(ACLMessage)proposes.get(numpropuesta);
                Action a=(Action)manager.extractContent(msg);
                est=(EstimacionRecorridoLlegada)a.getAction();
                l=est.getListaCallesServicio();

                RespuestaPeticion rp=new RespuestaPeticion(); //enviamos la respuesta
al CLIENTE
                rp.setRespuesta(true);
                rp.setIDTaxi((msg.getSender()).getName());
                rp.setPrecio(longitud_camino(l.getListaCalles())*PRECIOPASO);
                 rp.setListaCallesServicio(lista_servicio); //enviamos el servicio al
cliente
                 rp.setTiempo(tiempo_empleado); //enviamos los segundos q tardara en
llegar el taxi

                 try{
                  a=new Action(getAID(),rp);
                  manager.fillContent(info,a);
                 }catch(Exception e) {
                   e.printStackTrace();
                 }

        }else{                  //NO HAY TAXIS LIBRES
                info.setPerformative(ACLMessage.FAILURE);
        }
        send(info);

        numpropuesta=999;
           }catch(Exception e) {
         e.printStackTrace();
    }
}

 }// Fin de la clase CNInitiator


//*************************************************************************************
*****************************
 //Este BEHAVIOUR es el que controla la recepción y la respuesta DE LOS REQUESTS
 class RequestResponder extends AchieveREResponder {

public RequestResponder(Agent a, MessageTemplate mt) {
    super(a, mt);
}

//Método que se ejecuta automáticamente cuando recibe un REQUEST
protected ACLMessage prepareResponse(ACLMessage request) {
    ACLMessage response = request.createReply();
         try{
```

```
          Action a=(Action)manager.extractContent(request);

               if (a.getAction() instanceof BuscarTaxi){
              lista_clientes.add(response);
            System.out.println("["+getLocalName()+"]:"+" HA RECIBIDO UN REQUEST DE
BUSCARTAXI Y VA A INICIAR UN CFP");

              BuscarTaxi tmp;tmp=(BuscarTaxi)a.getAction();
              Peticion tmp2=new Peticion();tmp2=tmp.getpeticion();
              Behaviour dp;
              ACLMessage                        dibpers                        =
EnviarMsgVisualizador("DIBUJAR",tmp2.getposcliente());
              dp = new RequestInitiator(myCentralita, dibpers);
              addBehaviour(dp);

              Behaviour b;
               ACLMessage CfpMsg = EnviarMsgTaxis(a);
               b = new CNInitiator(myCentralita, CfpMsg);
               addBehaviour(b);
              response.setPerformative(ACLMessage.AGREE);
                }
          }catch(Exception e) {
        e.printStackTrace();
    }
    return response;
} //Fin de prepareResponse

//Método que se ejecuta automáticamente cuando recibe un REQUEST
protected ACLMessage prepareResultNotification(ACLMessage request,ACLMessage
response) {
    ACLMessage inf = response.createReply();
    return inf;
}

 } // Fin de la clase RequestResponder


//**************************************************************************************
*****************************
 //Método que crea un nuevo mensaje, aquí le indicaremos las acciones que deseamos
realizar al visualizador
 ACLMessage EnviarMsgVisualizador(String accion,Posicion pos){

    //Creamos el mensaje y rellenamos los campos claves
    ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
    AID                   receiver                   =                    new
AID(getAgentDF(TaxiOntology.SERVICE_TYPE,TaxiOntology.VISUALIZADOR_SERVICE),false);
    msg.setSender(getAID());
    msg.addReceiver(receiver);
    msg.setLanguage(codec.getName());
    msg.setOntology(ontology.getName());
    msg.setProtocol(JADE.proto.FIPAProtocolNames.FIPA_REQUEST);
```

```
    if(accion=="DIBUJAR"){
       DibujarPersona dp=new DibujarPersona();
       dp.setposicion(pos);
       try{
        //Rellenamos el contenido (SL requires actions to be included into the
action construct)
        Action a=new Action(getAID(),dp);
        manager.fillContent(msg,a);
       }catch(Exception e) {
          e.printStackTrace();
       }
    }else {
       BorrarPersona bp=new BorrarPersona();
       bp.setposicion(pos);
       try{
        //Rellenamos el contenido (SL requires actions to be included into the
action construct)
        Action a=new Action(getAID(),bp);
        manager.fillContent(msg,a);
       }catch(Exception e) {
          e.printStackTrace();
       }
    }
    return msg;
 }// Fin de EnviarMsgVisualizador


//*************************************************************************************
*****************************
  //Clase que inicia el Request NOTA:NO CAL HACER EL SEND YA QUE LO HACE
AUTOMÁTICAMENTE
 class RequestInitiator extends AchieveREInitiator {

public RequestInitiator(Agent a, ACLMessage req) {
    super(a, req);
}

protected void handleAgree(ACLMessage agree) {
    //System.out.println("["+getLocalName()+"]:"+" HA RECIBIDO UN AGREE DEL
VISUALIZADOR, HA DIBUJADO A UN CLIENTE");
}

 } // Fin de la clase RequestInitiator

 //*************************************************************************
 //Metodo setup que todo Agente tiene
 //*************************************************************************
 public void setup() {
       String dest=(String)getArguments()[0]; //cogemos el nombre del fichero de
calles q usara la centralita
     dm=new DatosMapa(dest);
```

```
   RegistrarDF();
   Behaviour          rq          =          new          RequestResponder(this,
   AchieveREResponder.createMessageTemplate(FIPAProtocolNames.FIPA_REQUEST));
   addBehaviour(rq);
    }// Fin del setup()

}//end of Agent Centralita
```

### 9.2.3. AGENTECLIENTE.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA

AGENTE CLIENTE
*****************************************************/
package Agentes;

import Ontology.*;

import java.io.*;
import java.lang.String;
import JADE.util.LEAP.Iterator;
import JADE.core.*;
import JADE.core.behaviours.*;
import JADE.core.Agent;
import JADE.core.AID;
import JADE.domain.FIPAException;
import java.util.*;

import JADE.lang.acl.*;
import JADE.lang.acl.ACLMessage;

import JADE.proto.*;
import JADE.domain.FIPAAgentManagement.*;
import JADE.domain.*;
import JADE.content.*;
import JADE.content.onto.basic.*;
import JADE.content.onto.*;
import JADE.content.lang.*;
import JADE.content.lang.sl.*;
import JADE.content.lang.Codec.*;
import JADE.content.onto.basic.*;

public class AgenteCliente extends Agent {

    private boolean continuar;

    Behaviour z;
    Agent miCliente=this;
    //Característica del Agente Cliente. PETICIÓN (Todo cliente debe tener una petición
   en curso)
```

```
 Peticion pet= new Peticion();

 //Para pintar los gráficos de la interficie
 AccionARealizar ar;
 VentanaCliente v=VentanaCliente.getInstance();
 VentanaMapa vm=VentanaMapa.getInstance();

 private ContentManager manager  = (ContentManager) getContentManager();
 private Codec codec=new SLCodec();
 private Ontology ontology=TaxiOntology.instance();


//*****************************************************************************
*****************************
 //Método que se encarga de buscar los agentes que deseo en el DF
 //TaxiOntology.SERVICE_TYPE,TaxiOntology.TAXI_SERVICE
 private String getAgentDF(String type,String name) {
String agentFound = new String();
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(type);
sd.setName(name);
dfd.addServices(sd);

try {
  SearchConstraints c = new SearchConstraints();
       c.setMaxResults(new Long(-1));  //buscaremos infinitos resultados (estilo
JADE2.x)
  DFAgentDescription[] DFAgents = DFService.search(this,dfd,c);
  int i=0;
  while ((DFAgents != null) && (i<DFAgents.length)) {
    DFAgentDescription agent = DFAgents[i];
    i++;
    Iterator services = agent.getAllServices();
    boolean found = false;
    ServiceDescription service = null;
    while (services.hasNext() && !found) {
      service = (ServiceDescription)services.next();
      found = (service.getType().equals(type) && service.getName().equals(name));
    }
    if (found) {
      AID aux=agent.getName();
           agentFound=aux.getName();    //Introduzco en agentsFound el AID de un
agente TAXI
    }
  }
} catch (FIPAException e) {
    System.out.println("["+getLocalName()+"]:"+"Se  ha  producido  un  error:
"+e.getMessage());
     }
return agentFound;
 }
```

```
//*************************************************************************************
*****************************
 //Método que crea un nuevo mensaje (una nueva petición para enviar a centralita)
 ACLMessage EnviarMsgPeticion(){

    //Creamos el mensaje y rellenamos los campos claves
    ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
    AID                    receiver                    =                    new
AID(getAgentDF(TaxiOntology.SERVICE_TYPE,TaxiOntology.CENTRALITA_SERVICE),false);

    Posicion p1=new Posicion();
    Posicion p2=new Posicion();

    msg.setSender(getAID());
    msg.addReceiver(receiver);
    msg.setLanguage(codec.getName());
    msg.setOntology(ontology.getName());
    msg.setProtocol(JADE.proto.FIPAProtocolNames.FIPA_REQUEST);

    int o=v.get_crew();
    pet.setnumpersonas(o);
    o=v.get_inicix();
    p1.setx(o);
    o=v.get_iniciy();
    p1.sety(o);
    o=v.get_destix();
    p2.setx(o);
    o=v.get_destiy();
    p2.sety(o);
    pet.setposcliente(p1);
    pet.setdestino(p2);

    //Rellenamos la ACCION BUSCARTAXI
    BuscarTaxi b=new BuscarTaxi(pet);

    try{
  //Rellenamos el contenido (SL requires actions to be included into the action
construct)
  Action a=new Action(getAID(),b);
  manager.fillContent(msg,a);
    }catch(Exception e) {
       e.printStackTrace();
    }

    //Devolvemos el ACLMessage (msg)
    return msg;
 }// Fin de EnviarMsgPeticion
```

```
//*************************************************************************************
*****************************
 //Metodo que registra en el DF
 void RegistrarDF() {
//REGISTRAMOS la ontología y el lenguaje
manager.registerLanguage(codec,codec.getName());
manager.registerOntology(ontology,ontology.getName());

     //Preparación para dar de alta en el DF
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setName(TaxiOntology.CLIENTE_SERVICE);
sd.setType(TaxiOntology.SERVICE_TYPE);
dfd.addServices(sd);
dfd.setName(getAID());

try {
    DFService.register(this, dfd);
    System.out.println("["+getLocalName()+"]:"+"SE HA REGISTRADO EN EL DF");
    }
         catch (JADE.domain.FIPAException e) {
         System.out.println("["+getLocalName()+"]:"+"NO SE HA PODIDO REGISTRAR EN EL
DF");
         doDelete();
         }

  //   ar = new AccionARealizar(1,0,0,null,null);
// v.addFigura(ar);
// map.pinta();

 }//Fin de RegistrarDF



//*************************************************************************************
*****************************
  //Clase que inicia el Request NOTA:NO CAL HACER EL SEND YA QUE LO HACE
AUTOMÁTICAMENTE
 class RequestInitiator extends AchieveREInitiator {

public RequestInitiator(Agent a, ACLMessage req) {
    super(a, req);
}

protected void handleAgree(ACLMessage agree) {}

protected void handleFailure(ACLMessage failure) {
    System.out.println("["+getLocalName()+"]:"+"     HA     RECIBIDO     UN
FAILURE$$$$$$$$$$$$$$$$$$$$$$$$$$$$.....(NO HAY TAXIS LIBRES EN LA CIUDAD)");
    Envio();
```

```
}

protected void handleInform(ACLMessage Inform) {
    System.out.println("["+getLocalName()+"]:"+"     HA     RECIBIDO     UN
INFORM.............(SE HA REALIZADO EL SERVICIO CORRECTAMENTE)");
    System.out.println("-----------------------------------------------------------
-----------------------------------------");
    try{
          Action a=(Action)manager.extractContent(Inform);
    RespuestaPeticion rp=(RespuestaPeticion)a.getAction();
    System.out.println("EL TAXI QUE LE IRA A BUSCAR ES: "+rp.getIDTaxi());
    System.out.println("EL SERVICIO LE COSTARA:(centimos)");
    System.out.println(rp.getPrecio());
    float minutos=rp.getTiempo()/60;
    System.out.println("el taxi tardara:"+minutos+" minutos");
  //  System.out.println("EL RECORRIDO ES:");
  //  System.out.println(rp.getListaCallesServicio().toString());
    JADE.util.LEAP.List  calles=rp.getListaCallesServicio().getListaCalles();
//sacamos la lista con las calles
    vm.muestra_camino(calles);
    v.muestra_resultado(rp.getIDTaxi(),rp.getPrecio(),minutos); //mostramos el
resultado por al interface
    }catch(Exception e){}
  //  takeDown();
}

 } // Fin de la clase RequestInitiator

 //*****************************************************************************
 public void Envio() {
     try{Thread.sleep(5000);}catch(Exception e){}
Behaviour b;
ACLMessage request = EnviarMsgPeticion();
b = new RequestInitiator(this, request);
addBehaviour(b);
 }

    public void qtejodan() {
      continuar=true;
    }


    class RBehaviour extends SimpleBehaviour {

     public RBehaviour(Agent a) {
         super(a);
     }

     public void action() {
       if (continuar) {
         continuar=false;
         Envio();
```

```
            }
          }

          public boolean done() {
              return false;
          }

      } // End of internal RBehaviour class


      //*****************************************************************************
      //Metodo setup que todo Agente tiene
      //*****************************************************************************
      public void setup() {
        v.initialise(this);
      //    v.addDibujo(map);
        RegistrarDF();
        z = new RBehaviour(this);
        addBehaviour(z);

      }

      //*****************************************************************************
      //Metodo takeDown
      //*****************************************************************************
      public void takeDown() {
     try {
         DFService.deregister(this);
         System.out.println("["+getLocalName()+"]:"+"EL AGENTE HA SIDO ELIMINADO DEL
     DF");
     } catch (JADE.domain.FIPAException e) {}
     doDelete();
      }
  }//Fin de la clase Initiator
```

### 9.2.4. AGENTETAXI.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

AGENTE TAXI
*****************************************************/
package Agentes;

import Ontology.*;

import java.io.*;
import java.util.*;

import JADE.core.*;
```

```
import JADE.core.behaviours.*;
import JADE.proto.*;
import JADE.lang.acl.*;
import JADE.domain.FIPAAgentManagement.*;
import JADE.domain.*;
import JADE.content.*;
import JADE.content.onto.basic.*;
import JADE.content.onto.*;
import JADE.content.lang.*;
import JADE.content.lang.sl.*;
import JADE.content.lang.sl.*;
import JADE.content.lang.Codec.*;
import JADE.content.onto.basic.*;

public class AgenteTaxi extends Agent {

    //Característica del Agente Taxi (es del tipo TaxiType
   (idtaxi,propietario,numlicencia,matricula,marca,modelo,turno,posactual,ocupado))
    TaxiType caract_taxi;
    Agent miTaxi=this;

    ListaCalles ListaCallesAnuladas=new ListaCalles();
    //String Turno_Actual="mañana";
    String Turno_Actual=null;
    boolean Registrado=false;
    //private final long MILLIS_PER_8HOURS = 60 * 1000;
    private final long MILLIS_PER_8HOURS = 28800 * 1000;  //8 horas de las de verdad.
    Date deathLine = new Date(System.currentTimeMillis() + MILLIS_PER_8HOURS);
   // DatosMapa dm=new DatosMapa();
     DatosMapa dm;
    public int TAMAÑO_LCAMI=1000;
    public int TAMAÑO_CAMI=121;
    Coordenadas2 coor2=new Coordenadas2();
    int nodo_final=0;
    JADE.util.LEAP.List prop=new JADE.util.LEAP.ArrayList();
    JADE.util.LEAP.List serv=new JADE.util.LEAP.ArrayList();


    private ContentManager manager  = (ContentManager) getContentManager();
    private Codec codec=new SLCodec();
    private Ontology ontology=TaxiOntology.instance();


   //*************************************************************************
   ******************************
    //Metodo que registra en el DF
    void RegistrarDF() {
        //Preparación para dar de alta en el DF
   DFAgentDescription dfd = new DFAgentDescription();
   ServiceDescription sd = new ServiceDescription();
   sd.setName(TaxiOntology.TAXI_SERVICE);
   sd.setType(TaxiOntology.SERVICE_TYPE);
```

```
dfd.addServices(sd);
dfd.setName(getAID());

try {
    DFService.register(this, dfd);
    System.out.println("["+getLocalName()+"]:"+" SE HA REGISTRADO EN EL DF");
    }
         catch (JADE.domain.FIPAException e) {
        System.out.println("["+getLocalName()+"]:"+"No se ha podido registrar en el
DF");
         doDelete();
         }
     try {
         Thread.sleep(1000);
     }
     catch (Exception e) {
     }

 }//Fin de RegistrarDF


//*************************************************************************************
*****************************
 //Metodo que elimina del DF al taxi
 void DeRegistrarDF() {
try {
    DFService.deregister(this);
    System.out.println("["+getLocalName()+"]:"+"EL AGENTE HA SIDO ELIMINADO DEL
DF");

         Behaviour dt;
         ACLMessage                          dibtaxi                          =
EnviarMsgVisualizador("BORRAR",caract_taxi.getPosActual());
         dt = new RequestInitiator(miTaxi, dibtaxi);
         addBehaviour(dt);

} catch (Exception e) {
         System.out.println("["+getLocalName()+"]:"+"No se ha podido eliminar del
DF");
     }
 }//Fin de DeRegistrarDF


//*************************************************************************************
*****************************
 //Método que se encarga de buscar los agentes que deseo en el DF
 //TaxiOntology.SERVICE_TYPE,TaxiOntology.TAXI_SERVICE
 private String getAgentDF(String type,String name) {
String agentFound = new String();
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(type);
```

```
sd.setName(name);
dfd.addServices(sd);

try {
  SearchConstraints c = new SearchConstraints();
       c.setMaxResults(new Long(-1));  //buscaremos infinitos resultados (estilo
JADE2.x)
  DFAgentDescription[] DFAgents = DFService.search(this,dfd,c);
  int i=0;
  while ((DFAgents != null) && (i<DFAgents.length)) {
    DFAgentDescription agent = DFAgents[i];
    i++;
    Iterator services = agent.getAllServices();
    boolean found = false;
    ServiceDescription service = null;
    while (services.hasNext() && !found) {
       service = (ServiceDescription)services.next();
       found = (service.getType().equals(type) && service.getName().equals(name));
    }
    if (found) {
       AID aux=agent.getName();
            agentFound=aux.getName();    //Introduzco en agentsFound el AID de un
agente TAXI
    }
  }
} catch (FIPAException e) {
    System.out.println("["+getLocalName()+"]:"+"Se  ha  producido  un  error:
"+e.getMessage());
       }
return agentFound;
 }


//*************************************************************************************
*****************************
 //Metodo que obtiene la información del taxi de un fichero (NOMBRE DEL FICHERO:
<NombreAgente>.txt)
 TaxiType readInfoTaxi(String pathname) throws Exception
 {
     FileReader profile = new FileReader(pathname);
     BufferedReader b= new BufferedReader(profile);
String IDTaxi,Propietario,NumLicencia,Matricula,Marca,Modelo,Turno;
Posicion posactual=new Posicion();

TaxiType taxi=null;String s=new String();
String miS=null;
IDTaxi=b.readLine();
Propietario=b.readLine();
NumLicencia=b.readLine();
Matricula=b.readLine();
Marca=b.readLine();
Modelo=b.readLine();
```

```
Turno=b.readLine();

miS=b.readLine();s=s+miS.charAt(0);
posactual.setx(Integer.parseInt(s));
s=new String();s=s+miS.charAt(2);
posactual.sety(Integer.parseInt(s));

taxi=new
TaxiType(IDTaxi,Propietario,NumLicencia,Matricula,Marca,Modelo,Turno,posactual,fals
e);
     return taxi;
 }//Fin de readPeticion


//*************************************************************************************
*****************************
 //Método que crea el mensaje para enviar la acción DAMECALLESANULADAS a Trafico
 ACLMessage EnviarMsgDameCalles(){

    //Creamos el mensaje y rellenamos los campos claves
    ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
    AID                   receiver                    =                    new
AID(getAgentDF(TaxiOntology.SERVICE_TYPE,TaxiOntology.TRAFICO_SERVICE),false);
    msg.setSender(getAID());
    msg.addReceiver(receiver);
    msg.setLanguage(codec.getName());
    msg.setOntology(ontology.getName());
    msg.setProtocol(JADE.proto.FIPAProtocolNames.FIPA_REQUEST);

    //Rellenamos la ACCION BUSCARTAXI
    DameCallesAnuladas d=new DameCallesAnuladas();

    try{
  //Rellenamos el contenido (SL requires actions to be included into the action
construct)
  Action a=new Action(getAID(),d);
  manager.fillContent(msg,a);
    }catch(Exception e) {
       e.printStackTrace();
    }

    //Devolvemos el ACLMessage (msg)
    return msg;
 }// Fin de EnviarMsgPeticion


//*************************************************************************************
*****************************
  //Clase que inicia el Request NOTA:NO CAL HACER EL SEND YA QUE LO HACE
AUTOMÁTICAMENTE
 class RequestInitiator extends AchieveREInitiator {
```

```
public RequestInitiator(Agent a, ACLMessage req) {
    super(a, req);
}

protected void handleInform(ACLMessage inform) {
    AID                                                                  traf=new
AID(getAgentDF(TaxiOntology.SERVICE_TYPE,TaxiOntology.TRAFICO_SERVICE),false);
    AID s=inform.getSender();
    String k1=s.getName();
    String k2=traf.getName();
    if(k1.equals(k2)){
            try{
                Action a=(Action)manager.extractContent(inform);

                 if (a.getAction() instanceof ListaCalles){
                  System.out.println("["+getLocalName()+"]:"+" HA RECIBIDO LAS
CALLES ANULADAS DE TRAFICO Y ACTUALIZO EL MAPA");
                  ListaCallesAnuladas=(ListaCalles)a.getAction();
                  JADE.util.LEAP.List tmp = new JADE.util.LEAP.ArrayList();
                  tmp=ListaCallesAnuladas.getListaCalles();
                  dm.ActualizarCallesAnuladas(tmp);
                 }else{
                  System.out.println("["+getLocalName()+"]:"+"NO HA RECIBIDO...");
                }
            }catch(Exception e) {
              e.printStackTrace();
           }
     }
}

 } // Fin de la clase RequestInitiator


//*************************************************************************************
*******************************
 //Método para transformar un Array en una Colección
 public JADE.util.LEAP.List Array_to_List(int[] camino){
JADE.util.LEAP.List lista=new JADE.util.LEAP.ArrayList();
int i=0;int j=1;Integer num;

while(camino[j]!=-1){
       num=new Integer(camino[i]);
       lista.add(num);
       num=new Integer(camino[j]);
       lista.add(num);
       i++;j=i+1;
}

return lista;
 }
```

```
//*************************************************************************************
*****************************
 //Este BEHAVIOUR es el que controla la recepción de los CONTRACT-NET
 class CNResponder extends ContractNetResponder {

public CNResponder(Agent a, MessageTemplate mt) {
    super(a, mt);
}

//Este método se ejecuta cuando se recibe un mensaje CFP y se tiene que dar una
propuesta
protected ACLMessage prepareResponse (ACLMessage cfp) {
         System.out.println("["+getLocalName()+"]:"+"  HA  RECIBIDO  EL  CFP
CORRECTAMENTE Y VAMOS A ENVIAR UNA PROPUESTA");
    ACLMessage propose = cfp.createReply();
    EstimacionRecorridoLlegada est=new EstimacionRecorridoLlegada();

       try{
              Action a=(Action)manager.extractContent(cfp);

              BuscarTaxi bt;
              bt=(BuscarTaxi)a.getAction();Peticion p;p=bt.getpeticion();
              Posicion pos_cliente=new Posicion();pos_cliente=p.getposcliente();
              Posicion pos_taxi=new Posicion();pos_taxi=caract_taxi.getPosActual();

 prop=Array_to_List(Aestrella(dm.mapa_aux,Codifica(pos_taxi.getx(),pos_taxi.g
ety()),Codifica(pos_cliente.getx(),pos_cliente.gety())));

              Posicion pos_llegada=new Posicion();pos_llegada=p.getdestino();

 serv=Array_to_List(Aestrella(dm.mapa_aux,Codifica(pos_cliente.getx(),pos_cli
ente.gety()),Codifica(pos_llegada.getx(),pos_llegada.gety())));


              //Rellenamos el contenido (SL requires actions to be included into
the action construct)
              ListaCalles ListaCallesCamino=new ListaCalles();
              ListaCalles ListaCallesServicio=new ListaCalles();
              ListaCallesServicio.setListaCalles(serv);
              ListaCallesCamino.setListaCalles(prop);

              est.setListaCallesServicio(ListaCallesServicio);
              est.setListaCallesCamino(ListaCallesCamino);
              est.setOcupado(caract_taxi.getOcupado());
              Action ac=new Action(getAID(),est);
              manager.fillContent(propose,ac);

        }catch(Exception e) {
           e.printStackTrace();
       }
       propose.setPerformative(ACLMessage.PROPOSE);
```

```
    return propose;
     }//Fin de prepareResponse

//Este método te da la posicion del taxi después de hacer el servicio
public Posicion ultima_pos(JADE.util.LEAP.List l){
       Coordenadas2 c2=new Coordenadas2();
       Iterator it=l.iterator();Integer n=new Integer(0);
       while(it.hasNext()){
             n=(Integer)it.next();
       }
       Posicion                                                    pos=new
Posicion();pos.setx(c2.getCol(n.intValue()));pos.sety(c2.getFil(n.intValue()));
       return pos;
}

//Este método se ejecuta cuando se recibe un ACCEPTPROPOSAL. Devuelve un INFORM o
FAILURE
protected ACLMessage prepareResultNotification (ACLMessage cfp, ACLMessage propose,
ACLMessage accept) {
         System.out.println("["+getLocalName()+"]:"+"HA      RECIBIDO      UN
ACCEPTPROPOSAL/////////////////////////////////////");
    caract_taxi.setOcupado(true);
    ACLMessage responseinf = accept.createReply();
         responseinf.setPerformative(ACLMessage.INFORM);

    try{Thread.sleep(1500);}catch(Exception e){}
    ListaCalles l1=new ListaCalles();l1.setListaCalles(prop);
    ListaCalles l2=new ListaCalles();l2.setListaCalles(serv);

    Behaviour dr;
         ACLMessage dibrecor = EnviarMsgVisRec("DIBUJAR",5,5,l1,l2);
         dr = new RequestInitiator(miTaxi, dibrecor);
    addBehaviour(dr);

    Behaviour bc;
         ACLMessage                        borcliente                        =
EnviarMsgVisualizador("BORRAR",ultima_pos(l1.getListaCalles()));
         bc = new RequestInitiator(miTaxi, borcliente);
    addBehaviour(bc);

          caract_taxi.setPosActual(ultima_pos(l2.getListaCalles()));
    Behaviour        rq        =        new        RequestResponder(miTaxi,
AchieveREResponder.createMessageTemplate(FIPAProtocolNames.FIPA_REQUEST));
    addBehaviour(rq);
         return responseinf;
     }//Fin de prepareResultNotification

//Este método se ejecuta cuando se recibe un REJECTPROPOSAL
protected void handleRejectProposal (ACLMessage cfp, ACLMessage propose, ACLMessage
reject) {
       System.out.println("["+getLocalName()+"]:"+"HA RECIBIDO UN REJECT");
     }//Fin de handleRejectProposal
```

```
 } // Fin de la clase CNResponder


//*************************************************************************************
*****************************
 //Método que crea un nuevo mensaje, aquí le indicaremos las acciones que deseamos
realizar al visualizador
 ACLMessage EnviarMsgVisualizador(String accion,Posicion pos){

    //Creamos el mensaje y rellenamos los campos claves
    ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
    AID                    receiver                    =                    new
AID(getAgentDF(TaxiOntology.SERVICE_TYPE,TaxiOntology.VISUALIZADOR_SERVICE),false);
    msg.setSender(getAID());
    msg.addReceiver(receiver);
    msg.setLanguage(codec.getName());
    msg.setOntology(ontology.getName());
    msg.setProtocol(JADE.proto.FIPAProtocolNames.FIPA_REQUEST);

    if(accion=="DIBUJAR"){
       DibujarTaxi dt=new DibujarTaxi();
       dt.setposicion(pos);
       try{
         //Rellenamos el contenido (SL requires actions to be included into the
action construct)
         Action a=new Action(getAID(),dt);
         manager.fillContent(msg,a);
       }catch(Exception e) {
          e.printStackTrace();
       }
    }else {
       BorrarTaxi bt=new BorrarTaxi();
       bt.setposicion(pos);
       try{
         //Rellenamos el contenido (SL requires actions to be included into the
action construct)
         Action a=new Action(getAID(),bt);
         manager.fillContent(msg,a);
       }catch(Exception e) {
          e.printStackTrace();
       }
    }
    return msg;
 }// Fin de EnviarMsgVisualizador


//*************************************************************************************
*****************************
 //Método que crea un nuevo mensaje, aquí le indicaremos las acciones que deseamos
realizar al visualizador
 ACLMessage      EnviarMsgVisRec(String      accion,int      costebuscar,int
```

```
costeservir,ListaCalles buscar,ListaCalles servir){

    //Creamos el mensaje y rellenamos los campos claves
    RecorridoTaxi rt=new RecorridoTaxi();
    ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
    AID                  receiver                  =                    new
AID(getAgentDF(TaxiOntology.SERVICE_TYPE,TaxiOntology.VISUALIZADOR_SERVICE),false);
    msg.setSender(getAID());
    msg.addReceiver(receiver);
    msg.setLanguage(codec.getName());
    msg.setOntology(ontology.getName());
    msg.setProtocol(JADE.proto.FIPAProtocolNames.FIPA_REQUEST);

    if(accion=="DIBUJAR"){
       DibujarRecorridoTaxi drt=new DibujarRecorridoTaxi();
       rt.setCosteBuscar(costebuscar);
       rt.setCosteServir(costeservir);
       rt.setListaCallesBuscar(buscar);
       rt.setListaCallesServir(servir);
       drt.setRecorridoTaxi(rt);
       try{
        //Rellenamos el contenido (SL requires actions to be included into the
action construct)
        Action a=new Action(getAID(),drt);
        manager.fillContent(msg,a);
       }catch(Exception e) {
          e.printStackTrace();
       }
    }else {
       BorrarRecorridoTaxi brt=new BorrarRecorridoTaxi();
       rt.setCosteBuscar(costebuscar);
       rt.setCosteServir(costeservir);
       rt.setListaCallesBuscar(buscar);
       rt.setListaCallesServir(servir);
       brt.setRecorridoTaxi(rt);
       try{
        //Rellenamos el contenido (SL requires actions to be included into the
action construct)
        Action a=new Action(getAID(),brt);
        manager.fillContent(msg,a);
       }catch(Exception e) {
          e.printStackTrace();
       }
    }
    return msg;
 }// Fin de EnviarMsgVisRec


//*************************************************************************************
*****************************
 //Este BEHAVIOUR es el que controla la recepción y la respuesta DE LOS REQUESTS
 class RequestResponder extends AchieveREResponder {
```

```
public RequestResponder(Agent a, MessageTemplate mt) {
    super(a, mt);
}

//Método que se ejecuta automáticamente cuando recibe un REQUEST DE VISUALIZADOR
INDICANDO QUE EL TAXI ESTA DESOCUPADO
protected ACLMessage prepareResponse(ACLMessage request) {
    ACLMessage response = request.createReply();
         try{
         Action a=(Action)manager.extractContent(request);
        caract_taxi.setOcupado(false);
        response.setPerformative(ACLMessage.AGREE);
          }catch(Exception e) {
       e.printStackTrace();
    }
    return response;
} //Fin de prepareResponse

//Se ejecuta automáticamente solo cuadno el método anterior (prepareResponse) ha
dado una respuesta AGREE
protected ACLMessage prepareResultNotification(ACLMessage request, ACLMessage
response) throws FailureException {
    ACLMessage resNot = request.createReply();
    Desocupado doc=new Desocupado();
    try{Action                                                    a=new
Action(getAID(),doc);manager.fillContent(resNot,a);}catch(Exception e) {}
    return resNot;
}// Fin de prepareResultNotification

 } // Fin de la clase RequestResponder



//*************************************************************************************
**********************************
 //método que actualiza la variable Turno_Actual dándole el valor dependiendo de la
hora del sistema
 private String ObtenerTurnoActual()
 {
String t=null;
Calendar fecha=Calendar.getInstance();
int hora=fecha.get(fecha.HOUR_OF_DAY);
if(hora>=8 && hora<16) {
       t="mañana";
}else if(hora>=16 && hora<24){
       t="tarde";
}else if(hora>=0 && hora<8){
       t="noche";
}
return t;
 }
```

```
//*************************************************************************************
*****************************
 //behaviour que mira los turnos de los taxis y registra el que en ese momento le
toque
 private class RegistroTurno extends OneShotBehaviour {
String Turno_Actual;
RegistroTurno(Agent agent,String turno) {
   super(agent);
   this.Turno_Actual=turno;
}

public void action() {

  if(Registrado) {
      DeRegistrarDF();Registrado=false;
  }
  else {
    if(Turno_Actual.equals(caract_taxi.getTurno())) {
      RegistrarDF();Registrado=true;       //SE REGISTRA EN EL DF
      Behaviour dt;
      ACLMessage                          dibtaxi                          =
EnviarMsgVisualizador("DIBUJAR",caract_taxi.getPosActual());
           dt = new RequestInitiator(miTaxi, dibtaxi);
           addBehaviour(dt);
           ACLMessage request = EnviarMsgDameCalles();
      Behaviour b = new RequestInitiator(miTaxi, request);  //Envia un mensaje a
Trafico para que le de calles anuladas
      addBehaviour(b);
    }
  }
       try {Thread.sleep(1000);} catch (Exception e) {}
  new turnoUPDAte(deathLine,(Agent)myAgent);
     }
 }


//*************************************************************************************
*****************************
 //Thread que actualiza los turnos de los TAXIS
 class turnoUPDAte extends Thread implements Runnable {
     long millis;
Agent myAgent;

turnoUPDAte(Date deathLine, Agent agent) {
   super();
   millis = deathLine.getTime() - System.currentTimeMillis();
   if(millis < 0)
       {
          millis=0;
       }
```

```
      myAgent = agent;
      this.start();
   }

   public void run(){
      try {
         //Cuando entra aquí es que debe hacer una nueva actualización (ya ha pasado
   el deathLine)
         Thread.sleep(millis);

               if(Turno_Actual.equals("mañana")) {Turno_Actual="tarde";}
          else if(Turno_Actual.equals("tarde")) {Turno_Actual="noche";}
          else if(Turno_Actual.equals("noche")) {Turno_Actual="mañana";}

         addBehaviour(new RegistroTurno(AgenteTaxi.this,Turno_Actual));
         deathLine = new Date(System.currentTimeMillis() + MILLIS_PER_8HOURS);
      } catch (Exception e) { }
   }
    }

//*************************************************************************************
   *********************************
//       CODIGO PARA LA IMPLEMENTACIÓN DEL ALGORITMO A* + Manhattan
//*************************************************************************************
   *********************************



   //*************************************************************************************
   *********************************
    //Método que te da el valor absoluto de un número
    public int abs(int x){
   if (x<0) {return x*-1;}
   else {return x;}
    }


   //*************************************************************************************
   *********************************
    //Método que te da la distancia de Manhattan
    public int g(int inici_x,int inici_y,int final_x, int final_y) {
   int resultat=abs(inici_x-final_x)+abs(inici_y-final_y);
   return resultat;
    }


   //*************************************************************************************
   *********************************
    //Método que codifica la coordenada en número de nodo
    public int Codifica(int col,int fil) {
   return ((fil*11)+col);
    }
```

```
//*************************************************************************************
*******************************
 //Método que presenta por pantalla el vector dado
 public void presenta(int[] v,int pausa) {
int i=0;
System.out.println("Es la lista LCami en este momento del programa:");
while (v[i]!=-1){
//while (v[i]!=-2){
       System.out.println(v[i]);
       i++;
}
try{
       Thread.sleep(pausa);
}catch(Exception e){}
 }


//*************************************************************************************
*******************************
 //Método que te da el primer recorrido de LCami
 public int[] primer(int[] c) {
int resultat[]=new int[TAMAÑO_CAMI];
int i=0;

while (c[i]!=-1) {
       resultat[i]=c[i];
       i++;
}
resultat[i]=c[i];
return resultat;
 }


//*************************************************************************************
*******************************
 //Método que te da el resto de LCami
 public int[] reste(int[] c) {
int resultat[]=new int[TAMAÑO_LCAMI];
int i=0;

while (c[i]!=-1) {
       i++;
}

i++;int j=0;
while(c[i]!=-2){
       resultat[j]=c[i];
       i++;j++;
}
resultat[j]=-2;
```

```
return resultat;
 }


//*************************************************************************************
*******************************
 //Método que te dice si el último nodo de camino es el objetivo
 public boolean objectiu(int[] c,int nfinal) {
int ultimo;

int i=0;
while (c[i]!=-1){
       i++;
}
i--;ultimo=c[i];
if(ultimo==nfinal){return true;}
else {return false;}
 }


//*************************************************************************************
*******************************
 //Método que te dice si el nodo ya pertenece al camino
 public boolean pertenece(int num,int[] c) {
boolean pertenece=false;
int i=0;

while(c[i]!=-1){
    if(c[i]==num){
       pertenece=true;
    }
    i++;
}
return pertenece;
 }


//*************************************************************************************
*******************************
 //Método que te concatena el camino con sus sucesores
 public int[] concatenar(int[] c,InfoMap[] s,int[] lc) {
int result[]=new int[TAMAÑO_LCAMI];

int i=0;int j=0;
while (s[i]!=null){
   if(!pertenece(s[i].succ,c)){
       int n=0;
       while (c[n]!=-1){
              result[j]=c[n];
              n++;
              j++;
       }
```

```
       result[j]=s[i].succ;j++;
       result[j]=-1;j++;
   }
   i++;
}
i=0;
while(lc[i]!=-2){
       result[j]=lc[i];
       j++;i++;
}
result[j]=-2;
return result;
 }


//*************************************************************************************
*******************************
 //Método que te crea los sucesores del camino actual y también te los concatena con
el camino ya realizado
 public int[] successors(int[] c,int[] lc) {
int ultimo;
int result[]=new int[TAMAÑO_LCAMI];
InfoMap succ[]=new InfoMap[4];

int i=0;
while (c[i]!=-1){
       i++;
}
i--;ultimo=c[i];

succ=dm.getInfoSucc(ultimo);
result=concatenar(c,succ,lc);
return result;
 }


//*************************************************************************************
*******************************
 //Método que esborra fisicament el cami redundant
 public int[] BorrarRed(int puntero,int[] c){
int resultat[]=new int[TAMAÑO_LCAMI];
int i=0;int j=0;
//while(c[i]!=-2 && c[j]!=-2){
while(c[i]!=-2){
       if(i==puntero){
              while(c[j]!=-1){j++;}
              j++;
       }
       resultat[i]=c[j];
       i++;j++;
}
return resultat;
```

```
 }


//*************************************************************************************
*******************************
 //Método que mira quin cami es redundant
 public int[] mirar_redundants(int puntero,int fi,int[] c){
int final_cami,punter_inici_cami;
int i=puntero;  //Hago esto para comenzar en el siguiente camino (sino compararia
el mismo)


while(c[i]!=-2){
       punter_inici_cami=i;
       while(c[i]!=-1 && c[i]!=-2){
              i++;
       }
       i--;final_cami=c[i];
       if(fi==final_cami || final_cami==c[punter_inici_cami]){
              c=BorrarRed(punter_inici_cami,c);
              i=punter_inici_cami-2; //Lo hago para que cuando sume 2 se coloque
donde estaba antes
       }
       i++;i++; //Hago esto para pasar al siguiente camino
}
return c;
 }


//*************************************************************************************
*******************************
 //Método que te elimina los caminos redundantes
 public int[] eliminar_redundants(int[] c) {
int final_cami;int principi_cami;
int i=0;

while(c[i]!=-2){
       principi_cami=c[0];
       while(c[i]!=-1){
              i++;
       }
       i--;final_cami=c[i];
       i++;i++;//Hago esto para pasar al siguiente camino
       //if(principi_cami==final_cami){c=BorrarRed(0,c);}
       c=mirar_redundants(i,final_cami,c);
}
return c;
 }


//*************************************************************************************
*******************************
```

```
     //Te da el coste total de ese camino
     public int heuristica(int puntero,int[] l){
    int respuesta=0;int i=puntero;
    while(l[i]!=-1){
           i++;
    }
    i--;//para saber el nodo del final del camino
respuesta=g(coor2.getCol(l[i]),coor2.getFil(l[i]),coor2.getCol(nodo_final),coor2.ge
    tFil(nodo_final));
    return respuesta;
     }


    //*********************************************************************************
    ******************************
     //Método que te da el puntero del siguiente elemento
     public int siguiente_elemento(int puntero,int[] l) {
    int i=puntero;
    while(l[i]!=-1){
           i++;
    }
    i++;
    return i;
     }


    //*********************************************************************************
    ******************************
     //Método que te inserta un camino en resultat
     public int[] Insertar(int puntero,int[] l,int[] resultat) {
    int i=0;

    while(resultat[i]!=-2){
           i++;
    }
    while(l[puntero]!=-1){
           resultat[i]=l[puntero];
           puntero++;i++;
    }
    resultat[i]=-1;i++;resultat[i]=-2;
    return resultat;
     }


    //*********************************************************************************
    ******************************
     //Método que te ordena LCami
     public int[] ordenar(int[] l) {
    int coste_minimo=1000; //Valor simbólico para luego cambiar
    int coste_aux;
    int puntero=0;
    int resultat[]=new int[TAMAÑO_LCAMI];resultat[0]=-2;
```

```
int i=0;
while(l[i]!=-2){
       while(l[i]!=-2){
              coste_aux=heuristica(i,l);
              if(coste_aux < coste_minimo) {
                     puntero=i;
                     coste_minimo=coste_aux;
              }
              i=siguiente_elemento(i,l);
       }
       resultat=Insertar(puntero,l,resultat);
       l=BorrarRed(puntero,l);
       i=0;coste_minimo=1000;
}
return resultat;
 }


//*************************************************************************************
*******************************
 //Método para hacer el algorismo A*
 public int[] Aestrella(InfoMap[][] mapa,int ninici,int nfinal) {
     int LCami[]=new int[TAMAÑO_LCAMI];
int CAMI[]=new int[TAMAÑO_CAMI];nodo_final=nfinal;

int coste=0;
boolean trobat=false;
LCami[0]=ninici;LCami[1]=-1;LCami[2]=-2; //Inicializo LCami con el nodo inicial y
pongo una marca (-1 y -2)

while (LCami[0]!=-1 && !trobat) {
       CAMI=primer(LCami);
       LCami=reste(LCami);

       if (objectiu(CAMI,nfinal)) {
              trobat=true;
              System.out.println("["+getLocalName()+"]:"+"             CAMINO
ENCONTRADO!!!!!!!!!");
       }else{
              LCami=successors(CAMI,LCami);
              LCami=eliminar_redundants(LCami);
              LCami=ordenar(LCami);
       }
}
return CAMI;
 }

//*************************************************************************************
   *******************************
//       FINAL DEL CODIGO A*
//*************************************************************************************
```

```
   *********************************


    //************************************************************************
    //Metodo setup que todo Agente tiene
    //************************************************************************
    public void setup() {

         String dest=(String)getArguments()[0]; //cogemos el nombre del fichero de
   calles q usara el taxi
          dm=new DatosMapa(dest);


   //REGISTRAMOS la ontología y el lenguaje
   manager.registerLanguage(codec,codec.getName());
   manager.registerOntology(ontology,ontology.getName());

        try {
            caract_taxi=readInfoTaxi(".\\pfc\\Agentes\\"+this.getLocalName()+".txt");
        }catch(Exception e) {
       System.out.println("["+getLocalName()+"]:"+"Error  leyendo  del  fichero
   "+this.getLocalName()+".txt"+":"+e.getMessage());
        }

   Turno_Actual=ObtenerTurnoActual();
   Behaviour rt = new RegistroTurno(this,Turno_Actual);
   addBehaviour(rt);
   Behaviour           cn           =           new           CNResponder(this,
   ContractNetResponder.createMessageTemplate(FIPAProtocolNames.FIPA_CONTRACT_NET));
   addBehaviour(cn);

    }// Fin del setup()

}//Fin del agente Taxi
```

### 9.2.5. AGENTETRAFICO.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

AGENTE TRAFICO
*****************************************************/
package Agentes;

import Ontology.*;

import java.io.*;
import java.util.*;

import JADE.core.*;
```

```
import JADE.core.behaviours.*;
import JADE.proto.*;
import JADE.lang.acl.*;
import JADE.domain.FIPAAgentManagement.*;
import JADE.domain.*;
import JADE.content.*;
import JADE.content.onto.basic.*;
import JADE.content.onto.*;
import JADE.content.lang.*;
import JADE.content.lang.sl.*;
import JADE.content.lang.sl.*;
import JADE.content.lang.Codec.*;
import JADE.content.onto.basic.*;

public class AgenteTrafico extends Agent {

    ListaCalles ListaCallesAnuladas=new ListaCalles();
    private final long MILLIS_PER_8HOURS = 60 * 1000;
    Date deathLine = new Date(System.currentTimeMillis() + MILLIS_PER_8HOURS);

    private ContentManager manager  = (ContentManager) getContentManager();
    private Codec codec=new SLCodec();
    private Ontology ontology=TaxiOntology.instance();


   //*************************************************************************************
   ******************************
    //Metodo que registra en el DF
    void RegistrarDF() {
   //REGISTRAMOS la ontología y el lenguaje
   manager.registerLanguage(codec,codec.getName());
   manager.registerOntology(ontology,ontology.getName());

        //Preparación para dar de alta en el DF
   DFAgentDescription dfd = new DFAgentDescription();
   ServiceDescription sd = new ServiceDescription();
   sd.setName(TaxiOntology.TRAFICO_SERVICE);
   sd.setType(TaxiOntology.SERVICE_TYPE);
   dfd.addServices(sd);
   dfd.setName(getAID());

   try {
       DFService.register(this, dfd);
       System.out.println("["+getLocalName()+"]:"+" SE HA REGISTRADO EN EL DF");
       }
            catch (JADE.domain.FIPAException e) {
           System.out.println("["+getLocalName()+"]:"+"No se ha podido registrar en el
   DF");
            doDelete();
            }
        try {
            Thread.sleep(1000);
```

```
     }
     catch (Exception e) {
     }
 }//Fin de RegistrarDF


//*************************************************************************************
*****************************
 //Este BEHAVIOUR es el que controla la recepción del REQUEST
 class RequestResponder extends AchieveREResponder {

public RequestResponder(Agent a, MessageTemplate mt) {
    super(a, mt);
}

//Método que se ejecuta automáticamente cuando recibe un REQUEST
protected ACLMessage prepareResponse(ACLMessage request) {
    ACLMessage response = request.createReply();
         try{
         Action a=(Action)manager.extractContent(request);

              if (a.getAction() instanceof DameCallesAnuladas){
           System.out.println("["+getLocalName()+"]:"+" HA RECIBIDO UN REQUEST DE
DAMECALLESANULADAS Y VA A ENVIAR LAS CALLES ANULADAS");
           response.setPerformative(ACLMessage.AGREE);
              }else{
           System.out.println("["+getLocalName()+"]:"+"NO  HA  RECIBIDO  UN
DAMECALLESANULADAS");
           response.setPerformative(ACLMessage.REFUSE);
         }

         }catch(Exception e) {
       e.printStackTrace();
    }
    return response;
} //Fin de prepareResponse

//Se ejecuta automáticamente solo cuadno el método anterior (prepareResponse) ha
dado una respuesta AGREE
protected ACLMessage prepareResultNotification(ACLMessage request, ACLMessage
response) throws FailureException {
    ACLMessage resNot = request.createReply();
    try{
        Action a=(Action)manager.extractContent(request);

        if (a.getAction() instanceof DameCallesAnuladas) {
            try{
                //Rellenamos el contenido (SL requires actions to be included into
the action construct)
                //EnviarCalles aux=new EnviarCalles();
                //aux.setlistacalles(ListaCallesAnuladas);
                a=new Action(getAID(),ListaCallesAnuladas);
```

```
                manager.fillContent(resNot,a);

              }catch(Exception e) {
                 e.printStackTrace();
              }
            resNot.setPerformative(ACLMessage.INFORM);
             System.out.println("["+getLocalName()+"]:"+"EL ENVIO DE LA LISTA DE
CALLES SE HA HECHO CORRECTAMENTE");
        }else{
            System.out.println("["+getLocalName()+"]:"+"EL ENVIO DE LA LISTA DE
CALLES ANULADAS HA FALLADO");
            resNot.setPerformative(ACLMessage.FAILURE);
        }
         }catch(Exception e) {
       e.printStackTrace();
    }
    return resNot;
}// Fin de prepareResultNotification

 } // Fin de la clase RequestResponder


//*************************************************************************************
*****************************
 //Metodo que obtiene la lista de calles anuladas.
 JADE.util.LEAP.List readCallesAnuladas(String pathname) throws Exception
 {
     FileReader profile = new FileReader(pathname);
     BufferedReader b= new BufferedReader(profile);
JADE.util.LEAP.List calles= new JADE.util.LEAP.ArrayList();int i=0;
String miS=null;String s=new String();Integer x,y;

miS=b.readLine();
while(miS.charAt(i)!='*') {

       while (miS.charAt(i)!=' ' && miS.charAt(i)!='*') {
              s=s+miS.charAt(i);
              i++;
       }
       if(miS.charAt(i)!='*') {i++;}
       x=Integer.decode(s);
       calles.add(x);s="";
}
     return calles;
 }//Fin de readCallesAnuladas


//*************************************************************************************
*****************************
 //comportamiento que obtiene las calles anuladas de un fichero y lanza el thread
runnable
 private class DarCalles extends OneShotBehaviour {
```

```
DarCalles(Agent agent) {
   super(agent);
}

public void action() {
   try {
       Thread.sleep(1000);
   } catch (Exception e) {}
   //obtengo la lista de las calles anuladas dependiendo de la hora del sistema
   ListaCallesAnuladas=obtenerCalles();
   new horaUPDAte(deathLine,(Agent)myAgent);
}
 }


//*************************************************************************************
*******************************
 //Thread que actualiza la "cache" de TAXIS
 class horaUPDAte extends Thread implements Runnable {
     long millis;
Agent myAgent;

horaUPDAte(Date deathLine, Agent agent) {
   super();
   millis = deathLine.getTime() - System.currentTimeMillis();
   if(millis < 0)
       {
          millis=0;
       }
   myAgent = agent;
   this.start();
}

public void run(){
   try {
       //Cuando entra aquí es que debe hacer una nueva actualización (ya ha pasado
el deathLine)
       Thread.sleep(millis);
       addBehaviour(new DarCalles(AgenteTrafico.this));
       deathLine = new Date(System.currentTimeMillis() + MILLIS_PER_8HOURS);
       System.out.println("["+myAgent.getLocalName()+"]:"+"FICHERO     CALLES
CORTADAS/OBRAS ACTUALIZADO. Proxima actualizacion: "+ deathLine + ".");
   } catch (Exception e) { }
}
 }


//*************************************************************************************
***********************************
 //método que dependiendo de la hora del sistema coge un fichero u otro
 private ListaCalles obtenerCalles()
 {
```

```
ListaCalles l=new ListaCalles();

        try {
          Thread.sleep(1000);
        } catch (Exception e) {}
       Calendar fecha=Calendar.getInstance();
       int hora=fecha.get(fecha.HOUR_OF_DAY);
       if(hora>=8 && hora<16) {
               try {
                  System.out.println("****************SE HA USADO EL FICHERO DE
CALLES MAÑANA.TXT**********");

l.setListaCalles(readCallesAnuladas(".\\pfc\\Agentes\\mañana.txt"));
               }catch(Exception e) {
                  System.out.println("["+getLocalName()+"]:"+"Error leyendo del
fichero "+"mañana.txt"+":"+e.getMessage());
               }
       }else if(hora>=16 && hora<24){
              try {
                  System.out.println("****************SE HA USADO EL FICHERO DE
CALLES TARDE.TXT**********");

l.setListaCalles(readCallesAnuladas(".\\pfc\\Agentes\\tarde.txt"));
               }catch(Exception e) {
                  System.out.println("["+getLocalName()+"]:"+"Error leyendo del
fichero "+"tarde.txt"+":"+e.getMessage());
               }
       }else if(hora>=0 && hora<8){
               try {
                  System.out.println("****************SE HA USADO EL FICHERO DE
CALLES NOCHE.TXT**********");

l.setListaCalles(readCallesAnuladas(".\\pfc\\Agentes\\noche.txt"));
               }catch(Exception e) {
                  System.out.println("["+getLocalName()+"]:"+"Error leyendo del
fichero "+"noche.txt"+":"+e.getMessage());
               }
       }
return l;
 }

 //*************************************************************************
 //Metodo setup que todo Agente tiene
 //*************************************************************************
 public void setup() {
RegistrarDF();
//ListaCallesAnuladas=obtenerCalles();
Behaviour dc = new DarCalles(this);
addBehaviour(dc);
Behaviour          rq          =          new          RequestResponder(this,
AchieveREResponder.createMessageTemplate(FIPAProtocolNames.FIPA_REQUEST));
addBehaviour(rq);
```

```
    }// Fin del setup()

}//Fin del agente Trafico
```

### 9.2.6. AGENTEVISUALIZADOR.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)

AGENTE VISUALIZADOR
*****************************************************/
package Agentes;

import Ontology.*;

import java.io.*;
import java.util.*;

import JADE.core.*;
import JADE.core.behaviours.*;
import JADE.proto.*;
import JADE.lang.acl.*;
import JADE.domain.FIPAAgentManagement.*;
import JADE.domain.*;
import JADE.content.*;
import JADE.content.onto.basic.*;
import JADE.content.onto.*;
import JADE.content.lang.*;
import JADE.content.lang.sl.*;
import JADE.content.lang.sl.*;
import JADE.content.lang.Codec.*;
import JADE.content.onto.basic.*;

import java.awt.*;
import java.awt.Graphics;
import java.awt.Canvas;
import java.awt.event.*;

public class AgenteVisualizador extends Agent {

    //Para pintar los gráficos de la interficie
    AccionARealizar ar;
    Ventana v=Ventana.getInstance();
    Mapa map=new Mapa();

    //ListaCalles ListaCallesAnuladas=new ListaCalles();
    ListaCalles ListaCallesAnuladas=null;
    private final long MILLIS_PER_8HOURS = 60 * 1000;
    Date deathLine = new Date(System.currentTimeMillis() + MILLIS_PER_8HOURS);
```

```
 private ContentManager manager  = (ContentManager) getContentManager();
 private Codec codec=new SLCodec();
 private Ontology ontology=TaxiOntology.instance();


//*****************************************************************************
*****************************
 //Metodo que registra en el DF
 void RegistrarDF() {
//REGISTRAMOS la ontología y el lenguaje
manager.registerLanguage(codec,codec.getName());
manager.registerOntology(ontology,ontology.getName());

     //Preparación para dar de alta en el DF
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setName(TaxiOntology.VISUALIZADOR_SERVICE);
sd.setType(TaxiOntology.SERVICE_TYPE);
dfd.addServices(sd);
dfd.setName(getAID());

try {
    DFService.register(this, dfd);
    System.out.println("["+getLocalName()+"]:"+" SE HA REGISTRADO EN EL DF");
    }
          catch (JADE.domain.FIPAException e) {
          System.out.println("["+getLocalName()+"]:"+"No se ha podido registrar en el
DF");
          doDelete();
          }
     try {
          Thread.sleep(1000);
     }
     catch (Exception e) {
     }
ar = new AccionARealizar(1,0,0,null,null);
v.addFigura(ar);
map.pinta();

 }//Fin de RegistrarDF


//*****************************************************************************
*****************************
 //Método que se encarga de buscar los agentes que deseo en el DF
 //TaxiOntology.SERVICE_TYPE,TaxiOntology.TAXI_SERVICE
 private String getAgentDF(String type,String name) {
String agentFound = new String();
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(type);
sd.setName(name);
```

```
dfd.addServices(sd);
try {
       SearchConstraints c = new SearchConstraints();
       c.setMaxResults(new Long(-1));  //buscaremos infinitos resultados (estilo
JADE2.x)
  DFAgentDescription[] DFAgents = DFService.search(this,dfd,c);
  int i=0;
  while ((DFAgents != null) && (i<DFAgents.length)) {
    DFAgentDescription agent = DFAgents[i];
    i++;
    Iterator services = agent.getAllServices();
    boolean found = false;
    ServiceDescription service = null;
    while (services.hasNext() && !found) {
       service = (ServiceDescription)services.next();
       found = (service.getType().equals(type) && service.getName().equals(name));
    }
    if (found) {
       AID aux=agent.getName();
            agentFound=aux.getName();    //Introduzco en agentsFound el AID de un
agente TAXI
    }
  }
     } catch (FIPAException e) {
    System.out.println("["+getLocalName()+"]:"+"Se  ha  producido  un  error:
"+e.getMessage());
       }
return agentFound;
 }



//*************************************************************************************
*******************************
 //Método para transformar un Array en una Colección
 public JADE.util.LEAP.List Array_to_List(int[] camino){
JADE.util.LEAP.List lista=new JADE.util.LEAP.ArrayList();
int i=0;int j=1;Integer num;

while(camino[j]!=-1){
       num=new Integer(camino[i]);
       lista.add(num);
       num=new Integer(camino[j]);
       lista.add(num);
       i++;j=i+1;
}

return lista;
 }


//*************************************************************************************
```

```
*****************************
 //Método que crea el mensaje para informar al taxi de que ya no está ocupado
 ACLMessage EnviarMsgDesocupado(ACLMessage req){

    //Creamos el mensaje y rellenamos los campos claves
    ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
    msg.setSender(getAID());
    msg.addReceiver(req.getSender());
    msg.setLanguage(codec.getName());
    msg.setOntology(ontology.getName());
    msg.setProtocol(JADE.proto.FIPAProtocolNames.FIPA_REQUEST);
    Desocupado doc=new Desocupado();
    try{
  Action a=new Action(getAID(),doc);
  manager.fillContent(msg,a);
    }catch(Exception e) {
       e.printStackTrace();
    }
    return msg;
 }// Fin de EnviarMsgDesocupado


//*****************************************************************************
*****************************
 //Este método te da la posicion del taxi después de hacer el servicio
 public Posicion ultima_pos(JADE.util.LEAP.List l){
Coordenadas2 c2=new Coordenadas2();
Iterator it=l.iterator();Long n=new Long(0);
while(it.hasNext()){
      n=(Long)it.next();
}
Posicion                                                          pos=new
Posicion();pos.setx(c2.getCol(n.intValue()));pos.sety(c2.getFil(n.intValue()));
return pos;
 }


//*****************************************************************************
*****************************
 //Este método te da la posicion del taxi después de hacer el servicio
 public Posicion primera_pos(JADE.util.LEAP.List l){
Coordenadas2 c2=new Coordenadas2();
Long n=(Long)l.get(0);
Posicion                                                          pos=new
Posicion();pos.setx(c2.getCol(n.intValue()));pos.sety(c2.getFil(n.intValue()));
return pos;
 }


//*****************************************************************************
*****************************
 //Este BEHAVIOUR es el que controla la recepción del REQUEST
```

```
 class RequestResponder extends AchieveREResponder {

public RequestResponder(Agent a, MessageTemplate mt) {
    super(a, mt);
}

//Método que se ejecuta automáticamente cuando recibe un REQUEST
protected ACLMessage prepareResponse(ACLMessage request) {
    ACLMessage response = request.createReply();
         response.setPerformative(ACLMessage.AGREE);
    return response;
} //Fin de prepareResponse

//Se ejecuta automáticamente solo cuadno el método anterior (prepareResponse) ha
dado una respuesta AGREE
protected ACLMessage prepareResultNotification(ACLMessage request, ACLMessage
response) throws FailureException {
    ACLMessage resNot = request.createReply();
    try{
        Action a=(Action)manager.extractContent(request);

             if (a.getAction() instanceof DibujarPersona){
            DibujarPersona                                          dp=new
DibujarPersona();dp=(DibujarPersona)a.getAction();
            Posicion p=new Posicion();p=dp.getposicion();
            ar = new AccionARealizar(2,p.getx(),p.gety(),Color.green,null);
            v.addFigura(ar);
            map.pinta();
             }else if(a.getAction() instanceof BorrarPersona) {
            BorrarPersona bp=new BorrarPersona();bp=(BorrarPersona)a.getAction();
            Posicion p=new Posicion();p=bp.getposicion();
            ar = new AccionARealizar(2,p.getx(),p.gety(),Color.green,null);
            v.addFigura(ar);
            map.pinta();
             }else if(a.getAction() instanceof DibujarTaxi) {
            DibujarTaxi dt=new DibujarTaxi();dt=(DibujarTaxi)a.getAction();
            Posicion p=new Posicion();p=dt.getposicion();
            ar = new AccionARealizar(2,p.getx(),p.gety(),Color.red,null);
            v.addFigura(ar);
            map.pinta();
        }else if(a.getAction() instanceof BorrarTaxi) {
            BorrarTaxi bt=new BorrarTaxi();bt=(BorrarTaxi)a.getAction();
            Posicion p=new Posicion();p=bt.getposicion();
            ar = new AccionARealizar(2,p.getx(),p.gety(),Color.white,null);
            v.addFigura(ar);
            map.pinta();
        }else if(a.getAction() instanceof DibujarRecorridoTaxi) {
            DibujarRecorridoTaxi drt=new DibujarRecorridoTaxi();
            drt=(DibujarRecorridoTaxi)a.getAction();
            RecorridoTaxi rt;rt=drt.getRecorridoTaxi();
            ListaCalles l1=new ListaCalles();l1=rt.getListaCallesBuscar();
            ListaCalles l2=new ListaCalles();l2=rt.getListaCallesServir();
```

```
                ar = new AccionARealizar(3,0,0,Color.blue,l1.getListaCalles());
                JADE.util.LEAP.List aux=l1.getListaCalles();
                if(aux!=null) {
                  if(aux.size()!=0){
                    v.addFigura(ar);
                    map.pinta();
                  }
                }
                try{Thread.sleep(3000);}catch(Exception e){}
                ar = new AccionARealizar(3,0,0,Color.orange,l2.getListaCalles());
                v.addFigura(ar);
                map.pinta();
                try{Thread.sleep(8000);}catch(Exception e){}
                ar = new AccionARealizar(3,0,0,Color.white,l1.getListaCalles());
                if(aux!=null) {
                  if(aux.size()!=0){
                    v.addFigura(ar);
                    map.pinta();
                  }
                }
                ar = new AccionARealizar(3,0,0,Color.white,l2.getListaCalles());
                v.addFigura(ar);
                map.pinta();
                if(aux!=null) {
                  if(aux.size()!=0){
                       Posicion p=new Posicion();p=primera_pos(l1.getListaCalles());
//Para borrar al taxi
                       ar                          =                          new
AccionARealizar(2,p.getx(),p.gety(),Color.white,null);
                       v.addFigura(ar);
                       map.pinta();
                  }else{
                       Posicion p=new Posicion();p=primera_pos(l2.getListaCalles());
//Para borrar al taxi
                       ar                          =                          new
AccionARealizar(2,p.getx(),p.gety(),Color.white,null);
                       v.addFigura(ar);
                       map.pinta();
                  }
                }

                Posicion                    p=ultima_pos(l2.getListaCalles());
//Para pintar el taxi en la nueva pos
                ar = new AccionARealizar(2,p.getx(),p.gety(),Color.red,null);
                v.addFigura(ar);
                map.pinta();

                //AQUI LE INDICAMOS QUE YA SE HA REALIZADO TODO EL SERVICIO Y EL TAXI
ESTÁ DESOCUPADO
                Behaviour b;
                ACLMessage desocupado = EnviarMsgDesocupado(request);
                b = new RequestInitiator(AgenteVisualizador.this, desocupado);
```

```
                addBehaviour(b);
            }

              }catch(Exception e) {
            e.printStackTrace();
        }
        resNot.setPerformative(ACLMessage.INFORM);
        return resNot;
    }// Fin de prepareResultNotification

     } // Fin de la clase RequestResponder


    //*************************************************************************************
    ******************************
     //Método que crea el mensaje para enviar la acción DAMECALLESANULADAS a Trafico
     ACLMessage EnviarMsgDameCalles(){

        //Creamos el mensaje y rellenamos los campos claves
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        AID                    receiver                    =                    new
    AID(getAgentDF(TaxiOntology.SERVICE_TYPE,TaxiOntology.TRAFICO_SERVICE),false);
        msg.setSender(getAID());
        msg.addReceiver(receiver);
        msg.setLanguage(codec.getName());
        msg.setOntology(ontology.getName());
        msg.setProtocol(JADE.proto.FIPAProtocolNames.FIPA_REQUEST);

        //Rellenamos la ACCION BUSCARTAXI
        DameCallesAnuladas d=new DameCallesAnuladas();

        try{
      //Rellenamos el contenido (SL requires actions to be included into the action
    construct)
      Action a=new Action(getAID(),d);
      manager.fillContent(msg,a);
        }catch(Exception e) {
            e.printStackTrace();
        }

        //Devolvemos el ACLMessage (msg)
        return msg;
     }// Fin de EnviarMsgPeticion


    //*************************************************************************************
    ******************************
      //Clase que inicia el Request NOTA:NO CAL HACER EL SEND YA QUE LO HACE
    AUTOMÁTICAMENTE
     class RequestInitiator extends AchieveREInitiator {

    public RequestInitiator(Agent a, ACLMessage req) {
```

```
    super(a, req);
}

protected void handleInform(ACLMessage inform) {
         try{
        if(ListaCallesAnuladas!=null){ //Para cuando cambia de fichero de calles
anuladas borre las calles anuladas antiguas
             ar                              =                              new
AccionARealizar(3,0,0,Color.white,ListaCallesAnuladas.getListaCalles());
              v.addFigura(ar);
              map.pinta();
              }
         Action a=(Action)manager.extractContent(inform);
          System.out.println("["+getLocalName()+"]:"+" HA RECIBIDO LAS CALLES
ANULADAS DE TRAFICO");
         ListaCallesAnuladas=(ListaCalles)a.getAction();
        ar                                =                                new
AccionARealizar(3,0,0,Color.black,ListaCallesAnuladas.getListaCalles());
        v.addFigura(ar);
        map.pinta();
         }catch(Exception e) {
       e.printStackTrace();
    }
}
 } // Fin de la clase RequestInitiator


//*************************************************************************************
*****************************
 //método que simplemente te monta TaxiCache y TaxiCacheIterator. Luego lanza el
UPDAteCache
 private class DarCalles extends OneShotBehaviour {
DarCalles(Agent agent) {
   super(agent);
}

public void action() {
   try {
       Thread.sleep(1000);
   } catch (Exception e) {}
   //obtengo la lista de las calles anuladas dependiendo de la hora del sistema
   ACLMessage request = EnviarMsgDameCalles();
   Behaviour b = new RequestInitiator(AgenteVisualizador.this, request);  //Envia
un mensaje a Trafico para que le de calles anuladas
   addBehaviour(b);
   new horaUPDAte(deathLine,(Agent)myAgent);
}
 }


//*************************************************************************************
*****************************
```

```
    //Thread que actualiza la "cache" de TAXIS
    class horaUPDAte extends Thread implements Runnable {
        long millis;
   Agent myAgent;

   horaUPDAte(Date deathLine, Agent agent) {
      super();
      millis = deathLine.getTime() - System.currentTimeMillis();
      if(millis < 0)
          {
             millis=0;
          }
      myAgent = agent;
      this.start();
   }

   public void run(){
      try {
          //Cuando entra aquí es que debe hacer una nueva actualización (ya ha pasado
   el deathLine)
          Thread.sleep(millis);
          addBehaviour(new DarCalles(AgenteVisualizador.this));
          deathLine = new Date(System.currentTimeMillis() + MILLIS_PER_8HOURS);
      } catch (Exception e) { }
   }
    }

    //*****************************************************************************
    //Metodo setup que todo Agente tiene
    //*****************************************************************************
    public void setup() {
   v.addDibujo(map);
   RegistrarDF();
   Behaviour dc = new DarCalles(this);
   addBehaviour(dc);

   Behaviour          rq          =          new          RequestResponder(this,
   AchieveREResponder.createMessageTemplate(FIPAProtocolNames.FIPA_REQUEST));
   addBehaviour(rq);
    }// Fin del setup()

}//Fin del agente Visualizador
```

### 9.2.7. COORDCLIENTE.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
*****************************************************/
package Agentes;

import java.util.*;
```

```
public class CoordCliente {

   int matriz[]=new int[11];
   int matrizpulsador[]=new int[11];

   public int getcoord(int x) {  //matriz para dibujar los puntos
     matriz[0]=6;
     matriz[1]=28;
     matriz[2]=50;
     matriz[3]=72;
     matriz[4]=94;
     matriz[5]=116;
     matriz[6]=138;
     matriz[7]=160;
     matriz[8]=182;
     matriz[9]=204;
     matriz[10]=225;
          return matriz[x];
        }

   public int getpulsa(int y) { //matriz para saber donde hemos pulsado(empieza en 6)
          matrizpulsador[0]=18;
          matrizpulsador[1]=40;
          matrizpulsador[2]=62;
          matrizpulsador[3]=84;
          matrizpulsador[4]=106;
          matrizpulsador[5]=128;
          matrizpulsador[6]=150;
          matrizpulsador[7]=172;
          matrizpulsador[8]=194;
          matrizpulsador[9]=216;
          matrizpulsador[10]=238;
                return matrizpulsador[y];
        }

}
/*
matriz[0]=3;
          matriz[1]=22;
          matriz[2]=41;
          matriz[3]=59;
          matriz[4]=77;
          matriz[5]=95;
          matriz[6]=113;
          matriz[7]=161;
          matriz[8]=183;
          matriz[9]=205;
          matriz[10]=227;
                return matriz[x];
*/
```

### 9.2.8. COORDENADAS2.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)
*****************************************************/
package Agentes;

import java.util.*;

public class Coordenadas2 {

   int vc[]={0,1,2,3,4,5,6,7,8,9,10,
          0,1,2,3,4,5,6,7,8,9,10,
          0,1,2,3,4,5,6,7,8,9,10,
          0,1,2,3,4,5,6,7,8,9,10,
          0,1,2,3,4,5,6,7,8,9,10,
          0,1,2,3,4,5,6,7,8,9,10,
          0,1,2,3,4,5,6,7,8,9,10,
          0,1,2,3,4,5,6,7,8,9,10,
          0,1,2,3,4,5,6,7,8,9,10,
          0,1,2,3,4,5,6,7,8,9,10};

   int vf[]={0,0,0,0,0,0,0,0,0,0,0,
          1,1,1,1,1,1,1,1,1,1,1,
          2,2,2,2,2,2,2,2,2,2,2,
          3,3,3,3,3,3,3,3,3,3,3,
          4,4,4,4,4,4,4,4,4,4,4,
          5,5,5,5,5,5,5,5,5,5,5,
          6,6,6,6,6,6,6,6,6,6,6,
          7,7,7,7,7,7,7,7,7,7,7,
          8,8,8,8,8,8,8,8,8,8,8,
          9,9,9,9,9,9,9,9,9,9,9,
          10,10,10,10,10,10,10,10,10,10,10};

   public int getCol(int x) {
                return vc[x];
        }

   public int getFil(int y) {
                return vf[y];
        }

}
```

### 9.2.9. COORDENADAS.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)
*****************************************************/
package Agentes;

import java.util.*;

public class Coordenadas {

   int matrizcol[]=new int[11];
   int matrizfil[]=new int[11];

   public int getCol(int x) {
          matrizcol[0]=70;
          matrizcol[1]=158;
          matrizcol[2]=246;
          matrizcol[3]=334;
          matrizcol[4]=422;
          matrizcol[5]=510;
          matrizcol[6]=598;
          matrizcol[7]=686;
          matrizcol[8]=774;
          matrizcol[9]=862;
          matrizcol[10]=950;
                return matrizcol[x];
        }

   public int getFil(int y) {
          matrizfil[0]=43;
          matrizfil[1]=100;
          matrizfil[2]=157;
          matrizfil[3]=214;
          matrizfil[4]=271;
          matrizfil[5]=328;
          matrizfil[6]=385;
          matrizfil[7]=442;
          matrizfil[8]=499;
          matrizfil[9]=556;
          matrizfil[10]=613;
                return matrizfil[y];
        }

}
```

## 9.2.10. DATOSMAPA.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)
*****************************************************/
package Agentes;

import java.util.*;
import java.io.*;

public class DatosMapa {

   InfoMap mapa[][]=new InfoMap[121][4];         //Matriz de 121 posiciones donde
   representaremos todo el mapa
   InfoMap mapa_aux[][]=new InfoMap[121][4];         //Lo utilizo para actualizarlo
   con las calles anuladas
   int traduccion[][]=new int[11][11];          //matriz para saber nodo i sus
   coordenadas en el mapa
   String nom_fitxer;

   public DatosMapa(String nom) {
          nom_fitxer=nom;
          inicia_traduccion();
           try {
              readInfoMapa();
           }catch(Exception e) {
              System.out.println("Error leyendo del fichero de DATOS DEL MAPA");
           }
   }

   public void inicia_traduccion() {
     int bledrup=0;
     for(int i=0;i<11;i++) {
       for(int j=0;j<11;j++) {
         traduccion[i][j]=bledrup;
         bledrup++;
       }
     }
   }

   public void readInfoMapa() throws Exception
   {
          FileReader profile = new FileReader(".\\pfc\\Agentes\\"+nom_fitxer);
           BufferedReader b= new BufferedReader(profile);

          String miS=null;String s=new String();
          int i=0;int numnodo=0;
          int nodoactual;
          int nodosucc;
```

```
	int coste;
	int numsucc=0;
	Integer x;

	while (numnodo<=120){
		miS=b.readLine();
		while (miS.charAt(i)!=' ' && miS.charAt(i)!='*') {
			s=s+miS.charAt(i);
			i++;
		}
		x=Integer.decode(s);
		nodoactual=x.intValue();i++;s=new String();
		while(miS.charAt(i)!='*') {
			InfoMap im=new InfoMap();
			while (miS.charAt(i)!=' ' && miS.charAt(i)!='*') {
				s=s+miS.charAt(i);
				i++;
			}
			x=Integer.decode(s);
			im.succ=x.intValue();i++;s=new String();
			while (miS.charAt(i)!=' ' && miS.charAt(i)!='*') {
				s=s+miS.charAt(i);
				i++;
			}

			x=Integer.decode(s);
			im.coste=x.intValue();if (miS.charAt(i)==' ') i++;s=new
String();

			mapa[nodoactual][numsucc]=im;  //Aquí le asigno los valores a
la matriz
			numsucc++;

		}
		i=0;numnodo++;numsucc=0;
	}
	mapa_aux=mapa;

}

public InfoMap[] getInfoSucc(int x) {
	return mapa_aux[x];
     }

public void presentaInfoNodo(int x) {
	InfoMap i[]=new InfoMap[4];
	i=getInfoSucc(x);
	int j=0;
	while (i[j]!=null){
		System.out.println(i[j].succ);
		System.out.println(i[j].coste);
		j++;
```

```
		}
	}

	public int CosteAresta(int i,int j){
		InfoMap mc[]=new InfoMap[4];
		int n=0;
		mc=mapa_aux[i];
		while(mc[n].succ!=j){
			n++;
		}
		return mc[n].coste;

	}

	public int DarCoste(int puntero, int[] l) {
		int i=puntero;int j=puntero+1;
		int coste=0;
		while (l[j]!=-1){
			coste=coste+CosteAresta(l[i],l[j]);
			j++;i++;
		}
		return coste;
	}

	public InfoMap[][] ActualizarCallesAnuladas(JADE.util.LEAP.List cortadas){
		InfoMap mapa_aux[][]=new InfoMap[121][4];
		InfoMap mc[]=new InfoMap[4];
		Long x1,x2;int i=0;int j=0;
		mapa_aux=mapa;

		Iterator it=cortadas.iterator();
		while (it.hasNext()){
			x1=(Long)it.next();
			x2=(Long)it.next();
			mc=mapa_aux[x1.intValue()];
		InfoMap mc_aux[]=new InfoMap[4];
			while(mc[i]!=null){
				if(mc[i].succ!=x2.intValue()){
					mc_aux[j]=mc[i];
					j++;
				}
				i++;
			}
			mapa_aux[x1.intValue()]=mc_aux;i=0;j=0;
		}
		return mapa_aux;
	}

}
```

### 9.2.11. INFOMAP.JAVA

```
/**************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)
**************************************************/
package Agentes;

import java.util.*;

public class InfoMap {

   int succ;
   int coste;
}
```

### 9.2.12. MAPA.JAVA

```
/**************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)
**************************************************/
package Agentes;

import java.awt.Graphics;
import java.awt.Canvas;
import java.awt.*;

import java.io.*;
import java.awt.Graphics;
import java.awt.Canvas;
import java.awt.Font;
import java.awt.Color;
import java.awt.Polygon;
import java.awt.image.*;
import java.awt.*;
import java.applet.Applet;
import JADE.util.LEAP.*;

class Mapa extends Canvas{

   boolean flag = false;
   Ventana v = null;
   Graphics g=null;

   int ancho=81;
   int alto=50;
   int calle=7;
   String numeros[]={"0","1","2","3","4","5","6","7","8","9","10"};
```

```
Coordenadas c=new Coordenadas();
Coordenadas2 co2=new Coordenadas2();


boolean db=false;
Image imgDobleBuffer=null;
Graphics contextoDobleBuffer=null;

public Mapa(){
	v = Ventana.getInstance();
	imgDobleBuffer=v.createImage(1024,739);
	contextoDobleBuffer=imgDobleBuffer.getGraphics();
	db=true;
}


public void pinta(){
	g=getGraphics();
	this.pinta(g);
}

public void pinta(Graphics g){
      if(v.figuras!=null) {
	java.util.Enumeration enumFiguras = v.figuras.elements();
	if(db && contextoDobleBuffer!=null){
		contextoDobleBuffer.clearRect(0,0,1024,739);

		while (enumFiguras.hasMoreElements()){
			AccionARealizar	acc	=	(AccionARealizar)
enumFiguras.nextElement();
			switch(acc.accion) {
				case 1:
contextoDobleBuffer=DibujaMapa(contextoDobleBuffer);break;
				case 2:
contextoDobleBuffer=DibujaItem(contextoDobleBuffer,acc.col,acc.fil,acc.color);break
;
				case 3:
contextoDobleBuffer=DibujaRecorrido(contextoDobleBuffer,acc.calles,acc.color);break
;
			}
		}
		g.drawImage(imgDobleBuffer,0,0,this);
	}
	    }
}


public void paint(Graphics g){
	if (flag){
		this.pinta();
	}
	flag = true;
```

```
    }


    //*************************************************************************************
    ******************************
    public Graphics DibujaItem(Graphics contextoDobleBuffer,int col,int fil,Color
    color) {

           contextoDobleBuffer.setColor(color);

     contextoDobleBuffer.fill3DRect(c.getCol(col),c.getFil(fil),calle,calle,true)
    ;
           return contextoDobleBuffer;
    }//Fin de DibujaRecorrido

//*************************************************************************************
    ******************************
    public int min(int x,int y) {
           int resultat;
           if(x>y){
              resultat=y;
           }else {resultat=x;}
           return resultat;
    }//Fin del minI


    //*************************************************************************************
    ******************************
    public Graphics DibujaRecorrido(Graphics contextoDobleBuffer,JADE.util.LEAP.List
    l,Color color) {

           contextoDobleBuffer.setColor(color);
           Iterator it=l.iterator();int i=0;

           if(l.get(i) instanceof Integer) {
                  Integer num1,num2;
                  int c1,f1,c2,f2;  //col fil
                  while (it.hasNext()) {
                         num1=(Integer)it.next();
                         num2=(Integer)it.next();
                         f1=co2.getFil(num1.intValue());f2=co2.getFil(num2.intValue());
                         c1=co2.getCol(num1.intValue());c2=co2.getCol(num2.intValue());

                         if(c1==c2){  //pintamos vertical

    contextoDobleBuffer.fill3DRect(c.getCol(c1),c.getFil(min(f1,f2)),calle,alto+calle,t
    rue);
                         }else { //pintamos horizontal

    contextoDobleBuffer.fill3DRect(c.getCol(min(c1,c2)),c.getFil(f1),ancho+calle,calle,
    true);
                         }
```

```
                }
        } else {
                Long num1,num2;
                int f1,c1,c2,f2;  //col fil
                while (it.hasNext()) {
                        num1=(Long)it.next();
                        num2=(Long)it.next();
                        f1=co2.getFil(num1.intValue());f2=co2.getFil(num2.intValue());
                        c1=co2.getCol(num1.intValue());c2=co2.getCol(num2.intValue());

                        if(c1==c2){

   contextoDobleBuffer.fill3DRect(c.getCol(c1),c.getFil(min(f1,f2)),calle,alto+calle,t
   rue);
                        }else{

   contextoDobleBuffer.fill3DRect(c.getCol(min(c1,c2)),c.getFil(f1),ancho+calle,calle,
   true);
                        }
                }
        }
        return contextoDobleBuffer;
   }//Fin de DibujaRecorrido


//*****************************************************************************
   ******************************
   //Método que se encarga de realizar la interficie principal (mapa, imágenes,
   leyenda...)
   public Graphics DibujaMapa(Graphics contextoDobleBuffer) {

        //para el cuadro de arriba con el título
                contextoDobleBuffer.setColor(Color.red);
        contextoDobleBuffer.setFont(new Font("Courier New",Font.BOLD,15));
        contextoDobleBuffer.drawString("Plataforma de SMA para gestión de
   Taxis",315,11);
        contextoDobleBuffer.drawLine(5,13,1013,13);

        //El cuadro del callejero
        contextoDobleBuffer.setColor(Color.black);
        contextoDobleBuffer.fill3DRect(60,33,909,600,false);
        contextoDobleBuffer.setColor(Color.white);
        contextoDobleBuffer.fill3DRect(70,43,890,580,true);

        //El cuadro de abajo con la leyenda
        contextoDobleBuffer.setColor(Color.black);
        contextoDobleBuffer.fill3DRect(60,639,909,63,false);
        contextoDobleBuffer.setColor(Color.white);
        contextoDobleBuffer.fill3DRect(63,642,903,57,true);

        //Leyenda
        contextoDobleBuffer.setColor(Color.green);
```

```
            contextoDobleBuffer.fill3DRect(83,651,15,15,false);
            contextoDobleBuffer.setColor(Color.red);
            contextoDobleBuffer.fill3DRect(83,676,15,15,false);
            contextoDobleBuffer.setColor(Color.blue);
            contextoDobleBuffer.fill3DRect(450,651,15,15,false);
            contextoDobleBuffer.setColor(Color.orange);
            contextoDobleBuffer.fill3DRect(450,676,15,15,false);
            contextoDobleBuffer.setColor(Color.black);
            contextoDobleBuffer.setFont(new Font("Arial Narrow",Font.BOLD,15));
            contextoDobleBuffer.drawString("Posición del Cliente",102,665);
            contextoDobleBuffer.drawString("Posición del Taxi",102,690);
            contextoDobleBuffer.drawString("Taxi en busca de un cliente",469,665);
            contextoDobleBuffer.drawString("Taxi   ofreciendo   el   servicio   al
      cliente",469,690);
            contextoDobleBuffer.fill3DRect(750,651,15,15,false);
            contextoDobleBuffer.drawString("Calle cortada o en obras",769,665);

            //Aqui va a dibujar las manzanas de las calles
            for(int j=50;j<600;j=j+alto) {
                  for(int i=77;i<900;i=i+ancho){
                        contextoDobleBuffer.setColor(Color.gray);
                        contextoDobleBuffer.fill3DRect(i,j,ancho,alto,false);

                        i=i+calle;
                  }
                  j=j+calle;
                  }
            int f1,c1,f2,c2;
            c1=40;f1=56;
            c2=71;f2=28;
            for(int i=0;i<11;i++){
                        contextoDobleBuffer.setColor(Color.black);
                        contextoDobleBuffer.setFont(new              Font("Arial
      Narrow",Font.BOLD,15));
                        contextoDobleBuffer.drawString(numeros[i],c1,f1);
                        contextoDobleBuffer.drawString(numeros[i],c2,f2);
                        f1=f1+alto+calle;
                        c2=c2+ancho+calle;
             }
            return contextoDobleBuffer;

      }//Fin de DibujaMapa

}
```

### 9.2.13. MAPACLIENTE.JAVA

```
/*************************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
*************************************************************/

package Agentes;

import java.awt.Graphics;
import java.awt.Canvas;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.awt.Font;
import java.awt.Color;
import JADE.util.LEAP.Iterator;
import JADE.util.LEAP.*;

/* This class exists solely to put a frame around the coordinate area. */
class MapaCliente extends Canvas {
    MapaCliente controller;
    Point punto = null;
    private Graphics g;
    int ancho=19;
    int alto=19;
    int calle=3;
    int accion=0; //pintar mapa inicialmente
    int turno=0; //incialmente pint
    boolean origen=true; //incialmente pincharemos para origen
    boolean
   repetimos=false,dibuja_recorrido=false,dibuja_origen=false,dibuja_destino=false,hay
   _fichero=false;
    boolean borrar_recorrido=false;
    CoordCliente co=new CoordCliente();
    Coordenadas2 co2=new Coordenadas2();
    VentanaCliente v=VentanaCliente.getInstance();
    private JADE.util.LEAP.List recorrido;
    private int co_x,co_y,dco_x,dco_y;
    String cruces_or,cruces_des;
    int edificios[][];

    public MapaCliente() {
        super();
        this.controller = controller;
        g=getGraphics();

        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
          int x = e.getX();
          int y = e.getY();
                if (punto == null) {
```

```
            punto = new Point(x, y);
        } else {
            punto.x = x;
            punto.y = y;
        }
        g=getGraphics();

        dibuja_recorrido=false;
        if (borrar_recorrido) { //para borrar repintaremos de blanco
          pinta_camino(recorrido,Color.white);
          borrar_recorrido=false;
        }

        if (repetimos) {
          dibuja_origen=false;
          dibuja_destino=false;
          repetimos=false;


//***************dibujamos cuadricula para pinchar
          for(int j=18;j<226;j=j+22) {
              g.setColor(Color.green);
              g.drawLine(j,6,j,226);
              g.drawLine(6,j,226,j);
          }

          g.setColor(Color.white);
          g.fill3DRect(6,233,222,30,true); //limpiamos cuadrado de abajo
          g.setColor(Color.black);
          g.setFont(new Font("Arial Narrow",Font.BOLD,10));
          g.drawString("Origen:  ",7,245);
          g.drawString("Destino: ",7,259);
//*********************************************/
        }
        else {
          if (origen) {
            dibuja_origen=true;
            co_x=dameCoord(punto.x);
            co_y=dameCoord(punto.y);
            int pos_x=donde_estoy(punto.x);
            int pos_y=donde_estoy(punto.y);
            cruces_or=v.cruces_instance(pos_x,pos_y);
            if (cruces_or!=null) hay_fichero=true;
            v.set_inicix(pos_x);
            v.set_iniciy(pos_y);
            origen=false;


 /************************************************/
              g.setColor(Color.yellow);
            g.draw3DRect(co_x,co_y,22,22,true);
            if (hay_fichero) { //si hay fichero cargado lo usamos
```

```
                    g.setColor(Color.black);
                    g.setFont(new Font("Arial Narrow",Font.BOLD,10));
                    g.drawString("Origen:  "+cruces_or,7,245);
                  }
            /*************************************************/


                }
                else { //dibujamos destino
                   dibuja_destino=true;
                  origen=true;
                  dco_x=dameCoord(punto.x);
                  dco_y=dameCoord(punto.y);
                  int dpos_x=donde_estoy(punto.x);
                  int dpos_y=donde_estoy(punto.y);
                  cruces_des=v.cruces_instance(dpos_x,dpos_y);
                  if (cruces_des!=null) hay_fichero=true;
                  v.set_destix(dpos_x);
                  v.set_destiy(dpos_y);
                  repetimos=true;

   /*************************************************/
                  g.setColor(Color.blue);
                  g.draw3DRect(dco_x,dco_y,22,22,true);
                  if (hay_fichero) { //si bay fichero cargado lo usamos
                    g.setColor(Color.black);
                    g.setFont(new Font("Arial Narrow",Font.BOLD,10));
                    g.drawString("Destino: "+cruces_des,7,259);
                  }
   /*************************************************/

                }
              }
    }
});
 }

 public void paint(Graphics g) {
   Dimension d = getSize();
   Color bg = getBackground();
   int[] xpuntos=new int[4];
   int[] ypuntos=new int[4];


    g.setColor(Color.black);
     g.fill3DRect(0,0,234,232,true); //cuadrado del mapa
     g.fill3DRect(0,232,234,35,true); //cuadrado de abajo
     g.setColor(Color.white);
     g.fill3DRect(6,6,222,222,true); //cuadrado del mapa
     g.fill3DRect(6,233,222,30,true); //cuadrado de abajo
```

```
//Aqui va a dibujar las manzanas de las calle


  edificios=v.genera_mapa(); //cogemos el mapa q hay q hay q dibujar
  g.setColor(Color.gray);

  int i=0;
  while (edificios[i][0]!=0) {
    xpuntos[0]=edificios[i][1];ypuntos[0]=edificios[i][2];
    xpuntos[1]=edificios[i][3];ypuntos[1]=edificios[i][4];
    xpuntos[2]=edificios[i][5];ypuntos[2]=edificios[i][6];
    if(edificios[i][0]==4) {
      xpuntos[3]=edificios[i][7];ypuntos[3]=edificios[i][8];
    }
    g.fillPolygon(xpuntos,ypuntos,edificios[i][0]);
    i++;
  }


//dibujamos cuadricula para pinchar
for(int j=18;j<226;j=j+22) {
   g.setColor(Color.green);
   g.drawLine(j,6,j,226);
   g.drawLine(6,j,226,j);
}
g.setColor(Color.black);
g.setFont(new Font("Arial Narrow",Font.BOLD,10));
g.drawString("Origen:  ",7,245);
g.drawString("Destino: ",7,259);

//************ DIBUJO SELECTIVO*****************/
if (dibuja_origen) {
   g.setColor(Color.yellow);
  g.draw3DRect(co_x,co_y,22,22,true);
  if (hay_fichero) { //si bay fichero cargado lo usamos
    g.setColor(Color.black);
    g.setFont(new Font("Arial Narrow",Font.BOLD,10));
    g.drawString("Origen:  "+cruces_or,7,245);
  }
}
if (dibuja_destino) {
  g.setColor(Color.blue);
  g.draw3DRect(dco_x,dco_y,22,22,true);
  if (hay_fichero) { //si bay fichero cargado lo usamos
    g.setColor(Color.black);
    g.setFont(new Font("Arial Narrow",Font.BOLD,10));
    g.drawString("Destino: "+cruces_des,7,259);
  }
}
if (dibuja_recorrido){
   pinta_camino(recorrido,Color.red);
}
```

```
}

public void camino_ontheway(JADE.util.LEAP.List calles) {
  recorrido=calles;
  dibuja_recorrido=true;

  pinta_camino(recorrido,Color.red);
  borrar_recorrido=true;
}



public int dameCoord(int x) {
  int i=0;
  int result=0;
  boolean trobat=false;
  while (!trobat&&i<11) {
    if (x<=co.getpulsa(i)) {
      trobat=true;
      result=co.getpulsa(i)-22;
      if (result<0)
        result=0;
      else
        if (result>200)
          result=210;
    }
    i++;
  }
  return result;
}

public int donde_estoy(int x) {
  int i=0;
  int result=0;
  boolean trobat=false;
  while (!trobat&&i<11) {
    if (x<=co.getpulsa(i)) {
      trobat=true;
      result=i;
    }
    else
      i++;
  }
  return result;
}

public void reinicia_booleans() {
  dibuja_recorrido=false;
  dibuja_origen=false;
  dibuja_destino=false;
}
```

```
    public int min(int x,int y) {
         int resultat;
         if(x>y){
            resultat=y;
         }else {resultat=x;}
         return resultat;
    }

    public void pinta_camino(JADE.util.LEAP.List calles,Color colorillo) {
      g=getGraphics();
      g.setColor(colorillo);
      Iterator it=calles.iterator();int i=0;

      if(calles.get(i) instanceof Integer) {
        Integer num1,num2;
   int c1,f1,c2,f2;  //col fil
   while (it.hasNext()) {
     num1=(Integer)it.next();
     num2=(Integer)it.next();
     f1=co2.getFil(num1.intValue());f2=co2.getFil(num2.intValue());
     c1=co2.getCol(num1.intValue());c2=co2.getCol(num2.intValue());
     g.drawLine(co.getcoord(c1),co.getcoord(f1),co.getcoord(c2),co.getcoord(f2));
   }
      }
      else {
        Long num1,num2;
   int f1,c1,c2,f2;  //col fil
   while (it.hasNext()) {
     num1=(Long)it.next();
     num2=(Long)it.next();
     f1=co2.getFil(num1.intValue());f2=co2.getFil(num2.intValue());
     c1=co2.getCol(num1.intValue());c2=co2.getCol(num2.intValue());
     g.drawLine(co.getcoord(c1),co.getcoord(f1),co.getcoord(c2),co.getcoord(f2));
   }
      }
   }
}
```

### 9.2.14. VENTANA.JAVA

```
/*****************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
basado en código de Juan Infante García(16-10-2002)
*****************************************************/
package Agentes;

import java.awt.*;
import java.awt.event.*;
import java.util.Vector;

public class Ventana extends Frame{
```

```
	private static Ventana ventana = new Ventana();

	public Vector figuras = null;

	public void addFigura (AccionARealizar a){
		if (figuras == null) figuras = new Vector();
		figuras.add(a);
	}



	private Ventana(){

		this.setSize(1024,739);
		this.setTitle("SMA para la gestión de TAXIS de una ciudad");
		this.setVisible(true);
		setLayout(new BorderLayout(20,20));
	}

	public static Ventana getInstance(){
		return ventana;
	}


	public void addDibujo(Mapa d) {

		add("Center",d);
		addWindowListener(
			new WindowAdapter() {
			public void windowClosing(WindowEvent e) {
				dispose();
				System.exit(0);
			}
		});
		setVisible(true);
	}
}
```

### 9.2.15. VENTANACLIENTE.JAVA

```
/*********************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
*********************************************************/

package Agentes;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
```

```
public class VentanaCliente extends Frame implements ActionListener{

   private static VentanaCliente ventanacliente = new VentanaCliente();
   private Label l1,l2,l3,l4,l5,tag_idtaxi,tag_precio,tag_tiempo,tag_callejero;
        private Button b_s,b_c,b_display;
        private TextField t,cliente_x,cliente_y,destino_x,destino_y,num_p;
        private AgenteCliente myAgent;
        private Panel recibir;
        private VentanaMapa vm;
        private MenuBar mbar;
        private FileDialog selec_ciudad;
   private String filename; //nombre de fichero con la info de la ciudad
   private MenuItem mi1_1,mi2_1;
   private String tabla_cruces[][]=new String[11][11];
   private int tabla_edificios[][]=new int[200][9];  //maximo 200 edificios

   private VentanaCliente(){
          //******************menus******************//
          mbar=new MenuBar();
          Menu m=new Menu("File");
          m.addActionListener(this);
          m.add(mi1_1=new MenuItem("Leer Ciudad"));
          mi1_1.setActionCommand("leer_ciudad");
          mbar.add(m);
          m=new Menu("Help");
          m.addActionListener(this);
          m.add(mi2_1=new MenuItem("About..."));
          mi2_1.setActionCommand("about");
          mbar.add(m);
          setMenuBar(mbar);

          selec_ciudad=new FileDialog(this,"selecciona callejero",FileDialog.LOAD);

          this.setSize(240,300);
          this.setTitle("Taxis en tu mano");
          setLayout(new BorderLayout(5,5));
          setBackground(Color.lightGray);
          setFont(new Font("Helvetica",Font.BOLD,12));


         //********panel enviar******************//
          Panel enviar=new Panel();
          enviar.setLayout(new GridLayout(6,2)); //repartidos en 6 filas i 2 columnas

          enviar.add(l1=new Label("num pers."));
          enviar.add(num_p= new TextField(2));

          enviar.add(l2=new Label("cliente x."));
          enviar.add(cliente_x=new TextField(2));
```

```
          enviar.add(l3=new Label("cliente y."));
        enviar.add(cliente_y=new TextField(2));

        enviar.add(l4=new Label("destino x"));
        enviar.add(destino_x=new TextField(2));

        enviar.add(l5=new Label("destino y"));
        enviar.add(destino_y=new TextField(2));

        enviar.add(b_s = new Button("Send"));
        enviar.add(b_c = new Button("Clear"));
        b_s.addActionListener(this);
        b_c.addActionListener(this);

          //*******panel recibir*************//

          recibir=new Panel();
        recibir.setLayout(new GridLayout(5,1)); //repartidos en 5 filas i 1 columnas
        recibir.add(tag_callejero=new Label("callejero en uso: "));
          recibir.add(tag_idtaxi=new Label("id taxi: "));
          recibir.add(tag_precio=new Label("Precio: "));
          recibir.add(tag_tiempo=new Label("Tiempo espera: "));
          recibir.add(b_display = new Button("mapa"));
          b_display.addActionListener(this);

        add(enviar,BorderLayout.NORTH);
        add(recibir,BorderLayout.SOUTH);

        addWindowListener(
               new WindowAdapter() {
               public void windowClosing(WindowEvent e) {
                      dispose();
                      System.exit(0);
               }
        });
        this.setVisible(true);

}

public void initialise(AgenteCliente agent) {
        myAgent=agent;
     }

public static VentanaCliente getInstance(){
       return ventanacliente;
}

      public void muestra_resultado(String idtaxi,int precio,float tiempo) {
//metodo para mostrar el resultado de la peticion
        tag_idtaxi.setText("id taxi: "+idtaxi);
        tag_precio.setText("Precio: "+precio+ " centimos");
        tag_tiempo.setText("Tiempo espera: "+tiempo+ " minutos");
```

```
    }

public void actionPerformed(ActionEvent e) {  //metodo para cuando hay una accion
  if (e.getActionCommand().equals("Send"))
        myAgent.qtejodan();
      else
        if (e.getActionCommand().equals("Clear")) {
          num_p.setText("");
          cliente_x.setText("");
          cliente_y.setText("");
          destino_x.setText("");
          destino_y.setText("");
          tag_idtaxi.setText("id taxi: ");
          tag_precio.setText("Precio: ");
          tag_tiempo.setText("Tiempo: ");
        }
        else
          if (e.getActionCommand().equals("mapa")) {
            if (vm==null)
              vm=VentanaMapa.getInstance();
            else {
              vm.show(); //si la ventana esta oculta la mostramos.
            }
          }
          else
            if (e.getActionCommand().equals("leer_ciudad")) {
              selec_ciudad.show();
        filename=selec_ciudad.getFile();
        String directorio=selec_ciudad.getDirectory();
        directorio=directorio+filename;
        try {
          leer_calles(directorio);
        }catch(Exception ex) {}
        tag_callejero.setText("callejero en uso: "+filename);
        vm.reinicia_mapa();
      }
      else
        if (e.getActionCommand().equals("about")) {
                new DialogWindow();
      }
    }

    public int get_crew() {
      int x=Integer.parseInt(num_p.getText());
      return x;
}

    public int get_inicix() {
      int x=Integer.parseInt(cliente_x.getText());
      return x;
}
```

```
public int get_iniciy() {
       int x=Integer.parseInt(cliente_y.getText());
       return x;
}

public int get_destix() {
       int x=Integer.parseInt(destino_x.getText());
       return x;
}

public int get_destiy() {
       int x=Integer.parseInt(destino_y.getText());
       return x;
}
//----------------------------------------//
public void set_inicix(int x) {
       cliente_x.setText(Integer.toString(x));
}

public void set_iniciy(int x) {
       cliente_y.setText(Integer.toString(x));
}

public void set_destix(int x) {
       destino_x.setText(Integer.toString(x));
}

public void set_destiy(int x) {
       destino_y.setText(Integer.toString(x));
}


//*****************************************************************************
*****************************
// enviamos las calles de la coordenada
  public String cruces_instance(int x,int y) {
    return tabla_cruces[x][y];
  }

  public int[][] genera_mapa() {
    return tabla_edificios;
  }




//*****************************************************************************
*****************************
  //Metodo que lee el callejero de un fichero
public void leer_calles(String pathname) throws Exception {

       FileReader profile = new FileReader(pathname);
```

```
        BufferedReader b= new BufferedReader(profile);
    int i=0;
    String miS=null;String s=new String();Integer aux;
    int x=0,y=0,n_edif=0,indice=0;


    for(int p=0;p<200;p++)
      tabla_edificios[p][0]=0; //inicializamos tabla edificios


    while ((miS = b.readLine()) != null) {
      i=0;
      if(miS.charAt(i)!='%') {  //si no tiene % es q son nombres de calles
        for (int o=0;o<2;o++) {  //repetimos dos veces para coger la x i la y
          while (miS.charAt(i)!=' ') {
            s=s+miS.charAt(i);
            i++;
          }
          i++;
          aux=Integer.decode(s); //cogemos la coordenada
          s="";
          if (o==0)
            x=aux.intValue();
          else
            y=aux.intValue();
        }
        //estamos apuntando a la primera calle
        while (miS.charAt(i)!='#') {
          s=s+miS.charAt(i);
          i++;
        }
        tabla_cruces[x][y]=s;
        s="";
        i=0;
      }
      else { //tiene % delante, es info del mapa a dibujar
        i++; //nos saltamos el %
        s="";
        s=s+miS.charAt(i);
        aux=Integer.decode(s);
         tabla_edificios[n_edif][indice]=aux.intValue();//cogemos el numero de
coordenadas
        indice++;s="";
        i+=2; //nos saltamos num_arest i el $
        while (miS.charAt(i)!='#') {
         while (miS.charAt(i)!='$') {
             s=s+miS.charAt(i);
             i++;
          }
          aux=Integer.decode(s);
          tabla_edificios[n_edif][indice]=aux.intValue(); //coord. para dibujar
          indice++;i++;s="";
```

```
            }
            i=0;s="";
            indice=0;n_edif++;
          }
        }
        b.close();
           }
}

class DialogWindow extends Frame implements ActionListener {

    private TextArea text_area;

    public DialogWindow() {
          setTitle("About...");
          setBackground(Color.lightGray);
          setLayout(new BorderLayout(5,5));
         text_area = new TextArea(5,25);
         text_area.setEditable(false);
         text_area.setText("AgenteCliente Grafico(AWT)\nVersion: 1.0\nAlexandre Viejo
   Galicia\nETSE,Departamento de IA");
         add(text_area,BorderLayout.CENTER);
         Button boton = new Button("Aceptar");
   boton.addActionListener(this);
         add(boton,BorderLayout.SOUTH);
         setVisible(true);
         pack();

   addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent e) {
         dispose();
       }
   });
    }

    public void actionPerformed(ActionEvent ev) {
        if (ev.getActionCommand().equals("Aceptar"))
             dispose();
    }
}
```

## 9.2.16. VENTANAMAPA.JAVA

```
/*************************************************************
Alexandre Viejo Galicia, 6-5-2003
PFC SMA para la gestión de Taxis sobre dispositivo movil/PDA
*************************************************************/

package Agentes;

import java.awt.*;
import java.awt.event.*;
```

```
import java.util.Vector;

public class VentanaMapa extends Frame {

   private static VentanaMapa ventanamapa = new VentanaMapa();


   public Vector figuras = null;
   private MapaCliente map;


   public void addFigura (AccionARealizar a){
          if (figuras == null) figuras = new Vector();
          figuras.add(a);
   }

   private VentanaMapa(){

          this.setSize(240,300);
          this.setTitle("Mapa del Cliente");
          setLayout(new BorderLayout());

          addWindowListener(
                 new WindowAdapter() {
                 public void windowClosing(WindowEvent e) {
                   hide(); //ocultamos la ventana
                 }
          });
          this.setVisible(true);

          map=new MapaCliente();
          addDibujo(map);
   }

   public static VentanaMapa getInstance(){
          return ventanamapa;
   }


   public void addDibujo(MapaCliente d) {
          add(d,BorderLayout.CENTER); //colocamos el mapa a lo grande
          setVisible(true);
   }

   public void muestra_camino(JADE.util.LEAP.List calles) {
     map.camino_ontheway(calles);
   }

   public void reinicia_mapa() {
     map.reinicia_booleans();
   }
}
```

# 10. VALORACIÓN Y CONCLUSIONES

Los agentes son sin duda alguna uno de los campos principales donde la Inteligencia Artificial va a crecer mas en un futuro próximo, la idea de tener varios agentes colaborando para resolver una tarea común que individualmente sería demasiado costosa, nos hace pensar que acabaremos aplicando estos novedosos sistemas en muchos y variados ámbitos. No obstante no podemos olvidar que estamos hablando de una tecnología en su infancia, hay que ver como evoluciona y si definitivamente despega, en ámbitos mas cuotidianos, puesto que en la actualidad es poco mas que una herramienta de investigación. Agradecer desde aquí el apoyo y fuerza que la FIPA proporciona a los SMA, puesto que gracias a esta organización, los avances serán más rápidos y con mayor proyección a escala mundial.

Personalmente he de decir que este proyecto me entusiasmó desde el principio, por el reto que representaba introducir los SMAs en dispositivos móviles. El hecho de tener que realizar una investigación previa, fue sin duda uno de sus atractivos más importantes, sin olvidar la sensación que estábamos escapando de los entornos convencionales, agentes en un solo PC, para movernos por un sendero más realista, un cliente con su PDA.

Las mayores dificultades fueron encontradas en la parte de investigación previa, en cuyo periodo nos tuvimos que adaptar a varios cambios de software que se produjeron, en especial destacar el paso de JADE2.5 a JADE3.01b que resultó ser ciertamente critico, puesto que mucha de la información recogida y procedimientos adaptados, solo eran validos para la versión antigua, de todos modos, el cambio fue a mejor, puesto que la nueva versión de JADE es muy superior tanto a la versión 2.5 como a la 2.61.

Los conocimientos adquiridos con la realización de este proyecto se pueden dividir en dos ámbitos:

En el entorno de la implementación he obtenido una mayor fluidez en el uso de Java, el lenguaje utilizado en la implementación. Conocer y entender bs principios de los SMAs, dominio de la plataforma para la creación de SMAs JADE y dominio a su vez de su addon para dispositivos con recursos limitados LEAP.

En el ámbito de la investigación realizada he obtenido un mayor conocimiento del entorno wireless, especialmente en cuanto a conexiones Bluetooth. También he aprendido mucho respecto a los dispositivos móviles (teléfonos, agendas electrónicas…) y su software/hardware asociado.

El hecho de haber trabajado en un proyecto con tecnología muy actual y con un gran futuro por delante, me hace ser optimista sobre lo que personalmente puedo aportar a las empresas interesadas en este ámbito, por ello estoy convencido que me facilitará la tarea de buscar trabajo.

En cuanto a la incidencia de lo aprendido durante la carrera a la hora de realizar este proyecto, personalmente creo que todas las asignaturas cursadas me han resultado útiles, puesto que para realizar una investigación con éxito es indispensable poseer una densa base de conocimientos. No obstante siempre podemos destacar algunas que por su contenido han resultado de mayor utilidad: *Redes de Computadores*, *Periféricos* e *Introducción a la Inteligencia Artificial.* Todas ellas cursadas dentro de los estudios de Ingeniería Técnica en Informática de Sistemas.

# 11. FUTUROS PROYECTOS

Desde el punto de vista de un SMA funcionando sobre un móvil/PDA, el paso siguiente ha realizar sería olvidarnos de la conexión Bluetooth y utilizar GPRS, para darle una verdadera funcionalidad y proyección a todos los SMA que creemos, no solo éste. Naturalmente centrarnos en el desarrollo sobre móvil quizás sería más interesante, puesto que la proliferación de estos dispositivos supera en mucho a las PDAs mucho más caras, y de una utilidad puesta en entredicho por muchos.

Desde el punto de vista del gestor de taxis creado como ejemplo, los posibles ampliaciones son varias, siempre teniendo en cuenta que con el uso de PDAs o móviles, las posibilidades aumentan. Varias ideas ya se han comentado a lo largo de este informe. Crear un editor de mapas sería el paso lógico para facilitar la complicada tarea de crear nuevos mapas, aumentar los detalles del servicio ofrecido al cliente sería también otra posibilidad, mejorar el algoritmo que calcula los trayectos, o el algoritmo para seleccionar el taxi idóneo, no son mas que mejoras obvias.

Sin embargo lo que tenemos en mente es continuar avanzando por el campo del mapa en la PDA, en la actualidad el mapa por donde nos movemos cabe en su totalidad en la pantalla del dispositivo, esto, aunque nos abre puertas, también nos impone unas fuertes limitaciones a la hora de crear los mapas, alejándonos dramáticamente de la realidad. Nosotros proponemos que la PDA solo muestre una parte de ese mapa, y que con unos sencillos cursores nos podamos desplazar por el mapa, buscando cualquier zona que nos interese. Sin duda, esto le daría gran fuerza al proyecto, además de colocarlo en muy buena posición para tener una utilidad real. Naturalmente no podemos olvidar que habría que mejorar el tipo de edificios representables, ya que aunque ahora esta bastante desarrollado, se siguen encontrando a faltar curvas, lo cual nos permitiría tener rotondas, ahora simplemente podemos intuir que están en un nodo, como si de una plaza se tratase, pero visualmente mejoraría mucho poder ver esa rotonda dibujada.

Otra opción interesante, sería cargar los mapas como imágenes GIF o JPEG, lo cual agilizaría el proceso de dibujado de mapas y le daría un mejor entorno visual. Estas imágenes podrían ser descargadas directamente de Internet y el mismo agente personal se podría ocupar de ello.

Un último apunte de interés, sería la posibilidad de integrar el gestor de taxis en un entorno de SMAs enfocado al transporte público en una ciudad. Las características de los SMAs jugarían a nuestro favor en unas condiciones de este tipo.

# 12. AGRADECIMIENTOS

Para finalizar, me gustaría agradecer el interés mostrado por mis tutores Antonio Moreno Ribas y Aïda Valls Mateu, aportando documentación y material indispensable para la correcta evolución del proyecto. Sin salir del GruSMA quiero destacar a David Isern y David Sánchez, que de una forma u otra han ayudado a la finalización de esta tarea. Agradecimientos especiales, para Ernest Folch Espallargues por su apoyo en la fase de investigación y sus comentarios al respecto y un ultimo agradecimiento a Juan Infante García, programador del gestor de taxis que hemos seleccionado y adaptado para las pruebas, que gracias a su clara documentación, ha facilitado las modificaciones realizadas.

Por último, agradecer también a todos aquellos que con sus comentarios, programas, documentos, pruebas, etc. Hicieron posible la realización de este proyecto, entre estos quiero destacar al profesor *David Grimshaw* de la *Ryerson Polytechnic University* [Grimshaw02] por sus tutoriales que fueron de gran ayuda en los primeros pasos del estudio.

# 13. BIBLIOGRAFÍA

[ActiveSync03] Aplicación de Microsoft para la conectividad entre PC y PDA.
http://www.microsoft.com/mobile/pocketpc/downloads/

[AgentCities03] AgentCities, proyecto europeo para la implantación de SMAs en ciudades
http://www.agentcities.org/

[Banzai03] Banzai, Grupo de investigación en Inteligencia Artificial, DEIM, Universitat Rovira i Virgili
http://www.etse.urv.es/recerca/banzai

[Bluetooth03] tecnología wireless para la conexión entre plataformas
http://www.monografias.com/trabajos12/tecninal/tecninal.shtml
http://www.Bluetooth.com

[FIPA03] Foundation for Intelligent Physical Agents, http://www.fipa.org/

[Grimshaw02] documentación sobre entorno JADE-LEAP
http://www.ryerson.ca/~dgrimsha/JADE/index.html

[GruSMA03] GruSMA, Grup de Sistemes Multi-Agent, DEIM, Universitat Rovira i Virgili, plana web,
http://www.etse.urv.es/recerca/rgai/toni/MAS/

[guide03] documentos de apoyo al programador de entornos JADE.
http://JADE.cselt.it/doc/programmersguide.pdf

[Infante02] Sistema multi-agente para la gestion de taxis de una ciudad

[Isern et al., 2001] Isern, D., Moreno, A., Valls, A., Desenvolupament de Sistemes MultiAgent en JADE, Report de Recerca, DEIM-RR-01-003, Universitat Rovira i Virgili, 2001

[JADE03] plataforma para SMAs según la FIPA http://sharon.cselt.it

[JadeApi03] Api con las clases del entorno JADE http://JADE.cselt.it/doc/api/index.html

[JADE.2xto3_03 ] documentación de apoyo para la promoción de código entre versiones de JADE
http://sharon.cselt.it/projects/JADE/doc/tutorials/toJADE30.html

[Java03] lenguaje programación orientado a objetos de Sun http://java.sun.com

[JeodeVM03] Maquina virtual de Insignia para PocketPC2002 http://www.insignia.com/

[J2ME03] java enfocado a la programación para móviles http://java.sun.com/j2me/

[KOBJECTS03] organización dedicada a la programación en j2me http://kobjects.org/

[LEAP02] iniciadores del proyecto LEAP http://LEAP.crm-paris.com/

[Mocha03] empresa creadora de aplicaciones para la conectividad entre estaciones
http://www.mochasoft.dk/

[PALM03] fabricante de PDAs http://www.palmsource.com/

[specs03] especificaciones de la FIPA http://www.fipa.org/specs/fipa00061/

[vigo02] departamento dedicado a SMAs de la universidad de Vigo
http://montealegre.ei.uvigo.es/agentes/

[WCE3.0+PPP03] documentación sobre las posibilidades de uso del protocolo PPP sobre el SO windows CE3.0 (pocketPC2002)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcecomm/htm/_wcesdk_Point_to_Point_Protocol.asp

[Wooldridge02] Wooldridge, M., *An introduction to multiagent systems*, John Wiley Ed., 2002. ISBN 0-471-49691-X.

[Zimm02] Marc Zimmermann creador aplic. pockethosts http://www.zimac.de/cestuff.htm/