

PROYECTO FINAL DE CARRERA



UNIVERSITAT
ROVIRA I VIRGILI



Sistema multi-agente
para la gestion de taxis
de una ciudad

Juan Infante García
jinf@menta.net

Directores de proyecto:
Aïda Valls Mateu

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

A mi familia

ÍNDICE

1. INTRODUCCIÓN	5
2. AGENTES Y SISTEMAS MULTI-AGENTES	7
2.1. Propiedades de los agentes	7
2.2. Arquitectura de los agentes	7
2.3. Tipos de agentes	8
2.4. Sistemas Multi-Agente	8
2.4.1. Ventajas de un Sistema Multi-Agente	8
2.4.2. Gestión de un Sistema Multi-Agente	9
2.4.3. Lenguajes de comunicación entre agentes ACL	10
2.4.3.1. Elemento de un mensaje	10
2.4.3.2. Protocolos de comunicación	12
3. JADE	13
3.1. Paquetes de JADE	13
3.2. Componentes principales de JADE	13
3.3. Agentes y Comportamientos (Behaviours)	14
3.3.1. Comportamientos Simples	15
3.3.2. Comportamientos Compuestos	15
3.4. Mensajes	16
3.5. Ontología	16
3.6. Protocolos de comunicación	18
3.7. Tipos de protocolos	19
3.7.1. FIPA REQUEST	19
3.7.2. FIPA CONTRACT-NET	20
3.8. Herramientas de JADE	23
4. DESCRIPCIÓN DEL SMA DE GESTIÓN DE TAXIS	28
4.1. Descripción general del problema	28
4.2. Justificación en la utilización de un SMA	28
4.3. Arquitectura del sistema	29
4.4. Agente Cliente	29
4.5. Agente Centralita	30
4.6. Agente Taxi	30
4.7. Agente Visualizador	31
4.8. Agente Tráfico	33
4.9. La simulación	33
5. COMUNICACIÓN ENTRE LOS AGENTES	36
5.1. Intercambio de mensajes entre los agentes	36
5.2. Ontología	39
5.2.1. Los conceptos de la ontología	39
5.2.2. Las acciones de la ontología	41
5.2.3. Relación: Ontología - Servicios Agentes – DF Agent	42

6. IMPLEMENTACIÓN	44
6.1. Clases y directorios	44
6.1.1. AgenteCliente.java	44
6.1.2. AgenteCentralita.java	44
6.1.3. AgenteTaxi.java	45
6.1.4. AgenteTráfico.java	46
6.1.5. AgenteVisualizador.java	46
7. PRUEBAS.....	49
7.1. Seguimiento de un ejemplo	49
8. CÓDIGO FUENTE	62
8.1. TaxiOntology.java	62
8.2. TaxiType.java	65
8.3. RespuestaPetición.java	68
8.4. RecorridoTaxi.java	69
8.5. Posición.java	70
8.6. Petición.java	70
8.7. ListaCalles.java	71
8.8. EstimaciónRecorridoLlegada.java	72
8.9. DibujarTaxi.java	72
8.10. DibujarRecorridoTaxi.java	73
8.11. DibujarPersona.java	74
8.12. Desocupado.java	74
8.13. DameCallesAnuladas.java	75
8.14. BuscarTaxi.java	75
8.15. BorrarTaxi.java	75
8.16. BorrarRecorridoTaxi.java	76
8.17. BorrarPersona.java	76
8.18. AgenteCentralita.java	77
8.19. AgenteCliente.java	87
8.20. AgenteTaxi.java	92
8.21. AgenteTráfico.java	111
8.22. AgenteVisualizador.java	117
8.23. AcciónARealizar.java	126
8.24. Coordenadas.java	126
8.25. Coordenadas2.java	127
8.26. DatosMapa.java	128
8.27. InfoMap.java	131
8.28. Mapa.java	131
8.29. Ventana.java	136
9. VALORACIÓN Y CONCLUSIONES	137
10. FUTUROS PROYECTOS	138
11. BIBLIOGRAFÍA	139

1. INTRODUCCIÓN

Uno de los objetivos de las nuevas tecnologías es ahorrar tiempo en la realización de ciertas tareas y sobretodo conseguir que las personas tengan una vida más cómoda. Una de estas nuevas tecnologías son los SMA (Sistemas Multi-Agente), que engloban un conjunto de agentes con objetivos comunes. Estos agentes se podrían definir como un proceso computacional capaz de realizar tareas de forma autónoma y que se comunica con otros agentes para resolver problemas mediante cooperación, coordinación y negociación. Para obtener buenos resultados utilizando dicha tecnología, debe existir un estándar que lo controle para asegurarnos su buen funcionamiento. Para definir estándares (tanto de arquitectura, lenguajes, protocolos...) se creó la FIPA, a partir de la cual se han desarrollado todo un conjunto de proyectos, como por ejemplo una iniciativa europea denominada *AgentCities*, con la idea de ofrecer mediante la utilización de Sistemas Multi-Agente los servicios que una ciudad realiza, como por ejemplo, ocio, hostelería, restauración, sanitarios, transporte... (Para más información sobre este proyecto europeo se puede consultar: [AgentCities]).

El grupo de Sistemas Multi-Agente de la Universitat Rovira i Virgili (GRUSMA) ha implementado un SMA con el objetivo de ampliar la oferta de servicios en *AgentCities* con la *simulación de la gestión de Taxis de una ciudad* cualquiera, aunque en dicho sistema se ha tomado como modelo un sistema de calles como el del Eixample de Barcelona, donde las calles forman una cuadrícula, ya que, facilitaba su implementación.

GRUSMA proporciona un entorno de trabajo en el cual los alumnos de Informática de la ETSE, interesados en las nuevas tecnologías de la Inteligencia Artificial, pueden compartir conocimientos y experiencias en el campo de los Sistemas Multi-Agente [GruSMA]. Este marco permite a los alumnos realizar un proyecto Final de Carrera innovador con la ayuda de otros compañeros interesados en SMA. Desde hace tiempo, el grupo de Sistemas Multi-Agente, trabaja en el desarrollo de Sistemas con dicha tecnología implementadas en JADE (v2.6) y por ello este documento, además de hacer una introducción a cómo se desarrollan este tipo de sistemas, intenta hacer una introducción a cómo se implementan utilizando JADE.

El sistema que hemos desarrollado 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 abordó. El objetivo de nuestro sistema no es otro que la utilización de las ventajas que nos proporciona los Sistemas Multi-Agente para optimizar el proceso anteriormente descrito.

Sistemas con características similares ya han sido creados por otras empresas, las cuales proporcionan una reducción considerable en el tiempo de espera del cliente, la posibilidad de pagar de forma remota a través de la tarjeta de crédito, realización de reservas, control del mantenimiento del vehículo y una serie de propiedades que nos aseguran la utilidad del sistema creado [Mobile Computer System, Inc], [SIGTEC]. En otras direcciones de interés encontramos páginas con formularios los cuales realizan reservas de taxis, lo cual, se aleja de nuestra finalidad pero que no se descarta como ampliación para futuros proyectos [FST]. Como más adelante se describe, la utilización de un sistema como el que hemos desarrollado necesita de una serie de hardware que a continuación se detalla:



Foto 1- Ordenador de abordó tipo 1



Foto 2- Ordenador de abordó tipo 2



Foto 3- Ordenador de abordó tipo 3

A continuación se muestran tres tipos diferenciados de ordenadores de abordó que podrían utilizar nuestro sistema para la realización de propuestas junto con una serie de características que dependerían del modelo a utilizar. El de la fotografía número 1 sería el más completo ya que podría gestionar sus turnos, controlar la posición del taxi (GPS), controlar el funcionamiento del taxi, realización de recorridos alternativos los cuales se visualizarían en la pantalla y una infinidad de posibilidades. En este caso también incluye un lector de tarjetas de crédito para facilitar el pago a los clientes.

Este documento 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 Mutli-Agente utilizada (JADE). A partir de aquí se hablará de todo el trabajo realizado en el proyecto final de carrera. En el capítulo 4 se explica con todo detalle la descripción del sistema. En el capítulo 5 se explica la comunicación que existe entre los diferentes agentes. En el capítulo 6 está dedicado a la explicación de la funciones y las principales características de cada uno de los módulos que forman la implementación del sistema. El capítulo 7 está destinado a la prueba final del sistema. En el capítulo 8 encontramos todo el código fuente del sistema. En el capítulo 9 hacemos una valoración del trabajo realizado, conclusiones y futuros proyectos, y Finalmente, el capítulo 10 lo destinamos a la Bibliografía.

2. AGENTES Y SISTEMAS MULTI-AGENTE

Antes de entrar a definir que son los Sistemas Multiagente, hemos de centrarnos en el estudio de su núcleo: el AGENTE.

Un agente inteligente es un proceso computacional capaz de realizar tareas de forma autónoma y que se comunica con otros agentes para resolver problemas mediante cooperación, coordinación y negociación. Todo agente que habita en un entorno complejo y dinámico con el cuál interacciona en tiempo real para conseguir un conjunto de objetivos. [Wooldridge,2002]

2.1. Propiedades de los Agentes

Las propiedades más importantes que pueden tener los agentes 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.
- **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.

No todos los agentes cumplen todas estas propiedades. Los agentes de nuestro sistema 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 de nuestro sistema siguen el model 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.

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

2.4. Sistemas MultiAgente

Un sistema MultiAgente es aquel en el que un conjunto de agentes cooperan, coordinan y se comunican para conseguir un objetivo común [MAS].

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 model lógico para la creación, registro, comunicación, movilidad y destrucción de agentes. En este proyecto vamos a seguir los estandares 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 [FIPA]:

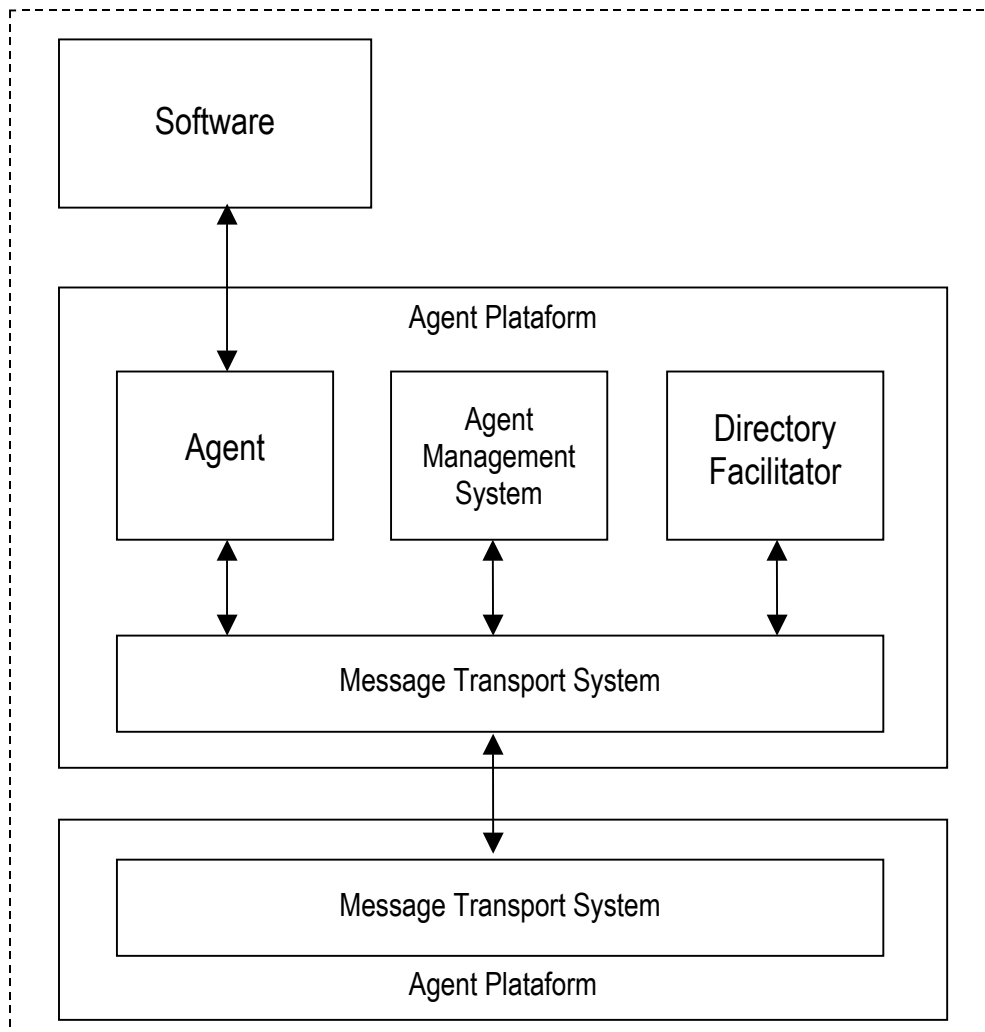


Fig. 1 – Interior de un Sistema MultiAgente

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 utilizando un lenguaje común: ACL (*Agent Communication Language*).

El elemento básico de la comunicación es el mensaje. La FIPA determina que forma ha de tener un mensaje y su utilización [FIPA, 2000a].

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

- Definición del tipo de mensaje: indica que tipo de comunicación deseamos hacer.
 - **accept-proposal:** aceptamos una propuesta hecha anteriormente para realizar una acción
 - **agree:** aceptación de una acción
 - **cancel:** cancelación de una acción
 - **cfp:** para proponer una acción
 - **confirm:** el emisor informa al receptor que una proposición es cierta, cuando el receptor dudaba
 - **disconfirm:** el emisor informa al receptor que una proposición es falsa, cuando el receptor pensaba que era cierta.
 - **Failure:** indica que la acción pedida anteriormente ha fallado por algún motivo.
 - **Inform:** el emisor informa al receptor que una acción se ha realizado correctamente.
 - **Not-understood:** el emisor informa al receptor que no ha entendido una petición realizada anteriormente.
 - **Propose:** acción de proponer la realización de una acción, dadas ciertas precondiciones.
 - **Query-if:** acción de preguntar a otro agente si un hecho es cierto o no.
 - **Refuse:** negación a la realización de una acción dada.
 - **Request:** el emisor pide al receptor la realización de una acción.
 - **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.
 - **FIPA-CCL (Constrant Choice Language):** define una semántica que permite especificar predicados con restricciones.
 - **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 he utilizado en este proyecto.
 - **FIPA-KIF (Knowledge Interchange Format):** permite expresar objetos como términos y proposiciones como sentencias.
 - **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:
 - **Language**: en qué está escrito el contenido, en este proyecto se ha utilizado FIPA-SL.
 - **Encoding**: indica si se utiliza algún tipo de codificación especial.
 - **Ontology**: es el campo más importante, ya que, define la ontología utilizada para dar significado al contenido del mensaje.
- Control de la conversación: está destinado a identificar qué tipo de conversaciones se mantienen. Sus campos son:
 - **Protocolo** utilizado en el mensaje.
 - **Conversation-id**: identificador de la conversación.
 - **Reply-with**: identificador del mensaje que se utiliza para seguir los diferentes pasos de la conversación.
 - **In-reply-to**: identifica una acción pasada a la cual este mensaje es la respuesta.
 - **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



Logotipo de JADE
<http://sharon.cselt.it/projects/jade/>

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

La versión utilizada para la realización de este proyecto ha sido la 2.6 [Sánchez, 2001].

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 interfaz gráfica de JADE para la administración y control del sistema.
 - *Dummy Agent*: es una herramienta de monitorización y depuración, compuesta por una interfaz 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.

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.

Tal como se define en la FIPA [FIPA, 1998b], 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.
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.

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 que 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 **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 inician y finalizan el agente respectivamente.

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.

La figura 2 muestra la estructura de un agente:

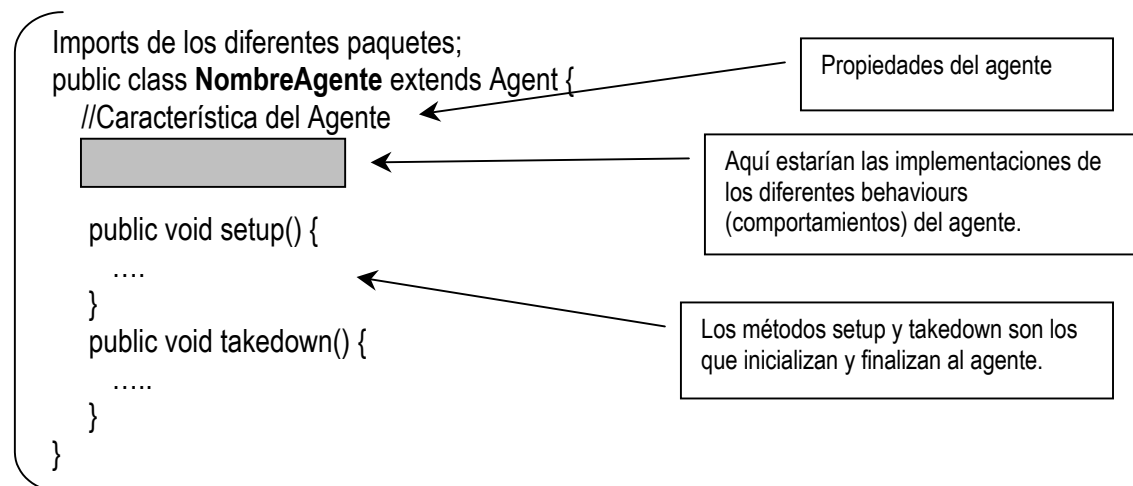


Fig. 2 – Estructura básica de un agente. Ahora vamos a profundizar un poco más en el tema de los Behaviours. JADE nos

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.behaviours.ComplexBehaviour*.

Los métodos que se pueden sobrecargar son **preAction()** i **postAction()**. El primero en ejecutarse será el **preAction()**, 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 **postAction()**. Hay dos tipos de comportamientos complejos:

- SequencialBehaviour: ejecuta los hijos de manera secuencial.
- NonDeterministicBehaviour: ejecuta los hijos según una política de Round Robin.

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 nueva versión del Jade (2.6), la implementación de la ontología cambia bastante 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_sequence, 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

La nueva versión de JADE (2.6) nos proporciona los elementos necesarios para definir los protocolos descritos por la FIPA [FIPA, 1998b]. Con la nueva versión se pueden implementar más protocolos: (FIPA-Request, FIPA-Propose, FIPA-Query, FIPA-Request-When, FIPA-Recruiting, FIPA-Brokering,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 la versión antigua 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 todos 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.

3.7. Tipos de protocolos

En Jade 2.6 se pueden implementar una gran variedad de protocolos, aunque, en este proyecto sólo hemos utilizado FIPA-Request y FIPA-Contract-Net.

A continuación explicaremos con más detalle cada uno de dichos protocolos.

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**.
- 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.

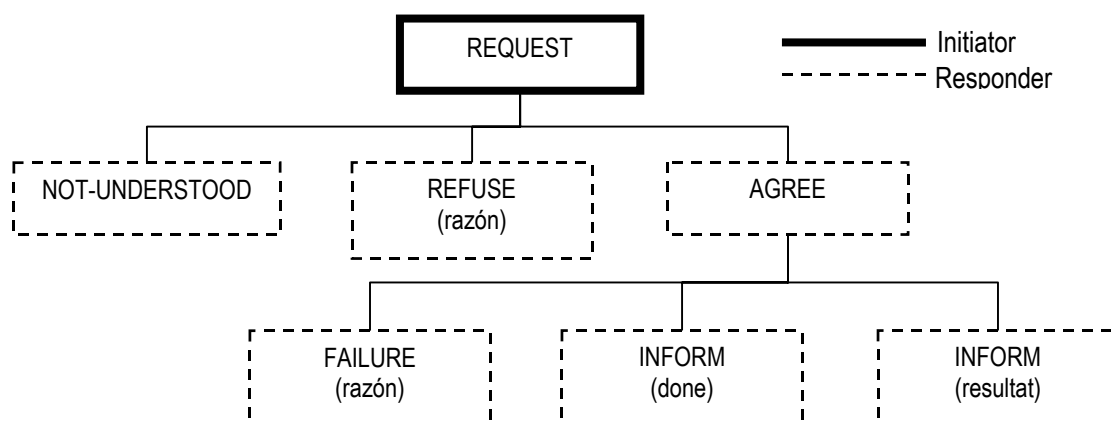


Fig. 3 – Protocolo FIPA Request

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 seguir 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.

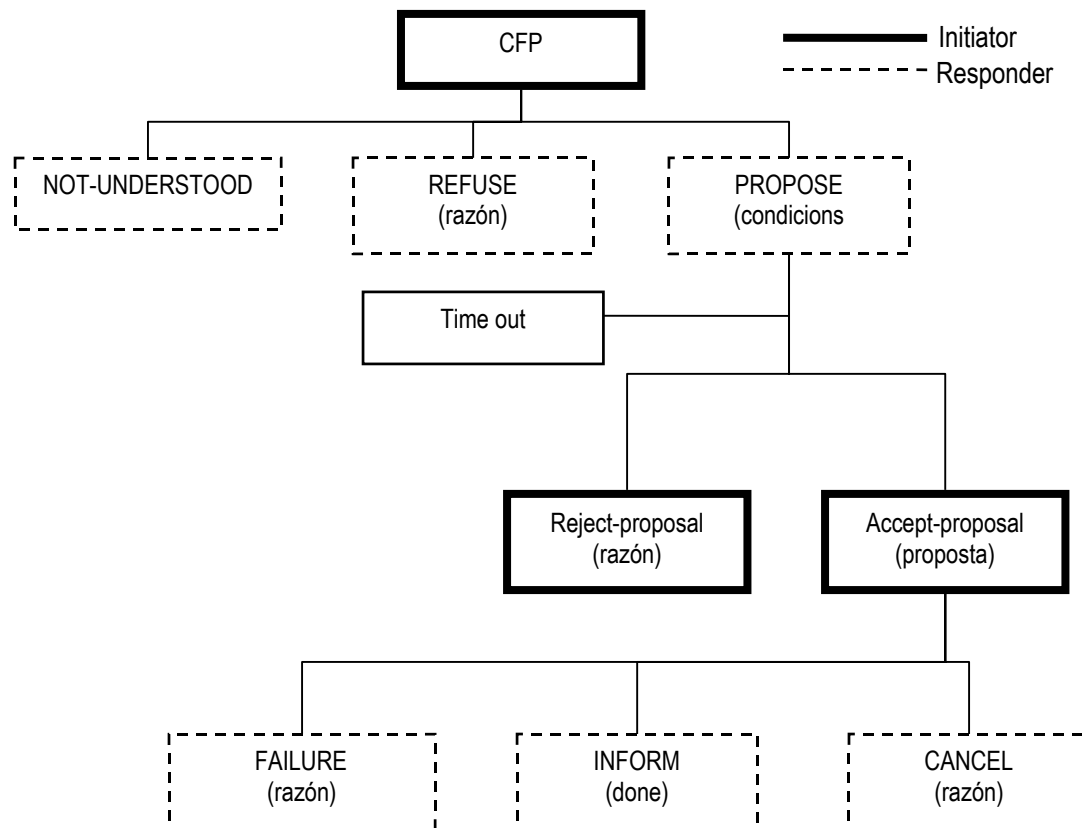


Fig. 4 – Protocolo FIPA Contract-Net

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 CNIInitiator extends ContractNetInitiator
{
    public CNIInitiator (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) { ... }
}

```

NOTA: A continuación les haremos una breve explicación de los tres métodos en cursiva del cuadro anterior:

- **prepareCfps:** este método se utiliza para enviar los 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 te lo hace 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 linea de comandos:

```
java jade.Boot [options] -platform [lista 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 por un espacio 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 linea de texto: <NombreAgente>:PackageClase.ClaseAgenteJava.

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.

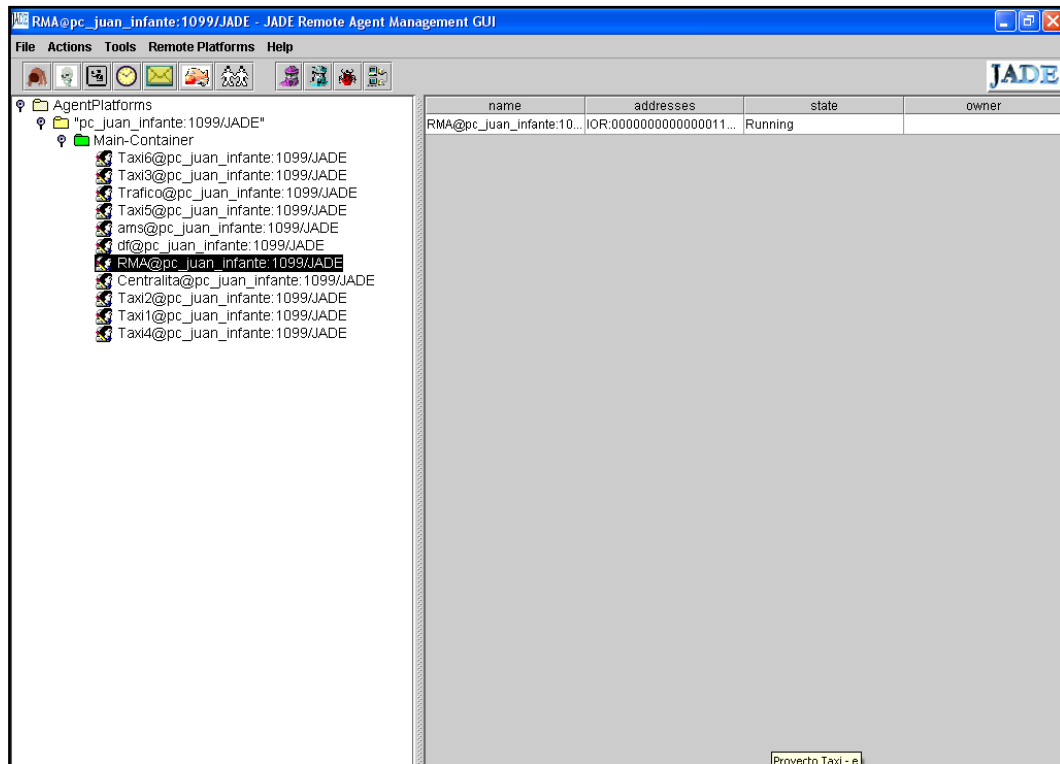


Fig. 5 – Pantalla principal de JADE

Cuando iniciamos el entorno gráfico, tal como se muestra en la figura 5, aunque no inicialicemos ningún agente, se crean algunos automáticamente. Ahora explicaremos la función de dichos agentes:

- 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.

The image shows a Java Swing window titled "ACL Message". It has a standard title bar with a close button. Inside, there are two tabs: "ACLMessage" and "Envelope". The "ACLMessage" tab is active and contains several labeled fields:

- Sender:** A text field containing "@pc_juan_infante:1099/JADE" with a "View" button next to it.
- Receivers:** A text field containing "df@pc_juan_infante:1099/JADE".
- Reply-to:** An empty text field.
- Communicative act:** A dropdown menu showing "request".
- Content:** A large text area containing the string "sequence) :language (sequence) :ontology (sequence)))".
- Language:** A text field containing "FIPA-SLO".
- Encoding:** An empty text field.
- Ontology:** A text field containing "FIPA-Agent-Management".
- Protocol:** A dropdown menu showing "fipa-request".
- Conversation-id:** A text field containing "n_infante:1099/JADE1037323056348".
- In-reply-to:** An empty text field.
- Reply-with:** A text field containing "n_infante:1099/JADE1037323056348".
- Reply-by:** An empty text field with a "View" button next to it.
- User Properties:** An empty text field.

At the bottom center of the dialog is an "OK" button.

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.

Otra posibilidad es guardar los mensajes y posteriormente poderlos abrir.

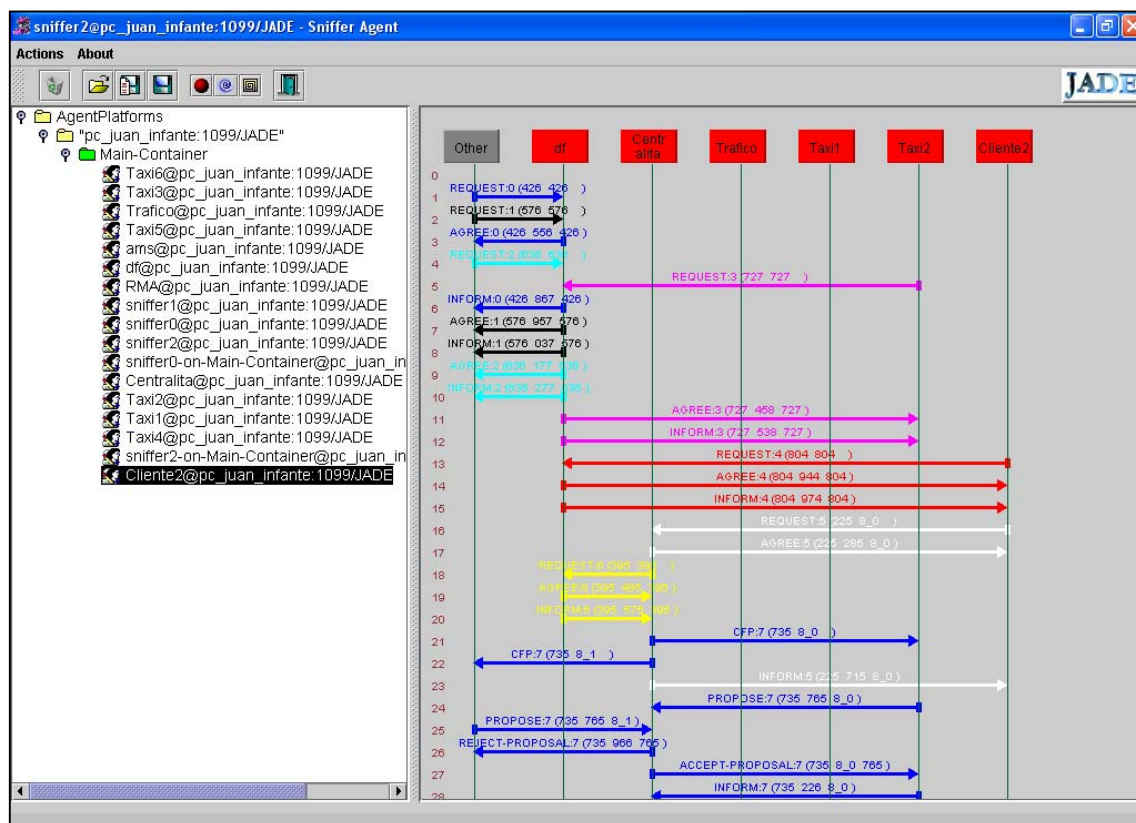


Fig. 7 – Ventana del sniffer

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

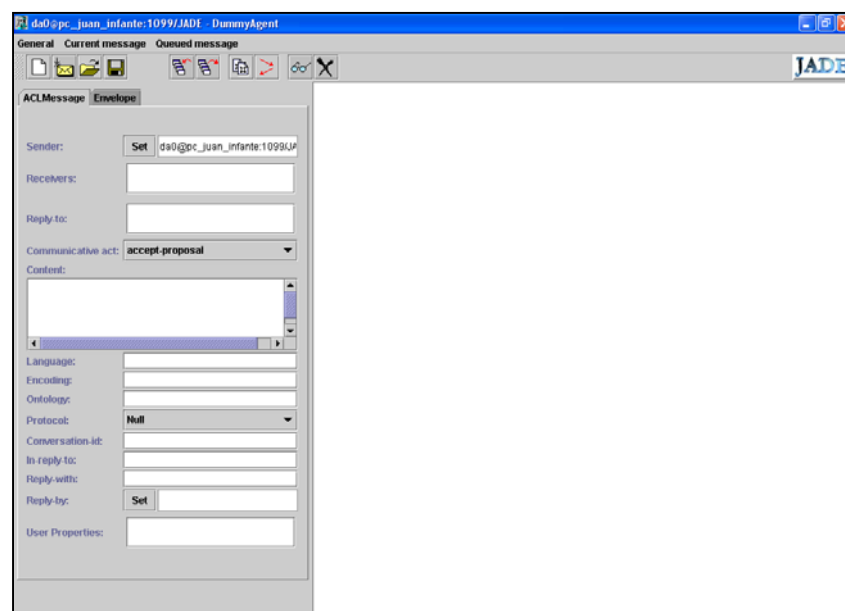


Fig. 8 – Ventana de dummy-agent

Otra herramienta muy útil para la realización de este tipo de sistemas, es la activación de la interfie 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.

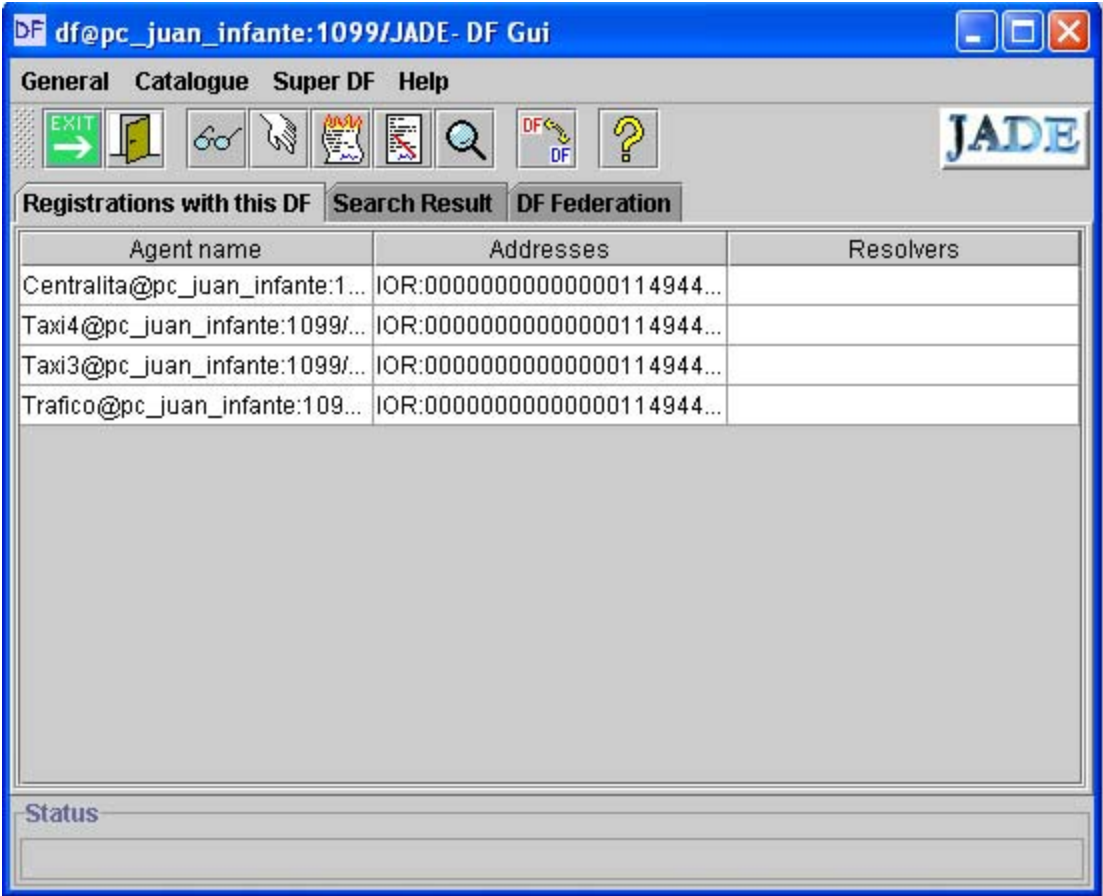


Fig. 9 – Int.gráfica del agente DF

4. DESCRIPCIÓN DEL SMA DE GESTIÓN DE TAXIS

4.1. Descripción general del problema

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 un 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 substituir el sistema actual de radio por un sistema informático que operaría con ordenadores de abordo.

A continuación les haremos una breve descripción del funcionamiento del sistema. El cliente realizará una petición mediante su agente personal que puede estar ejecutándose en un móvil, un PDA o un portátil. Cuando la centralita reciba una petición se pondrá en contacto con todos los taxis que esten 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 unos criterios.

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.

4.2. Justificación en 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 si 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.
- **COOPERACION:** 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.

4.3. Arquitectura del sistema

A continuación se muestra en esta figura la arquitectura completa del sistema. Representando cada agente y sus diferentes mensajes entre ellos.

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

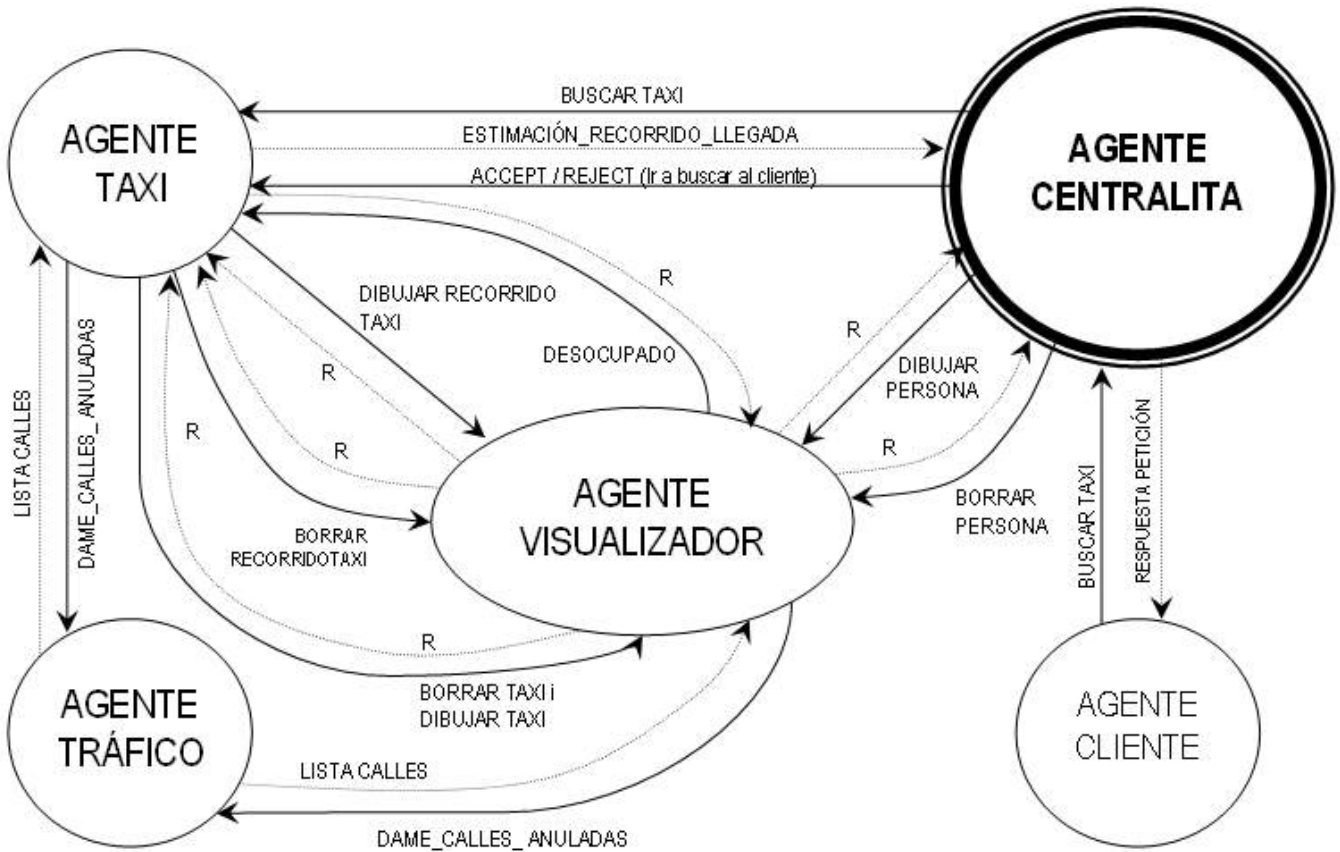


Fig. 10 – Arquitectura del sistema

4.4. 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 (GPS) del cliente y el destino (GPS). Para simplificar el problema suponemos que las peticiones se realizaran siempre desde una esquina entre 2 calles.

4.5. 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.

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 se pueden modificar para perfeccionarlos en futuras actualizaciones. El criterio que hemos utilizado para implementar la centralita, es el de escoger al taxi más cercano al cliente.

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.

4.6. 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 (GPS) 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.

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, puesto que és una buena estimación para ciudades con una distribución de calles en forma de cuadrícula, que es el modelo que nosotros hemos tomado (p.e. el Eixample de Barcelona). Una vez ha calculado el recorrido, el agente taxi se lo envía a la Centralita, que posteriormente se encarga de seleccionar el que mejor le convenga, y así conseguir un resultado satisfactorio.

4.7. Agente Visualizador

El agente VISUALIZADOR es el encargado de representar la interficie gráfica. Su función principal es la de ofrecer los resultados en pantalla para poder hacer un seguimiento de la posición de los taxis, la posición de los clientes y los recorridos que realizan en la ciudad.

Para el desarrollo del sistema hemos tomado como modelo un sistema de calles como el del Eixample de Barcelona, donde las calles forman una cuadrícula, ya que, facilitaba su implementación.

Si consideramos que los únicos puntos donde se recoge a los clientes es la intersección entre dos calles, podemos entender el mapa de la ciudad en cuestión como un *grafo* donde los nodos son las intersecciones y los arcos son las calles. Dicho grafo sería dirigido puesto que las calles pueden tener un solo sentido. A continuación se les muestra el grafo utilizado para la simulación:

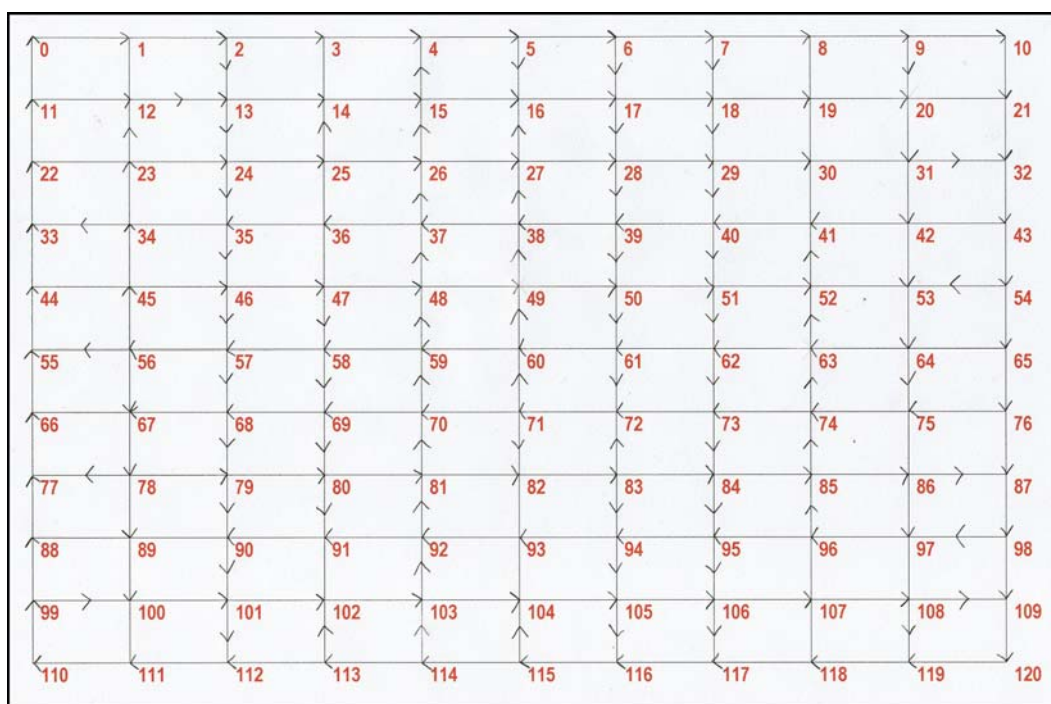


Fig. 11 – Grafo dirigido de la ciudad utilizada en la simulación

Para representar los taxis y los clientes en las diferentes posiciones, hemos creado dos matrices con las coordenadas de los píxeles dentro de la ventana. Así, cuando deseamos representar un taxi o cliente en la posición 7,6 de la ciudad, obtenemos el valor del píxel de la columna 7 y el valor del píxel de la fila 6, y así poder representarlo en la posición correcta del mapa. Los valores de ambas matrices son integers con los siguientes rangos, es decir, el ancho en píxeles de la ventana es 1023 y el alto en píxeles de la ventana es 710:

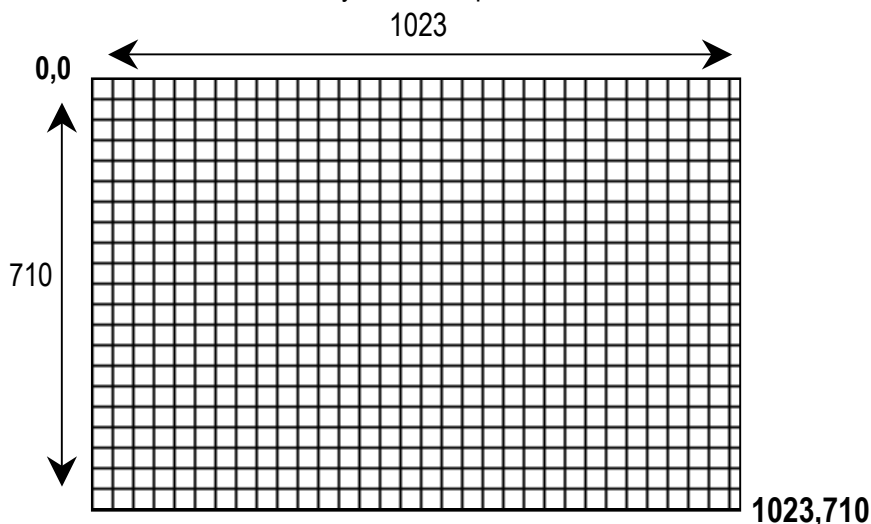


Fig. 12 – Rango de píxeles de la ventana

Para la representación de los recorridos, también utilizamos ambas matrices, ya que, la técnica es muy similar. En el caso de los recorridos hemos de tener en cuenta la *siguiente coordenada* para saber si hemos de dibujar esa parte del recorrido en horizontal o en vertical.

En el caso de la representación de las calles cortadas, utilizamos el mismo método encargado para pintar los recorridos, ya que, conceptualmente, pintar un recorrido debería hacerse igual que pintar una serie de calles que nos interesen.

A continuación les mostramos un pequeño ejemplo de la interficie en la *fig. 14*.

Como se puede observar, hemos utilizado diferentes colores para representar los resultados obtenidos. En la siguiente tabla les haremos una breve explicación de dicha leyenda.

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. 13 – Leyenda de la interficie

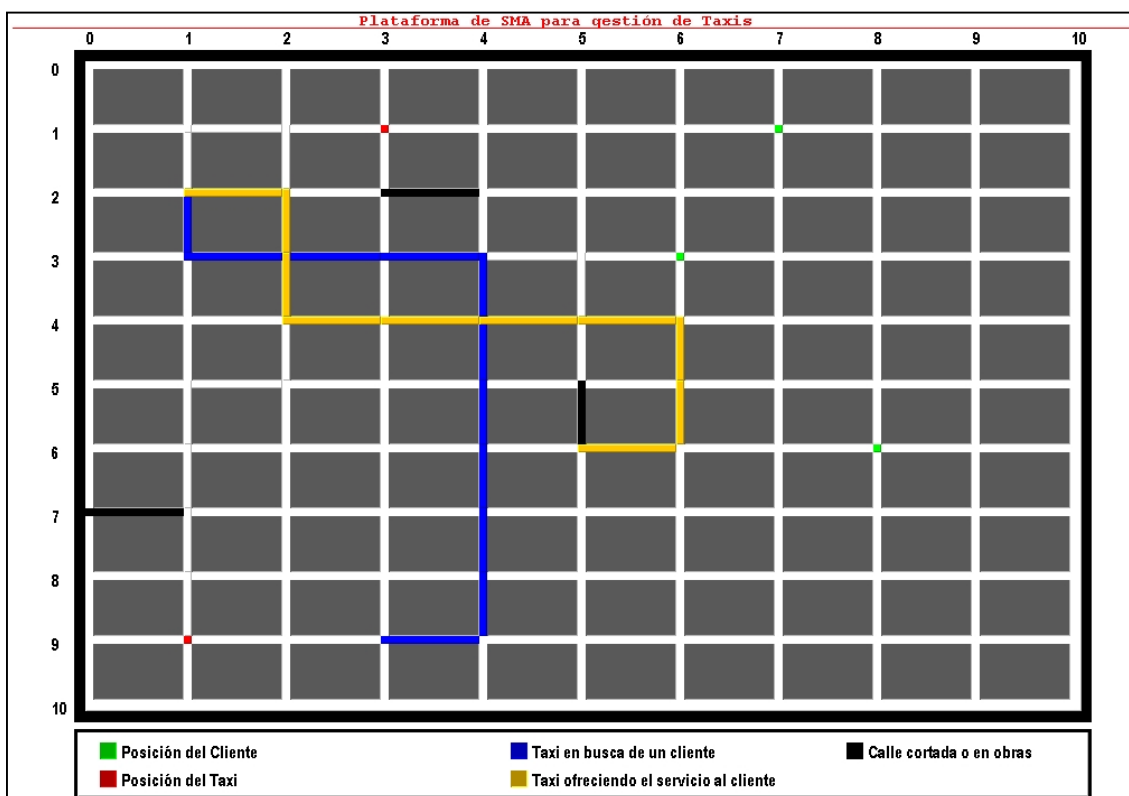


Fig. 14 – Ejemplo de la interficie gráfica

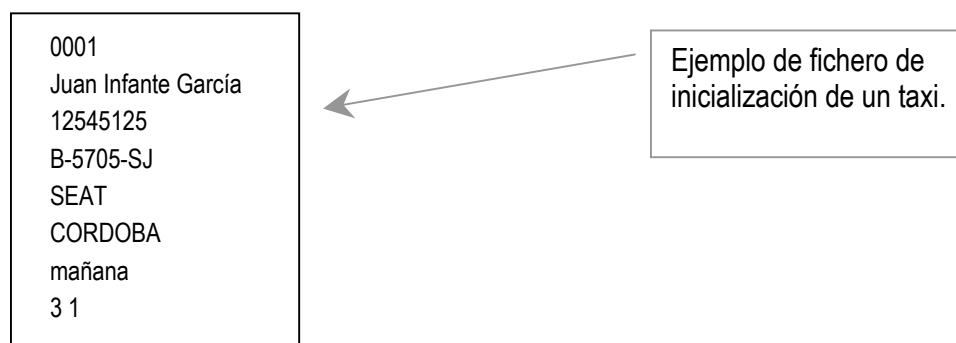
4.8. Agente Tráfico

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.

4.9. 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, todos los taxis se inicializan al ejecutar el sistema. Dependiendo de la hora en que se ejecute (lo averiguamos a través del reloj del sistema), sabemos qué taxis de todos los cargados en el sistema 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 60 segundos (ya que es una simulación, en su ejecución real debemos poner 8 horas) 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 (véase pàg.10) y así conseguimos que los únicos taxis dados de alta en el DIRECTORY FACILITATOR sean los que en ese momento realizan su turno. Así conseguimos que a la centralita le sea 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 es idéntica a la de los taxis, es decir, también se realiza utilizando un fichero de inicialización. El fichero también es de formato texto y sólo utiliza la primera línea, ya que, es suficiente para representar toda la información necesaria para un cliente. Los datos facilitados en el fichero son los siguientes: número de personas (la primera cifra), posición donde está el cliente (las dos cifras siguientes) y posición donde quiere ir el cliente (las dos últimas cifras).

Ejemplo: 3 1 2 5 6

Observamos que hay tres personas que quieren ir de la coordenada 1 2 hasta la coordenada 5 6 del mapa.

La centralita gestiona los clientes conforme van entrando en el sistema. 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 más cerca esté del cliente. Este proceso lo repite mientras haya algún taxi libre. 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. Con esto conseguimos la seguridad de servir a todos los clientes que utilicen el sistema.

Para la representación del mapa de la ciudad hemos utilizado una estructura de datos *grafo* (véase ap. 4.7). Dicha estructura es cargada a través de un fichero de inicialización donde

obtenemos los diferentes nodos (intersecciones), calles, direcciones y el coste asignado a cada calle. El (*) indica que la información de dicho nodo ha finalizado. A continuación podréis observar 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 hacia 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 vertice 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.

Como hemos explicado anteriormente, esto es una simulación de un caso real. En el momento en el que el sistema se ejecute en un entorno real hemos de aclarar que los agentes *clientes* realizarían sus peticiones a través del móvil, PDA u ordenadores y los agentes *taxistas* estarían ejecutándose en un ordenador de abordo, es decir, todo sería distribuido. Los cambios en el código son mínimos porque Jade está pensado para ejecutar agentes de forma distribuida.

La simulación y todo el proceso de pruebas ha sido ejecutado en un ordenador con procesador Intel Pentium III 600Mghz, 128 Mb de RAM y con el sistema operativo Windows XP.

5. COMUNICACIÓN ENTRE LOS AGENTES

5.1. 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 Visualizador ha dibujado el servicio (Taxi termina 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, por eso 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.

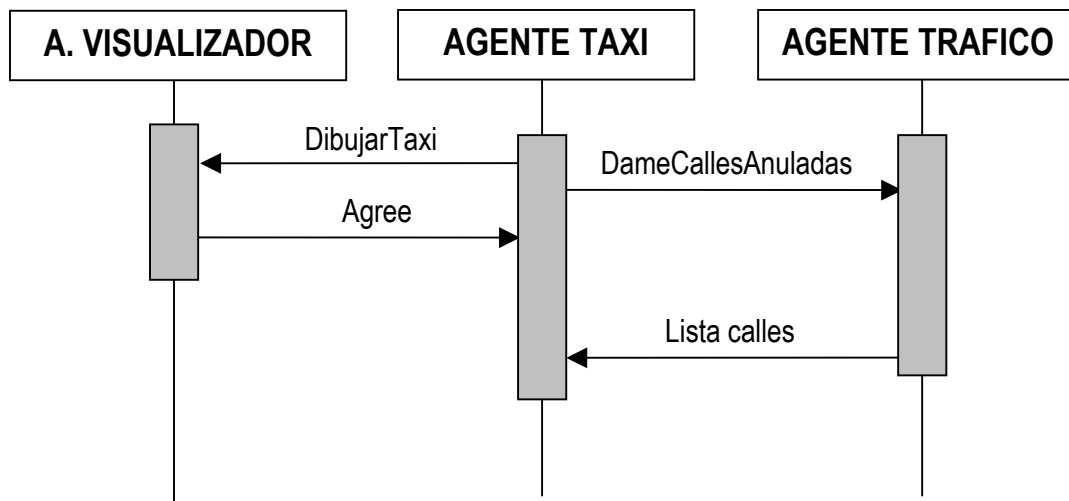


Fig. 15

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.

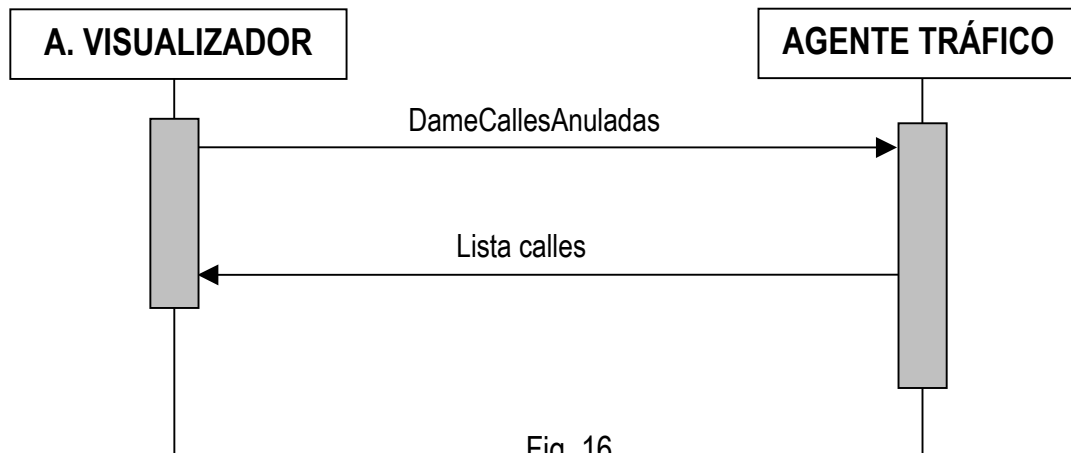


Fig. 16

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 realizador 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.

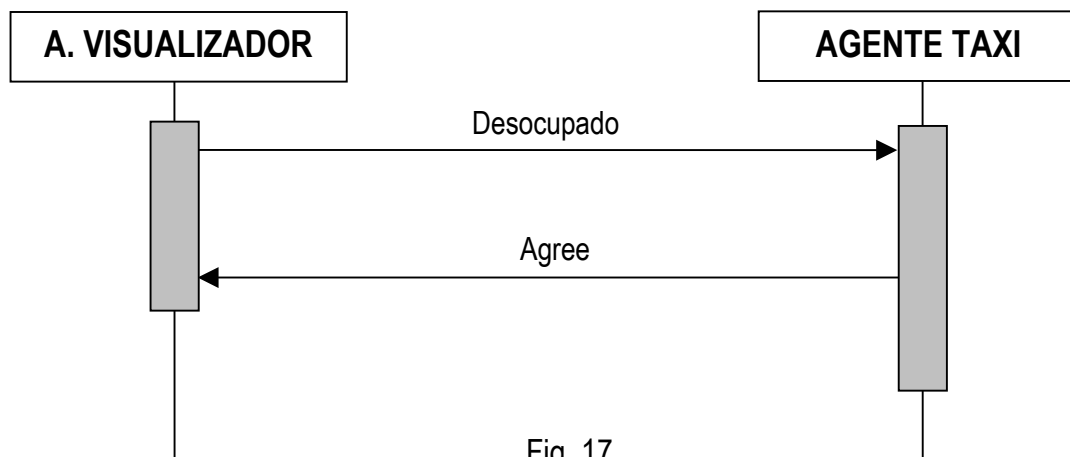


Fig. 17

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

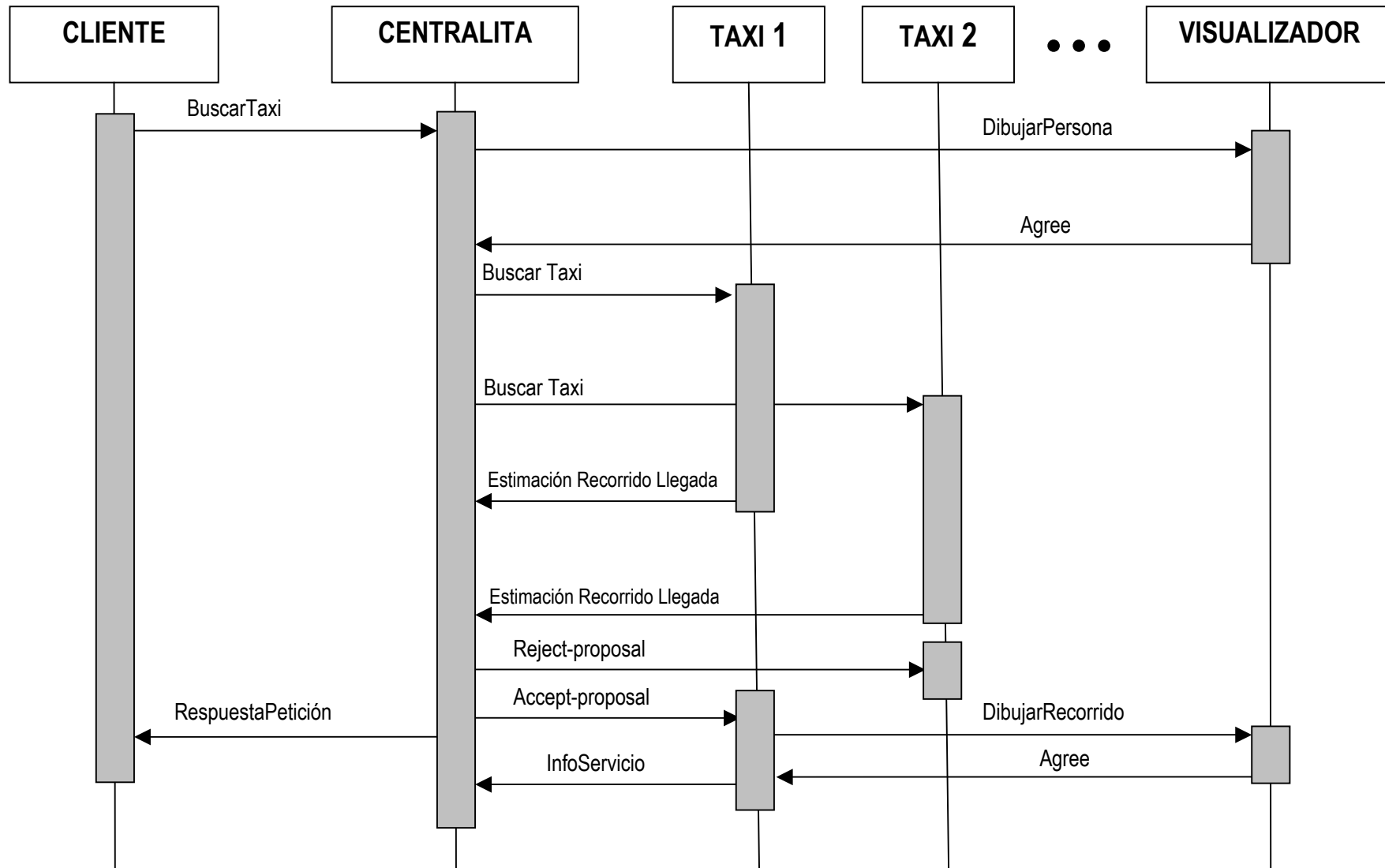


Fig. 18

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

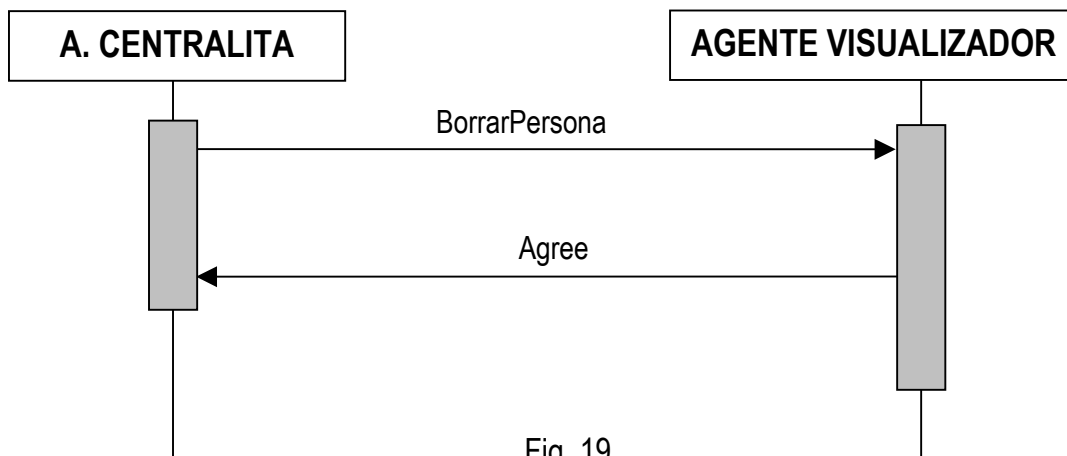


Fig. 19

5. 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.

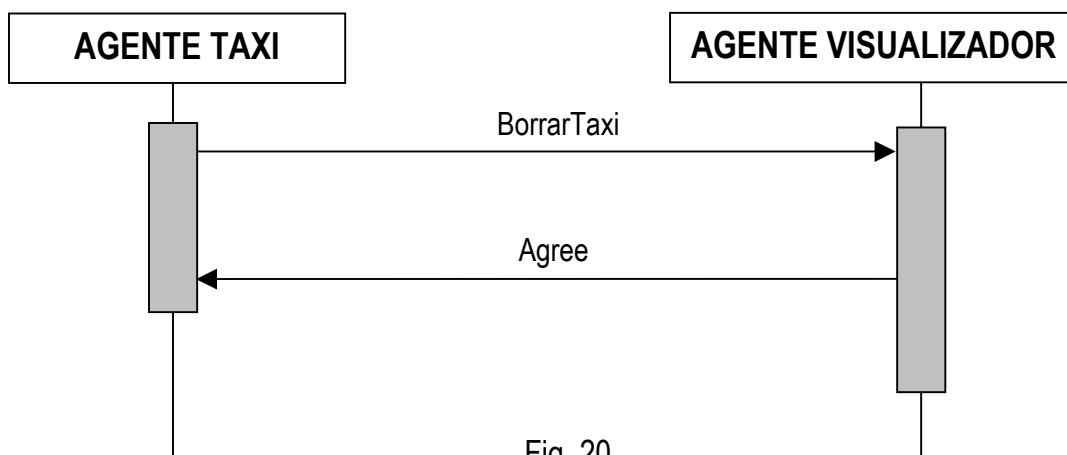


Fig. 20

5.2. Ontología

Como he comentado anteriormente, la definición de la ontología es primordial, ya que, define el vocabulario para que los agentes puedan entenderse.

A continuación se muestran los CONCEPTOS, PREDICADOS y ACCIONES que se han utilizado en el sistema:

5.2.1. 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. Son los objetos que se incluyen en los mensajes. Los objetos de esta ontología son:

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

Nombre Slot	Tipo
X	Integer
Y	Integer

- **PETICIÓN:** Objeto que se utiliza para indicar que un cliente necesita un taxi.

Nombre Slot	Tipo
Numpersonas	Integer
PosCliente	Posición
Destino	Posición

- **TAXITYPE:** Objeto que nos proporciona toda la información que se necesita 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 (o no) el taxi que le ofrecerá el servicio.

Nombre Slot	Tipo
Respuesta	Boolean
IdTaxi	String
Precio	Integer

- **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

5.2.2. 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 (si son necesarios). Las líneas de código dónde realmente indican qué hace dicha acción las tendrían los *agentes receptores*.

- **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 tiene ningún argumento, ya que, no los necesita por que con el nombre de la acción ya se sabe qué se ha de hacer.
- **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 tiene ningún argumento, ya que, no los necesita por que con el nombre de la acción ya se sabe qué se ha de hacer.
- **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 los mismos que BorrarPersona.

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 los mismos que la acción anterior.

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

5.2.3. 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. Para poder diferenciar un agente de otro lo haremos a través de esos parámetros. Los servicios los necesitamos para registrarnos en el DF, ya que, como hemos dicho anteriormente es como “*las páginas amarillas*” del sistema.

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 registrarte en el DF se necesita indicar el TYPE y el NAME del servicio. Ahora les mostraremos un caso práctico para dar un ejemplo. Registrar un agente taxi. En este caso el TYPE=SERVICE_TYPE y el NAME=TAXI_SERVICE. Si necesitamos indicar más información para ser más concretos también podemos crear propiedades de los distintos servicios.

```
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 registrar al DF");
    doDelete();
}
```

6. IMPLEMENTACIÓN

6.1. Clases y directorios

Para estructurar todo el sistema descrito anteriormente, el código se ha dividido en dos módulos bien diferenciados (AGENTES y ONTOLOGY) que pasaremos a describir a continuación.

En el package *Agentes*, (colocado en un 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 las calles y todas las clases encargadas de la interficie gráfica. A continuación haremos una breve descripción de las clases de este package:

6.1.1. AgenteCliente.java: este es el agente asociado a los clientes del sistema. Tiene un comportamiento simple que se encarga de enviar las peticiones a la centralita. El protocolo de comunicación que utiliza es del tipo FIPA-Request y las partes principales que lo forman son:

void RegistrarDF(): método que registra en el DF.

public void Envio(): método que se encarga de enviar la petición a la centralita utilizando el protocolo FIPA-Request.

Peticion readPeticion(String pathname): método que se encarga de leer de fichero la petición del cliente deseado.

6.1.2. 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. 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.

6.1.3. 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 són 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.

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 eliminar 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 aristas, *ninici* es el nodo donde iniciará 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.

6.1.4. 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.

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.

6.1.5. AgenteVisualizador.java: este es el objeto encargado de controlar los comportamientos del agente Visualizador. Tiene dos 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 deseamos 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.

void RegistrarDF(): método que registra en el DF.

ACLMessage EnviarMsgDameCalles(): método que crea el mensaje para enviar la acción *DameCallesAnuladas* a Tráfico.

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.

En dicho módulo también encontramos todos los ficheros de inicialización e información necesarios para el correcto funcionamiento del sistema. En el apartado 4.9 (La Simulación) podréis encontrar toda la información necesaria sobre los ficheros utilizados.

En el package *Ontology*, (colocado en un directorio con su mismo nombre) encontramos todas las clases referentes a la ontología del sistema (*ver apartado 5.2*). A continuación haremos una breve descripción de las clases de este package:

1. **TaxiOntology.java:** clase java donde encontraremos 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 EstimaciónRecorridoLlegada.
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 más información sobre el contenido de cada una de las clases del package ONTOLOGY ver el apartado 5.2 (Ontología).

7. PRUEBAS

A continuación se describe la evolución de un ejemplo, así como un conjunto de pruebas realizadas para comprobar el correcto funcionamiento del sistema.

7.1. Seguimiento de un ejemplo

Para conseguir el caso más general posible se han introducido un conjunto de datos iniciales sobre TAXIS, CLIENTES que realizará una prueba bastante fiable sobre la evolución del sistema. Recordamos que tanto el mapa de la ciudad como las calles anuladas también son inicializadas mediante sus correspondientes ficheros que a continuación se muestran:

Información sobre el mapa de la ciudad utilizado por el sistema (véase apartado 4.9)

DatosCalles.txt:

0 1 0*	31 32 0 42 0*	62 61 4 73 3*	93 92 3*
1 2 0*	32 43 0*	63 52 5 62 3*	94 93 2*
2 3 0 13 1*	33 22 0*	64 63 2 75 3*	95 94 1 106 0*
3 4 0*	34 23 2 33 0*	65 76 0*	96 85 5 95 1*
4 5 0*	35 34 1 46 1*	66 55 0*	97 96 1*
5 6 0 16 2*	36 35 2*	67 78 1*	98 97 2 109 0*
6 7 0 17 0*	37 26 0 36 3*	68 67 0 79 0*	99 88 0 100 1*
7 8 0 18 0*	38 27 5 37 4*	69 68 0 80 2*	100 101 1*
8 9 0*	39 38 1 50 0*	70 59 1 69 0*	101 102 1 112 0*
9 10 0 20 2*	40 39 2 51 2*	71 70 0 82 0*	102 103 1*
10 21 0*	41 40 0*	72 71 1*	103 92 2 104 1*
11 0 0 12 2*	42 41 1 53 0*	73 72 2 84 2*	104 105 1*
12 13 1*	43 54 0*	74 63 5 72 2*	105 106 1 116 0*
13 14 1 24 2*	44 33 0*	75 74 2*	106 107 1 117 0*
14 15 2*	45 34 1 46 3*	76 75 3 87 0*	107 108 1*
15 4 0 16 1*	46 47 1 57 1*	77 66 0*	108 109 0 119 1*
16 17 1*	47 48 0 58 2*	78 77 1 79 1 89 1*	109 120 0*
17 18 1 28 1*	48 37 0 49 2*	79 80 2 90 1*	110 99 0*
18 19 1 29 0*	49 38 5 50 4*	80 81 3 91 2*	111 110 0*
19 20 2*	50 51 1 61 2*	81 70 0 82 3*	112 111 0*
20 31 0*	51 52 2 62 2*	82 83 3*	113 102 1 112 0*
21 32 0*	52 41 5*	83 72 0 84 3 94 1*	114 103 1 113 0*
22 11 0*	53 64 0*	84 85 3 95 0*	115 104 0 114 0*
23 12 0 24 0*	54 53 2 65 0*	85 74 5 86 2*	116 115 0*
24 25 0 35 0*	55 44 0*	86 87 1 97 1*	117 116 0*
25 14 0 26 1*	56 45 1 55 0 67 1*	87 98 0*	118 117 0*
26 15 1 27 1*	57 56 2 68 0*	88 77 0*	119 118 0*
27 16 5 28 0*	58 57 0 69 2*	89 100 1*	120 119 0*
28 29 2 39 1*	59 48 1 58 0*	90 101 3*	
29 30 2 40 1*	60 49 5 59 1*	91 90 4*	
30 31 1*	61 60 5 72 1*	92 81 1 91 2*	

Con el fichero de inicialización anterior conseguiríamos la siguiente estructura grafo, es decir, las calles y la dirección de las mismas de la ciudad:

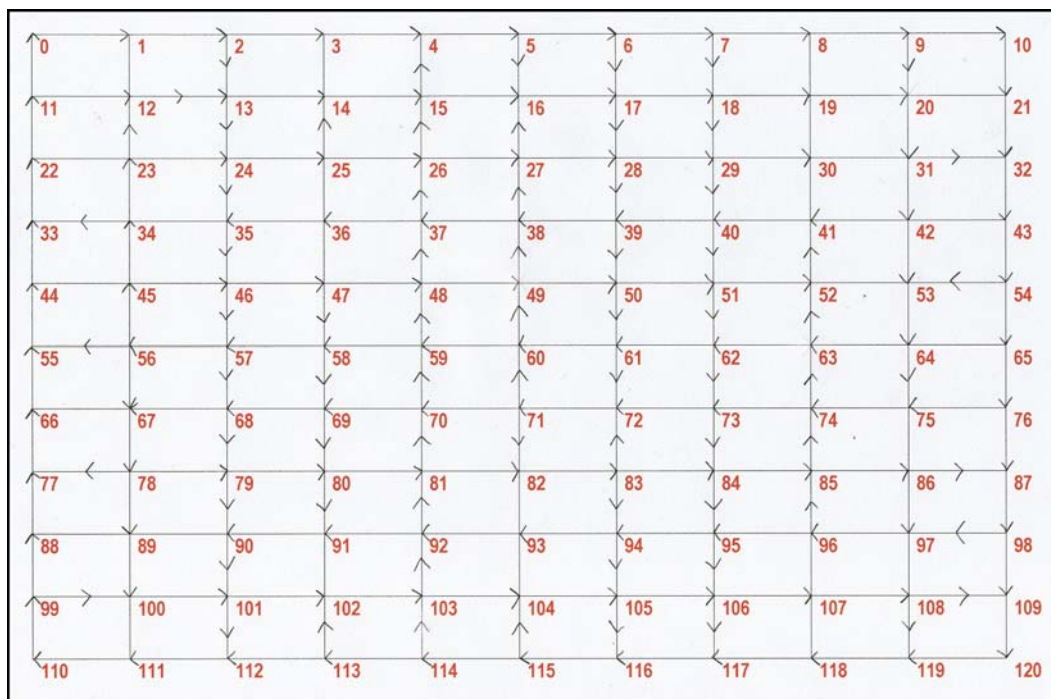


Fig. 21 – Grafo obtenido del fichero de inicialización *DatosCalles.txt*

Información sobre las calles anuladas.

mañana.txt: 25 26 71 60 78 77*

tarde.txt: 18 29 50 51 101 102*

noche.txt: 2 13 86 87 91 92*

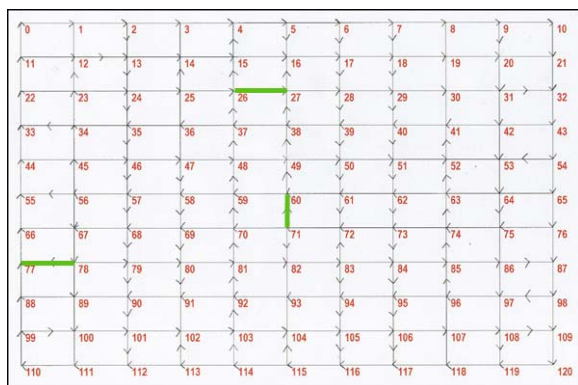


Fig. 22 – Calles anuladas de *mañana.txt*

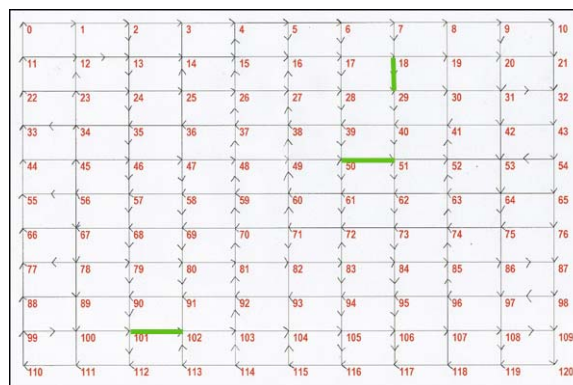


Fig. 23 – Calles anuladas de *tarde.txt*

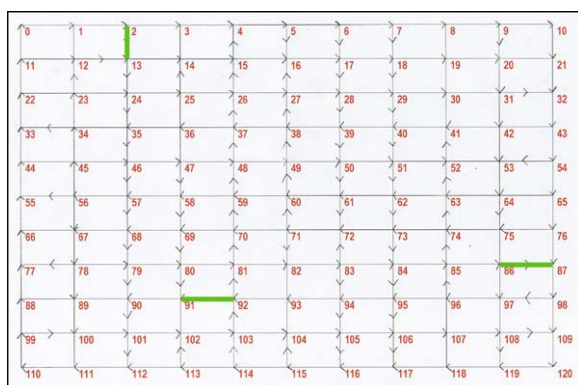


Fig. 24 – Calles anuladas de *noche.txt*

Información sobre los diferentes taxis que se cargarán en el sistema.

taxi1.txt:	taxi4.txt:	taxi7.txt:
0001	0004	0007
Juan Infante García	Daniel Casanova Haba	Maribel Clemente Cascales
12545125	97347168	19436759
B-5705-SJ	B-8912-SW	L-1234-SW
SEAT	PEUGEOT	WOLKSWAGEN
CORDOBA	206	BEATTLE
mañana	tarde	noche
3 1	3 7	1 1
taxi2.txt:	taxi5.txt:	taxi8.txt:
0002	0005	0008
Manolo Castillo Ramos	Israel Infante García	Juan Infante García
11237658	33224567	12545125
B-4418-LM	M-1354-SJ	B-5705-SJ
RENAULT	FORD	SEAT
CLIO	MONDEO	CORDOBA
noche	mañana	mañana
8 5	3 9	5 4
taxi3.txt:	taxi6.txt:	taxi9.txt:
0003	0006	0009
Isaac Infante García	Maribel Clemente Cascales	Manolo Castillo Ramos
39446688	19436759	11237658
B-1234-OM	L-1234-SW	B-4418-LM
FORD	WOLKSWAGEN	RENAULT
SCORT	BEATTLE	CLIO
tarde	noche	tarde
9 7	7 2	7 6

Información sobre los diferentes clientes que pedirán un taxi en el sistema.

Cliente1.txt: 3 1 2 5 6	Cliente6.txt: 5 1 1 1 9	Cliente11.txt: 4 5 5 1 10
Cliente2.txt: 4 7 1 4 6	Cliente7.txt: 4 0 0 1 5	Cliente12.txt: 3 9 2 5 6
Cliente3.txt: 1 8 6 10 10	Cliente8.txt: 1 5 6 7 2	Cliente13.txt: 4 6 6 4 6
Cliente4.txt: 1 6 3 8 2	Cliente9.txt: 1 10 10 9 9	Cliente14.txt: 4 10 10 9 10
Cliente5.txt: 5 1 1 1 9	Cliente10.txt: 1 0 0 5 4	Cliente15.txt: 1 8 8 10 10

A continuación se muestra un ejemplo de la interface tal y como queda dependiendo de la hora de iniciación del sistema (mañana, tarde, noche) con los taxis, clientes y calles anuladas anteriormente indicados:

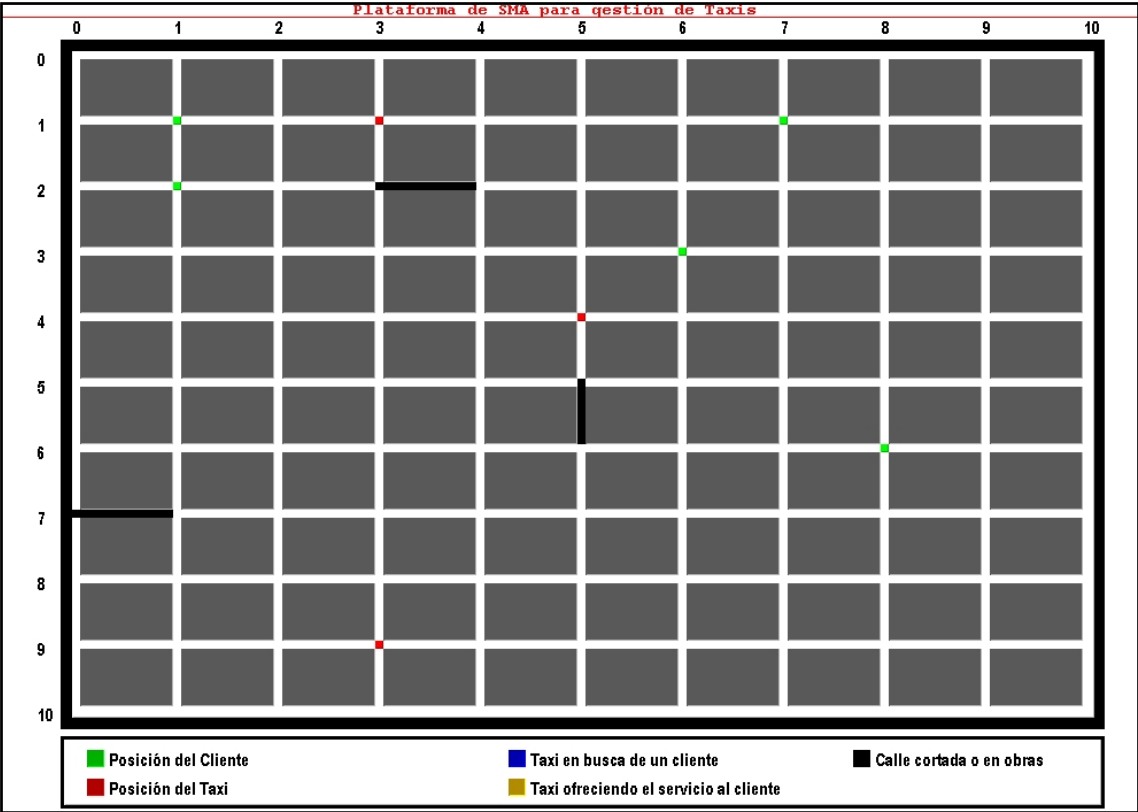


Fig. 25 – Ejemplo de la interficie iniciado por la mañana

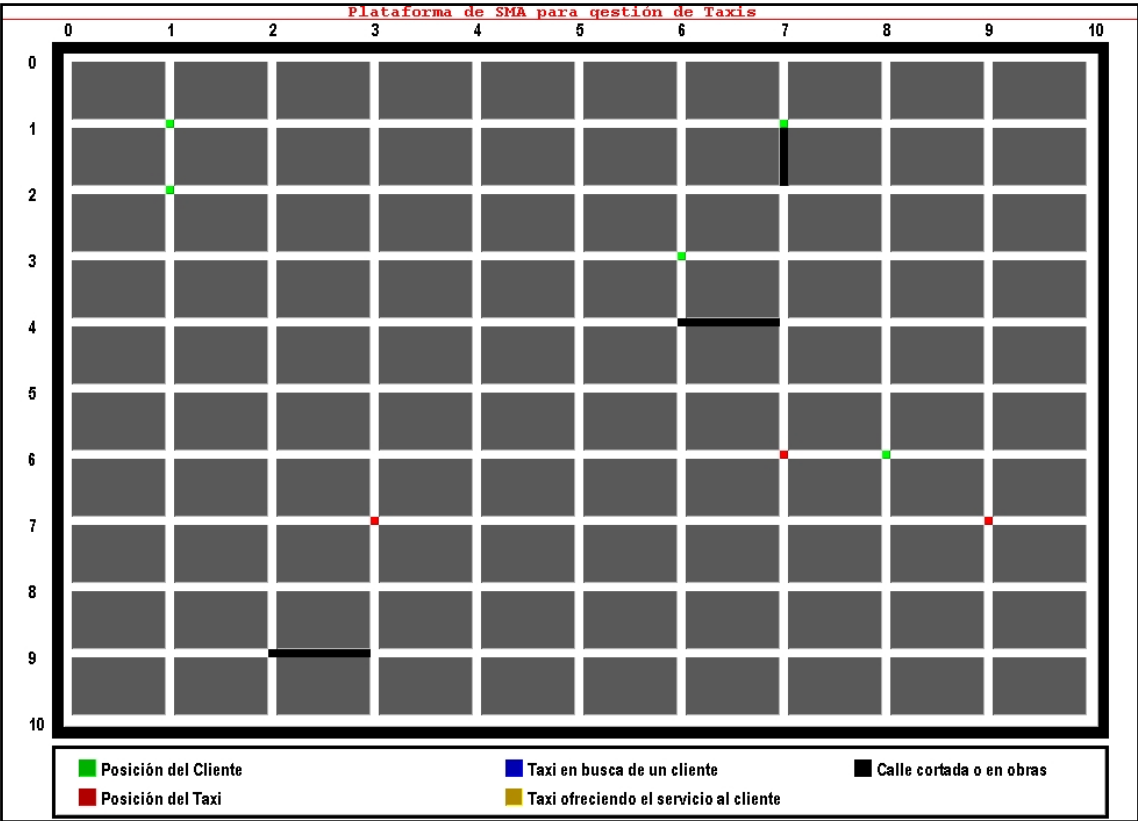
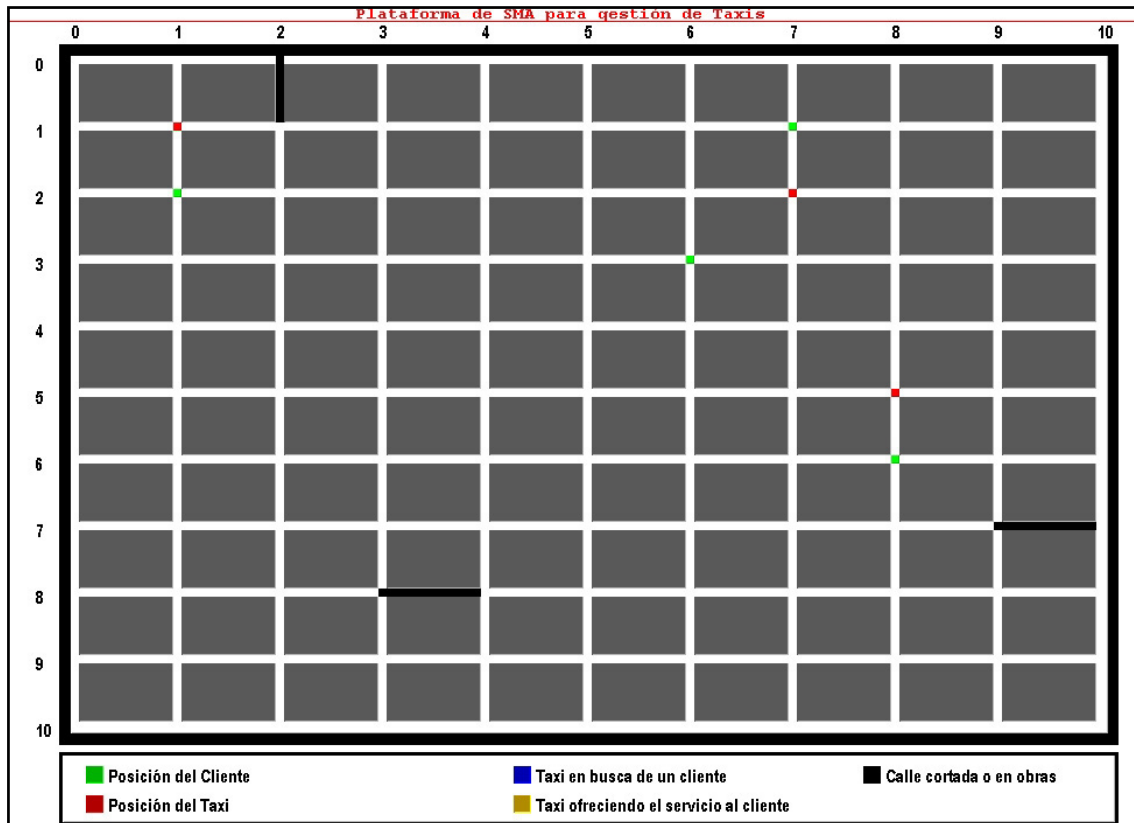


Fig. 26 – Ejemplo de la interficie iniciado por la tarde

Fig. 27 – Ejemplo de la interficie iniciado por la noche

Una vez definidos los datos iniciales de los ficheros, arrancaremos el sistema con el siguiente comando:

```
java jade.Boot -gui -platform Centralita:Agentes.AgenteCentralita Trafico:Agentes.AgenteTrafico
Visualizador:Agentes.AgenteVisualizador
Taxi1:Agentes.AgenteTaxi Taxi2:Agentes.AgenteTaxi Taxi3:Agentes.AgenteTaxi Taxi4:Agentes.AgenteTaxi
Taxi5:Agentes.AgenteTaxi Taxi6:Agentes.AgenteTaxi Taxi7:Agentes.AgenteTaxi Taxi8:Agentes.AgenteTaxi
Taxi9:Agentes.AgenteTaxi
Cliente1:Agentes.AgenteCliente Cliente2:Agentes.AgenteCliente Cliente3:Agentes.AgenteCliente
Cliente4:Agentes
```

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

-gui: especifica que deseamos visualizar el entorno gráfico de JADE, para posteriormente poder realizar más pruebas.

Centralita: Agentes.AgenteCentralita: 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.

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: 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.

Cliente1:Agentes.AgenteCliente: Indica que deseamos cargar en la plataforma un agente denominado CLIENTE1 que será controlado a través de la clase java AgenteCliente que se encuentra en un paquete llamado Agentes.

La simulación ha sido ejecutada a las 0:55h de la noche. Recordamos que la hora del sistema influye en su ejecución. A continuación observaréis en la figura 28 cual es la situación inicial de cada uno de los clientes y taxis en la ciudad.

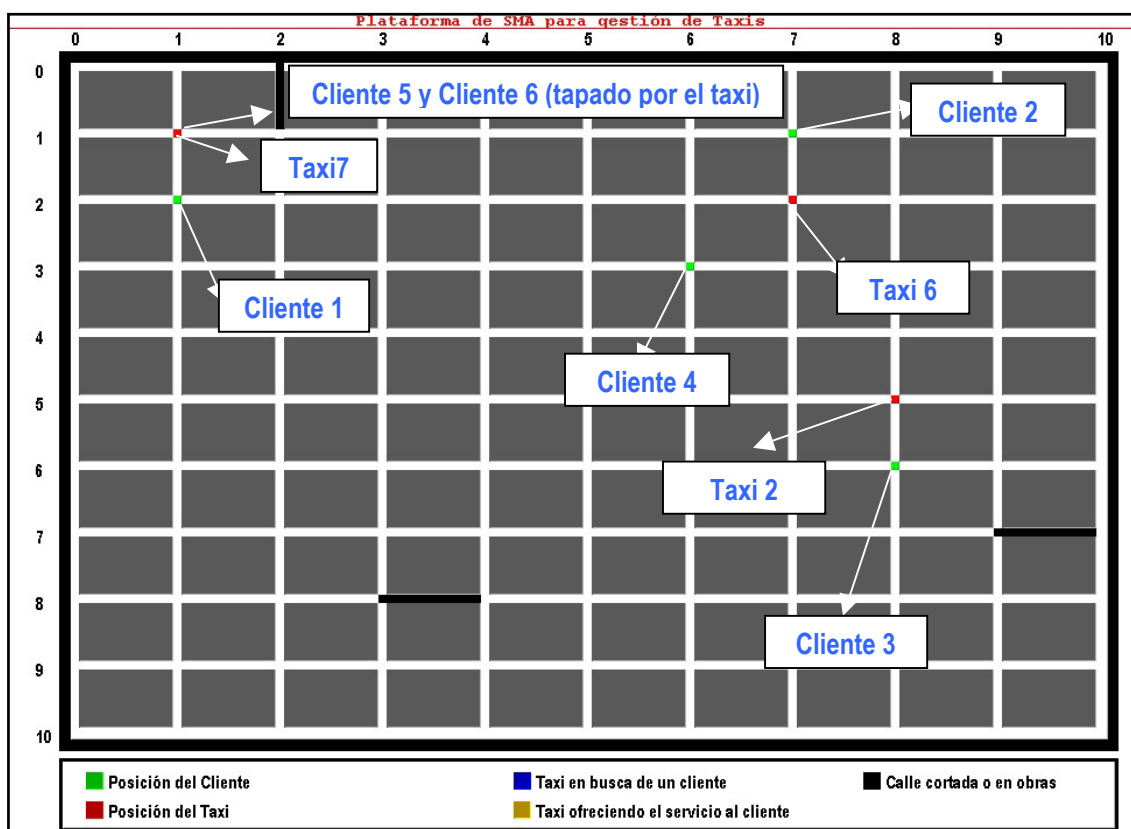


Fig. 28 – Situación inicial de cada uno de los items de la simulación

Llegados a este punto, los agentes Centralita, Tráfico, Visualizador, Cliente6, Cliente1, Cliente5, Cliente3, Cliente2, Cliente4 se registran en el DF. Como la simulación se ha realizado a las 0:55 solo los taxis 6,2,7 (turno de noche) se darán de alta en el DF. El orden de registro de los clientes influye en el resultado, ya que, se gestionan como una cola FIFO (First In First Out). Tráfico envía las calles anuladas al Visualizador y a los diferentes Taxis para que actualicen el mapa. Este proceso no se ejecutará hasta la próxima actualización (que en un principio deberían ser 8 horas, pero es un minuto aproximadamente para la simulación).

El Cliente 6 (el primero en registrarse), accede a su fichero de inicialización y realiza la petición a través de la Centralita. La Centralita avisa a los taxis que en ese momento trabajan (6,2,7), y eso lo sabe porque son los únicos que están dados de alta en el DF. Éstos realizan sus correspondientes propuestas dependiendo de dónde estén situados y a dónde deben dirigirse y se lo envían a la Centralita y ésta toma la decisión de quién debe ir a buscar al Cliente 6. En este caso como el Taxi 7 están en la misma posición que el Cliente 6, es éste el que se encarga de realizar el servicio (ya que es el que está más próximo). Centralita avisa a Taxi7 con un ACCEPTPROPOSAL y al resto de los taxis con un REJECTPROPOSAL. (véase figura 29)

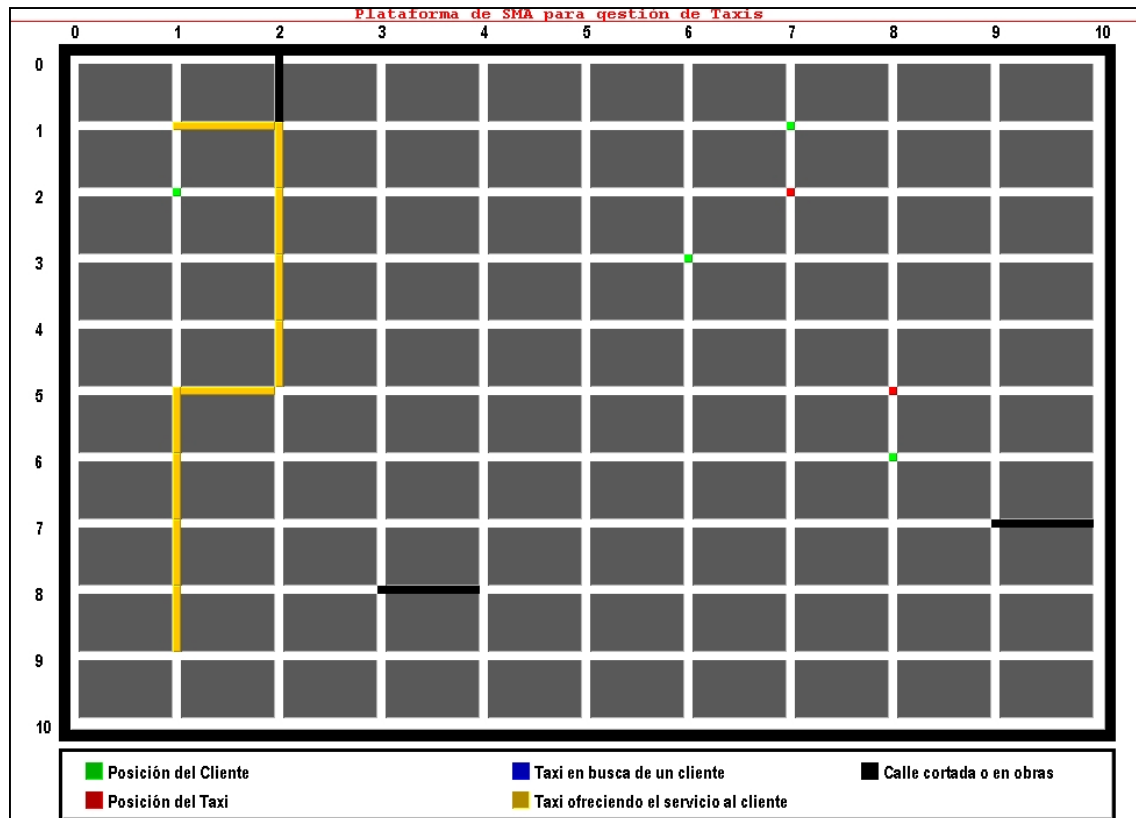


Fig. 29 – Taxi7 sirviendo al Cliente 6

El siguiente cliente a tratar es el Cliente1 (que está en el nodo 23, figura 21). Es exactamente el mismo procedimiento que anteriormente hemos descrito. Esta vez el taxi más cercano (y que no estaba ocupado con otro cliente en otro momento) es el Taxi6 (que se encuentra en el nodo 29, figura 21). Se puede observar que sigue el camino más corto teniendo en cuenta la dirección de las calles (figura 21).

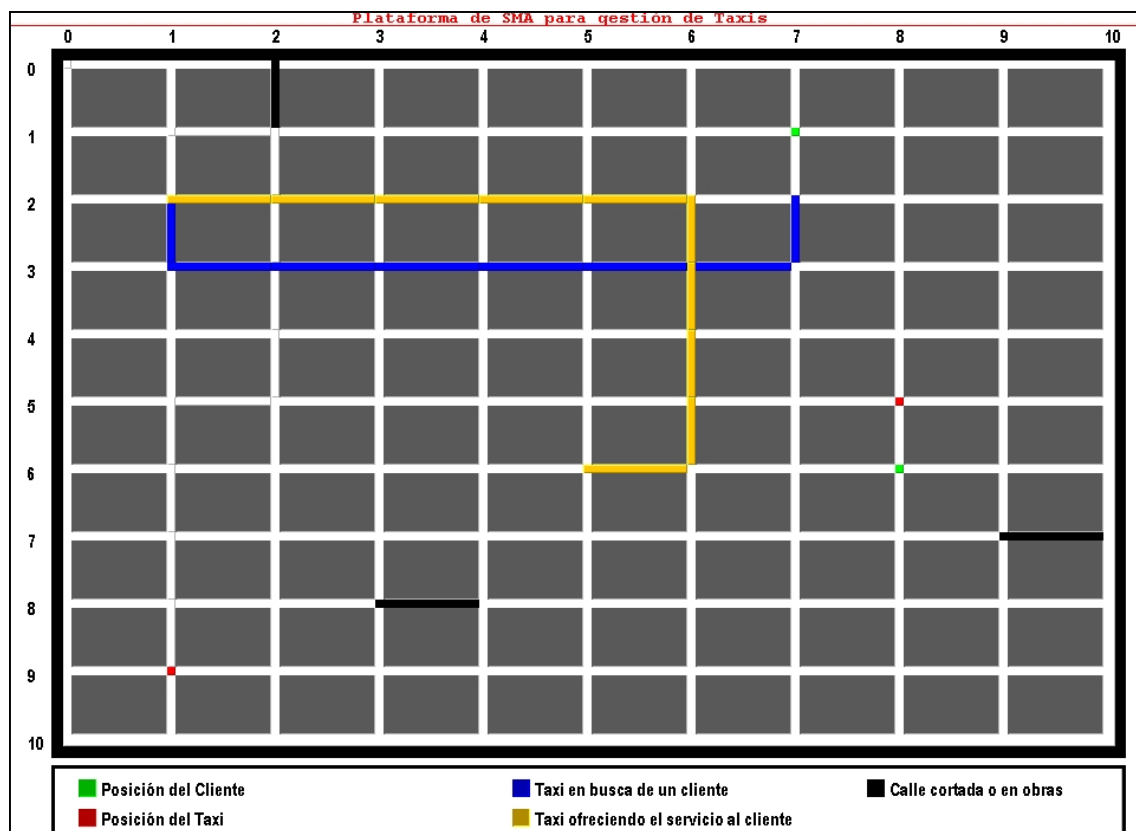


Fig. 30 – Taxi 6 sirviendo al Cliente 1

El siguiente cliente a tratar es el Cliente5. Esta vez el taxi más cercano (y que no estaba ocupado con otro cliente en otro momento) es el Taxi2.

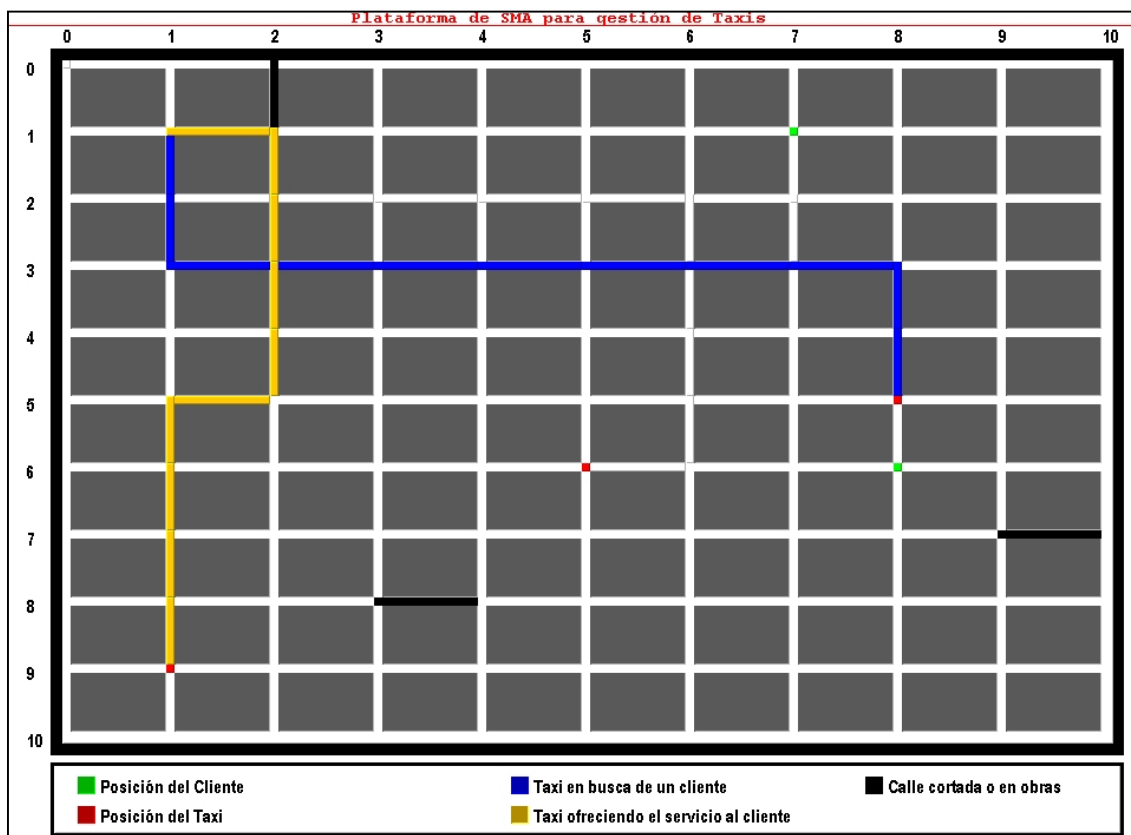


Fig. 31 – Taxi 2 sirviendo al Cliente 5

NOTA: Recordamos que el color azul en la interficie indicaba el recorrido que ha de realizar el taxi hasta llegar al cliente, y el color naranja indicaba el recorrido que el taxi realiza para llevar al cliente, es decir, recorrido de servicio.

Como hemos indicado anteriormente la Centralita trata a los clientes como una cola FIFO y va tratándolos mientras haya taxis libres. Ya se han realizado 3 servicios y sólo habían 3 taxis en la plataforma, es decir, los Clientes 3,2,4 no se les puede servir por ahora. No hay taxis libres. Se les avisa y posteriormente la Centralita vuelve a enviar la petición a los taxistas para saber si hay alguno libre. En este caso volvemos a tener una negativa, ya que, aún están ocupados. La Centralita vuelve a enviar las peticiones y es cuando el Taxi 7 se encuentra desocupado y puede servir al Cliente 3.

Recordamos que el Taxi7 anteriormente había servido al Cliente 6 entonces está situado en la posición 1,9. En la figura 32 se puede observar cual ha sido el recorrido utilizado por el Taxi7.

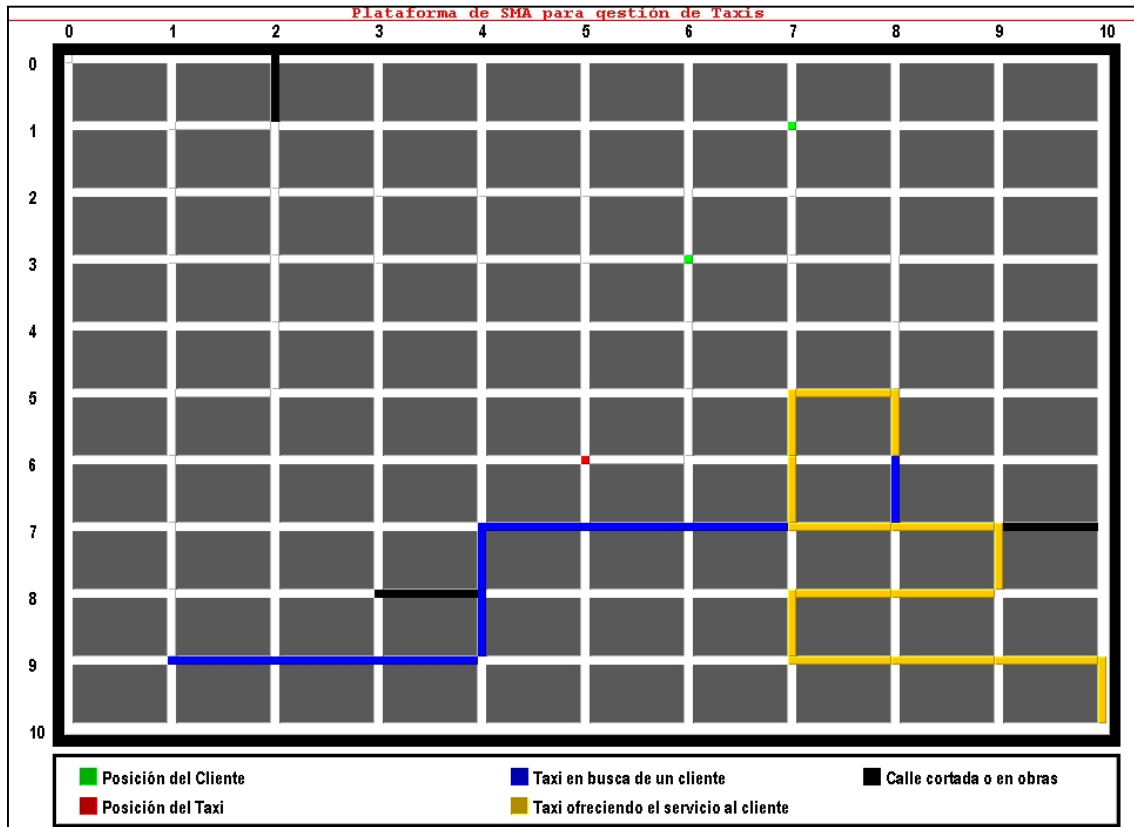


Fig. 32 – Taxi7 sirviendo al Cliente 2

Los Clientes 2,4 aún no se les ha podido realizar el servicio, ya que, todos los taxis de la plataforma están ocupados. La Centralita continúa enviando las peticiones.

Finalmente al Cliente4 se le puede servir. El Taxi6 está desocupado y es el que se va a encargar de realizar el servicio.

El único Cliente que queda en el sistema es el Cliente2 y al pasar unos segundos observamos como el Taxi2 se encarga de realizarle el servicio.

Una vez el sistema ha gestionado todas las peticiones que tenía resulta que ha pasado el tiempo estipulado para que se realice automáticamente el cambio de turno. Esto implica dos acciones relevantes:

- El cambio de turno de los taxis, es decir, se darán de baja (se eliminarán del DF) los Taxis 7,6,2 que son los que tenían el turno de noche y se darán de alta (se registrarán en el DF) los Taxis 8,5,1 que son los que realizan el turno de mañana.
- Actualización de las calles cortadas. Los Taxis que se han dado de alta en el DF (8,5,1) actualizarán sus calles cortadas por si ha habido algún cambio importante en ellas. El agente Visualizador, para que tenga concordancia con los datos de los taxis, también enviará un mensaje a Tráfico para que le envíe las calles anuladas.

En este momento la interfaz gráfica queda de la siguiente forma:

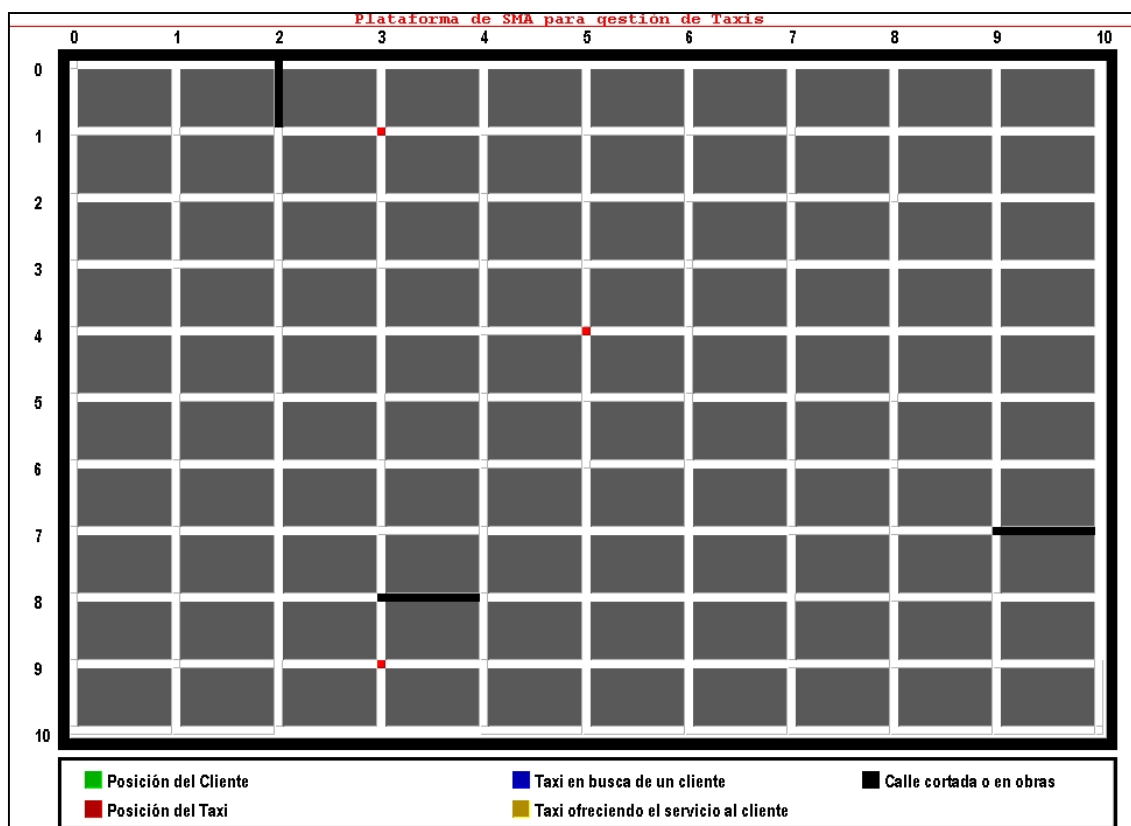


Fig. 33 – Cambio de turno de noche a mañana

Todos y cada uno de los clientes que han pasado por el sistema y se les ha realizado el servicio automáticamente se les elimina del mismo, no sin antes haberle enviado un mensaje con toda la información que les interesa saber. Dicha información contiene el taxi que les irá a buscar y el precio que les costará. En futuras actualizaciones la información que se le da al cliente puede ser ampliada.

Pasado un tiempo, se observa que cuando volvemos al turno de noche los taxis están en la misma posición en la que habían acabado su último turno.

Como se ha comentado anteriormente hemos ejecutado la línea de comandos con el parámetro **(-gui)** eso quería decir que se ha ejecutado la plataforma utilizando la interficie gráfica de JADE. Esto nos resulta muy útil, ya que, podemos realizar más pruebas si las necesitamos. En la siguiente figura se observa cómo podemos incluir nuevos clientes en la plataforma:

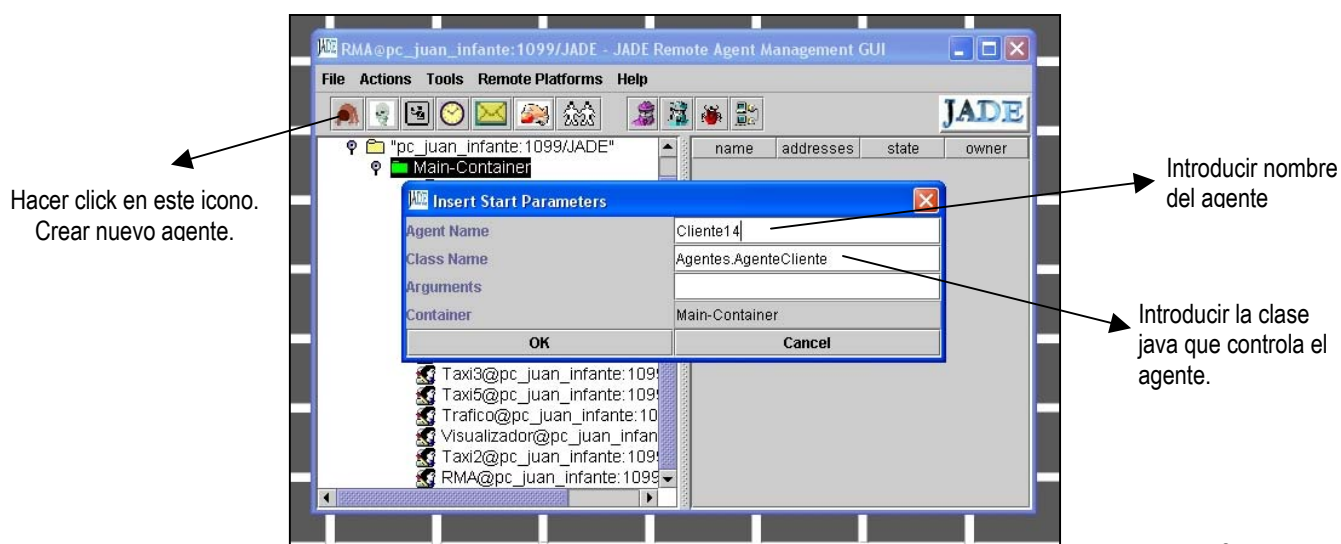


Fig. 34 – Int.gráfica de JADE para crear un nuevo agente

Todo el seguimiento que hemos realizado ha sido a través de la interficie gráfica del sistema, aunque otra posibilidad es realizar el seguimiento a través de la consola de mensajes. Dicha consola es una ventana de MS-DOS en la que se visualizan los mensajes que se van enviando los diferentes agentes para poder solucionar el problema. A continuación se muestran los diferentes mensajes que se pueden encontrar de cada uno de los agentes del sistema:

```

Proyecto Taxi
[Taxi7]: HA RECIBIDO EL CFP CORRECTAMENTE Y VAMOS A ENVIAR UNA PROPUESTA
[Taxi7]: CAMINO ENCONTRADO!!!!!!!
[Taxi7]: CAMINO ENCONTRADO!!!!!!!
[Centralita]: HA RECIBIDO UN REQUEST DE BUSCARTAXI Y VA A INICIAR UN CFP
[Centralita]:buscando los agentes taxis...
[Centralita]:ha encontrado 3 agentes TAXI.
[Centralita]:HA RECIBIDO TODAS LAS PROPUESTAS HABIDAS Y POR HABER
[Centralita]:HA ESCOGIDO UNA PROPUESTA DE TODAS LAS ENVIADAS.....
[Cliente2]: HA RECIBIDO UN INFORM.....(SE HA REALIZADO EL SERVICIO CORRECTAMENTE)
-----
EL TAXI QUE LE IRA A BUSCAR ES: Taxi6@pc_juan_infante:1099/JADE
EL SERVICIO LE COSTARA:(centimos)
720
[Taxi6]:HA RECIBIDO UN ACCEPTPROPOSAL////////////////////////////////////
[Taxi2]:HA RECIBIDO UN REJECT
[Taxi7]:HA RECIBIDO UN REJECT
[Taxi2]: HA RECIBIDO EL CFP CORRECTAMENTE Y VAMOS A ENVIAR UNA PROPUESTA
[Taxi2]: CAMINO ENCONTRADO!!!!!!!
[Taxi2]: CAMINO ENCONTRADO!!!!!!!
[Taxi7]: HA RECIBIDO EL CFP CORRECTAMENTE Y VAMOS A ENVIAR UNA PROPUESTA
[Taxi7]: CAMINO ENCONTRADO!!!!!!!
[Taxi7]: CAMINO ENCONTRADO!!!!!!!
[Cliente2]:EL AGENTE HA SIDO ELIMINADO DEL DF
[Taxi6]: HA RECIBIDO EL CFP CORRECTAMENTE Y VAMOS A ENVIAR UNA PROPUESTA
[Taxi6]: CAMINO ENCONTRADO!!!!!!!
[Taxi6]: CAMINO ENCONTRADO!!!!!!!
[Centralita]:HA RECIBIDO TODAS LAS PROPUESTAS HABIDAS Y POR HABER
[Cliente4]: HA RECIBIDO UN FAILURE$$$$$$$$$$$$$$$$$$$$$(NO HAY TAXIS LI
[Taxi2]:HA RECIBIDO UN REJECT
[Taxi7]:HA RECIBIDO UN REJECT
[Taxi6]:HA RECIBIDO UN REJECT
[Cliente4]:esta accediendo a la PETICION del fichero...
[Centralita]: HA RECIBIDO UN REQUEST DE BUSCARTAXI Y VA A INICIAR UN CFP
[Centralita]:buscando los agentes taxis...
[Centralita]:ha encontrado 3 agentes TAXI.
[Taxi2]: HA RECIBIDO EL CFP CORRECTAMENTE Y VAMOS A ENVIAR UNA PROPUESTA
[Taxi6]: HA RECIBIDO EL CFP CORRECTAMENTE Y VAMOS A ENVIAR UNA PROPUESTA
[Taxi6]: CAMINO ENCONTRADO!!!!!!!
[Taxi6]: CAMINO ENCONTRADO!!!!!!!
[Taxi2]: CAMINO ENCONTRADO!!!!!!!
[Taxi2]: CAMINO ENCONTRADO!!!!!!!

```

Fig. 35 – Mensajes de los agentes en la simulación

AGENTE CLIENTE

[Cliente1]: SE HA REGISTRADO EN EL DF: El Cliente1 desea un taxi y se inicia en el sistema. No será eliminado hasta haberle realizado dicho servicio.

[Cliente1]: EL AGENTE HA SIDO ELIMINADO DEL DF: Como ya le ha sido realizado el servicio se elimina del sistema.

[Cliente1]: ESTÁ ACCEDIENDO A LA PETICIÓN DEL FICHERO: Indica que el Cliente1 está leyendo su fichero de inicialización.

[Cliente1]: HA RECIBIDO UN INFORM.....(Se ha realizado el servicio correctamente): Centralita le está avisando al Cliente1 de que su servicio se ha realizado correctamente. En este momento se le informa al Cliente con el siguiente mensaje:

El taxi que le irá a buscar es: Taxi7@pc_juan_infante:1099/JADE

El servicio le costará (céntimos): 600

[Cliente1]: HA RECIBIDO UN FAILURE\$\$\$\$\$\$\$\$\$(no hay taxis libres en la ciudad): En ese momento no había ningún taxi libre en la ciudad y la centralita le avisa a través de un

FAILURE.

AGENTE CENTRALITA

[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 (la más cercana al cliente).

AGENTE TAXI

[Taxi1]: SE HA REGISTRADO EN EL DF: El Taxi1 se ha registrado en el DF, es decir, es su turno de trabajo.

[Taxi1]: 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.

[Taxi1]: HA RECIBIDO UN CFP CORRECTAMENTE Y VAMOS A ENVIAR UNA PROPUESTA: El Taxi1 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.

[Taxi1]: CAMINO ENCONTRADO !!!!!!!!: El agente Taxi1 ha encontrado el camino que le había pedido la Centralita (búsqueda de cliente (azul) o realización de servicio (naranja)).

[Taxi1]: HA RECIBIDO UN ACCEPT PROPOSAL !!!!!!!!!!!!!!!: La Centralita ha escogido al Taxi1 para que realice un servicio.

[Taxi1]: HA RECIBIDO UN REJECT: La Centralita no lo ha escogido para realizar un servicio a un cliente.

[Taxi1]: HA RECIBIDO LAS CALLES ANULADS DE TRÁFICO: Ha recibido correctamente las calles anuladas que tráfico le ha enviado.

AGENTE TRÁFICO

[Tráfico]: SE HA REGISTRADO EN EL DF: Indica que se ha iniciado en el sistema.

[Tráfico]: 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.

[Tráfico]: EL ENVÍO DE LA LISTA DE CALLES SE HA HECHO CORRECTAMENTE: El Taxi1 ha recibido las calles anuladas de Tráfico correctamente.

[Tráfico]: 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.

AGENTE VISUALIZADOR

[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. CÓDIGO FUENTE

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

8.1. TAXIONTOLOGY.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
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;
    }

    /**
     *
     *
     */
    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(RESPUESTAPETICION), RespuestaPeticion.class);
        add(new ConceptSchema(LISTACALLES), ListaCalles.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 (RESPUESTAPETICION);
        cs.add("respuesta", booleanSchema);
        cs.add("idtaxi", stringSchema);
        cs.add("precio", integerSchema);

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

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

    }
}

```

8.2. TAXITYPE.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
13-7-2002

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;
    }
}
```

8.3. RESPUESTAPETICION.JAVA

```
/******  
Juan Infante García  
PFC SMA para la gestión de Taxis de una ciudad  
13-7-2002  
  
ONTOLOGÍA (clases de java asociadas)  
*****/  
package Ontology;  
  
import jade.content.*;  
  
public class RespuestaPeticion implements Concept {  
  
    private boolean Respuesta;  
    private String IDTaxi;  
    private int Precio;  
  
    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;  
    }  
  
}
```

8.4. RECORRIDOTAXI.JAVA

```
/******  
Juan Infante García  
PFC SMA para la gestión de Taxis de una ciudad  
13-7-2002  
  
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 listacalleservir;  
  
    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 listacalleservir;  
    }  
}
```

```

    }

    public void setListaCallesServir(ListaCalles l) {
        this.listacallesServir=l;
    }

}

```

8.5. POSICION.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
13-7-2002

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;
    }
}

```

8.6. PETICION.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
13-7-2002

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;
    }
}
```

8.7. LISTACALLES.JAVA

```
/******
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
13-7-2002

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;
    }
}

```

8.8. ESTIMACIONRECORRIDOLLEGADA.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
13-7-2002

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;
    }
}

```

8.9. DIBUJARTAXI.JAVA

```

/*****
Juan Infante García

```

PFC SMA para la gestión de Taxis de una ciudad
13-7-2002

```
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;
    }
}
```

8.10. DIBUJARRECORRIDOTAXI.JAVA

```
/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
13-7-2002

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;
    }
}
```

8.11. DIBUJARPERSONA.JAVA

```
/******
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
13-7-2002

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;
    }
}
```

8.12. DESOCUPADO.JAVA

```
/******
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
13-7-2002

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

import jade.content.*;

public class Desocupado implements AgentAction {

    public Desocupado() { }
}
```

8.13. DAMECALLESANULADAS.JAVA

```
/******  
Juan Infante García  
PFC SMA para la gestión de Taxis de una ciudad  
13-7-2002  
  
ONTOLOGÍA (clases de java asociadas)  
*****/  
package Ontology;  
import jade.content.*;  
public class DameCallesAnuladas implements AgentAction {  
    public DameCallesAnuladas() { }  
}
```

8.14. BUSCARTAXI.JAVA

```
/******  
Juan Infante García  
PFC SMA para la gestión de Taxis de una ciudad  
13-7-2002  
  
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;  
    }  
}
```

8.15. BORRARTAXI.JAVA

```
/******  
Juan Infante García  
PFC SMA para la gestión de Taxis de una ciudad  
13-7-2002  
  
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;
    }
}
```

8.16. BORRARRECORRIDOTAXI.JAVA

```
/******
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
13-7-2002

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;
    }
}
```

8.17. BORRARPERSONA.JAVA

```
/******
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
13-7-2002

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;
    }
}
```

8.18. AGENTECENTRALITA.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
18-10-2002

AGENTE CENTRALITA
Agente que representa a la centralita de los taxis.
Su correcto funcionamiento es primordial para el buen
resultado de la plataforma.
*****/
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

    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 DFAgents = new ArrayList();
            List agentsFound = new ArrayList();
            DFAgentDescription dfd = new DFAgentDescription();
            ServiceDescription sd = new ServiceDescription();
            sd.setType(type);
            sd.setName(name);
            dfd.addServices(sd);

            try {
                DFAgents = DFServiceCommunicator.search(this,dfd);
            } catch (FIPAException e) {
                System.out.println "["+this.getLocalName()+"]:"+ "Se ha producido un error:
"+e.getMessage());
            }

            Iterator DFAgentsIt = DFAgents.iterator();
            while (DFAgentsIt.hasNext()) {
                DFAgentDescription agent = (DFAgentDescription)DFAgentsIt.next();
                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
                }

            }

            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) {
            List DFAgents = new ArrayList();
            String agentFound = new String();
            DFAgentDescription dfd = new DFAgentDescription();
            ServiceDescription sd = new ServiceDescription();
            sd.setType(type);

```

```

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

try {
    DFAgents = DFServiceCommunicator.search(this,dfd);
} catch (FIPAException e) {
    System.out.println("[ "+getLocalName()+" ]:"+ "Se ha producido un error: "+e.getMessage());
}

Iterator DFAgentsIt = DFAgents.iterator();
while (DFAgentsIt.hasNext()) {
    DFAgentDescription agent = (DFAgentDescription)DFAgentsIt.next();
    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
    }
}
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) {
    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();
    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;
            }
            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.getListasCallesServicio();
        RespuestaPeticion rp=new RespuestaPeticion();
        rp.setRespuesta(true);
        rp.setIDTaxi((msg.getSender()).getName());
        rp.setPrecio(longitud_camino(l.getListasCalles())*PRECIOPASO);

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

} //end of Agent Centralita

```

8.19. AGENTECLIENTE.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
18-10-2002

AGENTE CLIENTE
Agente que representa los clientes que solicitan los taxis.
Es el encargado de iniciar todo el proceso.
*****/

package Agentes;

import Ontology.*;

import java.io.*;
import java.lang.String;
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 {

    Agent miCliente=this;
    //Característica del Agente Cliente. PETICIÓN (Todo cliente debe tener una
    petición en curso)
    Peticion pet= new Peticion();

    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) {
        List DFAgents = new ArrayList();
        String agentFound = new String();
        DFAgentDescription dfd = new DFAgentDescription();
        ServiceDescription sd = new ServiceDescription();
        sd.setType(type);
        sd.setName(name);
        dfd.addServices(sd);

        try {
            DFAgents = DFServiceCommunicator.search(this,dfd);
        } catch (FIPAException e) {
            System.out.println("[ "+getLocalName()+" ]:"+ "Se ha producido un error:
"+e.getMessage());
        }

        Iterator DFAgentsIt = DFAgents.iterator();
        while (DFAgentsIt.hasNext()) {
            DFAgentDescription agent = (DFAgentDescription)DFAgentsIt.next();
            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
        }
    }
    return agentFound;
}

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

    //Creamos el mensaje y rellenamos los campos claves
    ACLMessage msg = new ACLMessage(ACLMMessage.REQUEST);

    AID receiver = new
AID(getAgentDF(TaxiOntology.SERVICE_TYPE,TaxiOntology.CENTRALITA_SERVICE),false);

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

    //Obtenemos la petición de un fichero
    try {
        System.out.println("[ "+getLocalName()+" ]: "+ "esta accediendo a la PETICION del
fichero...");
        pet= readPeticion(this.getLocalName()+".txt");
    }catch(Exception e) {
        System.out.println("[ "+getLocalName()+" ]: "+ "Error leyendo del fichero
"+this.getLocalName()+".txt "+": "+e.getMessage());
    }

    //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 ontologia 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();
        }

    } //Fin de RegistrarDF

//*****
//*****

    //Metodo que obtiene las peticiones de un fichero (NOMBRE DEL FICHERO:
<NombreAgente>.txt)
    Peticion readPeticion(String pathname) throws Exception
    {
        Peticion pet=new Peticion();
        Posicion p1=new Posicion();
        Posicion p2=new Posicion();

        FileReader profile = new FileReader(pathname);
        BufferedReader b= new BufferedReader(profile);

        String miS=null;
        miS=b.readLine();
        String s=new String();int i=0;
        while (miS.charAt(i)!=' ') { s=s+miS.charAt(i);i++;}
        int x = Integer.parseInt(s);s="";i++;
        pet.setnumpersonas(x);
        while (miS.charAt(i)!=' ') { s=s+miS.charAt(i);i++;}
    }

```

```
x = Integer.parseInt(s); s=""; i++;
p1.setx(x);
    while (miS.charAt(i)!=' ') { s=s+miS.charAt(i); i++;}
x = Integer.parseInt(s); s=""; i++;
p1.sety(y);
    while (miS.charAt(i)!=' ') { s=s+miS.charAt(i); i++;}
x = Integer.parseInt(s); s=""; i++;
p2.setx(x);
    while      (miS.charAt(i)!='          '           &&            ((i+1)!=miS.length()))
{s=s+miS.charAt(i); i++;}
    s=s+miS.charAt(i); x = Integer.parseInt(s);
p2.sety(y);
pet.setposcliente(p1);
pet.setdestino(p2);
    profile.close();
    return pet;
}

//*****
*****

//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());
    }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);
    }

    /*******
    //Metodo setup que todo Agente tiene
    /*******
    public void setup() {
        RegistrarDF();
        Envio();
    }

    /*******
    //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

```

8.20. AGENTETAXI.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
5-11-2002

AGENTE TAXI
Agente que representa los Taxis de la plataforma.
*****/
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;
    Date deathLine = new Date(System.currentTimeMillis() + MILLIS_PER_8HOURS);
    DatosMapa dm=new DatosMapa();
    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) {
List DFAgents = new ArrayList();
String agentFound = new String();
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(type);
sd.setName(name);
dfd.addServices(sd);

try {
    DFAgents = DFServiceCommunicator.search(this,dfd);
} catch (FIPAException e) {
    System.out.println "["+getLocalName()+"]:"+ "Se ha producido un error:
"+e.getMessage());
}

Iterator DFAgentsIt = DFAgents.iterator();
while (DFAgentsIt.hasNext()) {
    DFAgentDescription agent = (DFAgentDescription)DFAgentsIt.next();
    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
    }
}
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,false
        );

        return taxi;
    }//Fin de readPeticion

//*****
//*****
//Método que crea el mensaje para enviar la acción DAMECALLESANULADAS a Trafico
ACLMMessage 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.gety()
),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_cliente.
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.getListasCalles()));

        bc = new RequestInitiator(miTaxi, borcliente);
        addBehaviour(bc);

        caract_taxi.setPosActual(ultima_pos(l2.getListasCalles()));
        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 cuando 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 borra físicamente el cami redundante
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.
getFil(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 algoritmo 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() {
//REGISTRAMOS la ontologia y el lenguaje
manager.registerLanguage(codec,codec.getName());
manager.registerOntology(ontology,ontology.getName());

    try {
        caract_taxi=readInfoTaxi(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

```

8.21. AGENTETRAFICO.JAVA

```

//*****
Juan Infante Garcia
PFC SMA para la gestión de Taxis de una ciudad
09-11-2002

AGENTE TRAFICO
Agente que se encarga de informarnos de las anomalías que
se pueden encontnrar en las calles de la ciudad.
*****/

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 ontologia 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 cuando 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(ACLMMessage.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(ACLMMessage.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("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("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("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

```

8.22. AGENTEVISUALIZADOR.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
22-11-2002

AGENTE VISUALIZADOR
Agente que se encarga de realizar todas las tareas de
visualización del mapa, recorridos, clientes, taxis...
*****/

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) {
java.util.List DFAgents = new java.util.ArrayList();
String agentFound = new String();
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(type);
sd.setName(name);
dfd.addServices(sd);
try {
    DFAgents = DFServiceCommunicator.search(this,dfd);
} catch (FIPAException e) {
    System.out.println "["+getLocalName()+"]:"+ "Se ha producido un error:
"+e.getMessage());
}
Iterator DFAgentsIt = DFAgents.iterator();
while (DFAgentsIt.hasNext()) {
    DFAgentDescription agent = (DFAgentDescription)DFAgentsIt.next();
    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
    }
}
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
ACLMMessage EnviarMsgDesocupado(ACLMMessage req){

    //Creamos el mensaje y rellenamos los campos claves
    ACLMessage msg = new ACLMessage(ACLMMessage.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 cuando 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.getListasCallesBuscar();
        ListaCalles l2=new ListaCalles();l2=rt.getListasCallesServir();
        ar = new AccionARealizar(3,0,0,Color.blue,l1.getListasCalles());
        jade.util.leap.List aux=l1.getListasCalles(); //Esto lo hago por si
el taxi se encuentra en la misma
        if(aux.size()!=0){ //posición que el
cliente (long de la lista 0) para que
            v.addFigura(ar);
            map.pinta();
        }
        try{Thread.sleep(3000);}catch(Exception e){}
        ar = new AccionARealizar(3,0,0,Color.orange,l2.getListasCalles());
        v.addFigura(ar);
        map.pinta();
        try{Thread.sleep(8000);}catch(Exception e){}
        ar = new AccionARealizar(3,0,0,Color.white,l1.getListasCalles());
        if(aux.size()!=0){ //Esto lo hago por si el
taxi se encuentra en la misma
            v.addFigura(ar); //posición que el cliente
(long de la lista 0) para que
            map.pinta();
        }
        ar = new AccionARealizar(3,0,0,Color.white,l2.getListasCalles());
        v.addFigura(ar);
        map.pinta();
        if(aux.size()!=0){
            Posicion p=new
Posicion();p=primera_pos(l1.getListasCalles()); //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.getListasCalles()); //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,ListasCallesAnuladas.getListasCalles());
                v.addFigura(ar);
                map.pinta();
            }
            Action a=(Action)manager.extractContent(inform);
            System.out.println("[ "+getLocalName()+" ]:"+" HA RECIBIDO LAS CALLES
ANULADAS DE TRAFICO");
            ListasCallesAnuladas=(ListasCalles)a.getAction();
            ar = new
AccionARealizar(3,0,0,Color.black,ListasCallesAnuladas.getListasCalles());
            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

```

8.23. ACCIONAREALIZAR.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
10-11-2002
*****/

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;
    }
}

```

8.24. COORDENADAS.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
10-11-2002
*****/

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];
    }
}

```

8.25. COORDENADAS2.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
10-11-2002
*****/

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,
        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];
    }
}

```

8.26. DATOSMAPA.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
10-11-2002
*****/
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

    public DatosMapa() {
        try {

```

```

        readInfoMapa();
    }catch(Exception e) {
        System.out.println("Error leyendo del fichero de DATOS DEL MAPA");
    }
}

public void readInfoMapa() throws Exception
{
    FileReader profile = new FileReader("DatosCalles.txt");
    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;
}
}

```

8.27. INFOMAP.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
10-11-2002
*****/
package Agentes;
import java.util.*;
public class InfoMap {
    int succ;
    int coste;
}

```

8.28. MAPA.JAVA

```

/*****
Juan Infante García
PFC SMA para la gestión de Taxis de una ciudad
10-11-2002
*****/
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) {
        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){

                    contextoDobleBuffer.fill3DRect(c.getCol(c1),c.getFil(min(f1,f2)),calle,alto+calle,tru
ue);

                }else {

```

```

contextoDobleBuffer.fill3DRect(c.getCol(min(c1,c2)),c.getFil(f1),ancho+calle,calle,t
rue);

        }

    }
} 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,tr
ue);

        }else{

contextoDobleBuffer.fill3DRect(c.getCol(min(c1,c2)),c.getFil(f1),ancho+calle,calle,t
rue);

        }

    }

}

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);

//Aquí 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;
}
}

```

8.29. VENTANA.JAVA

```
/******  
Juan Infante García  
PFC SMA para la gestión de Taxis de una ciudad  
10-11-2002  
*****/  
  
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. VALORACIÓN Y CONCLUSIONES

Los agentes suponen un gran avance en la Inteligencia Artificial. El hecho de que diversos individuos trabajen conjuntamente para conseguir un objetivo común, les asigna una capacidad de reacción y de decisión bastante compleja. Un agente por si solo no tendría mucho éxito al intentar resolver problemas relativamente complejos, pero la combinación y cooperación de un grupo de ellos hace que sus comportamientos sean bastante inteligentes. Es en este punto donde los sistemas Multi-Agente demuestran que pueden ser útiles para resolver ciertos problemas que, individualmente, serían demasiado complicados de resolver.

Aunque cada vez se consiguen más méritos en SMA es un campo muy novedoso de la Inteligencia Artificial y aún está en proceso de desarrollo, aunque gracias a organismos internacionales como la FIPA que definen estándares y promueve su utilización, se consiguen avances con mucha más rapidez.

La realización de este proyecto ha supuesto todo un reto personal por varias razones. Requería un estudio previo y un aprendizaje de una materia completamente desconocida, ya que, sólo se dieron pinceladas teóricas en la asignatura de Introducción a la Inteligencia Artificial que Aïda Valls y Toni Moreno impartían. El montar desde cero un sistema de estas características, implicaba la visión global de toda la carrera, y me ha resultado tremendamente útil saber que lo he estudiado durante estos últimos años lo he podido aplicar de forma práctica.

La mayor dificultad con la que me he hallado en el desarrollo del sistema ha sido el haber tenido que realizarlo desde Barcelona, ya que, eso ha hecho que cuando surgía algún problema o inconveniente tardase mucho más tiempo en resolverlos, sin embargo, a su vez, ha resultado muy positivo por todo lo que he aprendido.

Por último me gustaría agradecer el interés demostrado por mi tutora Aïda Valls, ya que, con la base teórica y la gran cantidad de documentación que me ha proporcionado ha hecho que las dificultades iniciales fuesen más llevaderas. Sin olvidarme de mis compañeros de GruSMA por la aportación de sus proyectos, en especial a David Sánchez por la realización de la documentación *How to Upgrade Source Code Written for Jade 2.3 to 2.6* que me resultó de gran ayuda.

10. FUTUROS PROYECTOS

Aún se puede realizar mucho trabajo en este proyecto para poder mejorarlo y darle más funcionalidad, tal y como se muestra a continuación:

Utilización de los costes de cada una de las calles de la ciudad. Con esto conseguimos obtener más información sobre el servicio a realizar y con ello conseguir unos mejores resultados para el cliente. Son muy útiles para obtener recorridos más cortos y menos costos y con ello reducir substancialmente el tiempo de espera del cliente.

Posibilidad de cambiar el criterio de asignación de taxi para un cliente. Es uno de los puntos clave del SMA. Con él podemos conseguir darle más funcionalidad e inteligencia al sistema, ya que, es el encargado de tomar la decisión final de asignación de vehículo.

Los clientes deberían estar informados con todo lujo de detalles sobre el servicio que se les va a realizar. Más adelante deberíamos ser más detallistas a la hora de facilitar la información al cliente. Un cliente informado es un cliente satisfecho.

Como hemos explicado en el apartado 7.1, la gestión de los diferentes clientes por parte del agente Centralita se realiza a través de una cola FIFO, es decir, el primer cliente que entra en el sistema será el primero en servir. Esto es susceptible de mejora si se estudiase previamente la cola de clientes y se gestionase de forma más inteligente. Con ello conseguiríamos una gran mejora en los resultados finales, tanto para el cliente como para el taxista, agente a tener más en cuenta en futuros cambios.

Para futuras actualizaciones deberíamos dotar de mayor capacidad de tomar decisiones al agente Taxi y hacer que entre ellos se comuniquen. También se deberían incluir en dicho agente varios algoritmos de búsqueda de recorrido dependiendo de las necesidades del cliente.

Personalizar más los servicios es algo que está a la orden del día. Cada cliente tiene unas características y necesidades totalmente diferentes de otro. Conseguir que el cliente personalice sus necesidades y se vean reflejadas en el servicio realizado es algo a tener en cuenta para cambios futuros.

Ya que las características de los SMA nos lo permiten, podemos realizar una combinación de varios SMA que ofrezcan servicios a la comunidad referente a transportes públicos. Un ejemplo sería combinar el sistema que hemos desarrollado junto a otro que ofrezca servicios de Autobuses y dotar al cliente la posibilidad de elección entre los dos tipos de transporte.

Como se puede observar, el trabajo a realizar es aún muy extenso y el hecho de que se trabaje con una tecnología tan nueva que aún está en periodo de cambios hace que el proceso de aprendizaje sea continuo y que se replanteen ciertas ideas y objetivos.

11. BIBLIOGRAFÍA

[AgentCities] AgentCities, web page, <http://www.agentcities.org/> , 15-12-2002

[FIPA] Foundation for Intelligent Physical Agents, web page, <http://www.fipa.org/>, 20-11-2002

[GruSMA] GruSMA, Grup de Sistemes Multi-Agent, DEIM, Universitat Rovira i Virgili, plana web, <http://www.etsr.urv.es/recerca/rgai/toni/MAS/>, 2-1-2003

[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

[Sánchez, 2001] Sánchez, D. (2001), Un sistema Multi-Agent d'ajuda a l'assignació d'òrgans en trasplantaments. El mètode ClusDM amb criteris lingüístics, Projecte Final de Carrera, Enginyeria Tècnica en Informàtica de Sistemes, Universitat Rovira i Virgili, 2001

[Sánchez, 2002] Sánchez, D. (2001), How to upgrade source code written for Jade 2.3 to 2.6, Universitat Rovira i Virgili, 2001

[MAS] Agentes y Sistemas Multiagente, <http://montealegre.ei.uvigo.es/agentes/>, 10-11-2002

[Wooldridge, 2002] Wooldridge, M., An introduction to multiagent systems, John Wiley & Sons (2002)

[SPB Software house] Automated airport taxis dispatch System, <http://www.softspb.com/enterprise/casestudies/airport.html> , 9-1-2003

[Mobile Computer System, Inc] TaxiCentral.com, <http://www.taxicentral.com/> , 9-1-2003

[SIGTEC] mobiCABII Taxi Dispatch System, http://www.sigtec.com.au/taxi_mobiCABii.htm , 9-1-2003

[FST] Five Sixes Taxi, <http://www.keywesttaxi.com/Reservation.htm> , 9-1-2003